

Givaro

Generated by Doxygen 1.9.5



<b>1 Givaro Documentation.</b>	<b>1</b>
1.1 Introduction . . . . .	1
1.2 Goals . . . . .	1
1.3 Design . . . . .	1
1.4 Using Givaro . . . . .	1
1.5 Contributing to Givaro, getting assistance. . . . .	2
<b>2 Tutorial.</b>	<b>3</b>
<b>3 Licence.</b>	<b>5</b>
<b>4 Installation.</b>	<b>7</b>
<b>5 Architecture.</b>	<b>9</b>
<b>6 Bibliography</b>	<b>11</b>
<b>7 Bug List</b>	<b>13</b>
<b>8 Todo List</b>	<b>15</b>
<b>9 Test List</b>	<b>17</b>
<b>10 Module Index</b>	<b>19</b>
10.1 Modules . . . . .	19
<b>11 Namespace Index</b>	<b>21</b>
11.1 Namespace List . . . . .	21
<b>12 Hierarchical Index</b>	<b>23</b>
12.1 Class Hierarchy . . . . .	23
<b>13 Data Structure Index</b>	<b>27</b>
13.1 Data Structures . . . . .	27
<b>14 File Index</b>	<b>31</b>
14.1 File List . . . . .	31
<b>15 Module Documentation</b>	<b>35</b>
15.1 Givaro . . . . .	35
15.1.1 Detailed Description . . . . .	35
15.2 bstruct . . . . .	35
15.2.1 Detailed Description . . . . .	36
15.3 GMP . . . . .	36
15.4 Integer . . . . .	36
15.5 Memory . . . . .	36
15.5.1 Detailed Description . . . . .	37
15.6 Rationals . . . . .	37

15.6.1 Detailed Description	37
15.7 ZRing	37
15.8 System	37
15.8.1 Detailed Description	38
<b>16 Namespace Documentation</b>	<b>39</b>
16.1 Givaro Namespace Reference	39
16.1.1 Detailed Description	45
16.1.2 Function Documentation	45
16.1.2.1 inv()	45
16.1.2.2 invin()	45
16.1.2.3 gcd()	46
16.1.2.4 pp()	46
16.1.2.5 lcm() [1/2]	46
16.1.2.6 lcm() [2/2]	46
16.1.2.7 pow() [1/2]	47
16.1.2.8 pow() [2/2]	47
16.1.2.9 powmod()	47
16.1.2.10 sign()	48
16.1.2.11 compare()	48
16.1.2.12 absCompare()	48
16.1.2.13 isZero()	49
16.1.2.14 nonZero()	49
16.1.2.15 isOne()	49
16.1.2.16 fact()	49
16.1.2.17 sqrt() [1/2]	49
16.1.2.18 sqrtrem() [1/2]	50
16.1.2.19 sqrt() [2/2]	50
16.1.2.20 sqrtrem() [2/2]	50
16.1.2.21 root()	50
16.1.2.22 logp()	51
16.1.2.23 logtwo()	51
16.1.2.24 naturallog()	51
16.1.2.25 swap()	51
16.1.2.26 isOdd()	52
16.1.2.27 length()	52
16.1.2.28 operator>>()	53
16.1.2.29 operator<<()	53
16.1.2.30 absOutput()	53
16.1.2.31 operator+()	53
16.1.2.32 operator"!=()	54
16.1.2.33 operator==( ) [1/2]	54

16.1.2.34 operator>()	54
16.1.2.35 operator<()	55
16.1.2.36 operator>=()	55
16.1.2.37 operator<=()	55
16.1.2.38 operator%()	55
16.1.2.39 operator*()	56
16.1.2.40 operator-()	56
16.1.2.41 operator==( ) [ 2 / 2 ]	56
16.2 Reclnt Namespace Reference	56
16.2.1 Detailed Description	56
16.3 std Namespace Reference	57
16.3.1 Detailed Description	57
16.3.2 Function Documentation	57
16.3.2.1 operator<<() [ 1 / 4 ]	57
16.3.2.2 operator<<() [ 2 / 4 ]	58
16.3.2.3 operator<<() [ 3 / 4 ]	58
16.3.2.4 operator<<() [ 4 / 4 ]	58
<b>17 Data Structure Documentation</b>	<b>61</b>
17.1 __giv_map_less_ith< T, UNARYOP, ith > Struct Template Reference	61
17.1.1 Detailed Description	61
17.2 __givdom_trait_name< T > Struct Template Reference	61
17.2.1 Detailed Description	61
17.3 _perfArray0< T > Struct Template Reference	62
17.3.1 Detailed Description	62
17.4 Array0< T > Class Template Reference	62
17.4.1 Detailed Description	64
17.4.2 Member Function Documentation	64
17.4.2.1 allocate()	64
17.4.2.2 reallocate()	64
17.4.2.3 copy()	64
17.4.3 Field Documentation	65
17.4.3.1 _size	65
17.5 Array0Tag Class Reference	65
17.5.1 Detailed Description	65
17.6 ArrayAllocatort< T, Tag > Class Template Reference	65
17.6.1 Detailed Description	67
17.6.2 Member Function Documentation	67
17.6.2.1 allocate()	67
17.6.2.2 reallocate()	67
17.6.2.3 copy()	67
17.6.3 Field Documentation	68

17.6.3.1 <code>_size</code>	68
17.7 <code>ArrayFixed&lt; T, SIZE &gt;</code> Class Template Reference	68
17.7.1 Detailed Description	68
17.8 <code>BaseDomain&lt; T &gt;</code> Class Template Reference	69
17.8.1 Detailed Description	69
17.9 <code>BaseTimer</code> Class Reference	69
17.9.1 Detailed Description	69
17.10 <code>Bits</code> Class Reference	69
17.10.1 Detailed Description	70
17.11 <code>BlocFreeList</code> Class Reference	70
17.11.1 Detailed Description	70
17.12 <code>ChineseRemainder&lt; Ring, Domain, REDUCE &gt;</code> Struct Template Reference	70
17.12.1 Detailed Description	70
17.13 <code>ChineseRemainder&lt; Ring, Domain, false &gt;</code> Struct Template Reference	71
17.13.1 Detailed Description	71
17.14 <code>Degree</code> Class Reference	71
17.14.1 Detailed Description	71
17.15 <code>ElemConstRef&lt; T &gt;</code> Struct Template Reference	71
17.15.1 Detailed Description	72
17.16 <code>ElemRef&lt; T &gt;</code> Struct Template Reference	72
17.16.1 Detailed Description	72
17.17 <code>Extension&lt; BFT &gt;</code> Class Template Reference	72
17.17.1 Detailed Description	72
17.18 <code>FermatDom</code> Class Reference	73
17.18.1 Detailed Description	73
17.19 <code>GeneralRingNonZeroRandIter&lt; Ring, RandIter &gt;</code> Class Template Reference	73
17.19.1 Detailed Description	73
17.20 <code>GeneralRingRandIter&lt; Ring &gt;</code> Class Template Reference	74
17.20.1 Detailed Description	74
17.21 <code>GF2</code> Class Reference	74
17.21.1 Detailed Description	77
17.21.2 Constructor & Destructor Documentation	78
17.21.2.1 <code>GF2()</code>	78
17.21.3 Member Function Documentation	78
17.21.3.1 <code>operator=()</code>	78
17.21.3.2 <code>init()</code>	78
17.21.3.3 <code>convert()</code>	79
17.21.3.4 <code>assign()</code>	79
17.21.3.5 <code>cardinality()</code>	80
17.21.3.6 <code>characteristic()</code>	80
17.21.3.7 <code>areEqual()</code>	80
17.21.3.8 <code>isZero()</code>	81

17.21.3.9 isOne()	81
17.21.3.10 isUnit()	82
17.21.3.11 isMOne()	82
17.21.3.12 write() [1/2]	82
17.21.3.13 read() [1/3]	83
17.21.3.14 write() [2/2]	83
17.21.3.15 read() [2/3]	83
17.21.3.16 read() [3/3]	84
17.21.3.17 add() [1/2]	84
17.21.3.18 add() [2/2]	85
17.21.3.19 sub() [1/2]	85
17.21.3.20 sub() [2/2]	86
17.21.3.21 mul() [1/2]	86
17.21.3.22 mul() [2/2]	87
17.21.3.23 div() [1/2]	87
17.21.3.24 div() [2/2]	87
17.21.3.25 neg() [1/2]	88
17.21.3.26 neg() [2/2]	88
17.21.3.27 inv() [1/2]	89
17.21.3.28 inv() [2/2]	89
17.21.3.29 axpy() [1/2]	90
17.21.3.30 axpy() [2/2]	90
17.21.3.31 axmy() [1/2]	90
17.21.3.32 axmy() [2/2]	91
17.21.3.33 maxpy() [1/2]	91
17.21.3.34 maxpy() [2/2]	92
17.21.3.35 addin() [1/2]	92
17.21.3.36 addin() [2/2]	93
17.21.3.37 subin() [1/2]	93
17.21.3.38 subin() [2/2]	94
17.21.3.39 mulin() [1/2]	94
17.21.3.40 mulin() [2/2]	94
17.21.3.41 divin() [1/2]	95
17.21.3.42 divin() [2/2]	95
17.21.3.43 negin() [1/2]	96
17.21.3.44 negin() [2/2]	96
17.21.3.45 invin() [1/2]	96
17.21.3.46 invin() [2/2]	97
17.21.3.47 axpyin() [1/2]	97
17.21.3.48 axpyin() [2/2]	98
17.21.3.49 axmyin() [1/2]	98
17.21.3.50 axmyin() [2/2]	98

17.21.3.51 maxpyin() [1/2]	99
17.21.3.52 maxpyin() [2/2]	99
17.21.3.53 maxCardinality()	100
17.22 GFqDom< TT > Class Template Reference	100
17.22.1 Detailed Description	100
17.23 GFqExt< TT > Class Template Reference	101
17.23.1 Detailed Description	101
17.24 GFqExtFast< TT > Class Template Reference	101
17.24.1 Detailed Description	102
17.25 GFqKronecker< TT, Ints > Struct Template Reference	102
17.25.1 Detailed Description	102
17.26 GIV_Extensionrandlter< ExtensionField, Type > Class Template Reference	102
17.26.1 Detailed Description	103
17.26.2 Member Typedef Documentation	103
17.26.2.1 Element	103
17.26.3 Constructor & Destructor Documentation	103
17.26.3.1 GIV_Extensionrandlter() [1/2]	104
17.26.3.2 GIV_Extensionrandlter() [2/2]	105
17.26.3.3 ~GIV_Extensionrandlter()	105
17.26.4 Member Function Documentation	105
17.26.4.1 random()	105
17.26.4.2 operator>() [1/2]	106
17.26.4.3 operator>() [2/2]	106
17.26.4.4 ring()	106
17.27 GIV_randlter< Ring, Type > Class Template Reference	106
17.27.1 Detailed Description	107
17.27.2 Member Typedef Documentation	107
17.27.2.1 Element	107
17.27.3 Constructor & Destructor Documentation	108
17.27.3.1 GIV_randlter() [1/2]	108
17.27.3.2 GIV_randlter() [2/2]	108
17.27.3.3 ~GIV_randlter()	108
17.27.4 Member Function Documentation	109
17.27.4.1 operator=()	109
17.27.4.2 operator>() [1/2]	109
17.27.4.3 random() [1/2]	109
17.27.4.4 operator>() [2/2]	109
17.27.4.5 random() [2/2]	110
17.27.4.6 ring()	110
17.28 GivaroAppli Class Reference	110
17.28.1 Detailed Description	110
17.29 GivaroMain Class Reference	110

17.29.1 Detailed Description	111
17.30 GivaroMM< T > Class Template Reference	111
17.30.1 Detailed Description	111
17.30.2 Member Function Documentation	111
17.30.2.1 allocate()	111
17.31 GivaroNoInit Class Reference	112
17.31.1 Detailed Description	112
17.32 GivBadFormat Class Reference	112
17.32.1 Detailed Description	112
17.33 GivError Class Reference	112
17.33.1 Detailed Description	113
17.34 GivMathDivZero Class Reference	113
17.34.1 Detailed Description	113
17.35 GivMathError Class Reference	113
17.35.1 Detailed Description	113
17.36 GivMMFreeList Class Reference	114
17.36.1 Detailed Description	114
17.37 GivMMInfo Class Reference	114
17.37.1 Detailed Description	114
17.38 GivMMRefCount Class Reference	114
17.38.1 Detailed Description	115
17.39 GivModule Class Reference	115
17.39.1 Detailed Description	115
17.40 givNoCopy Class Reference	115
17.40.1 Detailed Description	115
17.41 givNoInit Class Reference	116
17.41.1 Detailed Description	116
17.42 GivRandom Class Reference	116
17.42.1 Detailed Description	116
17.43 givWithCopy Class Reference	116
17.43.1 Detailed Description	116
17.44 HashTable< T, Key > Class Template Reference	117
17.44.1 Detailed Description	117
17.45 Indeter Class Reference	117
17.45.1 Detailed Description	117
17.46 InitAfter Class Reference	117
17.46.1 Detailed Description	118
17.47 Integer Class Reference	118
17.47.1 Detailed Description	136
17.47.2 Constructor & Destructor Documentation	136
17.47.2.1 Integer() [1/13]	136
17.47.2.2 Integer() [2/13]	137

17.47.2.3 Integer() [3/13]	137
17.47.2.4 Integer() [4/13]	137
17.47.2.5 Integer() [5/13]	138
17.47.2.6 Integer() [6/13]	138
17.47.2.7 Integer() [7/13]	138
17.47.2.8 Integer() [8/13]	138
17.47.2.9 Integer() [9/13]	139
17.47.2.10 Integer() [10/13]	139
17.47.2.11 Integer() [11/13]	139
17.47.2.12 Integer() [12/13]	140
17.47.2.13 Integer() [13/13]	140
17.47.3 Member Function Documentation	140
17.47.3.1 operator=()	140
17.47.3.2 logcopy()	140
17.47.3.3 copy()	141
17.47.3.4 isleq()	141
17.47.3.5 operator>=() [1/7]	141
17.47.3.6 operator>=() [2/7]	142
17.47.3.7 operator>=() [3/7]	142
17.47.3.8 operator>=() [4/7]	142
17.47.3.9 operator>=() [5/7]	143
17.47.3.10 operator>=() [6/7]	143
17.47.3.11 operator>=() [7/7]	143
17.47.3.12 operator<=() [1/7]	143
17.47.3.13 operator<=() [2/7]	144
17.47.3.14 operator<=() [3/7]	144
17.47.3.15 operator<=() [4/7]	144
17.47.3.16 operator<=() [5/7]	145
17.47.3.17 operator<=() [6/7]	145
17.47.3.18 operator<=() [7/7]	145
17.47.3.19 operator"! =() [1/7]	145
17.47.3.20 operator"! =() [2/7]	146
17.47.3.21 operator"! =() [3/7]	146
17.47.3.22 operator"! =() [4/7]	146
17.47.3.23 operator"! =() [5/7]	147
17.47.3.24 operator"! =() [6/7]	147
17.47.3.25 operator"! =() [7/7]	147
17.47.3.26 operator==( ) [1/7]	147
17.47.3.27 operator==( ) [2/7]	148
17.47.3.28 operator==( ) [3/7]	148
17.47.3.29 operator==( ) [4/7]	148
17.47.3.30 operator==( ) [5/7]	149

17.47.3.31 operator==( ) [ 6 / 7 ] . . . . .	149
17.47.3.32 operator==( ) [ 7 / 7 ] . . . . .	149
17.47.3.33 operator>( ) [ 1 / 7 ] . . . . .	149
17.47.3.34 operator>( ) [ 2 / 7 ] . . . . .	150
17.47.3.35 operator>( ) [ 3 / 7 ] . . . . .	150
17.47.3.36 operator>( ) [ 4 / 7 ] . . . . .	150
17.47.3.37 operator>( ) [ 5 / 7 ] . . . . .	151
17.47.3.38 operator>( ) [ 6 / 7 ] . . . . .	151
17.47.3.39 operator>( ) [ 7 / 7 ] . . . . .	151
17.47.3.40 operator<( ) [ 1 / 7 ] . . . . .	151
17.47.3.41 operator<( ) [ 2 / 7 ] . . . . .	152
17.47.3.42 operator<( ) [ 3 / 7 ] . . . . .	152
17.47.3.43 operator<( ) [ 4 / 7 ] . . . . .	152
17.47.3.44 operator<( ) [ 5 / 7 ] . . . . .	153
17.47.3.45 operator<( ) [ 6 / 7 ] . . . . .	153
17.47.3.46 operator<( ) [ 7 / 7 ] . . . . .	153
17.47.3.47 operator^( ) [ 1 / 3 ] . . . . .	153
17.47.3.48 operator^( ) [ 2 / 3 ] . . . . .	154
17.47.3.49 operator^( ) [ 3 / 3 ] . . . . .	154
17.47.3.50 operator^=( ) [ 1 / 3 ] . . . . .	154
17.47.3.51 operator^=( ) [ 2 / 3 ] . . . . .	155
17.47.3.52 operator^=( ) [ 3 / 3 ] . . . . .	155
17.47.3.53 operator"   ( ) [ 1 / 3 ] . . . . .	155
17.47.3.54 operator"   ( ) [ 2 / 3 ] . . . . .	155
17.47.3.55 operator"   ( ) [ 3 / 3 ] . . . . .	156
17.47.3.56 operator"  = ( ) [ 1 / 3 ] . . . . .	156
17.47.3.57 operator"  = ( ) [ 2 / 3 ] . . . . .	156
17.47.3.58 operator"  = ( ) [ 3 / 3 ] . . . . .	157
17.47.3.59 operator&( ) [ 1 / 3 ] . . . . .	157
17.47.3.60 operator&( ) [ 2 / 3 ] . . . . .	157
17.47.3.61 operator&( ) [ 3 / 3 ] . . . . .	157
17.47.3.62 operator&=( ) [ 1 / 3 ] . . . . .	158
17.47.3.63 operator&=( ) [ 2 / 3 ] . . . . .	158
17.47.3.64 operator&=( ) [ 3 / 3 ] . . . . .	158
17.47.3.65 operator<<( ) [ 1 / 4 ] . . . . .	159
17.47.3.66 operator<<( ) [ 2 / 4 ] . . . . .	159
17.47.3.67 operator<<( ) [ 3 / 4 ] . . . . .	159
17.47.3.68 operator<<( ) [ 4 / 4 ] . . . . .	159
17.47.3.69 operator<=<( ) [ 1 / 4 ] . . . . .	160
17.47.3.70 operator<=<( ) [ 2 / 4 ] . . . . .	160
17.47.3.71 operator<=<( ) [ 3 / 4 ] . . . . .	160
17.47.3.72 operator<=<( ) [ 4 / 4 ] . . . . .	161

17.47.3.73 operator>>() [1/4]	161
17.47.3.74 operator>>() [2/4]	161
17.47.3.75 operator>>() [3/4]	161
17.47.3.76 operator>>() [4/4]	162
17.47.3.77 operator>>=() [1/4]	162
17.47.3.78 operator>>=() [2/4]	162
17.47.3.79 operator>>=() [3/4]	163
17.47.3.80 operator>>=() [4/4]	163
17.47.3.81 addin() [1/5]	163
17.47.3.82 addin() [2/5]	163
17.47.3.83 addin() [3/5]	164
17.47.3.84 addin() [4/5]	164
17.47.3.85 addin() [5/5]	164
17.47.3.86 add() [1/5]	165
17.47.3.87 add() [2/5]	165
17.47.3.88 add() [3/5]	165
17.47.3.89 add() [4/5]	166
17.47.3.90 add() [5/5]	166
17.47.3.91 subin() [1/5]	166
17.47.3.92 subin() [2/5]	167
17.47.3.93 subin() [3/5]	167
17.47.3.94 subin() [4/5]	167
17.47.3.95 subin() [5/5]	167
17.47.3.96 sub() [1/5]	168
17.47.3.97 sub() [2/5]	168
17.47.3.98 sub() [3/5]	168
17.47.3.99 sub() [4/5]	169
17.47.3.100 sub() [5/5]	169
17.47.3.101 negin()	169
17.47.3.102 neg()	170
17.47.3.103 mulin() [1/5]	170
17.47.3.104 mulin() [2/5]	170
17.47.3.105 mulin() [3/5]	171
17.47.3.106 mulin() [4/5]	171
17.47.3.107 mulin() [5/5]	171
17.47.3.108 mul() [1/5]	172
17.47.3.109 mul() [2/5]	172
17.47.3.110 mul() [3/5]	172
17.47.3.111 mul() [4/5]	173
17.47.3.112 mul() [5/5]	173
17.47.3.113 operator+() [1/5]	173
17.47.3.114 operator+() [2/5]	174

17.47.3.115 operator+() [3/5]	174
17.47.3.116 operator+() [4/5]	174
17.47.3.117 operator+() [5/5]	175
17.47.3.118 operator+__() [1/6]	175
17.47.3.119 operator+__() [2/6]	175
17.47.3.120 operator+__() [3/6]	176
17.47.3.121 operator+__() [4/6]	176
17.47.3.122 operator+__() [5/6]	176
17.47.3.123 operator+__() [6/6]	177
17.47.3.124 operator-() [1/6]	177
17.47.3.125 operator-() [2/6]	177
17.47.3.126 operator-() [3/6]	178
17.47.3.127 operator-() [4/6]	178
17.47.3.128 operator-() [5/6]	179
17.47.3.129 operator-__() [1/6]	179
17.47.3.130 operator-__() [2/6]	179
17.47.3.131 operator-__() [3/6]	180
17.47.3.132 operator-__() [4/6]	180
17.47.3.133 operator-__() [5/6]	180
17.47.3.134 operator-__() [6/6]	181
17.47.3.135 operator-() [6/6]	181
17.47.3.136 operator*() [1/5]	181
17.47.3.137 operator*() [2/5]	182
17.47.3.138 operator*() [3/5]	182
17.47.3.139 operator*() [4/5]	182
17.47.3.140 operator*() [5/5]	183
17.47.3.141 operator*__() [1/6]	183
17.47.3.142 operator*__() [2/6]	183
17.47.3.143 operator*__() [3/6]	184
17.47.3.144 operator*__() [4/6]	184
17.47.3.145 operator*__() [5/6]	184
17.47.3.146 operator*__() [6/6]	185
17.47.3.147 axpy() [1/2]	185
17.47.3.148 axpy() [2/2]	186
17.47.3.149 axpyin() [1/2]	186
17.47.3.150 axpyin() [2/2]	186
17.47.3.151 maxpy() [1/2]	187
17.47.3.152 maxpy() [2/2]	187
17.47.3.153 maxpyin() [1/2]	187
17.47.3.154 maxpyin() [2/2]	188
17.47.3.155 axmy() [1/2]	188
17.47.3.156 axmy() [2/2]	188

17.47.3.157 axmyin() [1/2]	189
17.47.3.158 axmyin() [2/2]	189
17.47.3.159 divin() [1/3]	190
17.47.3.160 divin() [2/3]	190
17.47.3.161 divin() [3/3]	190
17.47.3.162 div() [1/4]	191
17.47.3.163 div() [2/4]	191
17.47.3.164 div() [3/4]	191
17.47.3.165 div() [4/4]	192
17.47.3.166 divexact() [1/6]	192
17.47.3.167 divexact() [2/6]	192
17.47.3.168 divexact() [3/6]	193
17.47.3.169 divexact() [4/6]	193
17.47.3.170 divexact() [5/6]	193
17.47.3.171 divexact() [6/6]	194
17.47.3.172 crem() [1/3]	194
17.47.3.173 frem() [1/3]	194
17.47.3.174 crem() [2/3]	195
17.47.3.175 frem() [2/3]	195
17.47.3.176 crem() [3/3]	195
17.47.3.177 frem() [3/3]	195
17.47.3.178 operator/() [1/5]	196
17.47.3.179 operator/() [2/5]	196
17.47.3.180 operator/() [3/5]	196
17.47.3.181 operator/() [4/5]	197
17.47.3.182 operator/() [5/5]	197
17.47.3.183 operator/=( ) [1/6]	197
17.47.3.184 operator/=( ) [2/6]	197
17.47.3.185 operator/=( ) [3/6]	198
17.47.3.186 operator/=( ) [4/6]	198
17.47.3.187 operator/=( ) [5/6]	198
17.47.3.188 operator/=( ) [6/6]	199
17.47.3.189 modin() [1/3]	199
17.47.3.190 modin() [2/3]	199
17.47.3.191 modin() [3/3]	199
17.47.3.192 mod() [1/5]	200
17.47.3.193 mod() [2/5]	200
17.47.3.194 mod() [3/5]	200
17.47.3.195 mod() [4/5]	201
17.47.3.196 mod() [5/5]	201
17.47.3.197 divmod() [1/3]	201
17.47.3.198 divmod() [2/3]	202

17.47.3.199 divmod() [3/3]	202
17.47.3.200 ceil() [1/2]	202
17.47.3.201 floor() [1/2]	203
17.47.3.202 trunc() [1/2]	203
17.47.3.203 ceil() [2/2]	203
17.47.3.204 floor() [2/2]	204
17.47.3.205 trunc() [2/2]	204
17.47.3.206 operator%() [1/8]	204
17.47.3.207 operator%() [2/8]	204
17.47.3.208 operator%() [3/8]	205
17.47.3.209 operator%() [4/8]	205
17.47.3.210 operator%() [5/8]	205
17.47.3.211 operator%() [6/8]	205
17.47.3.212 operator%() [7/8]	206
17.47.3.213 operator%() [8/8]	206
17.47.3.214 operator%=( ) [1/6]	206
17.47.3.215 operator%=( ) [2/6]	207
17.47.3.216 operator%=( ) [3/6]	207
17.47.3.217 operator%=( ) [4/6]	207
17.47.3.218 operator%=( ) [5/6]	207
17.47.3.219 operator%=( ) [6/6]	208
17.47.3.220 size_in_base()	208
17.47.3.221 bitsize()	208
17.47.3.222 operator[]()	208
17.47.3.223 random_lessthan()	209
17.47.3.224 print()	209
17.47.4 Friends And Related Function Documentation	209
17.47.4.1 compare	209
17.47.4.2 absCompare [1/8]	210
17.47.4.3 absCompare [2/8]	210
17.47.4.4 absCompare [3/8]	210
17.47.4.5 absCompare [4/8]	211
17.47.4.6 absCompare [5/8]	211
17.47.4.7 absCompare [6/8]	211
17.47.4.8 absCompare [7/8]	212
17.47.4.9 absCompare [8/8]	212
17.47.4.10 isOne	213
17.47.4.11 isMOne	213
17.47.4.12 nonZero	213
17.47.4.13 isZero [1/7]	213
17.47.4.14 isZero [2/7]	214
17.47.4.15 isZero [3/7]	214

17.47.4.16 isZero [4/7]	214
17.47.4.17 isZero [5/7]	215
17.47.4.18 isZero [6/7]	215
17.47.4.19 isZero [7/7]	216
17.47.4.20 operator>= [1/6]	216
17.47.4.21 operator>= [2/6]	216
17.47.4.22 operator>= [3/6]	217
17.47.4.23 operator>= [4/6]	217
17.47.4.24 operator>= [5/6]	217
17.47.4.25 operator>= [6/6]	217
17.47.4.26 operator<= [1/6]	218
17.47.4.27 operator<= [2/6]	218
17.47.4.28 operator<= [3/6]	218
17.47.4.29 operator<= [4/6]	219
17.47.4.30 operator<= [5/6]	219
17.47.4.31 operator<= [6/6]	219
17.47.4.32 operator!= [1/6]	219
17.47.4.33 operator!= [2/6]	220
17.47.4.34 operator!= [3/6]	220
17.47.4.35 operator!= [4/6]	220
17.47.4.36 operator!= [5/6]	221
17.47.4.37 operator!= [6/6]	221
17.47.4.38 operator== [1/6]	221
17.47.4.39 operator== [2/6]	222
17.47.4.40 operator== [3/6]	222
17.47.4.41 operator== [4/6]	222
17.47.4.42 operator== [5/6]	222
17.47.4.43 operator== [6/6]	223
17.47.4.44 operator> [1/6]	223
17.47.4.45 operator> [2/6]	223
17.47.4.46 operator> [3/6]	224
17.47.4.47 operator> [4/6]	224
17.47.4.48 operator> [5/6]	224
17.47.4.49 operator> [6/6]	224
17.47.4.50 operator< [1/6]	225
17.47.4.51 operator< [2/6]	225
17.47.4.52 operator< [3/6]	225
17.47.4.53 operator< [4/6]	226
17.47.4.54 operator< [5/6]	226
17.47.4.55 operator< [6/6]	226
17.47.4.56 operator+ [1/4]	226
17.47.4.57 operator+ [2/4]	227

17.47.4.58 operator+ [3/4]	227
17.47.4.59 operator+ [4/4]	227
17.47.4.60 operator- [1/4]	228
17.47.4.61 operator- [2/4]	228
17.47.4.62 operator- [3/4]	228
17.47.4.63 operator- [4/4]	229
17.47.4.64 operator* [1/4]	229
17.47.4.65 operator* [2/4]	229
17.47.4.66 operator* [3/4]	230
17.47.4.67 operator* [4/4]	230
17.47.4.68 operator/ [1/2]	230
17.47.4.69 operator/ [2/2]	231
17.47.4.70 operator% [1/4]	231
17.47.4.71 operator% [2/4]	231
17.47.4.72 operator% [3/4]	232
17.47.4.73 operator% [4/4]	232
17.47.4.74 gcd [1/4]	232
17.47.4.75 gcd [2/4]	233
17.47.4.76 gcd [3/4]	233
17.47.4.77 gcd [4/4]	234
17.47.4.78 inv	234
17.47.4.79 invin	234
17.47.4.80 pp	235
17.47.4.81 lcm [1/2]	235
17.47.4.82 lcm [2/2]	235
17.47.4.83 pow [1/9]	236
17.47.4.84 pow [2/9]	236
17.47.4.85 pow [3/9]	236
17.47.4.86 pow [4/9]	237
17.47.4.87 pow [5/9]	237
17.47.4.88 pow [6/9]	238
17.47.4.89 pow [7/9]	238
17.47.4.90 pow [8/9]	238
17.47.4.91 pow [9/9]	239
17.47.4.92 powmod	239
17.47.4.93 fact	239
17.47.4.94 sqrt [1/2]	240
17.47.4.95 sqrt [2/2]	240
17.47.4.96 sqrtrem [1/2]	240
17.47.4.97 sqrtrem [2/2]	241
17.47.4.98 root	241
17.47.4.99 logp	241

17.47.4.100 logtwo . . . . .	241
17.47.4.101 naturallog . . . . .	242
17.47.4.102 swap . . . . .	242
17.47.4.103 sign . . . . .	242
17.47.4.104 length . . . . .	243
17.47.4.105 isOdd . . . . .	243
17.47.4.106 operator>> . . . . .	243
17.47.4.107 operator<< . . . . .	244
17.47.4.108 absOutput . . . . .	244
17.47.4.109 Protected::importWords . . . . .	244
17.48 IntegerDom Class Reference . . . . .	245
17.48.1 Detailed Description . . . . .	245
17.49 Interpolation< Domain, REDUCE > Struct Template Reference . . . . .	246
17.49.1 Detailed Description . . . . .	246
17.49.2 Member Function Documentation . . . . .	246
17.49.2.1 setdegree() . . . . .	246
17.49.2.2 sqrfree() . . . . .	247
17.50 IntFactorDom< MyRandIter > Class Template Reference . . . . .	247
17.50.1 Detailed Description . . . . .	248
17.51 IntNumTheoDom< MyRandIter > Class Template Reference . . . . .	248
17.51.1 Detailed Description . . . . .	249
17.51.2 Member Function Documentation . . . . .	249
17.51.2.1 probable_prim_root() . . . . .	249
17.51.2.2 prim_inv() . . . . .	249
17.52 IntPrimeDom Class Reference . . . . .	250
17.52.1 Detailed Description . . . . .	250
17.53 IntRNSsystem< Container, Alloc > Class Template Reference . . . . .	250
17.53.1 Detailed Description . . . . .	251
17.54 IntRSADom< MyRandIter > Class Template Reference . . . . .	251
17.54.1 Detailed Description . . . . .	252
17.54.2 Member Function Documentation . . . . .	252
17.54.2.1 strong_prime() . . . . .	252
17.54.2.2 keys_gen() . . . . .	252
17.54.3 Field Documentation . . . . .	253
17.54.3.1 _fast_impl . . . . .	253
17.55 IntSqrtModDom< MyRandIter > Class Template Reference . . . . .	253
17.55.1 Detailed Description . . . . .	254
17.56 Key< T > Class Template Reference . . . . .	254
17.56.1 Detailed Description . . . . .	254
17.57 List0< T > Class Template Reference . . . . .	254
17.57.1 Detailed Description . . . . .	255
17.58 Modular< IntType, _Compute_t, Enable > Class Template Reference . . . . .	255

17.58.1 Detailed Description	255
17.59 Modular< _Storage_t, _Compute_t, typename std::enable_if< is_same_ruint< _Storage_t, _Compute_t >::value  is_smaller_ruint< _Storage_t, _Compute_t >::value >::type > Class Template Reference	256
17.59.1 Detailed Description	256
17.60 Modular< Integer > Class Reference	256
17.60.1 Detailed Description	256
17.61 Modular< Log16 > Class Reference	257
17.61.1 Detailed Description	257
17.62 Modular_implem< _Storage_t, _Compute_t, _Residu_t > Class Template Reference	257
17.62.1 Detailed Description	257
17.63 ModularRandIter< Ring > Class Template Reference	258
17.63.1 Detailed Description	258
17.63.2 Member Typedef Documentation	258
17.63.2.1 Element	259
17.63.3 Constructor & Destructor Documentation	259
17.63.3.1 ModularRandIter() [1/2]	259
17.63.3.2 ModularRandIter() [2/2]	259
17.63.3.3 ~ModularRandIter()	260
17.63.4 Member Function Documentation	260
17.63.4.1 operator=()	260
17.63.4.2 operator()() [1/2]	260
17.63.4.3 random() [1/2]	261
17.63.4.4 operator()() [2/2]	261
17.63.4.5 random() [2/2]	261
17.64 Montgomery< int32_t > Class Reference	261
17.64.1 Detailed Description	261
17.65 Montgomery< Reclnt::ruint< K > > Class Template Reference	262
17.65.1 Detailed Description	262
17.66 Neutral Class Reference	262
17.66.1 Detailed Description	262
17.67 NewtonInterpGeom< Domain, REDUCE > Struct Template Reference	262
17.67.1 Detailed Description	263
17.67.2 Member Function Documentation	263
17.67.2.1 setdegree()	263
17.67.2.2 sqrfree()	263
17.68 NewtonInterpGeomMultip< Domain, REDUCE > Struct Template Reference	264
17.68.1 Detailed Description	264
17.68.2 Member Function Documentation	264
17.68.2.1 setdegree()	264
17.68.2.2 sqrfree()	265
17.69 ObjectInit Class Reference	265
17.69.1 Detailed Description	265

17.70 OMPTimer Struct Reference . . . . .	266
17.70.1 Detailed Description . . . . .	266
17.71 Pair< T1, T2 > Struct Template Reference . . . . .	266
17.71.1 Detailed Description . . . . .	266
17.72 Poly1CRT< Field > Class Template Reference . . . . .	266
17.72.1 Detailed Description . . . . .	267
17.73 Poly1Dom< Domain, Dense > Class Template Reference . . . . .	267
17.73.1 Detailed Description . . . . .	267
17.73.2 Member Function Documentation . . . . .	267
17.73.2.1 setdegree() . . . . .	267
17.73.2.2 sqfree() . . . . .	268
17.74 Poly1FactorDom< Domain, Tag, RandomIterator > Class Template Reference . . . . .	268
17.74.1 Detailed Description . . . . .	269
17.74.2 Constructor & Destructor Documentation . . . . .	269
17.74.2.1 Poly1FactorDom() . . . . .	269
17.75 Poly1PadicDom< Domain, Dense > Class Template Reference . . . . .	270
17.75.1 Detailed Description . . . . .	270
17.75.2 Member Function Documentation . . . . .	270
17.75.2.1 setdegree() . . . . .	270
17.75.2.2 sqfree() . . . . .	271
17.76 Primes16 Class Reference . . . . .	271
17.76.1 Detailed Description . . . . .	271
17.77 QField< Rational > Class Reference . . . . .	271
17.77.1 Detailed Description . . . . .	272
17.78 RandomIntegerIterator< _Unsigned, _Exact_Size > Class Template Reference . . . . .	272
17.78.1 Detailed Description . . . . .	272
17.78.2 Constructor & Destructor Documentation . . . . .	273
17.78.2.1 RandomIntegerIterator() [1/2] . . . . .	273
17.78.2.2 RandomIntegerIterator() [2/2] . . . . .	273
17.78.3 Member Function Documentation . . . . .	273
17.78.3.1 operator=() . . . . .	273
17.78.3.2 operator*() . . . . .	275
17.78.3.3 randomInteger() . . . . .	275
17.78.3.4 setSeed() . . . . .	275
17.79 Rational Class Reference . . . . .	275
17.79.1 Detailed Description . . . . .	276
17.79.2 Constructor & Destructor Documentation . . . . .	276
17.79.2.1 Rational() . . . . .	276
17.80 RealTimer Class Reference . . . . .	277
17.80.1 Detailed Description . . . . .	277
17.81 RefCounter Class Reference . . . . .	277
17.81.1 Detailed Description . . . . .	277

17.82 RefCountPtr< T > Class Template Reference . . . . .	277
17.82.1 Detailed Description . . . . .	277
17.83 RNSsystem< RING, Domain > Class Template Reference . . . . .	278
17.83.1 Detailed Description . . . . .	278
17.84 RNSsystemFixed< Ints > Class Template Reference . . . . .	278
17.84.1 Detailed Description . . . . .	279
17.85 Stack< THING > Class Template Reference . . . . .	279
17.85.1 Detailed Description . . . . .	279
17.86 StaticElement< DomainStyle > Struct Template Reference . . . . .	279
17.86.1 Detailed Description . . . . .	279
17.87 SysTimer Class Reference . . . . .	280
17.87.1 Detailed Description . . . . .	280
17.88 Timer Class Reference . . . . .	280
17.88.1 Detailed Description . . . . .	281
17.88.2 Member Function Documentation . . . . .	281
17.88.2.1 clear() . . . . .	281
17.88.2.2 start() . . . . .	282
17.88.2.3 stop() . . . . .	282
17.88.2.4 usertime() . . . . .	283
17.88.2.5 systime() . . . . .	283
17.88.2.6 realtime() . . . . .	283
17.88.2.7 userElapsedTime() . . . . .	284
17.88.2.8 sysElapsedTime() . . . . .	284
17.88.2.9 realElapsedTime() . . . . .	284
17.89 UserTimer Class Reference . . . . .	285
17.89.1 Detailed Description . . . . .	285
17.90 VectorDom< Domain, StorageTag > Class Template Reference . . . . .	285
17.90.1 Detailed Description . . . . .	285
17.91 ZRing< _Element > Class Template Reference . . . . .	285
17.91.1 Detailed Description . . . . .	287
17.91.2 Member Function Documentation . . . . .	287
17.91.2.1 RationalReconstruction() . . . . .	287
17.91.2.2 write() . . . . .	287
17.91.2.3 read() . . . . .	288
<b>18 File Documentation</b>	<b>289</b>
18.1 all_field.C File Reference . . . . .	289
18.2 domain_to_operatorstyle.C File Reference . . . . .	289
18.3 exponentiation.C File Reference . . . . .	289
18.4 ff_arith.C File Reference . . . . .	290
18.5 GF128.C File Reference . . . . .	290
18.6 GFirreducible.C File Reference . . . . .	290

18.7 gfq_atomic.C File Reference . . . . .	290
18.8 Test_Extension.C File Reference . . . . .	290
18.9 zpz_atomic.C File Reference . . . . .	290
18.10 iexponentiation.C File Reference . . . . .	291
18.11 ifactor.C File Reference . . . . .	291
18.12 ifactor_lenstra.C File Reference . . . . .	291
18.13 igcd.C File Reference . . . . .	291
18.14 igcdext.C File Reference . . . . .	291
18.15 ilcm.C File Reference . . . . .	292
18.16 ispower.C File Reference . . . . .	292
18.17 isproot.C File Reference . . . . .	292
18.18 lambda.C File Reference . . . . .	292
18.19 lambda_inv.C File Reference . . . . .	292
18.20 ModularSquareRoot.C File Reference . . . . .	292
18.21 nb_primes.C File Reference . . . . .	293
18.22 nextprime.C File Reference . . . . .	293
18.23 order.C File Reference . . . . .	293
18.24 phi.C File Reference . . . . .	293
18.25 prevprime.C File Reference . . . . .	293
18.26 primitiveelement.C File Reference . . . . .	293
18.27 primitiveroot.C File Reference . . . . .	294
18.28 probable_primroot.C File Reference . . . . .	294
18.29 ProbLucas.C File Reference . . . . .	294
18.30 RSA_breaking.C File Reference . . . . .	294
18.31 RSA_decipher.C File Reference . . . . .	294
18.32 RSA_encipher.C File Reference . . . . .	295
18.33 RSA_keys_generator.C File Reference . . . . .	295
18.34 highorder.C File Reference . . . . .	295
18.35 interpolate.C File Reference . . . . .	295
18.36 isirred.C File Reference . . . . .	295
18.37 isprimitive.C File Reference . . . . .	296
18.38 pol_arith.C File Reference . . . . .	296
18.39 pol_eval.C File Reference . . . . .	296
18.40 pol_factor.C File Reference . . . . .	296
18.41 PolynomialCRT.C File Reference . . . . .	296
18.42 trunc_arith.C File Reference . . . . .	297
18.43 iratrecon.C File Reference . . . . .	297
18.44 polydouble.C File Reference . . . . .	297
18.45 givarray0.h File Reference . . . . .	297
18.45.1 Detailed Description . . . . .	298
18.46 givarrayallocator.h File Reference . . . . .	298
18.46.1 Detailed Description . . . . .	298

18.47 givarrayfixed.h File Reference . . . . .	298
18.47.1 Detailed Description . . . . .	299
18.48 givbits.h File Reference . . . . .	299
18.48.1 Detailed Description . . . . .	299
18.49 givelem.h File Reference . . . . .	299
18.49.1 Detailed Description . . . . .	300
18.50 givhashtable.h File Reference . . . . .	300
18.50.1 Detailed Description . . . . .	300
18.51 givlist0.h File Reference . . . . .	301
18.51.1 Detailed Description . . . . .	301
18.52 givstack.h File Reference . . . . .	301
18.52.1 Detailed Description . . . . .	301
18.53 chineseremainder.h File Reference . . . . .	302
18.53.1 Detailed Description . . . . .	302
18.54 extension.h File Reference . . . . .	302
18.54.1 Detailed Description . . . . .	303
18.55 gfq.h File Reference . . . . .	303
18.55.1 Detailed Description . . . . .	304
18.56 gfqext.h File Reference . . . . .	304
18.56.1 Detailed Description . . . . .	304
18.57 gfqkronecker.h File Reference . . . . .	304
18.57.1 Detailed Description . . . . .	305
18.58 givprimes16.h File Reference . . . . .	305
18.58.1 Detailed Description . . . . .	305
18.59 givrns.h File Reference . . . . .	305
18.59.1 Detailed Description . . . . .	306
18.60 givrnsfixed.h File Reference . . . . .	306
18.60.1 Detailed Description . . . . .	306
18.61 StaticElement.h File Reference . . . . .	307
18.61.1 Detailed Description . . . . .	307
18.62 gmp++_int.h File Reference . . . . .	307
18.62.1 Detailed Description . . . . .	308
18.63 gmp++_int_add.C File Reference . . . . .	308
18.63.1 Detailed Description . . . . .	309
18.64 gmp++_int_compare.C File Reference . . . . .	309
18.64.1 Detailed Description . . . . .	309
18.65 gmp++_int_cstor.C File Reference . . . . .	309
18.65.1 Detailed Description . . . . .	310
18.66 gmp++_int_div.C File Reference . . . . .	310
18.66.1 Detailed Description . . . . .	310
18.67 gmp++_int_gcd.C File Reference . . . . .	310
18.67.1 Detailed Description . . . . .	311

18.68 gmp++_int_io.C File Reference . . . . .	311
18.68.1 Detailed Description . . . . .	311
18.69 gmp++_int_lib.C File Reference . . . . .	311
18.69.1 Detailed Description . . . . .	311
18.70 gmp++_int_misc.C File Reference . . . . .	312
18.70.1 Detailed Description . . . . .	312
18.71 gmp++_int_mod.C File Reference . . . . .	312
18.71.1 Detailed Description . . . . .	313
18.72 gmp++_int_mul.C File Reference . . . . .	313
18.72.1 Detailed Description . . . . .	313
18.73 gmp++_int_pow.C File Reference . . . . .	313
18.73.1 Detailed Description . . . . .	314
18.74 gmp++_int_rand.inl File Reference . . . . .	314
18.74.1 Detailed Description . . . . .	314
18.75 gmp++_int_sub.C File Reference . . . . .	314
18.75.1 Detailed Description . . . . .	315
18.76 givinteger.h File Reference . . . . .	315
18.76.1 Detailed Description . . . . .	315
18.77 givintfactor.h File Reference . . . . .	315
18.77.1 Detailed Description . . . . .	316
18.78 givintnumtheo.h File Reference . . . . .	316
18.78.1 Detailed Description . . . . .	317
18.79 givintprime.h File Reference . . . . .	317
18.79.1 Detailed Description . . . . .	317
18.80 givintrns.h File Reference . . . . .	318
18.80.1 Detailed Description . . . . .	318
18.81 givintrsa.h File Reference . . . . .	318
18.81.1 Detailed Description . . . . .	319
18.82 givintsqrootmod.h File Reference . . . . .	319
18.82.1 Detailed Description . . . . .	319
18.83 givaromm.h File Reference . . . . .	319
18.83.1 Detailed Description . . . . .	320
18.84 givpointer.h File Reference . . . . .	320
18.84.1 Detailed Description . . . . .	321
18.85 givref_count.h File Reference . . . . .	321
18.85.1 Detailed Description . . . . .	321
18.86 givrational.h File Reference . . . . .	321
18.86.1 Detailed Description . . . . .	322
18.87 modular-implem.h File Reference . . . . .	322
18.87.1 Detailed Description . . . . .	322
18.88 modular-integral.h File Reference . . . . .	323
18.88.1 Detailed Description . . . . .	323

18.89 modular.h File Reference . . . . .	323
18.89.1 Detailed Description . . . . .	323
18.90 montgomery.h File Reference . . . . .	323
18.90.1 Detailed Description . . . . .	324
18.91 givbasictype.h File Reference . . . . .	324
18.91.1 Detailed Description . . . . .	324
18.92 givcaster.h File Reference . . . . .	324
18.92.1 Detailed Description . . . . .	325
18.93 givconfig.h File Reference . . . . .	325
18.93.1 Detailed Description . . . . .	325
18.94 giverror.h File Reference . . . . .	325
18.94.1 Detailed Description . . . . .	326
18.95 givgenarith.h File Reference . . . . .	326
18.95.1 Detailed Description . . . . .	326
18.96 givinit.h File Reference . . . . .	327
18.96.1 Detailed Description . . . . .	327
18.97 givmodule.h File Reference . . . . .	327
18.97.1 Detailed Description . . . . .	328
18.98 givperf.h File Reference . . . . .	328
18.98.1 Detailed Description . . . . .	328
18.99 givpower.h File Reference . . . . .	328
18.99.1 Detailed Description . . . . .	328
18.100 givprint.h File Reference . . . . .	328
18.100.1 Detailed Description . . . . .	329
18.101 givranditer.h File Reference . . . . .	329
18.101.1 Detailed Description . . . . .	330
18.102 givrandom.h File Reference . . . . .	330
18.102.1 Detailed Description . . . . .	330
18.103 givtimer.h File Reference . . . . .	330
18.103.1 Detailed Description . . . . .	331
18.104 givdegree.h File Reference . . . . .	331
18.104.1 Detailed Description . . . . .	332
18.105 givindeter.h File Reference . . . . .	332
18.105.1 Detailed Description . . . . .	332
18.106 givinterp.h File Reference . . . . .	332
18.106.1 Detailed Description . . . . .	333
18.107 givinterpgeom-multip.h File Reference . . . . .	333
18.107.1 Detailed Description . . . . .	333
18.108 givinterpgeom.h File Reference . . . . .	334
18.108.1 Detailed Description . . . . .	334
18.109 givpoly1.h File Reference . . . . .	334
18.109.1 Detailed Description . . . . .	335

18.110 givpoly1crt.h File Reference . . . . .	335
18.110.1 Detailed Description . . . . .	335
18.111 givpoly1dense.h File Reference . . . . .	335
18.111.1 Detailed Description . . . . .	336
18.112 givpoly1factor.h File Reference . . . . .	336
18.112.1 Detailed Description . . . . .	336
18.113 givpoly1padic.h File Reference . . . . .	337
18.113.1 Detailed Description . . . . .	337
18.114 test-crt.C File Reference . . . . .	337
18.114.1 Detailed Description . . . . .	337
18.115 test-integer.C File Reference . . . . .	338
18.115.1 Detailed Description . . . . .	338
18.115.2 Function Documentation . . . . .	338
18.115.2.1 main() . . . . .	338
18.116 test-modsqroot.C File Reference . . . . .	338
18.116.1 Detailed Description . . . . .	339
18.117 test-random.C File Reference . . . . .	339
18.117.1 Detailed Description . . . . .	339
18.117.2 Function Documentation . . . . .	339
18.117.2.1 test2() . . . . .	340
18.117.2.2 test3() . . . . .	340
<b>19 Example Documentation . . . . .</b>	<b>341</b>
19.1 examples/FiniteField/all_field.C . . . . .	341
19.2 examples/FiniteField/domain_to_operatorstyle.C . . . . .	342
19.3 examples/FiniteField/exponentiation.C . . . . .	343
19.4 examples/FiniteField/ff_arith.C . . . . .	343
19.5 examples/FiniteField/GF128.C . . . . .	345
19.6 examples/FiniteField/GFirreducible.C . . . . .	346
19.7 examples/FiniteField/gfq_atomic.C . . . . .	347
19.8 examples/FiniteField/Test_Extension.C . . . . .	349
19.9 examples/FiniteField/zpz_atomic.C . . . . .	349
19.10 examples/Integer/ieponentiation.C . . . . .	351
19.11 examples/Integer/ifactor.C . . . . .	351
19.12 examples/Integer/ifactor_lenstra.C . . . . .	352
19.13 examples/Integer/igcd.C . . . . .	353
19.14 examples/Integer/igcdext.C . . . . .	353
19.15 examples/Integer/ilcm.C . . . . .	354
19.16 examples/Integer/ispower.C . . . . .	354
19.17 examples/Integer/isprime.C . . . . .	355
19.18 examples/Integer/isroot.C . . . . .	355
19.19 examples/Integer/lambda.C . . . . .	356

19.20 examples/Integer/lambda_inv.C . . . . .	356
19.21 examples/Integer/ModularSquareRoot.C . . . . .	357
19.22 examples/Integer/nb_primes.C . . . . .	357
19.23 examples/Integer/nextprime.C . . . . .	358
19.24 examples/Integer/order.C . . . . .	358
19.25 examples/Integer/phi.C . . . . .	359
19.26 examples/Integer/prevprime.C . . . . .	359
19.27 examples/Integer/primitiveelement.C . . . . .	360
19.28 examples/Integer/primitiveroot.C . . . . .	360
19.29 examples/Integer/probable_primroot.C . . . . .	361
19.30 examples/Integer/ProbLucas.C . . . . .	362
19.31 examples/Integer/RSA_breaking.C . . . . .	365
19.32 examples/Integer/RSA_decipher.C . . . . .	365
19.33 examples/Integer/RSA_encipher.C . . . . .	366
19.34 examples/Integer/RSA_keys_generator.C . . . . .	367
19.35 examples/Polynomial/pol_arith.C . . . . .	367
19.36 examples/Polynomial/highorder.C . . . . .	368
19.37 examples/Polynomial/interpolate.C . . . . .	370
19.38 examples/Polynomial/isprimitive.C . . . . .	371
19.39 examples/Polynomial/isirred.C . . . . .	371
19.40 examples/Polynomial/pol_eval.C . . . . .	372
19.41 examples/Polynomial/pol_factor.C . . . . .	373
19.42 examples/Polynomial/PolynomialCRT.C . . . . .	373
19.43 examples/Polynomial/trunc_arith.C . . . . .	376
19.44 examples/Rational/iratrecon.C . . . . .	379
19.45 examples/Rational/polydouble.C . . . . .	380
<b>Index</b>	<b>381</b>



# Chapter 1

## Givaro Documentation.

[Givaro](#) is a C++ library for arithmetic and algebraic computations.

### 1.1 Introduction

[Givaro](#) can do fast arithmetic computations with

- Integers. The Integer class uses GMP integers.
- Rationals.
- Rings of the form  $\mathbb{Z}/m\mathbb{Z}$ .
- Prime fields and extensions.
- Polynomials.

Non-standard arithmetic tools include truncated arithmetic.

### 1.2 Goals

### 1.3 Design

### 1.4 Using Givaro

- [Licence](#)..
- [Tutorial](#).. This is a brief introduction to LinBox capabilities.
- [Installation](#).. Explains how to install from sources or from the latest cvs version.
- [Architecture](#).. Describes how [Givaro](#) is organized
- [Documentation for Users](#). If everything around is blue, then you are reading the lighter, user-oriented, documentation.
- [Documentation for Developers](#). If everything around is green, then you can get to everything (not necessarily yet) documented.

## 1.5 Contributing to Givaro, getting assistance.

Version

4.0.1

## Chapter 2

## Tutorial.

Empty.



## Chapter 3

## Licence.

Empty.



## Chapter 4

# Installation.

Empty.



## Chapter 5

# Architecture.

Empty.



## Chapter 6

# Bibliography

File [gfqext.h](#)

JG Dumas, *Q-adic Transform Revisited*, ISSAC 2008

File [givinterpgeom-multip.h](#)

A Bostan and E Schost, *Polynomial evaluation and interpolation on special sets of points*, Journal of Complexity 21(4): 420-446, 2005.

File [givinterpgeom.h](#)

A Bostan and E Schost, *Polynomial evaluation and interpolation on special sets of points*, Journal of Complexity 21(4): 420-446, 2005.

File [givrandom.h](#)

Fishman, GS *Multiplicative congruential random number generators...* Math. Comp. 54:331-344 (1990).

Global [IntRSADom](#)< [MyRandIter](#) >::[strong\\_prime](#) (random\_generator &g, int64\_t psize, Element &p) const

J. Gordon, *Strong Primes Are Easy to Find*, EUROCRYPT'84, LNCS 209.

Global [Rational::Rational](#) (const [Integer](#) &f, const [Integer](#) &m, const [Integer](#) &k, bool recurs=false)

von zur Gathen & Gerhard *Modern Computer Algebra*, 5.10, Cambridge Univ. Press 1999]

Global [ZRing](#)< [\\_Element](#) >::[RationalReconstruction](#) (Element &a, Element &b, const Element &f, const Element &m, const Element &k, bool forcedreduce, bool recursive) const

von zur Gathen & Gerhard, *Modern Computer Algebra*, 5.10, Cambridge Univ. Press 1999



## Chapter 7

# Bug List

Global `Givaro::length` (const `Integer` &a)

JGD 23.04.2012: shouldn't it be "mp\_limb\_t" instead of "uint64\_t"?

Global `Givaro::operator==` (const `Indeter` &i1, const `Indeter` &i2)

put elsewhere. Inline members functions :

Global `GivaroMM< T >::allocate` (const `size_t` s)

implem does not belong here

Global `Integer::random_lesssthan` (`Integer` &r, const `Integer` &m)

m **has** to be an integer here.



## Chapter 8

# Todo List

Namespace **Givaro**

use NTL if available ?

Global **main** (int argc, char \*\*argv)

test gcd...



## Chapter 9

# Test List

File [test-integer.C](#)

tests integer.h fuctions not tested elsewhere.

File [test-random.C](#)

we test bounds for random Integers



## Chapter 10

# Module Index

### 10.1 Modules

Here is a list of all modules:

Givaro . . . . .	35
bstruct . . . . .	35
GMP . . . . .	36
Integer . . . . .	36
Memory . . . . .	36
Rationals . . . . .	37
System . . . . .	37
ZRing . . . . .	37



## Chapter 11

# Namespace Index

### 11.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

<a href="#">Givaro</a>	Namespace in which the whole <a href="#">Givaro</a> library resides . . . . .	<a href="#">39</a>
<a href="#">RecInt</a>	NOTE : For this common file, either basic/reduc.h or mg/reduc.h has to be pre-included . . . .	<a href="#">56</a>
<a href="#">std</a>	STL namespace . . . . .	<a href="#">57</a>



## Chapter 12

# Hierarchical Index

### 12.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<code>__giv_map_less_ith&lt; T, UNARYOP, ith &gt;</code>	61
<code>__givdom_trait_name&lt; T &gt;</code>	61
<code>_perfArray0&lt; T &gt;</code>	62
<code>Array0&lt; T &gt;</code>	62
<code>ArrayAllocatort&lt; T, Tag &gt;</code>	65
<code>Array0&lt; Bits::base &gt;</code>	62
<code>Array0&lt; Domain &gt;</code>	62
<code>Array0&lt; Indice_t &gt;</code>	62
<code>Array0&lt; Modular&lt; Ints &gt; &gt;</code>	62
<code>Array0&lt; modulo &gt;</code>	62
<code>Array0Tag</code>	65
<code>ArrayFixed&lt; T, SIZE &gt;</code>	68
<code>BaseDomain&lt; T &gt;</code>	69
<code>BaseTimer</code>	69
<code>RealTimer</code>	277
<code>SysTimer</code>	280
<code>UserTimer</code>	285
<code>Bits</code>	69
<code>BlocFreeList</code>	70
<code>ChineseRemainder&lt; Ring, Domain, REDUCE &gt;</code>	70
<code>ChineseRemainder&lt; Ring, Domain, false &gt;</code>	71
<code>Degree</code>	71
<code>ElemConstRef&lt; T &gt;</code>	71
<code>ElemRef&lt; T &gt;</code>	72
<code>Extension&lt; BFT &gt;</code>	72
<code>GeneralRingNonZeroRandlter&lt; Ring, Randlter &gt;</code>	73
<code>GeneralRingRandlter&lt; Ring &gt;</code>	74
<code>GF2</code>	74
<code>GFqDom&lt; TT &gt;</code>	100
<code>GFqExtFast&lt; TT &gt;</code>	101
<code>GFqExt&lt; TT &gt;</code>	101
<code>GFqKronecker&lt; TT, Ints &gt;</code>	102
<code>GFqDom&lt; int64_t &gt;</code>	100
<code>GIV_Extensionrandlter&lt; ExtensionField, Type &gt;</code>	102

GIV_randIter< Ring, Type > . . . . .	106
GivaroMain . . . . .	110
GivaroAppli . . . . .	110
GivaroMM< T > . . . . .	111
GivaroNoInit . . . . .	112
GivError . . . . .	112
GivBadFormat . . . . .	112
GivMathDivZero . . . . .	113
GivMathError . . . . .	113
GivMMFreeList . . . . .	114
GivMMInfo . . . . .	114
GivMMRefCount . . . . .	114
GivModule . . . . .	115
givNoCopy . . . . .	115
givNoInit . . . . .	116
GivRandom . . . . .	116
givWithCopy . . . . .	116
HashTable< T, Key > . . . . .	117
HashTable< T, Givaro::Key > . . . . .	117
Indeter . . . . .	117
InitAfter . . . . .	117
Integer . . . . .	118
IntegerDom . . . . .	245
FermatDom . . . . .	73
IntPrimeDom . . . . .	250
IntFactorDom< GivRandom > . . . . .	247
IntFactorDom< MyRandIter > . . . . .	247
IntNumTheoDom< MyRandIter > . . . . .	248
IntRSADom< MyRandIter > . . . . .	251
IntSqrtModDom< MyRandIter > . . . . .	253
IntRNSsystem< Container, Alloc > . . . . .	250
Poly1PadicDom< Domain, Dense > . . . . .	270
Key< T > . . . . .	254
List0< T > . . . . .	254
Modular< Log16 > . . . . .	257
Modular_implem< _Storage_t, _Compute_t, _Residu_t > . . . . .	257
Modular< Ints > . . . . .	255
Modular_implem< _Storage_t, _Compute_t, _Storage_t > . . . . .	257
Modular< _Storage_t, _Compute_t, typename std::enable_if< is_same_ruint< _Storage_t, _↵ Compute_t >::value  is_smaller_ruint< _Storage_t, _Compute_t >::value >::type > . . . . .	256
Modular_implem< _Storage_t, _Compute_t, make_unsigned_int< _Storage_t >::type > . . . . .	257
Modular_implem< _Storage_t, std::make_unsigned< _Compute_t >::type, std::make_unsigned< _↵ Storage_t >::type > . . . . .	257
Modular_implem< Integer, Integer, Integer > . . . . .	257
Modular< Integer > . . . . .	256
Modular_implem< Ints, _Compute_t, Ints > . . . . .	257
Modular_implem< IntType, _Compute_t, IntType > . . . . .	257
Modular< IntType, _Compute_t, Enable > . . . . .	255
ModularRandIter< Ring > . . . . .	258
Montgomery< int32_t > . . . . .	261
Montgomery< Reclnt::ruint< K > > . . . . .	262
Neutral . . . . .	262
ObjectInit . . . . .	265
OMPTimer . . . . .	266
Pair< T1, T2 > . . . . .	266
Poly1CRT< Field > . . . . .	266

Poly1Dom< Domain, Dense > . . . . .	267
Interpolation< Domain, REDUCE > . . . . .	246
Poly1PadicDom< Domain, Dense > . . . . .	270
Poly1FactorDom< Domain, Tag, RandomIterator > . . . . .	268
Poly1FactorDom< GFqDom< int64_t >, Dense > . . . . .	268
Primes16 . . . . .	271
QField< Rational > . . . . .	271
RandomIntegerIterator< _Unsigned, _Exact_Size > . . . . .	272
Rational . . . . .	275
RefCounter . . . . .	277
RefCountPtr< T > . . . . .	277
RNSsystem< RING, Domain > . . . . .	278
RNSsystem< Ints, Modular< Ints > > . . . . .	278
RNSsystemFixed< Ints > . . . . .	278
Stack< THING > . . . . .	279
StaticElement< DomainStyle > . . . . .	279
StaticElement< Field > . . . . .	279
Timer . . . . .	280
VectorDom< Domain, StorageTag > . . . . .	285
VectorDom< Domain, Dense > . . . . .	285
ZRing< _Element > . . . . .	285



## Chapter 13

# Data Structure Index

### 13.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">__giv_map_less_ith&lt; T, UNARYOP, ith &gt;</a>	61
Map opcode on all Elements less or requal that ith	
<a href="#">__givdom_trait_name&lt; T &gt;</a>	61
Give a name for /read/write	
<a href="#">_perfArray0&lt; T &gt;</a>	62
Defined by marco GIVARO_PERF_DEFCLASS. ref counting and stuff	
<a href="#">Array0&lt; T &gt;</a>	62
NODOC	
<a href="#">Array0Tag</a>	65
<a href="#">Array0Tag</a>	
<a href="#">ArrayAllocatort&lt; T, Tag &gt;</a>	65
ArrayAllocator: class for allocation of arrays	
<a href="#">ArrayFixed&lt; T, SIZE &gt;</a>	68
<a href="#">ArrayFixed</a>	
<a href="#">BaseDomain&lt; T &gt;</a>	69
Base Domain	
<a href="#">BaseTimer</a>	69
Base for class <a href="#">RealTimer</a> ; class <a href="#">SysTimer</a> ; class <a href="#">UserTimer</a> ;	
<a href="#">Bits</a>	69
<a href="#">Bits</a>	
<a href="#">BlocFreeList</a>	70
Data structure of a bloc	
<a href="#">ChineseRemainder&lt; Ring, Domain, REDUCE &gt;</a>	70
CRA	
<a href="#">ChineseRemainder&lt; Ring, Domain, false &gt;</a>	71
CRA2	
<a href="#">Degree</a>	71
<a href="#">Degree</a> type for polynomials	
<a href="#">ElemConstRef&lt; T &gt;</a>	71
Elem const Ref	
<a href="#">ElemRef&lt; T &gt;</a>	72
Elem Ref	
<a href="#">Extension&lt; BFT &gt;</a>	72
<a href="#">Extension</a>	
<a href="#">FermatDom</a>	73
Fermat numbers	

<a href="#">GeneralRingNonZeroRandIter&lt; Ring, RandIter &gt;</a>	
Random iterator for nonzero random numbers	73
<a href="#">GeneralRingRandIter&lt; Ring &gt;</a>	
UnparametricRandIter	74
<a href="#">GF2</a>	
Integers modulo 2	74
<a href="#">GFqDom&lt; TT &gt;</a>	
Class <a href="#">GFqDom</a>	100
<a href="#">GFqExt&lt; TT &gt;</a>	
GFq Ext (other)	101
<a href="#">GFqExtFast&lt; TT &gt;</a>	
GFq Ext	101
<a href="#">GFqKronecker&lt; TT, Ints &gt;</a>	
<a href="#">GFqKronecker</a>	102
<a href="#">GIV_ExtensionrandIter&lt; ExtensionField, Type &gt;</a>	
Extension rand iters	102
<a href="#">GIV_randIter&lt; Ring, Type &gt;</a>	
Random ring Element generator	106
<a href="#">GivaroAppli</a>	
Main application class Could be not used	110
<a href="#">GivaroMain</a>	
Initialisation of GIVARO	110
<a href="#">GivaroMM&lt; T &gt;</a>	
Memory manager that allocates array of object of type T for	111
<a href="#">GivaroNoInit</a>	
<a href="#">GivaroNoInit</a>	112
<a href="#">GivBadFormat</a>	
Exception thrown in input of data structure	112
<a href="#">GivError</a>	
Base class for exeception handling in <a href="#">Givaro</a>	112
<a href="#">GivMathDivZero</a>	
Div by 0	113
<a href="#">GivMathError</a>	
Math error	113
<a href="#">GivMMFreeList</a>	
Implementation of a memory manager with free-lists	114
<a href="#">GivMMInfo</a>	
Static informations of memory allocation	114
<a href="#">GivMMRefCount</a>	
Memory management with reference counter on allocated data	114
<a href="#">GivModule</a>	
<a href="#">GivModule</a>	115
<a href="#">givNoCopy</a>	
Used to call ctor without copy	115
<a href="#">givNoInit</a>	
Used to build no initialized object as static object	116
<a href="#">GivRandom</a>	
<a href="#">GivRandom</a>	116
<a href="#">givWithCopy</a>	
Used to call ctor with copy	116
<a href="#">HashTable&lt; T, Key &gt;</a>	
Hash table	117
<a href="#">Indeter</a>	
Indeterminate	117
<a href="#">InitAfter</a>	
<a href="#">InitAfter</a>	117
<a href="#">Integer</a>	
This is the <a href="#">Integer</a> class	118

IntegerDom	
Integer Domain	245
Interpolation< Domain, REDUCE >	
Interpolation	246
IntFactorDom< MyRandIter >	
Integer Factor Domain	247
IntNumTheoDom< MyRandIter >	
Num theory Domain	248
IntPrimeDom	
Primality tests	250
IntRNSsystem< Container, Alloc >	
RNS system class. No doc	250
IntRSADom< MyRandIter >	
RSA domain	251
IntSqrtModDom< MyRandIter >	
Modular square roots	253
Key< T >	
The class Key	254
List0< T >	
ListO	254
Modular< IntType, _Compute_t, Enable >	
Forward declaration for Givaro::Modular	255
Modular< _Storage_t, _Compute_t, typename std::enable_if< is_same_ruint< _Storage_t, _Compute_t >::value  is_smaller_ru	
The standard arithmetic in modular rings using fixed size precision	256
Modular< Integer >	
This class implement the standard arithmetic with Modulo Elements	256
Modular< Log16 >	
This class implement the standard arithmetic with Modulo Elements	257
Modular_implem< _Storage_t, _Compute_t, _Residu_t >	
This class implement the standard arithmetic with Modulo Elements	257
ModularRandIter< Ring >	
Random ring Element generator	258
Montgomery< int32_t >	
This class implements the standard arithmetic with Modulo Elements	261
Montgomery< Reclnt::ruint< K > >	
The recint-based Montgomery ring	262
Neutral	
Neutral type	262
NewtonInterpGeom< Domain, REDUCE >	
Newton	262
NewtonInterpGeomMultip< Domain, REDUCE >	
Newton (multip)	264
ObjectInit	
GivModule	265
OMPTimer	
OMP timer	266
Pair< T1, T2 >	
Pair	266
Poly1CRT< Field >	
Poly1 CRT	266
Poly1Dom< Domain, Dense >	
Class Poly1Dom	267
Poly1FactorDom< Domain, Tag, RandomIterator >	
Poly1FactorDom	268
Poly1PadicDom< Domain, Dense >	
Poly1 p-adic	270
Primes16	
Class Primes16	271

<a href="#">QField&lt; Rational &gt;</a>	
<a href="#">Rational Domain</a>	271
<a href="#">RandomIntegerIterator&lt; _Unsigned, _Exact_Size &gt;</a>	
Random <a href="#">Integer</a> Iterator	272
<a href="#">Rational</a>	
Rationals. No doc	275
<a href="#">RealTimer</a>	
Real timer	277
<a href="#">RefCounter</a>	
Ref counter	277
<a href="#">RefCountPtr&lt; T &gt;</a>	
RefCount Pointer	277
<a href="#">RNSsystem&lt; RING, Domain &gt;</a>	
Class <a href="#">RNSsystem</a>	278
<a href="#">RNSsystemFixed&lt; Ints &gt;</a>	
NO DOC	278
<a href="#">Stack&lt; THING &gt;</a>	
Stack	279
<a href="#">StaticElement&lt; DomainStyle &gt;</a>	
Static Element	279
<a href="#">SysTimer</a>	
Sys timer	280
<a href="#">Timer</a>	
Timer	280
<a href="#">UserTimer</a>	
User timer	285
<a href="#">VectorDom&lt; Domain, StorageTag &gt;</a>	
VectorDom<Domain,StorageTag>	285
<a href="#">ZRing&lt; _Element &gt;</a>	
Class <a href="#">ZRing</a>	285

## Chapter 14

# File Index

### 14.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">all_field.C</a>	289
<a href="#">domain_to_operatorstyle.C</a>	289
<a href="#">exponentiation.C</a>	289
<a href="#">ff_arith.C</a>	290
<a href="#">GF128.C</a>	290
<a href="#">GFirreducible.C</a>	290
<a href="#">gfq_atomic.C</a>	290
<a href="#">Test_Extension.C</a>	290
<a href="#">zpz_atomic.C</a>	290
<a href="#">iexponentiation.C</a>	291
<a href="#">ifactor.C</a>	291
<a href="#">ifactor_lenstra.C</a>	291
<a href="#">igcd.C</a>	291
<a href="#">igcdext.C</a>	291
<a href="#">ilcm.C</a>	292
<a href="#">ispower.C</a>	292
<a href="#">isproot.C</a>	292
<a href="#">lambda.C</a>	292
<a href="#">lambda_inv.C</a>	292
<a href="#">ModularSquareRoot.C</a>	292
<a href="#">nb_primes.C</a>	293
<a href="#">nextprime.C</a>	293
<a href="#">order.C</a>	293
<a href="#">phi.C</a>	293
<a href="#">prevprime.C</a>	293
<a href="#">primitiveelement.C</a>	293
<a href="#">primitiveroot.C</a>	294
<a href="#">probable_primroot.C</a>	294
<a href="#">ProbLucas.C</a>	294
<a href="#">RSA_breaking.C</a>	294
<a href="#">RSA_decipher.C</a>	294
<a href="#">RSA_encipher.C</a>	295
<a href="#">RSA_keys_generator.C</a>	295
<a href="#">highorder.C</a>	295
<a href="#">interpolate.C</a>	295

<a href="#">isirred.C</a>	295
<a href="#">isprimitive.C</a>	296
<a href="#">pol_arith.C</a>	296
<a href="#">pol_eval.C</a>	296
<a href="#">pol_factor.C</a>	296
<a href="#">PolynomialCRT.C</a>	296
<a href="#">trunc_arith.C</a>	297
<a href="#">iratrecon.C</a>	297
<a href="#">polydouble.C</a>	297
<a href="#">givarray0.h</a>	
Array of type T with reference mecanism	297
<a href="#">givarrayallocator.h</a>	
NO DOC	298
<a href="#">givarrayfixed.h</a>	
ArrayFixed of type T with fixed dimension	298
<a href="#">givbits.h</a>	
Field of n bits, for any n	299
<a href="#">givelem.h</a>	
Definition of a reference to an object	299
<a href="#">givhashtable.h</a>	
Hash table	300
<a href="#">givlist0.h</a>	
List of type T with double link and various insert/get/rmv method	301
<a href="#">givstack.h</a>	
No doc	301
<a href="#">chineseremainder.h</a>	
Chinese Remainder Algorithm for 2 Elements	302
<a href="#">extension.h</a>	
NO DOX	302
<a href="#">gfq.h</a>	
Arithmetic on $GF(p^k)$ , with p a prime number less than $2^{16}$	303
<a href="#">gfqext.h</a>	
Arithmetic on $GF(p^k)$ , with p a prime number less than $2^{15}$	304
<a href="#">gfkroncker.h</a>	
Arithmetic on $GF(p^k)$ , with dynamic Kronecker substitution	304
<a href="#">givprimes16.h</a>	
Set of primes less than $2^{16}$	305
<a href="#">givrns.h</a>	
Modular arithmetic for GIVARO	305
<a href="#">givrnsfixed.h</a>	
Chinese Remainder Algorithm	306
<a href="#">StaticElement.h</a>	
NO DOC	307
<a href="#">gmp++_int.h</a>	
Core gmp++_int.h	307
<a href="#">gmp++_int_add.C</a>	
Adding stuff	308
<a href="#">gmp++_int_compare.C</a>	
Routines to compare integers	309
<a href="#">gmp++_int_cstor.C</a>	
Cstoring stuff	309
<a href="#">gmp++_int_div.C</a>	
Diving stuff	310
<a href="#">gmp++_int_gcd.C</a>	
Gcding stuff	310
<a href="#">gmp++_int_io.C</a>	
Ioing stuff	311

<a href="#">gmp++_int_lib.C</a>	
Libing stuff . . . . .	311
<a href="#">gmp++_int_misc.C</a>	
Miscing stuff . . . . .	312
<a href="#">gmp++_int_mod.C</a>	
Moding stuff . . . . .	312
<a href="#">gmp++_int_mul.C</a>	
Muling stuff . . . . .	313
<a href="#">gmp++_int_pow.C</a>	
Powing stuff . . . . .	313
<a href="#">gmp++_int_rand.inl</a>	
Randing stuff . . . . .	314
<a href="#">gmp++_int_sub.C</a>	
Subing stuff . . . . .	314
<a href="#">givinteger.h</a>	
Integer Domain class definition . . . . .	315
<a href="#">givintfactor.h</a>	
Factorisation . . . . .	315
<a href="#">givintnumtheo.h</a>	
Num theory . . . . .	316
<a href="#">givintprime.h</a>	
Primes . . . . .	317
<a href="#">givintrns.h</a>	
Arithmetic for RNS representations . . . . .	318
<a href="#">givintrsa.h</a>	
RSA scheme . . . . .	318
<a href="#">givintsqrootmod.h</a>	
Modular square roots . . . . .	319
<a href="#">givaromm.h</a>	
Memory management in <a href="#">Givaro</a> two memory managers: . . . . .	319
<a href="#">givpointer.h</a>	
Auto ptr management . . . . .	320
<a href="#">givref_count.h</a>	
Definition of the Counter class, Counter . . . . .	321
<a href="#">givrational.h</a>	
Rationals (and domain), composed of an integer (numerator), and a positive integer (denominator) NO DOC . . . . .	321
<a href="#">modular-implem.h</a>	
Generic implementation of Modular . . . . .	322
<a href="#">modular-integral.h</a>	
Representation of $\mathbb{Z}/m\mathbb{Z}$ over int types . . . . .	323
<a href="#">modular.h</a>	
Family of arithmetics over $\mathbb{Z}_p$ ( $\mathbb{Z}/p\mathbb{Z}$ ) . . . . .	323
<a href="#">montgomery.h</a>	
Family of arithmetics over $\mathbb{Z}_p$ ( $\mathbb{Z}/p\mathbb{Z}$ ) . . . . .	323
<a href="#">givbasictype.h</a>	
NO DOC . . . . .	324
<a href="#">givcaster.h</a>	
NO DOC . . . . .	324
<a href="#">givconfig.h</a>	
Configuration file for <a href="#">Givaro</a> . . . . .	325
<a href="#">giverror.h</a>	
Error exception . . . . .	325
<a href="#">givgenarith.h</a>	
Domain definition for basic type of the language . . . . .	326
<a href="#">givinit.h</a>	
NO DOC . . . . .	327

<a href="#">givmodule.h</a>		
NO DOC	.....	327
<a href="#">givperf.h</a>		
Performance analysis	.....	328
<a href="#">givpower.h</a>		
NO DOC	.....	328
<a href="#">givprint.h</a>		
Helper print for containers	.....	328
<a href="#">givranditer.h</a>		
NO DOC <a href="#">Givaro</a> ring Elements generator	.....	329
<a href="#">givrandom.h</a>		
NO DOC	.....	330
<a href="#">givtimer.h</a>		
Timer	.....	330
<a href="#">givdegree.h</a>		
NO DOC opaque class for Degree of polynomial	.....	331
<a href="#">givindeter.h</a>		
Indeterminates for polynomial manipulation	.....	332
<a href="#">givinterp.h</a>		
NO DOC	.....	332
<a href="#">givinterpgeom-multip.h</a>		
Interpolation at geometric points	.....	333
<a href="#">givinterpgeom.h</a>		
Interpolation at geometric points	.....	334
<a href="#">givpoly1.h</a>		
NO DOC	.....	334
<a href="#">givpoly1crt.h</a>		
Polynomial Chinese Remaindering of degree 1	.....	335
<a href="#">givpoly1dense.h</a>		
Univariate polynomial over T	.....	335
<a href="#">givpoly1factor.h</a>		
NO DOC	.....	336
<a href="#">givpoly1padic.h</a>		
NO DOC	.....	337
<a href="#">test-crt.C</a>		
NO DOC	.....	337
<a href="#">test-integer.C</a>		
Tests integer.h fucntions not tested elsewhere	.....	338
<a href="#">test-modsqroot.C</a>		
NO DOC	.....	338
<a href="#">test-random.C</a>		
We test bounds for random Integers	.....	339

## Chapter 15

# Module Documentation

### 15.1 Givaro

[Givaro](#) library starts here.

Collaboration diagram for Givaro:

#### Modules

- [bstruct](#)  
*NO DOC.*
- [GMP](#)  
*Wrapper (and more) around GMP integer interface.*
- [Integer](#)  
*NO DOC NO DOC.*
- [Memory](#)  
*NO DOC.*
- [Rationals](#)  
*NO DOC.*
- [System](#)  
*NO DOC.*

#### 15.1.1 Detailed Description

[Givaro](#) library starts here.

### 15.2 bstruct

NO DOC.

Collaboration diagram for bstruct:

## Files

- file [givarrrayallocator.h](#)  
*NO DOC.*
- file [givarrrayfixed.h](#)  
*ArrayFixed of type T with fixed dimension.*
- file [givbits.h](#)  
*field of n bits, for any n*
- file [givhashtable.h](#)  
*hash table*
- file [givlist0.h](#)  
*List of type T with double link and various insert/get/rmv method.*
- file [givstack.h](#)  
*no doc.*

### 15.2.1 Detailed Description

NO DOC.

NO DOC

## 15.3 GMP

Wrapper (and more) around GMP integer interface.

Collaboration diagram for GMP: Wrapper (and more) around GMP integer interface.

## 15.4 Integer

NO DOC NO DOC.

Collaboration diagram for Integer: NO DOC NO DOC.

## 15.5 Memory

NO DOC.

Collaboration diagram for Memory:

## Files

- file [givaromm.h](#)  
*Memory management in [Givaro](#) two memory managers:*
- file [givpointer.h](#)  
*auto ptr management*
- file [givref\\_count.h](#)  
*Definition of the Counter class, Counter.*

### 15.5.1 Detailed Description

NO DOC.

## 15.6 Rationals

NO DOC.

Collaboration diagram for Rationals:

### Files

- file [givrational.h](#)

*Rationals (and domain), composed of an integer (numerator), and a positive integer (denominator) NO DOC.*

### 15.6.1 Detailed Description

NO DOC.

## 15.7 ZRing

Unparameterized field adapter.

Unparameterized field adapter.

A field having an interface similar to that of floats is adapted to LinBox.

Used to generate efficient field classes for unparameterized fields (or hidden parameter fields).

Some fields are implemented by definition of the C++ arithmetic operators, such as  $z = x*y$ , for  $z, y, z$  instances of a type  $K$ . The LinBox field `Unparametric<K>` is the adaptation to LinBox.

For a typical unparametric field, some of the methods must be defined in a specialization.

## 15.8 System

NO DOC.

Collaboration diagram for System:

## Files

- file [givbasictype.h](#)  
*NO DOC.*
- file [givcaster.h](#)  
*NO DOC.*
- file [givconfig.h](#)  
*configuration file for [Givaro](#)*
- file [giverror.h](#)  
*error exception.*
- file [givgenarith.h](#)  
*Domain definition for basic type of the language.*
- file [givinit.h](#)  
*NO DOC.*
- file [givmodule.h](#)  
*NO DOC.*
- file [givtimer.h](#)  
*timer*
- file [givperf.h](#)  
*performance analysis*
- file [givpower.h](#)  
*NO DOC.*
- file [givrandom.h](#)  
*NO DOC.*
- file [givtimer.h](#)  
*timer*

### 15.8.1 Detailed Description

NO DOC.

## Chapter 16

# Namespace Documentation

### 16.1 Givaro Namespace Reference

Namespace in which the whole [Givaro](#) library resides.

#### Data Structures

- struct [\\_\\_giv\\_map\\_less\\_ith](#)  
*Map opcode on all Elements less or requal that ith.*
- struct [\\_\\_givdom\\_trait\\_name](#)  
*give a name for /read/write*
- struct [\\_perfArray0< T >](#)  
*defined by marco GIVARO\_PERF\_DEFCLASS. ref counting and stuff.*
- class [Array0](#)  
*NODOC.*
- class [Array0Tag](#)  
*Array0Tag.*
- class [ArrayAllocatort](#)  
*ArrayAllocator: class for allocation of arrays.*
- class [ArrayFixed](#)  
*ArrayFixed.*
- class [BaseDomain](#)  
*Base Domain.*
- class [BaseTimer](#)  
*base for class [RealTimer](#); class [SysTimer](#); class [UserTimer](#);*
- class [Bits](#)  
*Bits.*
- class [BlocFreeList](#)  
*Data structure of a bloc.*
- struct [ChineseRemainder](#)  
*CRA.*
- struct [ChineseRemainder< Ring, Domain, false >](#)  
*CRA2.*
- class [Degree](#)  
*Degree type for polynomials.*

- struct [ElemConstRef](#)  
*Elem const Ref.*
- struct [ElemRef](#)  
*Elem Ref.*
- class [Extension](#)  
*Extension.*
- class [FermatDom](#)  
*Fermat numbers.*
- class [GeneralRingNonZeroRandIter](#)  
*Random iterator for nonzero random numbers.*
- class [GeneralRingRandIter](#)  
*UnparametricRandIter.*
- class [GF2](#)  
*Integers modulo 2.*
- class [GFqDom](#)  
*class GFqDom*
- class [GFqExt](#)  
*GFq Ext (other)*
- class [GFqExtFast](#)  
*GFq Ext.*
- struct [GFqKronecker](#)  
*GFqKronecker.*
- class [GIV\\_ExtensionrandIter](#)  
*Extension rand iters.*
- class [GIV\\_randIter](#)  
*Random ring Element generator.*
- class [GivaroAppli](#)  
*Main application class Could be not used.*
- class [GivaroMain](#)  
*Initialisation of GIVARO .*
- class [GivaroMM](#)  
*Memory manager that allocates array of object of type T for.*
- class [GivaroNoInit](#)  
*GivaroNoInit.*
- class [GivBadFormat](#)  
*Exception thrown in input of data structure.*
- class [GivError](#)  
*Base class for exeception handling in Givaro.*
- class [GivMathDivZero](#)  
*Div by 0.*
- class [GivMathError](#)  
*Math error.*
- class [GivMMFreeList](#)  
*Implementation of a memory manager with free-lists.*
- class [GivMMInfo](#)  
*Static informations of memory allocation.*
- class [GivMMRefCount](#)  
*Memory management with reference counter on allocated data.*
- class [GivModule](#)  
*GivModule.*
- class [givNoCopy](#)

- Used to call cstor without copy.*
- class [givNoInit](#)
  - Used to build no initialized object as static object.*
- class [GivRandom](#)
  - GivRandom.*
- class [givWithCopy](#)
  - Used to call cstor with copy.*
- class [HashTable](#)
  - Hash table.*
- class [Indeter](#)
  - Indeterminate.*
- class [InitAfter](#)
  - InitAfter.*
- class [Integer](#)
  - This is the [Integer](#) class.*
- class [IntegerDom](#)
  - [Integer](#) Domain.*
- struct [Interpolation](#)
  - Interpolation.*
- class [IntFactorDom](#)
  - [Integer](#) Factor Domain.*
- class [IntNumTheoDom](#)
  - Num theory Domain.*
- class [IntPrimeDom](#)
  - Primality tests.*
- class [IntRNSsystem](#)
  - RNS system class. No doc.*
- class [IntRSADom](#)
  - RSA domain.*
- class [IntSqrtModDom](#)
  - [Modular](#) square roots.*
- class [Key](#)
  - The class [Key](#).*
- class [List0](#)
  - ListO.*
- class [Modular](#)
  - Forward declaration for [Givaro::Modular](#).*
- class [Modular< \\_Storage\\_t, \\_Compute\\_t, typename std::enable\\_if< is\\_same\\_ruint< \\_Storage\\_t, \\_Compute\\_t >::value||is\\_sm](#)
  - The standard arithmetic in modular rings using fixed size precision.*
- class [Modular< Integer >](#)
  - This class implement the standard arithmetic with Modulo Elements.*
- class [Modular< Log16 >](#)
  - This class implement the standard arithmetic with Modulo Elements.*
- class [Modular\\_implem](#)
  - This class implement the standard arithmetic with Modulo Elements.*
- class [ModularRandIter](#)
  - Random ring Element generator.*
- class [Montgomery< int32\\_t >](#)
  - This class implements the standard arithmetic with Modulo Elements.*
- class [Montgomery< Reclnt::ruint< K > >](#)
  - The recint-based Montgomery ring.*

- class [Neutral](#)  
*Neutral type.*
- struct [NewtonInterpGeom](#)  
*Newton.*
- struct [NewtonInterpGeomMultip](#)  
*Newton (multip)*
- class [ObjectInit](#)  
*GivModule.*
- struct [OMPTimer](#)  
*OMP timer.*
- struct [Pair](#)  
*Pair.*
- class [Poly1CRT](#)  
*Poly1 CRT.*
- class [Poly1Dom< Domain, Dense >](#)  
*Class Poly1Dom.*
- class [Poly1FactorDom](#)  
*Poly1FactorDom.*
- class [Poly1PadicDom< Domain, Dense >](#)  
*Poly1 p-adic.*
- class [Primes16](#)  
*class Primes16*
- class [QField< Rational >](#)  
*Rational Domain.*
- class [RandomIntegerIterator](#)  
*Random Integer Iterator.*
- class [Rational](#)  
*Rationals. No doc.*
- class [RealTimer](#)  
*Real timer.*
- class [RefCounter](#)  
*Ref counter.*
- class [RefCountPtr](#)  
*RefCount Pointer.*
- class [RNSsystem](#)  
*class RNSsystem.*
- class [RNSsystemFixed](#)  
*NO DOC.*
- class [Stack](#)  
*Stack.*
- struct [StaticElement](#)  
*Static Element.*
- class [SysTimer](#)  
*Sys timer.*
- class [Timer](#)  
*Timer.*
- class [UserTimer](#)  
*User timer.*
- class [VectorDom](#)  
*VectorDom< Domain, StorageTag>*
- class [ZRing](#)  
*Class ZRing.*

## Typedefs

- typedef [BaseDomain](#)< char > **CharDom**  
*char dom*
- typedef [BaseDomain](#)< short > **ShortDom**  
*short dom*
- typedef [BaseDomain](#)< int > **IntDom**  
*int dom*
- typedef [BaseDomain](#)< long > **LongDom**  
*long dom*
- typedef [BaseDomain](#)< float > **FloatDom**  
*float dom*
- typedef [BaseDomain](#)< double > **DoubleDom**  
*double dom*
- template<typename T , typename A = std::allocator<T>>  
using **givvector** = std::vector< T, A >  
*givvector*

## Functions

- template<class T1 , class T2 >  
std::ostream & **operator**<< (std::ostream &o, const [Pair](#)< T1, T2 > &p)  
*IO.*
- template<class T1 , class T2 >  
std::istream & **operator**>> (std::istream &fin, [Pair](#)< T1, T2 > &p)  
*IO.*
- template<class Rt >  
Rt **FF\_EXPONENT\_MAX** (const Rt p, const Rt maxe=21)  
*XXX.*
- template<class Rt >  
Rt **FF\_SUBEXPONENT\_MAX** (const Rt p, const Rt e)  
*XXX.*
- template<typename [Field](#) >  
int64\_t **Exponent\_Trait** (const [Field](#) &F)  
*XXX.*
- template<> int64\_t **Exponent\_Trait** (const [GFqDom](#)< int64\_t > &F)  
*XXX.*
- template<typename BaseField >  
int64\_t **Exponent\_Trait** (const [Extension](#)< BaseField > &F)  
*XXX.*
- [Integer](#) & **inv** ([Integer](#) &u, const [Integer](#) &a, const [Integer](#) &b)  
*Modular inverse.*
- [Integer](#) & **invin** ([Integer](#) &u, const [Integer](#) &b)
- [Integer](#) **gcd** (const [Integer](#) &a, const [Integer](#) &b)
- [Integer](#) **pp** (const [Integer](#) &P, const [Integer](#) &Q)
- [Integer](#) & **lcm** ([Integer](#) &g, const [Integer](#) &a, const [Integer](#) &b)
- [Integer](#) **lcm** (const [Integer](#) &a, const [Integer](#) &b)
- [Integer](#) & **pow** ([Integer](#) &Res, const [Integer](#) &n, const int64\_t l)
- [Integer](#) **pow** (const [Integer](#) &n, const int64\_t l)
- [Integer](#) **powmod** (const [Integer](#) &n, const uint64\_t e, const [Integer](#) &m)
- int32\_t **sign** (const [Integer](#) &a)
- int32\_t **compare** (const [Integer](#) &a, const [Integer](#) &b)

- `int32_t absCompare` (const `Integer` &a, const `Integer` &b)
- `int32_t isZero` (const `Integer` &a)
- `int32_t nonZero` (const `Integer` &a)
- `int32_t isOne` (const `Integer` &a)
- `Integer fact` (uint64\_t l)
- `Integer sqrt` (const `Integer` &p)
- `Integer sqrtrem` (const `Integer` &p, `Integer` &rem)
- `Integer & sqrt` (`Integer` &r, const `Integer` &p)
- `Integer & sqrtrem` (`Integer` &r, const `Integer` &p, `Integer` &rem)
- `bool root` (`Integer` &q, const `Integer` &, uint32\_t n)
- `int64_t logp` (const `Integer` &a, const `Integer` &p)
- `double logtwo` (const `Integer` &a)
- `double naturallog` (const `Integer` &a)
- `void swap` (`Integer` &, `Integer` &)
- `bool isOdd` (const `Integer` &a)

*Tests parity of an integer.*

- `uint64_t length` (const `Integer` &a)
- `std::istream & operator>>` (std::istream &i, `Integer` &n)
- `std::ostream & operator<<` (std::ostream &o, const `Integer` &n)
- `std::ostream & absOutput` (std::ostream &o, const `Integer` &n)
- `Integer operator+` (const int32\_t l, const `Integer` &n)
- `int32_t operator!=` (uint32\_t l, const `Integer` &n)
- `int32_t operator==` (uint32\_t l, const `Integer` &n)
- `int32_t operator>` (uint32\_t l, const `Integer` &n)
- `int32_t operator<` (uint32\_t l, const `Integer` &n)
- `int32_t operator>=` (uint32\_t l, const `Integer` &n)
- `int32_t operator<=` (uint32\_t l, const `Integer` &n)
- `Integer operator%` (const int64\_t l, const `Integer` &n)
- `Integer operator*` (const int32\_t l, const `Integer` &n)
- `Integer operator-` (const int32\_t l, const `Integer` &n)
- `std::ostream & operator<<` (std::ostream &o, const `GivMMInfo` &T)

*I/O.*

- `template<typename Storage_t >`  
`Storage_t & gcdext` (`Storage_t` &d, `Storage_t` &u, `Storage_t` &v, const `Storage_t` a, const `Storage_t` b)

*Generalized extended GCD used by specialized `Modular`.*

- `template<typename Storage_t >`  
`std::enable_if< std::is_floating_point< Storage_t >::value, Storage_t & >::type extended_euclid` (`Storage_t` &x, `Storage_t` &d, const `Storage_t` a, const `Storage_t` b)

*Extended Euclidean algorithm computing only the Bezout coefficient for a.*

- `template<typename Storage_t >`  
`Storage_t & invext` (`Storage_t` &x, `Storage_t` &d, const `Storage_t` a, const `Storage_t` b)

*Generalized inversion used by specialized `Modular`.*

- `template<typename T >`  
`unsigned GIVINTLOG` (const T &a)

*Integer log.*

- `template<class TT, class UU >`  
`TT power` (const TT n, const UU l)

*Powering.*

- `template<class D, class TT >`  
`TT & dom_power` (TT &res, const TT &n, long l, const D &F)

*dom\_power*

- `std::ostream & operator<<` (std::ostream &o, const `BaseTimer` &BT)

*I/O.*

- `std::ostream & operator<<` (std::ostream &o, const `Timer` &T)

- I/O.*
- `int64_t value` (const [Degree](#) &d)
- value*
- `int operator==` (const [Indeter](#) &i1, const [Indeter](#) &i2)

### 16.1.1 Detailed Description

Namespace in which the whole [Givaro](#) library resides.

[Todo](#) use NTL if available ?

### 16.1.2 Function Documentation

#### 16.1.2.1 `inv()`

```
Integer & inv (
    Integer & u,
    const Integer & a,
    const Integer & b )
```

[Modular](#) inverse.

Inverse.

Parameters

	<i>a</i>	
	<i>b</i>	
out	<i>u</i>	is set to $a^{-1}$ modulo b

#### 16.1.2.2 `invin()`

```
Integer & invin (
    Integer & u,
    const Integer & b )
```

Parameters

<i>u</i>	
<i>b</i>	

**16.1.2.3 gcd()**

```
Integer gcd (
    const Integer & a,
    const Integer & b )
```

**Parameters**

$a, b$	integers
--------	----------

**Returns**

gcd(a,b)

**16.1.2.4 pp()**

```
Integer pp (
    const Integer & P,
    const Integer & Q )
```

**Parameters**

$P, Q$	params
--------	--------

**16.1.2.5 lcm() [1/2]**

```
Integer & lcm (
    Integer & g,
    const Integer & a,
    const Integer & b )
```

**Parameters**

$g, a, b$	
-----------	--

**Returns**

g=lcm(a,b)

**16.1.2.6 lcm() [2/2]**

```
Integer lcm (
    const Integer & a,
    const Integer & b )
```

## Parameters

$a, b$	
--------	--

**16.1.2.7 pow()** [1/2]

```
Integer & pow (
    Integer & Res,
    const Integer & n,
    const int64_t l )
```

return  $n^l$

## Parameters

$Res, n, l$	
-------------	--

**16.1.2.8 pow()** [2/2]

```
Integer pow (
    const Integer & n,
    const int64_t l )
```

return  $n^l$

## Parameters

$n, l$	
--------	--

**16.1.2.9 powmod()**

```
Integer powmod (
    const Integer & n,
    const uint64_t e,
    const Integer & m )
```

return  $n^e \bmod m$ .

## Parameters

$n, e, m$	
-----------	--

#### 16.1.2.10 sign()

```
int32_t sign (  
    const Integer & a )
```

##### Parameters

<i>a</i>	
----------	--

#### 16.1.2.11 compare()

```
int32_t compare (  
    const Integer & a,  
    const Integer & b )
```

##### Parameters

<i>a</i>	integer
<i>b</i>	integer

##### Returns

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

#### 16.1.2.12 absCompare()

```
int32_t absCompare (  
    const Integer & a,  
    const Integer & b )
```

##### Parameters

<i>a</i>	integer
<i>b</i>	integer

##### Returns

1 if  $|a| > |b|$ , 0 if  $|a| = |b|$  and -1 otherwise.

### 16.1.2.13 isZero()

```
int32_t isZero (
    const Integer & a )
```

#### Parameters

<i>a</i>	
----------	--

### 16.1.2.14 nonZero()

```
int32_t nonZero (
    const Integer & a )
```

#### Parameters

<i>a</i>	
----------	--

### 16.1.2.15 isOne()

```
int32_t isOne (
    const Integer & a )
```

#### Parameters

<i>a</i>	
----------	--

### 16.1.2.16 fact()

```
Integer fact (
    uint64_t l )
```

#### Parameters

<i>l</i>	
----------	--

### 16.1.2.17 sqrt() [1/2]

```
Integer sqrt (
```

```
const Integer & p )
```

#### Parameters

$p$	
-----	--

#### 16.1.2.18 sqrtrem() [1/2]

```
Integer sqrtrem (
    const Integer & p,
    Integer & rem )
```

#### Parameters

$p, rem$	
----------	--

#### 16.1.2.19 sqrt() [2/2]

```
Integer & sqrt (
    Integer & r,
    const Integer & p )
```

#### Parameters

$r, p$	
--------	--

#### 16.1.2.20 sqrtrem() [2/2]

```
Integer & sqrtrem (
    Integer & r,
    const Integer & p,
    Integer & rem )
```

#### Parameters

$r, p, rem$	
-------------	--

#### 16.1.2.21 root()

```
bool root (
```

```
Integer & q,  
const Integer & a,  
uint32_t n )
```

**Parameters**

<i>q,a,n</i>	
--------------	--

**16.1.2.22 logp()**

```
int64_t logp (  
    const Integer & a,  
    const Integer & p )
```

**Parameters**

<i>a,p</i>	
------------	--

**16.1.2.23 logtwo()**

```
double logtwo (  
    const Integer & a )
```

**Parameters**

<i>a</i>	
----------	--

**16.1.2.24 naturallog()**

```
double naturallog (  
    const Integer & a )
```

**Parameters**

<i>a</i>	
----------	--

**16.1.2.25 swap()**

```
void swap (  

```

```
Integer & a,
Integer & b )
```

swap

#### Parameters

<i>a,b</i>	
------------	--

#### 16.1.2.26 isOdd()

```
bool isOdd (
    const Integer & a )
```

Tests parity of an integer.

parity of an integer

#### Parameters

<i>a</i>	integer
----------	---------

#### Returns

1 if odd, 0 if even

#### 16.1.2.27 length()

```
uint64_t length (
    const Integer & a )
```

#### Parameters

<i>a</i>	
----------	--

**Bug** JGD 23.04.2012: shouldn't it be "mp\_limb\_t" instead of "uint64\_t"?

**Bug** JGD 23.04.2012: shouldn't it be "mp\_limb\_t" instead of "uint64\_t"?

### 16.1.2.28 operator>>()

```
std::istream & operator>> (
    std::istream & i,
    Integer & n )
```

in operator.

#### Parameters

<i>i</i>	input stream
<i>n</i>	integer to be built

### 16.1.2.29 operator<<()

```
std::ostream & operator<< (
    std::ostream & o,
    const Integer & n )
```

#### Parameters

<i>o</i>	output stream
<i>n</i>	integer to be printed

### 16.1.2.30 absOutput()

```
std::ostream & absOutput (
    std::ostream & o,
    const Integer & n )
```

#### Parameters

<i>o</i>	output
<i>n</i>	integer

### 16.1.2.31 operator+()

```
Integer operator+ (
    const int32_t l,
    const Integer & n )
```

**Parameters**

<i>l,n</i>	to be added
------------	-------------

**16.1.2.32 operator"!=()**

```
int32_t operator!= (
    uint32_t l,
    const Integer & n )
```

**Parameters**

<i>l,n</i>	integer
------------	---------

**Returns**

1 iff l == n

**16.1.2.33 operator==( ) [1/2]**

```
int32_t operator== (
    uint32_t l,
    const Integer & n )
```

**Parameters**

<i>l,n</i>	integers to compare
------------	---------------------

**16.1.2.34 operator>()**

```
int32_t operator> (
    uint32_t l,
    const Integer & n )
```

**Parameters**

<i>l,n</i>	integers to compare
------------	---------------------

### 16.1.2.35 operator<()

```
int32_t operator< (
    uint32_t l,
    const Integer & n )
```

#### Parameters

<i>l,n</i>	integers to compare
------------	---------------------

### 16.1.2.36 operator>=()

```
int32_t operator>= (
    uint32_t l,
    const Integer & n )
```

#### Parameters

<i>l,n</i>	integers to compare
------------	---------------------

### 16.1.2.37 operator<=()

```
int32_t operator<= (
    uint32_t l,
    const Integer & n )
```

#### Parameters

<i>l,n</i>	integers to compare
------------	---------------------

### 16.1.2.38 operator%()

```
Integer operator% (
    const int64_t l,
    const Integer & n )
```

#### Parameters

<i>l</i>	
<i>n</i>	

**Returns**

n!

**16.1.2.39 operator\*()**

```
Integer operator* (
    const int32_t l,
    const Integer & n )
```

**Parameters**

<i>l,n</i>	to be multpct
------------	---------------

**16.1.2.40 operator-()**

```
Integer operator- (
    const int32_t l,
    const Integer & n )
```

**Parameters**

<i>l,n</i>	to be subtracted
------------	------------------

**16.1.2.41 operator==( ) [2/2]**

```
int operator== (
    const Indeter & i1,
    const Indeter & i2 ) [inline]
```

**Bug** put elsewhere. Inline members functions :

**16.2 RecInt Namespace Reference**

NOTE : For this common file, either basic/reduc.h or mg/reduc.h has to be pre-included.

**16.2.1 Detailed Description**

NOTE : For this common file, either basic/reduc.h or mg/reduc.h has to be pre-included.

The same goes for (basic|mg)/arith/mul.h

## 16.3 std Namespace Reference

STL namespace.

### Functions

- `template<class T , typename A = std::allocator<T>>>  
std::ostream & operator<< (std::ostream &out, const std::vector< T, A > &V)`  
*Prints a vector on output.*
- `template<class S , class T >  
std::ostream & operator<< (std::ostream &out, const std::pair< S, T > &C)`  
*Prints a pair.*
- `template<class T , typename A = std::allocator<T>>>  
std::ostream & operator<< (std::ostream &out, const std::list< T, A > &L)`  
*Prints a list.*
- `template<class T , typename A = std::allocator<T>>>  
std::ostream & operator<< (std::ostream &out, const std::set< T, A > &S)`  
*Prints a set.*

### 16.3.1 Detailed Description

STL namespace.

### 16.3.2 Function Documentation

#### 16.3.2.1 `operator<<()` [1/4]

```
std::ostream & operator<< (
    std::ostream & out,
    const std::vector< T, A > & V )
```

Prints a vector on output.

#### Parameters

<i>o</i>	output stream
<i>v</i>	vector

#### Warning

`<<(ostream&,T&)` exists !

**16.3.2.2 operator<<()** [2/4]

```
std::ostream & operator<< (
    std::ostream & out,
    const std::pair< S, T > & C )
```

Prints a pair.

**Parameters**

<i>o</i>	output stream
<i>C</i>	a pair

**Warning**

<<(ostream&,T&) exists !

**16.3.2.3 operator<<()** [3/4]

```
std::ostream & operator<< (
    std::ostream & out,
    const std::list< T, A > & L )
```

Prints a list.

**Parameters**

<i>o</i>	output stream
<i>C</i>	a pair

**Warning**

<<(ostream&,T&) exists !

**16.3.2.4 operator<<()** [4/4]

```
std::ostream & operator<< (
    std::ostream & out,
    const std::set< T, A > & S )
```

Prints a set.

**Parameters**

<i>o</i>	output stream
<i>C</i>	a pair

**Warning**

<<(ostream&,T&) exists !



## Chapter 17

# Data Structure Documentation

### 17.1 `__giv_map_less_ith`< T, UNARYOP, ith > Struct Template Reference

Map opcode on all Elements less or requal that ith.

#### 17.1.1 Detailed Description

```
template<class T, class UNARYOP, size_t ith>
struct Givaro::__giv_map_less_ith< T, UNARYOP, ith >
```

Map opcode on all Elements less or requal that ith.

Terminal recursion, specialization

The documentation for this struct was generated from the following file:

- [givarrayfixed.h \(4.1.1\)](#)

### 17.2 `__givdom_trait_name`< T > Struct Template Reference

give a name for /read/write

```
#include <givgenarith.h>
```

#### 17.2.1 Detailed Description

```
template<class T>
struct Givaro::__givdom_trait_name< T >
```

give a name for /read/write

The documentation for this struct was generated from the following file:

- [givgenarith.h \(4.1.1\)](#)

## 17.3 `_perfArray0< T >` Struct Template Reference

defined by marco GIVARO\_PERF\_DEFCLASS. ref counting and stuff.

```
#include <givarray0.h>
```

Inheritance diagram for `_perfArray0< T >`:

### 17.3.1 Detailed Description

```
template<class T>
struct Givaro::_perfArray0< T >
```

defined by marco GIVARO\_PERF\_DEFCLASS. ref counting and stuff.

The documentation for this struct was generated from the following file:

- [givarray0.h \(4.1.1\)](#)

## 17.4 `Array0< T >` Class Template Reference

NODOC.

```
#include <givarray0.h>
```

Inheritance diagram for `Array0< T >`:

Collaboration diagram for `Array0< T >`:

### Public Types

- typedef `Type_t` **value\_type**  
*STL compliance.*

## Public Member Functions

- **Array0** (size\_t s=0)  
*Default ctor : ctor of s size array.*
- **Array0** (const Self\_t &p, givNoCopy)  
*Recopy ctor : logical copy.*
- **Array0** (const Self\_t &p, givWithCopy)  
*Recopy ctor : physical copy.*
- **~Array0** ()  
*Destructor.*
- void **destroy** ()  
*Destroy of the array.*
- void **allocate** (size\_t s)  
*Allocation of an array of s Elements.*
- void **reallocate** (size\_t s)  
*Reallocation of an array of s Elements.*
- void **resize** (size\_t s)  
*resize*
- void **reserve** (size\_t s)  
*reserve*
- Self\_t & **copy** (const Self\_t &src)  
*Physical copy operator.*
- Self\_t & **logcopy** (const Self\_t &src)  
*Logical recopy operator: make an alias to src. Return dest.*
- Self\_t & **operator=** (const Self\_t &p)  
*assignement operator is physical copy*
- size\_t **size** () const  
*Return the occupied size of the array.*
- size\_t **phsize** () const  
*Return the physical size of the array (capacity)*
- Type\_t \* **baseptr** ()  
*Return the base ptr to the array.*
- const T & **operator[]** (Indice\_t i) const  
*Access to the ith Element:*
- const T & **front** () const  
*back/front*
- void **push\_back** (const T &a)  
*add one element at the end*
- void **write** (Indice\_t i, const Type\_t &val)  
*write*
- void **read** (Indice\_t i, Type\_t &val) const  
*read*
- Iterator\_t **begin** ()  
*Iterators.*

## Protected Attributes

- `int * _cnt`  
*reference counter on \_d*
- `size_t _size`  
*actual size of the array.*
- `size_t _psz`  
*physical size of the array*
- `T * _d`  
*ptr to the memory*

### 17.4.1 Detailed Description

```
template<class T>
class Givaro::Array0< T >
```

NODOC.

### 17.4.2 Member Function Documentation

#### 17.4.2.1 allocate()

```
void allocate (
    size_t s ) [inline]
```

Allocation of an array of s Elements.

if refcount>1 then it is always a creation of new array

#### 17.4.2.2 reallocate()

```
void reallocate (
    size_t s ) [inline]
```

Reallocation of an array of s Elements.

if refcount>1 then it is always a creation of new array + recopy

#### 17.4.2.3 copy()

```
Array0< T > & copy (
    const Self_t & src ) [inline]
```

Physical copy operator.

reallocate dest of the same size as src (if necessary) and apply GivaroCopyItem<Array<T>,T> on each Element. This class can be specialized. Return dest (i.e, \*this).

### 17.4.3 Field Documentation

#### 17.4.3.1 `_size`

```
size_t _size [protected]
```

actual size of the array.

If ==0 then `_psz=_d=_cnt=0`

The documentation for this class was generated from the following files:

- [givarray0.h \(4.1.1\)](#)
- [givarray0.inl \(4.1.1\)](#)

## 17.5 Array0Tag Class Reference

[Array0Tag.](#)

```
#include <givarrayallocator.h>
```

### 17.5.1 Detailed Description

[Array0Tag.](#)

The documentation for this class was generated from the following file:

- [givarrayallocator.h \(4.1.1\)](#)

## 17.6 ArrayAllocatort< T, Tag > Class Template Reference

ArrayAllocator: class for allocation of arrays.

```
#include <givarrayallocator.h>
```

Inheritance diagram for ArrayAllocatort< T, Tag >:

Collaboration diagram for ArrayAllocatort< T, Tag >:

### Public Types

- `typedef Type_t value_type`  
*STL compliance.*

## Public Member Functions

- void **destroy** ()  
*Destroy of the array.*
- void **allocate** (size\_t s)  
*Allocation of an array of s Elements.*
- void **realloc** (size\_t s)  
*Reallocation of an array of s Elements.*
- void **resize** (size\_t s)  
*resize*
- void **reserve** (size\_t s)  
*reserve*
- Self\_t & **copy** (const Self\_t &src)  
*Physical copy operator.*
- Self\_t & **logcopy** (const Self\_t &src)  
*Logical recopy operator: make an alias to src. Return dest.*
- size\_t **size** () const  
*Return the occupied size of the array.*
- size\_t **phsize** () const  
*Return the physical size of the array (capacity)*
- Type\_t \* **baseptr** ()  
*Return the base ptr to the array.*
- const T & **operator[]** (Indice\_t i) const  
*Access to the ith Element:*
- const T & **front** () const  
*back/front*
- void **push\_back** (const T &a)  
*add one element at the end*
- void **write** (Indice\_t i, const Type\_t &val)  
*write*
- void **read** (Indice\_t i, Type\_t &val) const  
*read*
- Iterator\_t **begin** ()  
*Iterators.*

## Protected Attributes

- int \* **\_cnt**  
*reference counter on \_d*
- size\_t **\_size**  
*actual size of the array.*
- size\_t **\_psz**  
*physical size of the array*
- T \* **\_d**  
*ptr to the memory*

### 17.6.1 Detailed Description

```
template<class T, class Tag>
class Givaro::ArrayAllocatort< T, Tag >
```

ArrayAllocator: class for allocation of arrays.

Specialization: for [Array0Tag](#).

Should have

- `allocate(size_n)`
- `resize(size_n)`
- `destroy`

### 17.6.2 Member Function Documentation

#### 17.6.2.1 `allocate()`

```
void allocate (
    size_t s ) [inline], [inherited]
```

Allocation of an array of s Elements.

if refcount>1 then it is always a creation of new array

#### 17.6.2.2 `reallocate()`

```
void reallocate (
    size_t s ) [inline], [inherited]
```

Reallocation of an array of s Elements.

if refcount>1 then it is always a creation of new array + recopy

#### 17.6.2.3 `copy()`

```
Array0< T > & copy (
    const Self\_t & src ) [inline], [inherited]
```

Physical copy operator.

reallocates dest of the same size as src (if necessary) and apply `GivaroCopyItem<Array<T>,T>` on each Element. This class can be specialized. Return dest (i.e, `*this`).

### 17.6.3 Field Documentation

#### 17.6.3.1 `_size`

```
size_t _size [protected], [inherited]
```

actual size of the array.

If ==0 then `_psz=_d=_cnt=0`

The documentation for this class was generated from the following file:

- [givarrayallocator.h \(4.1.1\)](#)

## 17.7 `ArrayFixed< T, SIZE >` Class Template Reference

[ArrayFixed.](#)

```
#include <givarrayfixed.h>
```

Inherits `_perfArrayFixed< T >`.

### Public Member Functions

- `template<class UNARYOP >`  
void **map** (UNARYOP &opcode)  
*Specialization.*
- `template<class UNARYOP >`  
void **map** (UNARYOP &opcode) const  
*Specialization.*

#### 17.7.1 Detailed Description

```
template<class T, size_t SIZE>
class Givaro::ArrayFixed< T, SIZE >
```

[ArrayFixed.](#)

The documentation for this class was generated from the following file:

- [givarrayfixed.h \(4.1.1\)](#)

## 17.8 BaseDomain< T > Class Template Reference

Base Domain.

```
#include <givgenarith.h>
```

### 17.8.1 Detailed Description

```
template<class T>  
class Givaro::BaseDomain< T >
```

Base Domain.

The documentation for this class was generated from the following file:

- [givgenarith.h \(4.1.1\)](#)

## 17.9 BaseTimer Class Reference

base for class [RealTimer](#); class [SysTimer](#); class [UserTimer](#);

```
#include <givtimer.h>
```

Inheritance diagram for BaseTimer:

### 17.9.1 Detailed Description

base for class [RealTimer](#); class [SysTimer](#); class [UserTimer](#);

Examples

[examples/Polynomial/highorder.C](#), and [examples/Polynomial/trunc\\_arith.C](#).

The documentation for this class was generated from the following files:

- [givtimer.h \(4.1.1\)](#)
- [givtimer.C \(4.1.1\)](#)

## 17.10 Bits Class Reference

[Bits](#).

```
#include <givbits.h>
```

Collaboration diagram for Bits:

### 17.10.1 Detailed Description

[Bits.](#)

The documentation for this class was generated from the following files:

- [givbits.h \(4.1.1\)](#)
- [givbits.C \(4.1.1\)](#)
- [givbits.inl \(4.1.1\)](#)

## 17.11 BlocFreeList Class Reference

Data structure of a bloc.

```
#include <givaromm.h>
```

### 17.11.1 Detailed Description

Data structure of a bloc.

Each bloc in TabFree[id] has a data field of size TabSize[id]

The documentation for this class was generated from the following files:

- [givaromm.h \(4.1.1\)](#)
- [givaromm.C \(4.1.1\)](#)

## 17.12 ChineseRemainder< Ring, Domain, REDUCE > Struct Template Reference

CRA.

```
#include <chineseremainder.h>
```

### 17.12.1 Detailed Description

```
template<class Ring, class Domain, bool REDUCE = true>
struct Givaro::ChineseRemainder< Ring, Domain, REDUCE >
```

CRA.

The documentation for this struct was generated from the following file:

- [chineseremainder.h \(4.1.1\)](#)

## 17.13 ChineseRemainder< Ring, Domain, false > Struct Template Reference

CRA2.

```
#include <chineseremainder.h>
```

### 17.13.1 Detailed Description

```
template<class Ring, class Domain>
struct Givaro::ChineseRemainder< Ring, Domain, false >
```

CRA2.

JGD 05.12.2007: not required anymore ...

The documentation for this struct was generated from the following file:

- [chineseremainder.h \(4.1.1\)](#)

## 17.14 Degree Class Reference

[Degree](#) type for polynomials.

```
#include <givdegree.h>
```

### 17.14.1 Detailed Description

[Degree](#) type for polynomials.

#### Examples

[examples/FiniteField/GF128.C](#), [examples/FiniteField/GFirreducible.C](#), [examples/Integer/ModularSquareRoot.C](#), [examples/Polynomial/PolynomialCRT.C](#), [examples/Polynomial/highorder.C](#), [examples/Polynomial/isprimitive.C](#), [examples/Polynomial/pol\\_arith.C](#), and [examples/Polynomial/trunc\\_arith.C](#).

The documentation for this class was generated from the following files:

- [givdegree.h \(4.1.1\)](#)
- [givdegree.C \(4.1.1\)](#)

## 17.15 ElemConstRef< T > Struct Template Reference

Elem const Ref.

```
#include <givelem.h>
```

### 17.15.1 Detailed Description

```
template<class T>
struct Givaro::ElemConstRef< T >
```

Elem const Ref.

The documentation for this struct was generated from the following file:

- [givelem.h \(4.1.1\)](#)

## 17.16 ElemRef< T > Struct Template Reference

Elem Ref.

```
#include <givelem.h>
```

### 17.16.1 Detailed Description

```
template<class T>
struct Givaro::ElemRef< T >
```

Elem Ref.

The documentation for this struct was generated from the following file:

- [givelem.h \(4.1.1\)](#)

## 17.17 Extension< BFT > Class Template Reference

[Extension.](#)

```
#include <extension.h>
```

Collaboration diagram for Extension< BFT >:

### 17.17.1 Detailed Description

```
template<class BFT = GFqDom<int64_t>>
class Givaro::Extension< BFT >
```

[Extension.](#)

Examples

[examples/FiniteField/Test\\_Extension.C](#), and [examples/Polynomial/PolynomialCRT.C](#).

The documentation for this class was generated from the following file:

- [extension.h \(4.1.1\)](#)

## 17.18 FermatDom Class Reference

Fermat numbers.

```
#include <givintprime.h>
```

Inheritance diagram for FermatDom:

Collaboration diagram for FermatDom:

### Public Member Functions

- **bool isUnit** (const [Rep](#) &x) const  
*isUnit*
- **bool isDivisor** (const [Element](#) &a, const [Element](#) &b) const  
*isDivisor (a, b) Test if  $b \mid a$ .*

#### 17.18.1 Detailed Description

Fermat numbers.

The documentation for this class was generated from the following files:

- [givintprime.h](#) (4.1.1)
- [givintprime.C](#) (4.1.1)

## 17.19 GeneralRingNonZeroRandIter< Ring, RandIter > Class Template Reference

Random iterator for nonzero random numbers.

```
#include <givranditer.h>
```

#### 17.19.1 Detailed Description

```
template<class Ring, class RandIter = typename Ring::RandIter>
class Givaro::GeneralRingNonZeroRandIter< Ring, RandIter >
```

Random iterator for nonzero random numbers.

Wraps around an existing random iterator and ensures that the output is entirely nonzero numbers. Imported from FFLAS-FFPACK NonzeroRandIter - AB 2015-01-12

The documentation for this class was generated from the following file:

- [givranditer.h](#) (4.1.1)

## 17.20 GeneralRingRandIter< Ring > Class Template Reference

UnparametricRandIter.

```
#include <givranditer.h>
```

### 17.20.1 Detailed Description

```
template<class Ring>
class Givaro::GeneralRingRandIter< Ring >
```

UnparametricRandIter.

Imported from FFLAS-FFPACK UnparametricRandIter - AB 2015-01-12

The documentation for this class was generated from the following file:

- [givranditer.h \(4.1.1\)](#)

## 17.21 GF2 Class Reference

Integers modulo 2.

```
#include <gf2.h>
```

### Public Types

- using **Self\_t** = [GF2](#)  
*Element type.*
- using **RandIter** = [GIV\\_randIter](#)< [Self\\_t](#), bool >  
*Random.*

### Public Member Functions

#### Object Management

- **GF2** ()  
*Default constructor.*
- **GF2** (int p, int exp=1)  
*Default constructor.*
- **GF2** (const [GF2](#) &F)  
*Copy constructor.*
- **GF2** & **operator=** (const [GF2](#) &F)  
*Assignment operator.*
- Element **minElement** () const  
*Accessors.*
- Element **maxElement** () const  
*Default constructor.*
- Residu\_t **residu** () const

- *Access to the modulus.*
- Residu\_t **size** () const  
*Default constructor.*
- Residu\_t **characteristic** () const  
*Default constructor.*
- Residu\_t **cardinality** () const  
*Default constructor.*
- template<class T >  
T & **characteristic** (T &p) const  
*Default constructor.*
- template<class T >  
T & **cardinality** (T &p) const  
*Default constructor.*
- Element & **init** (Element &x, const int32\_t &y) const  
*Initialization of field base element from an Integer.*
- Element & **init** (Element &x, const uint32\_t &y) const  
*Default constructor.*
- Element & **init** (Element &x, const int64\_t &y) const  
*Default constructor.*
- Element & **init** (Element &x, const uint64\_t &y) const  
*Default constructor.*
- Element & **init** (Element &x, const float &y) const  
*Default constructor.*
- Element & **init** (Element &x, const double &y) const  
*Default constructor.*
- Element & **init** (Element &x, const Integer &y) const  
*Default constructor.*
- Element & **init** (Element &x) const  
*Default constructor.*
- BitReference **init** (BitReference x, const Integer &y=0) const  
*Default constructor.*
- Integer & **convert** (Integer &x, const Element &y) const  
*Conversion of field base element to a template class T.*
- BitReference **convert** (BitReference x, const Element &y) const  
*Default constructor.*
- template<class T >  
T & **convert** (T &x, const Element &y) const  
*Default constructor.*
- Element & **assign** (Element &x, const Element &y) const  
*Assignment of one field base element to another.*
- BitReference **assign** (BitReference x, const Element &y) const  
*Default constructor.*
- Integer & **cardinality** (Integer &c) const  
*Cardinality.*
- Integer & **characteristic** (Integer &c) const  
*Characteristic.*

### Arithmetic Operations

$x <- y \text{ op } z$ ;  $x <- \text{op } y$  These operations require all elements, including  $x$ , to be initialized before the operation is called.

Uninitialized field base elements will give undefined results.

- bool **areEqual** (const Element &x, const Element &y) const  
*Equality of two elements.*
- bool **isZero** (const Element &x) const  
*Zero equality.*
- bool **isOne** (const Element &x) const  
*One equality.*

- bool `isUnit` (const Element &x) const  
*Invertibility.*
- bool `isMOne` (const Element &x) const  
*MOne equality.*
- Element & `add` (Element &x, const Element &y, const Element &z) const  
*Addition.*
- BitReference `add` (BitReference x, const Element &y, const Element &z) const  
*Equality of two elements.*
- Element & `sub` (Element &x, const Element &y, const Element &z) const  
*Subtraction.*
- BitReference `sub` (BitReference x, const Element &y, const Element &z) const  
*Equality of two elements.*
- Element & `mul` (Element &x, const Element &y, const Element &z) const  
*Multiplication.*
- BitReference `mul` (BitReference x, const Element &y, const Element &z) const  
*Equality of two elements.*
- Element & `div` (Element &x, const Element &y, const Element &z) const  
*Division.*
- BitReference `div` (BitReference x, const Element &y, const Element &z) const  
*Equality of two elements.*
- Element & `neg` (Element &x, const Element &y) const  
*Additive Inverse (Negation).*
- BitReference `neg` (BitReference x, const Element &y) const  
*Equality of two elements.*
- Element & `inv` (Element &x, const Element &y) const  
*Multiplicative Inverse.*
- BitReference `inv` (BitReference x, const Element &y) const  
*Equality of two elements.*
- BitReference `axpy` (BitReference r, const Element &a, const Element &x, const Element &y) const  
*Natural AXPY.*
- Element & `axpy` (Element &r, const Element &a, const Element &x, const Element &y) const  
*Equality of two elements.*
- BitReference `axmy` (BitReference r, const Element &a, const Element &x, const Element &y) const  
*Natural AXMY.*
- Element & `axmy` (Element &r, const Element &a, const Element &x, const Element &y) const  
*Equality of two elements.*
- BitReference `maxpy` (BitReference r, const Element &a, const Element &x, const Element &y) const  
*Natural MAXPY.*
- Element & `maxpy` (Element &r, const Element &a, const Element &x, const Element &y) const  
*Equality of two elements.*

### Input/Output Operations

- std::ostream & `write` (std::ostream &os) const  
*Print field.*
- std::istream & `read` (std::istream &is)  
*Read field.*
- std::ostream & `write` (std::ostream &os, const Element &x) const  
*Print field base element.*
- std::istream & `read` (std::istream &is, Element &x) const  
*Read field base element.*
- std::istream & `read` (std::istream &is, BitReference x) const  
*Print field.*

## Inplace Arithmetic Operations

$x \leftarrow x \text{ op } y$ ;  $x \leftarrow \text{op } x$

- Element & [addin](#) (Element &x, const Element &y) const  
*Inplace Addition.*
- BitReference [addin](#) (BitReference x, const Element &y) const  
*Inplace Addition.*
- Element & [subin](#) (Element &x, const Element &y) const  
*Inplace Subtraction.*
- BitReference [subin](#) (BitReference x, const Element &y) const  
*Inplace Addition.*
- Element & [mulin](#) (Element &x, const Element &y) const  
*Inplace Multiplication.*
- BitReference [mulin](#) (BitReference x, const Element &y) const  
*Inplace Addition.*
- Element & [divin](#) (Element &x, const Element &y) const  
*Inplace Division.*
- BitReference [divin](#) (BitReference x, const Element &y) const  
*Inplace Addition.*
- Element & [negin](#) (Element &x) const  
*Inplace Additive Inverse (Inplace Negation).*
- BitReference [negin](#) (BitReference x) const  
*Inplace Addition.*
- Element & [invin](#) (Element &x) const  
*Inplace Multiplicative Inverse.*
- BitReference [invin](#) (BitReference x) const  
*Inplace Addition.*
- Element & [axpyin](#) (Element &r, const Element &a, const Element &x) const  
*Inplace AXPY.*
- BitReference [axpyin](#) (BitReference r, const Element &a, const Element &x) const  
*Inplace Addition.*
- Element & [axmyin](#) (Element &r, const Element &a, const Element &x) const  
*Inplace AXMY.*
- BitReference [axmyin](#) (BitReference r, const Element &a, const Element &x) const  
*Inplace Addition.*
- Element & [maxpyin](#) (Element &r, const Element &a, const Element &x) const  
*Inplace MAXPY.*
- BitReference [maxpyin](#) (BitReference r, const Element &a, const Element &x) const  
*Inplace Addition.*
- static int [maxCardinality](#) ()  
*Inplace Addition.*

### 17.21.1 Detailed Description

Integers modulo 2.

This is a tuned implementation of the field of integers modulo

1. In particular, when one constructs a VectorDomain object over this field, highly optimized bit operations will be used to make vector arithmetic very fast.

## 17.21.2 Constructor & Destructor Documentation

### 17.21.2.1 GF2()

```
GF2 (
    const GF2 & F ) [inline]
```

Copy constructor.

Constructs [Givaro::GF2](#) object by copying the field. This is required to allow field objects to be passed by value into functions.

Parameters

<i>F</i>	<a href="#">Givaro::GF2</a> object.
----------	-------------------------------------

## 17.21.3 Member Function Documentation

### 17.21.3.1 operator=()

```
GF2 & operator= (
    const GF2 & F ) [inline]
```

Assignment operator.

Required by the archetype

Parameters

<i>F</i>	constant reference to <a href="#">Givaro::Modular</a> object
----------	--

Returns

reference to [Givaro::Modular](#) object for self

### 17.21.3.2 init()

```
Element & init (
    Element & x,
    const int32_t & y ) const [inline]
```

Initialization of field base element from an [Integer](#).

Behaves like C++ allocator construct. This function assumes the output field base element *x* has already been constructed, but that it is not already initialized. This is not a specialization of the template function because such a specialization is not allowed inside the class declaration.

#### Returns

reference to field base element.

#### Parameters

<i>x</i>	field base element to contain output (reference returned).
<i>y</i>	<a href="#">Integer</a> .

### 17.21.3.3 convert()

```
Integer & convert (
    Integer & x,
    const Element & y ) const [inline]
```

Conversion of field base element to a template class *T*.

This function assumes the output field base element *x* has already been constructed, but that it is not already initialized.

#### Returns

reference to template class *T*.

#### Parameters

<i>x</i>	template class <i>T</i> to contain output (reference returned).
<i>y</i>	constant field base element.

### 17.21.3.4 assign()

```
Element & assign (
    Element & x,
    const Element & y ) const [inline]
```

Assignment of one field base element to another.

This function assumes both field base elements have already been constructed and initialized.

#### Returns

reference to *x*

## Parameters

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.5 cardinality()**

```
Integer & cardinality (
    Integer & c ) const [inline]
```

Cardinality.

Return [Integer](#) representing cardinality of the domain. Returns a non-negative [Integer](#) for all domains with finite cardinality, and returns -1 to signify a domain of infinite cardinality.

## Returns

[Integer](#) representing cardinality of the domain

**17.21.3.6 characteristic()**

```
Integer & characteristic (
    Integer & c ) const [inline]
```

Characteristic.

Return [Integer](#) representing characteristic of the domain. Returns a positive [Integer](#) to all domains with finite characteristic, and returns 0 to signify a domain of infinite characteristic.

## Returns

[Integer](#) representing characteristic of the domain.

**17.21.3.7 areEqual()**

```
bool areEqual (
    const Element & x,
    const Element & y ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

## Returns

boolean true if equal, false if not.

**Parameters**

$x$	field base element
$y$	field base element

**17.21.3.8 isZero()**

```
bool isZero (  
    const Element & x ) const [inline]
```

Zero equality.

Test if field base element is equal to zero. This function assumes the field base element has already been constructed and initialized.

**Returns**

boolean true if equals zero, false if not.

**Parameters**

$x$	field base element.
-----	---------------------

**17.21.3.9 isOne()**

```
bool isOne (  
    const Element & x ) const [inline]
```

One equality.

Test if field base element is equal to one. This function assumes the field base element has already been constructed and initialized.

**Returns**

boolean true if equals one, false if not.

**Parameters**

$x$	field base element.
-----	---------------------

#### 17.21.3.10 isUnit()

```
bool isUnit (
    const Element & x ) const [inline]
```

Invertibility.

Test if field base element is invertible. This function assumes the field base element has already been constructed and initialized.

##### Returns

boolean true if equals one, false if not.

##### Parameters

<i>x</i>	field base element.
----------	---------------------

#### 17.21.3.11 isMOne()

```
bool isMOne (
    const Element & x ) const [inline]
```

MOne equality.

Test if field base element is equal to one. This function assumes the field base element has already been constructed and initialized.

##### Returns

boolean true if equals one, false if not.

##### Parameters

<i>x</i>	field base element.
----------	---------------------

#### 17.21.3.12 write() [1/2]

```
std::ostream & write (
    std::ostream & os ) const [inline]
```

Print field.

##### Returns

output stream to which field is written.

## Parameters

<i>os</i>	output stream to which field is written.
-----------	--

**17.21.3.13 read()** [1/3]

```
std::istream & read (  
    std::istream & is ) [inline]
```

Read field.

## Returns

input stream from which field is read.

## Parameters

<i>is</i>	input stream from which field is read.
-----------	--

**17.21.3.14 write()** [2/2]

```
std::ostream & write (  
    std::ostream & os,  
    const Element & x ) const [inline]
```

Print field base element.

This function assumes the field base element has already been constructed and initialized.

## Returns

output stream to which field base element is written.

## Parameters

<i>os</i>	output stream to which field base element is written.
<i>x</i>	field base element.

**17.21.3.15 read()** [2/3]

```
std::istream & read (  

```

```
std::istream & is,
Element & x ) const [inline]
```

Read field base element.

#### Precondition

This function assumes the field base element has already been constructed and initialized.

#### Returns

input stream from which field base element is read.

#### Parameters

<i>is</i>	input stream from which field base element is read.
<i>x</i>	field base element.

### 17.21.3.16 read() [3/3]

```
std::istream & read (
    std::istream & is,
    BitReference x ) const [inline]
```

Print field.

#### Returns

output stream to which field is written.

#### Parameters

<i>os</i>	output stream to which field is written.
-----------	--

### 17.21.3.17 add() [1/2]

```
GF2::Element & add (
    Element & x,
    const Element & y,
    const Element & z ) const [inline]
```

Addition.

$x = y + z$  This function assumes all the field base elements have already been constructed and initialized.

#### Returns

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.
<i>z</i>	field base element.

**17.21.3.18 add() [2/2]**

```
GF2::BitReference add (
    BitReference x,
    const Element & y,
    const Element & z ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

**Returns**

boolean true if equal, false if not.

**Parameters**

<i>x</i>	field base element
<i>y</i>	field base element

**17.21.3.19 sub() [1/2]**

```
GF2::Element & sub (
    Element & x,
    const Element & y,
    const Element & z ) const [inline]
```

Subtraction.

$x = y - z$  This function assumes all the field base elements have already been constructed and initialized.

**Returns**

reference to *x*.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.
<i>z</i>	field base element.

**17.21.3.20 sub() [2/2]**

```
GF2::BitReference sub (
    BitReference x,
    const Element & y,
    const Element & z ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

**Returns**

boolean true if equal, false if not.

**Parameters**

<i>x</i>	field base element
<i>y</i>	field base element

**17.21.3.21 mul() [1/2]**

```
GF2::Element & mul (
    Element & x,
    const Element & y,
    const Element & z ) const [inline]
```

Multiplication.

$x = y * z$  This function assumes all the field base elements have already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.
<i>z</i>	field base element.

**17.21.3.22 mul()** [2/2]

```
GF2::BitReference mul (
    BitReference x,
    const Element & y,
    const Element & z ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

**Returns**

boolean true if equal, false if not.

**Parameters**

<i>x</i>	field base element
<i>y</i>	field base element

**17.21.3.23 div()** [1/2]

```
GF2::Element & div (
    Element & x,
    const Element & y,
    const Element & z ) const [inline]
```

Division.

$x = y / z$  This function assumes all the field base elements have already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.
<i>z</i>	field base element.

**17.21.3.24 div()** [2/2]

```
GF2::BitReference div (
    BitReference x,
```

```
const Element & y,
const Element & z ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

#### Returns

boolean true if equal, false if not.

#### Parameters

<i>x</i>	field base element
<i>y</i>	field base element

### 17.21.3.25 `neg()` [1/2]

```
GF2::Element & neg (
    Element & x,
    const Element & y ) const [inline]
```

Additive Inverse (Negation).

$x = -y$  This function assumes both field base elements have already been constructed and initialized.

#### Returns

reference to *x*.

#### Parameters

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

### 17.21.3.26 `neg()` [2/2]

```
GF2::BitReference neg (
    BitReference x,
    const Element & y ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

#### Returns

boolean true if equal, false if not.

## Parameters

$x$	field base element
$y$	field base element

**17.21.3.27 inv()** [1/2]

```
GF2::Element & inv (
    Element & x,
    const Element & y ) const [inline]
```

Multiplicative Inverse.

$x = 1 / y$  This function assumes both field base elements have already been constructed and initialized.

## Returns

reference to x.

## Parameters

$x$	field base element (reference returned).
$y$	field base element.

**17.21.3.28 inv()** [2/2]

```
GF2::BitReference inv (
    BitReference x,
    const Element & y ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

## Returns

boolean true if equal, false if not.

## Parameters

$x$	field base element
$y$	field base element

**17.21.3.29 axpy()** [1/2]

```
GF2::BitReference axpy (
    BitReference r,
    const Element & a,
    const Element & x,
    const Element & y ) const [inline]
```

Natural AXPY.

**Returns**

reference to r.

**Parameters**

<i>r</i>	
<i>a</i>	
<i>x</i>	
<i>y</i>	

**17.21.3.30 axpy()** [2/2]

```
GF2::Element & axpy (
    Element & r,
    const Element & a,
    const Element & x,
    const Element & y ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

**Returns**

boolean true if equal, false if not.

**Parameters**

<i>x</i>	field base element
<i>y</i>	field base element

**17.21.3.31 axmy()** [1/2]

```
GF2::BitReference axmy (
    BitReference r,
```

```

const Element & a,
const Element & x,
const Element & y ) const [inline]

```

Natural AXMY.

#### Returns

reference to r.

#### Parameters

<i>r</i>	
<i>a</i>	
<i>x</i>	
<i>y</i>	

### 17.21.3.32 axmy() [2/2]

```

GF2::Element & axmy (
    Element & r,
    const Element & a,
    const Element & x,
    const Element & y ) const [inline]

```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

#### Returns

boolean true if equal, false if not.

#### Parameters

<i>x</i>	field base element
<i>y</i>	field base element

### 17.21.3.33 maxpy() [1/2]

```

GF2::BitReference maxpy (
    BitReference r,
    const Element & a,
    const Element & x,
    const Element & y ) const [inline]

```

Natural MAXPY.

**Returns**

reference to r.

**Parameters**

<i>r</i>	
<i>a</i>	
<i>x</i>	
<i>y</i>	

**17.21.3.34 maxpy() [2/2]**

```
GF2::Element & maxpy (
    Element & r,
    const Element & a,
    const Element & x,
    const Element & y ) const [inline]
```

Equality of two elements.

This function assumes both field base elements have already been constructed and initialized.

**Returns**

boolean true if equal, false if not.

**Parameters**

<i>x</i>	field base element
<i>y</i>	field base element

**17.21.3.35 addin() [1/2]**

```
GF2::Element & addin (
    Element & x,
    const Element & y ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to x.

## Parameters

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.36 addin()** [2/2]

```
GF2::BitReference addin (  
    BitReference x,  
    const Element & y ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

## Returns

reference to *x*.

## Parameters

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.37 subin()** [1/2]

```
GF2::Element & subin (  
    Element & x,  
    const Element & y ) const [inline]
```

Inplace Subtraction.

$x -= y$  This function assumes both field base elements have already been constructed and initialized.

## Returns

reference to *x*.

## Parameters

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.38 subin()** [2/2]

```
GF2::BitReference subin (
    BitReference x,
    const Element & y ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to  $x$ .

**Parameters**

$x$	field base element (reference returned).
$y$	field base element.

**17.21.3.39 mulin()** [1/2]

```
GF2::Element & mulin (
    Element & x,
    const Element & y ) const [inline]
```

Inplace Multiplication.

$x *= y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to  $x$ .

**Parameters**

$x$	field base element (reference returned).
$y$	field base element.

**17.21.3.40 mulin()** [2/2]

```
GF2::BitReference mulin (
    BitReference x,
    const Element & y ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.41 divin() [1/2]**

```
GF2::Element & divin (  
    Element & x,  
    const Element & y ) const [inline]
```

Inplace Division.

$x /= y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.42 divin() [2/2]**

```
GF2::BitReference divin (  
    BitReference x,  
    const Element & y ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.43 negin()** [1/2]

```
GF2::Element & negin (
    Element & x ) const [inline]
```

Inplace Additive Inverse (Inplace Negation).

$x = -x$  This function assumes the field base element has already been constructed and initialized.

**Returns**

reference to  $x$ .

**Parameters**

$x$	field base element (reference returned).
-----	--

**17.21.3.44 negin()** [2/2]

```
GF2::BitReference negin (
    BitReference x ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to  $x$ .

**Parameters**

$x$	field base element (reference returned).
$y$	field base element.

**17.21.3.45 invin()** [1/2]

```
GF2::Element & invin (
    Element & x ) const [inline]
```

Inplace Multiplicative Inverse.

$x = 1 / x$  This function assumes the field base element has already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
----------	--

**17.21.3.46 invin() [2/2]**

```
GF2::BitReference invin (
    BitReference x ) const [inline]
```

Inplace Addition.

$x \oplus= y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.47 axpyin() [1/2]**

```
GF2::Element & axpyin (
    Element & r,
    const Element & a,
    const Element & x ) const [inline]
```

Inplace AXPY.

**Returns**

reference to r.

**Parameters**

<i>r</i>	field element (reference returned).
<i>a</i>	field element.
<i>x</i>	field element.

**17.21.3.48 axpyin() [2/2]**

```
GF2::BitReference axpyin (
    BitReference r,
    const Element & a,
    const Element & x ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to x.

**Parameters**

$x$	field base element (reference returned).
$y$	field base element.

**17.21.3.49 axmyin() [1/2]**

```
GF2::Element & axmyin (
    Element & r,
    const Element & a,
    const Element & x ) const [inline]
```

Inplace AXMY.

**Returns**

reference to r.

**Parameters**

$r$	field element (reference returned).
$a$	field element.
$x$	field element.

**17.21.3.50 axmyin() [2/2]**

```
GF2::BitReference axmyin (
    BitReference r,
```

```
const Element & a,
const Element & x ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

#### Returns

reference to x.

#### Parameters

$x$	field base element (reference returned).
$y$	field base element.

#### 17.21.3.51 maxpyin() [1/2]

```
GF2::Element & maxpyin (
    Element & r,
    const Element & a,
    const Element & x ) const [inline]
```

Inplace MAXPY.

#### Returns

reference to r.

#### Parameters

$r$	field element (reference returned).
$a$	field element.
$x$	field element.

#### 17.21.3.52 maxpyin() [2/2]

```
GF2::BitReference maxpyin (
    BitReference r,
    const Element & a,
    const Element & x ) const [inline]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

#### Returns

reference to x.

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

**17.21.3.53 maxCardinality()**

```
static int maxCardinality ( ) [inline], [static]
```

Inplace Addition.

$x += y$  This function assumes both field base elements have already been constructed and initialized.

**Returns**

reference to  $x$ .

**Parameters**

<i>x</i>	field base element (reference returned).
<i>y</i>	field base element.

The documentation for this class was generated from the following files:

- [gf2.h](#) (4.1.1)
- [gf2.inl](#) (4.1.1)

**17.22 GFqDom< TT > Class Template Reference**

class [GFqDom](#)

```
#include <gfq.h>
```

Inheritance diagram for GFqDom< TT >:

**17.22.1 Detailed Description**

```
template<class TT>
class Givaro::GFqDom< TT >
```

class [GFqDom](#)

**Examples**

[examples/FiniteField/GF128.C](#), [examples/FiniteField/GFirreducible.C](#), [examples/FiniteField/Test\\_Extension.C](#), [examples/FiniteField/all\\_field.C](#), [examples/FiniteField/domain\\_to\\_operatorstyle.C](#), [examples/FiniteField/exponentiation.C](#), [examples/FiniteField/ff\\_arith.C](#), [examples/FiniteField/gfq\\_atomic.C](#), [examples/Polynomial/PolynomialCRT.C](#), [examples/Polynomial/highorder.C](#), [examples/Polynomial/isirred.C](#), [examples/Polynomial/isprimitive.C](#), [examples/Polynomial/pol\\_arith.C](#), [examples/Polynomial/pol\\_eval.C](#), [examples/Polynomial/pol\\_factor.C](#), and [examples/Polynomial/trunc\\_arith.C](#).

The documentation for this class was generated from the following files:

- [gfq.h \(4.1.1\)](#)
- [gfq.inl \(4.1.1\)](#)

**17.23 GFqExt< TT > Class Template Reference**

GFq Ext (other)

```
#include <gfqext.h>
```

Inheritance diagram for GFqExt< TT >:

Collaboration diagram for GFqExt< TT >:

**17.23.1 Detailed Description**

```
template<class TT>
class Givaro::GFqExt< TT >
```

GFq Ext (other)

**Examples**

[examples/FiniteField/ff\\_arith.C](#).

The documentation for this class was generated from the following file:

- [gfqext.h \(4.1.1\)](#)

**17.24 GFqExtFast< TT > Class Template Reference**

GFq Ext.

```
#include <gfqext.h>
```

Inheritance diagram for GFqExtFast< TT >:

Collaboration diagram for GFqExtFast< TT >:

### 17.24.1 Detailed Description

```
template<class TT>
class Givaro::GFqExtFast< TT >
```

GFq Ext.

The documentation for this class was generated from the following file:

- [gfqext.h \(4.1.1\)](#)

## 17.25 GFqKronecker< TT, Ints > Struct Template Reference

[GFqKronecker](#).

```
#include <gfqkronecker.h>
```

Inheritance diagram for GFqKronecker< TT, Ints >:

Collaboration diagram for GFqKronecker< TT, Ints >:

### 17.25.1 Detailed Description

```
template<class TT, class Ints>
struct Givaro::GFqKronecker< TT, Ints >
```

[GFqKronecker](#).

The documentation for this struct was generated from the following file:

- [gfqkronecker.h \(4.1.1\)](#)

## 17.26 GIV\_ExtensionrandIter< ExtensionField, Type > Class Template Reference

[Extension](#) rand iters.

```
#include <extension.h>
```

## Common Object Interface.

These methods are required of all LinBox random field Element generators.

- `typedef ExtensionField::PolElement Element`  
*Field Element type.*
- `GIV\_Extensionrandlter (const ExtensionField &F, const Type &size=0, const Type &seed=0)`  
*Constructor from field, sampling size, and seed.*
- `GIV\_Extensionrandlter (const GIV\_Extensionrandlter &R)`  
*Copy constructor.*
- `~GIV\_Extensionrandlter (void)`  
*Destructor.*
- `Element & random (Element &elt) const`  
*Random field Element creator with assignement.*
- `Element & operator\(\) (Element &elt) const`  
*Random field Element creator with assignement.*
- `Element & operator\(\) (void)`  
*Random field Element creator.*
- `const ExtensionField & ring ()`  
*Field Element type.*

### 17.26.1 Detailed Description

```
template<class ExtensionField, class Type>
class Givaro::GIV_Extensionrandlter< ExtensionField, Type >
```

[Extension](#) rand iters.

### 17.26.2 Member Typedef Documentation

#### 17.26.2.1 Element

```
typedef ExtensionField::PolElement Element
```

Field Element type.

The field Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

### 17.26.3 Constructor & Destructor Documentation

### 17.26.3.1 GIV\_ExtensionrandIter() [1/2]

```
GIV_ExtensionrandIter (
    const ExtensionField & F,
    const Type & size = 0,
    const Type & seed = 0 ) [inline]
```

Constructor from field, sampling size, and seed.

The random field Element iterator works in the field  $F$ , is seeded by seed, and it returns any one Element with probability no more than  $1/\min(\text{size}, F.\text{cardinality}())$ . A sampling size of zero means to sample from the entire field. A seed of zero means to use some arbitrary seed for the generator. This implementation sets the sampling size to be no more than the cardinality of the field.

## Parameters

<i>F</i>	LinBox field archetype object in which to do arithmetic
<i>size</i>	constant integer reference of sample size from which to sample (default = 0)
<i>seed</i>	constant integer reference from which to seed random number generator (default = 0)

## 17.26.3.2 GIV\_ExtensionrandIter() [2/2]

```
GIV_ExtensionrandIter (
    const GIV_ExtensionrandIter< ExtensionField, Type > & R ) [inline]
```

Copy constructor.

Constructs ALP\_randIter object by copying the random field Element generator. This is required to allow generator objects to be passed by value into functions. In this implementation, this means copying the random field Element generator to which R.\_randIter\_ptr points.

## Parameters

<i>R</i>	ALP_randIter object.
----------	----------------------

## 17.26.3.3 ~GIV\_ExtensionrandIter()

```
~GIV_ExtensionrandIter (
    void ) [inline]
```

Destructor.

This destructs the random field Element generator object. In this implementation, this destroys the generator by deleting the random generator object to which \_randIter\_ptr points.

## 17.26.4 Member Function Documentation

## 17.26.4.1 random()

```
Element & random (
    Element & elt ) const [inline]
```

Random field Element creator with assignment.

This returns a random field Element from the information supplied at the creation of the generator.

## Returns

random field Element

**17.26.4.2 operator>() [1/2]**

```
Element & operator() (
    Element & elt ) const [inline]
```

Random field Element creator with assignment.

This returns a random field Element from the information supplied at the creation of the generator.

**Returns**

random field Element

**17.26.4.3 operator>() [2/2]**

```
Element & operator() (
    void ) [inline]
```

Random field Element creator.

This returns a random field Element from the information supplied at the creation of the generator.

**Returns**

random field Element

**17.26.4.4 ring()**

```
const ExtensionField & ring ( ) [inline]
```

Field Element type.

The field Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

The documentation for this class was generated from the following file:

- [extension.h \(4.1.1\)](#)

**17.27 GIV\_randIter< Ring, Type > Class Template Reference**

Random ring Element generator.

```
#include <givranditer.h>
```

## Common Object Interface.

These methods are required of all LinBox random ring Element generators.

- typedef Ring::Element [Element](#)  
*Ring Element type.*
- [GIV\\_randlter](#) (const Ring &F, const size\_t size=0, const uint64\_t seed=0)  
*Constructor from ring, sampling size, and seed.*
- [GIV\\_randlter](#) (const [GIV\\_randlter](#) &R)  
*Copy constructor.*
- [~GIV\\_randlter](#) (void)  
*Destructor.*
- [GIV\\_randlter](#)< Ring, Type > & [operator=](#) (const [GIV\\_randlter](#)< Ring, Type > &R)  
*Assignment operator.*
- [Element](#) & [operator\(\)](#) ([Element](#) &elt) const  
*Random ring Element creator with assignement.*
- [Element](#) & [random](#) ([Element](#) &elt) const  
*Ring Element type.*
- [Element](#) [operator\(\)](#) () const  
*Ring Element type.*
- [Element](#) [random](#) () const  
*Ring Element type.*
- const Ring & [ring](#) () const  
*Ring Element type.*

### 17.27.1 Detailed Description

```
template<class Ring, class Type>
class Givaro::GIV_randlter< Ring, Type >
```

Random ring Element generator.

This class defines a ring Element generator for all givaro ring (Gfq and Zpz) through a template argument as a ring. The random generator used is the givrandom.

### 17.27.2 Member Typedef Documentation

#### 17.27.2.1 Element

```
typedef Ring::Element Element
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

### 17.27.3 Constructor & Destructor Documentation

#### 17.27.3.1 GIV\_randIter() [1/2]

```
GIV_randIter (
    const Ring & F,
    const size_t size = 0,
    const uint64_t seed = 0 ) [inline]
```

Constructor from ring, sampling size, and seed.

The random ring Element iterator works in the ring  $F$ , is seeded by  $seed$ , and it returns any one Element with probability no more than  $1/\min(\text{size}, F.\text{cardinality}())$ . A sampling size of zero means to sample from the entire ring. A seed of zero means to use some arbitrary seed for the generator. This implementation sets the sampling size to be no more than the cardinality of the ring.

##### Parameters

$F$	LinBox ring archetype object in which to do arithmetic
$size$	constant integer reference of sample size from which to sample (default = 0)
$seed$	constant integer reference from which to seed random number generator (default = 0)

#### 17.27.3.2 GIV\_randIter() [2/2]

```
GIV_randIter (
    const GIV_randIter< Ring, Type > & R ) [inline]
```

Copy constructor.

Constructs ALP\_randIter object by copying the random ring Element generator. This is required to allow generator objects to be passed by value into functions. In this implementation, this means copying the random ring Element generator to which  $R\_randIter\_ptr$  points.

##### Parameters

$R$	ALP_randIter object.
-----	----------------------

#### 17.27.3.3 ~GIV\_randIter()

```
~GIV_randIter (
    void ) [inline]
```

Destructor.

This destructs the random ring Element generator object. In this implementation, this destroys the generator by deleting the random generator object to which  $\_randIter\_ptr$  points.

## 17.27.4 Member Function Documentation

### 17.27.4.1 operator=()

```
GIV_randIter< Ring, Type > & operator= (
    const GIV_randIter< Ring, Type > & R ) [inline]
```

Assignment operator.

Assigns ALP\_randIter object R to generator. In this implementation, this means copying the generator to which R.\_randIter\_ptr points.

Parameters

<i>R</i>	ALP_randIter object.
----------	----------------------

### 17.27.4.2 operator>() [1/2]

```
Element & operator() (
    Element & elt ) const [inline]
```

Random ring Element creator with assignment.

This returns a random ring Element from the information supplied at the creation of the generator.

Returns

random ring Element

### 17.27.4.3 random() [1/2]

```
Element & random (
    Element & elt ) const [inline]
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

### 17.27.4.4 operator>() [2/2]

```
Element operator() ( ) const [inline]
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

**17.27.4.5 random() [2/2]**

```
Element random ( ) const [inline]
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

**17.27.4.6 ring()**

```
const Ring & ring ( ) const [inline]
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

The documentation for this class was generated from the following file:

- [givranditer.h \(4.1.1\)](#)

**17.28 GivaroAppli Class Reference**

Main application class Could be not used.

```
#include <givinit.h>
```

Inheritance diagram for GivaroAppli:

Collaboration diagram for GivaroAppli:

**17.28.1 Detailed Description**

Main application class Could be not used.

The documentation for this class was generated from the following files:

- [givinit.h \(4.1.1\)](#)
- [givinit.C \(4.1.1\)](#)

**17.29 GivaroMain Class Reference**

Initialisation of GIVARO .

```
#include <givinit.h>
```

Inheritance diagram for GivaroMain:

### 17.29.1 Detailed Description

Initialisation of GIVARO .

- handler to manage signal
- init the memory manager
- init all other modules

The documentation for this class was generated from the following files:

- [givinit.h \(4.1.1\)](#)
- [givinit.C \(4.1.1\)](#)

## 17.30 GivaroMM< T > Class Template Reference

Memory manager that allocates array of object of type T for.

```
#include <givaromm.h>
```

### Static Public Member Functions

- static T \* [allocate](#) (const size\_t s)

### 17.30.1 Detailed Description

```
template<class T>  
class Givaro::GivaroMM< T >
```

Memory manager that allocates array of object of type T for.

### 17.30.2 Member Function Documentation

#### 17.30.2.1 allocate()

```
T * allocate (  
    const size_t s ) [inline], [static]
```

**Bug** implem does not belong here

The documentation for this class was generated from the following file:

- [givaromm.h \(4.1.1\)](#)

## 17.31 GivaroNoInit Class Reference

[GivaroNoInit](#).

```
#include <givmodule.h>
```

### 17.31.1 Detailed Description

[GivaroNoInit](#).

Purpose: used to delay construction of object in the init function of a module definition.

The documentation for this class was generated from the following file:

- [givmodule.h \(4.1.1\)](#)

## 17.32 GivBadFormat Class Reference

Exception thrown in input of data structure.

```
#include <giverror.h>
```

Inheritance diagram for GivBadFormat:

Collaboration diagram for GivBadFormat:

### 17.32.1 Detailed Description

Exception thrown in input of data structure.

The documentation for this class was generated from the following files:

- [giverror.h \(4.1.1\)](#)
- [giverror.C \(4.1.1\)](#)

## 17.33 GivError Class Reference

Base class for exeception handling in [Givaro](#).

```
#include <giverror.h>
```

Inheritance diagram for GivError:

### 17.33.1 Detailed Description

Base class for exception handling in [Givaro](#).

#### Examples

[examples/Polynomial/highorder.C](#).

The documentation for this class was generated from the following files:

- [giverror.h \(4.1.1\)](#)
- [giverror.C \(4.1.1\)](#)

## 17.34 GivMathDivZero Class Reference

Div by 0.

```
#include <giverror.h>
```

Inheritance diagram for GivMathDivZero:

Collaboration diagram for GivMathDivZero:

### 17.34.1 Detailed Description

Div by 0.

The documentation for this class was generated from the following files:

- [giverror.h \(4.1.1\)](#)
- [giverror.C \(4.1.1\)](#)

## 17.35 GivMathError Class Reference

Math error.

```
#include <giverror.h>
```

Inheritance diagram for GivMathError:

Collaboration diagram for GivMathError:

### 17.35.1 Detailed Description

Math error.

The documentation for this class was generated from the following files:

- [giverror.h \(4.1.1\)](#)
- [giverror.C \(4.1.1\)](#)

## 17.36 GivMMFreeList Class Reference

Implementation of a memory manager with free-lists.

```
#include <givaromm.h>
```

Collaboration diagram for GivMMFreeList:

### 17.36.1 Detailed Description

Implementation of a memory manager with free-lists.

All members are static methods.

The documentation for this class was generated from the following files:

- [givaromm.h \(4.1.1\)](#)
- [givaromm.C \(4.1.1\)](#)

## 17.37 GivMMInfo Class Reference

Static informations of memory allocation.

```
#include <givaromm.h>
```

### 17.37.1 Detailed Description

Static informations of memory allocation.

The documentation for this class was generated from the following files:

- [givaromm.h \(4.1.1\)](#)
- [givaromm.C \(4.1.1\)](#)

## 17.38 GivMMRefCount Class Reference

Memory management with reference counter on allocated data.

```
#include <givaromm.h>
```

### 17.38.1 Detailed Description

Memory management with reference counter on allocated data.

The memory manager uses the [BlocFreeList](#) data structure and stores the refcounter in the field data[0]

The documentation for this class was generated from the following files:

- [givaromm.h](#) (4.1.1)
- [givaromm.C](#) (4.1.1)

## 17.39 GivModule Class Reference

[GivModule](#).

```
#include <givmodule.h>
```

### 17.39.1 Detailed Description

[GivModule](#).

Purpose: definition of module with precedence relation use to initialize them between different units compilation.

The documentation for this class was generated from the following files:

- [givmodule.h](#) (4.1.1)
- [givmodule.C](#) (4.1.1)

## 17.40 givNoCopy Class Reference

Used to call ctor without copy.

```
#include <givbasictype.h>
```

### 17.40.1 Detailed Description

Used to call ctor without copy.

The documentation for this class was generated from the following file:

- [givbasictype.h](#) (4.1.1)

## 17.41 givNoInit Class Reference

Used to build no initialized object as static object.

```
#include <givbasictype.h>
```

### 17.41.1 Detailed Description

Used to build no initialized object as static object.

The documentation for this class was generated from the following file:

- [givbasictype.h \(4.1.1\)](#)

## 17.42 GivRandom Class Reference

[GivRandom.](#)

```
#include <givrandom.h>
```

### 17.42.1 Detailed Description

[GivRandom.](#)

Examples

[examples/FiniteField/gfq\\_atomic.C](#), [examples/FiniteField/zpz\\_atomic.C](#), [examples/Integer/ProbLucas.C](#), [examples/Integer/RSA\\_keys\\_generator.C](#), [examples/Polynomial/PolynomialCRT.C](#), and [examples/Polynomial/highorder.C](#).

The documentation for this class was generated from the following file:

- [givrandom.h \(4.1.1\)](#)

## 17.43 givWithCopy Class Reference

Used to call cstor with copy.

```
#include <givbasictype.h>
```

### 17.43.1 Detailed Description

Used to call cstor with copy.

The documentation for this class was generated from the following file:

- [givbasictype.h \(4.1.1\)](#)

## 17.44 HashTable< T, Key > Class Template Reference

Hash table.

```
#include <givhashtable.h>
```

### 17.44.1 Detailed Description

```
template<class T, class Key>  
class Givaro::HashTable< T, Key >
```

Hash table.

The documentation for this class was generated from the following files:

- [givhashtable.h](#) (4.1.1)
- [givhashtable.inl](#) (4.1.1)

## 17.45 Indeter Class Reference

Indeterminate.

```
#include <givindeter.h>
```

### 17.45.1 Detailed Description

Indeterminate.

#### Examples

[examples/FiniteField/GFirreducible.C](#), [examples/Polynomial/highorder.C](#), [examples/Polynomial/interpolate.C](#), [examples/Polynomial/isirred.C](#), [examples/Polynomial/isprimitive.C](#), [examples/Polynomial/pol\\_arith.C](#), [examples/Polynomial/pol\\_eval.C](#), [examples/Polynomial/pol\\_factor.C](#), and [examples/Polynomial/trunc\\_arith.C](#).

The documentation for this class was generated from the following files:

- [givindeter.h](#) (4.1.1)
- [givindeter.C](#) (4.1.1)

## 17.46 InitAfter Class Reference

[InitAfter](#).

```
#include <givmodule.h>
```

Collaboration diagram for [InitAfter](#):

### 17.46.1 Detailed Description

[InitAfter](#).

Purpose: define a precedence relation between two modules.

The documentation for this class was generated from the following files:

- [givmodule.h](#) (4.1.1)
- [givmodule.C](#) (4.1.1)

## 17.47 Integer Class Reference

This is the [Integer](#) class.

```
#include <gmp++_int.h>
```

Inherited by `SpyInteger::InheritsInteger`.

Collaboration diagram for Integer:

### Public Types

- typedef std::vector< mp\_limb\_t > **vect\_t**  
*vector of limbs (ie a gmp number).*

### Public Member Functions

#### Constructor/Destructors.

*Constructors and destructor for an [Integer](#).*

- giv\_all\_inlined [Integer](#) (int32\_t n=0)  
*Constructor form a known type.*
- giv\_all\_inlined [Integer](#) (int64\_t n)  
*Constructor form a known type.*
- giv\_all\_inlined [Integer](#) (unsigned char n)  
*Constructor form a known type.*
- giv\_all\_inlined [Integer](#) (uint32\_t n)  
*Constructor form a known type.*
- giv\_all\_inlined [Integer](#) (uint64\_t n)  
*Constructor form a known type.*
- giv\_all\_inlined [Integer](#) (double n)  
*Constructor form a known type.*
- giv\_all\_inlined [Integer](#) (const char \*n)  
*Constructor form a known type.*
- giv\_all\_inlined [Integer](#) (const mpz\_class &a)  
*Constructor form a known type.*
- template<size\_t K>  
[Integer](#) (const Reclnt::ruint< K > &n)  
*Constructor form a known type.*
- template<size\_t K>  
[Integer](#) (const Reclnt::rint< K > &n)

- *Constructor from a known type.*  
giv\_all\_inlined **Integer** (const **Integer** &n)
- *Copy constructor.*  
giv\_all\_inlined **Integer** (uint64\_t \*d, int64\_t sz)
- *Creates a new Integer from pointers.*  
giv\_all\_inlined **Integer** (const vect\_t &v)
- *Creates a new Integers for a vector of limbs.*  
giv\_all\_inlined ~**Integer** ()
- *destructor*

### Assignment and copy operators

- giv\_all\_inlined **Integer** & operator= (const **Integer** &n)  
*copy from an integer.*
- giv\_all\_inlined **Integer** & logcopy (const **Integer** &n)  
*copy from an integer.*
- giv\_all\_inlined **Integer** & copy (const **Integer** &n)  
*copy from an integer.*

### Bit logic operators. <br>

- giv\_all\_inlined **Integer** operator^ (const **Integer** &a) const  
*XOR (^)*
- giv\_all\_inlined **Integer** operator^ (const uint64\_t &a) const  
*XOR (^)*
- giv\_all\_inlined **Integer** operator^ (const uint32\_t &a) const  
*XOR (^)*
- giv\_all\_inlined **Integer** & operator^= (const **Integer** &a)  
*XOR inplace (^=)*
- giv\_all\_inlined **Integer** & operator^= (const uint64\_t &a)  
*XOR (^)*
- giv\_all\_inlined **Integer** & operator^= (const uint32\_t &a)  
*XOR (^)*
- giv\_all\_inlined **Integer** operator| (const **Integer** &a) const  
*OR (|)*
- giv\_all\_inlined **Integer** operator| (const uint64\_t &a) const  
*XOR (^)*
- giv\_all\_inlined **Integer** operator| (const uint32\_t &a) const  
*XOR (^)*
- giv\_all\_inlined **Integer** & operator|= (const **Integer** &a)  
*OR inplace (|=)*
- giv\_all\_inlined **Integer** & operator|= (const uint64\_t &a)  
*XOR (^)*
- giv\_all\_inlined **Integer** & operator|= (const uint32\_t &a)  
*XOR (^)*
- giv\_all\_inlined **Integer** operator& (const **Integer** &a) const  
*AND (&)*
- giv\_all\_inlined uint32\_t operator& (const uint32\_t &a) const  
*XOR (^)*
- giv\_all\_inlined uint64\_t operator& (const uint64\_t &a) const  
*XOR (^)*
- giv\_all\_inlined **Integer** & operator&= (const **Integer** &a)  
*AND inplace (&=)*
- giv\_all\_inlined **Integer** & operator&= (const uint64\_t &a)  
*XOR (^)*
- giv\_all\_inlined **Integer** & operator&= (const uint32\_t &a)  
*XOR (^)*
- giv\_all\_inlined **Integer** operator~ () const

- *complement to 1 (~)*
- `giv_all_inlined Integer operator<< (int32_t l) const`  
*left shift (<<)*
- `giv_all_inlined Integer operator<< (int64_t l) const`  
*XOR (^)*
- `giv_all_inlined Integer operator<< (uint32_t l) const`  
*XOR (^)*
- `giv_all_inlined Integer operator<< (uint64_t l) const`  
*XOR (^)*
- `giv_all_inlined Integer & operator<<= (int32_t l)`  
*left shift inplace (<<=)*
- `giv_all_inlined Integer & operator<<= (int64_t l)`  
*XOR (^)*
- `giv_all_inlined Integer & operator<<= (uint32_t l)`  
*XOR (^)*
- `giv_all_inlined Integer & operator<<= (uint64_t l)`  
*XOR (^)*
- `giv_all_inlined Integer operator>> (int32_t l) const`  
*right shift (>>)*
- `giv_all_inlined Integer operator>> (int64_t l) const`  
*XOR (^)*
- `giv_all_inlined Integer operator>> (uint32_t l) const`  
*XOR (^)*
- `giv_all_inlined Integer operator>> (uint64_t l) const`  
*XOR (^)*
- `giv_all_inlined Integer & operator>>= (int32_t l)`  
*right shift inplace (>>=)*
- `giv_all_inlined Integer & operator>>= (int64_t l)`  
*XOR (^)*
- `giv_all_inlined Integer & operator>>= (uint32_t l)`  
*XOR (^)*
- `giv_all_inlined Integer & operator>>= (uint64_t l)`  
*XOR (^)*

### Increment/Decrement operators

- `Integer & operator++ ()`
- `Integer operator++ (int)`
- `Integer & operator-- ()`
- `Integer operator-- (int)`

### Cast operators.

Convert an *Integer* to a basic C++ type.

#### Warning

Cast towards **unsigned** consider only the absolute value

- `operator bool () const`
- `operator int16_t () const`
- `operator uint16_t () const`
- `operator unsigned char () const`
- `giv_all_inlined operator uint32_t () const`
- `giv_all_inlined operator int32_t () const`
- `operator signed char () const`
- `giv_all_inlined operator uint64_t () const`
- `giv_all_inlined operator int64_t () const`
- `giv_all_inlined operator std::string () const`
- `giv_all_inlined operator float () const`
- `giv_all_inlined operator double () const`
- `giv_all_inlined operator vect_t () const`
- `template<size_t K>`  
`operator RecInt::ruint< K > () const`
- `template<size_t K>`  
`operator RecInt::rint< K > () const`

## Static Public Member Functions

### Addition, subtraction, multiplication

- static giv\_all\_inlined [Integer](#) & [addin](#) ([Integer](#) &res, const [Integer](#) &n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [addin](#) ([Integer](#) &res, const int64\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [addin](#) ([Integer](#) &res, const uint64\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [addin](#) ([Integer](#) &res, const int32\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [addin](#) ([Integer](#) &res, const uint32\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [add](#) ([Integer](#) &res, const [Integer](#) &n1, const [Integer](#) &n2)  
*Addition res=n1+n2.*
- static giv\_all\_inlined [Integer](#) & [add](#) ([Integer](#) &res, const [Integer](#) &n1, const int64\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [add](#) ([Integer](#) &res, const [Integer](#) &n1, const uint64\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [add](#) ([Integer](#) &res, const [Integer](#) &n1, const int32\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [add](#) ([Integer](#) &res, const [Integer](#) &n1, const uint32\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [subin](#) ([Integer](#) &res, const [Integer](#) &n)  
*Subtraction (inplace) res-=n.*
- static giv\_all\_inlined [Integer](#) & [subin](#) ([Integer](#) &res, const int64\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [subin](#) ([Integer](#) &res, const uint64\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [subin](#) ([Integer](#) &res, const int32\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [subin](#) ([Integer](#) &res, const uint32\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [sub](#) ([Integer](#) &res, const [Integer](#) &n1, const [Integer](#) &n2)  
*Subtraction res=n1-n2.*
- static giv\_all\_inlined [Integer](#) & [sub](#) ([Integer](#) &res, const [Integer](#) &n1, const int64\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [sub](#) ([Integer](#) &res, const [Integer](#) &n1, const uint64\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [sub](#) ([Integer](#) &res, const [Integer](#) &n1, const int32\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [sub](#) ([Integer](#) &res, const [Integer](#) &n1, const uint32\_t n2)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [negin](#) ([Integer](#) &res)  
*Negation (inplace) res=-res.*
- static giv\_all\_inlined [Integer](#) & [neg](#) ([Integer](#) &res, const [Integer](#) &n)  
*Negation res=-n.*
- static giv\_all\_inlined [Integer](#) & [mulin](#) ([Integer](#) &res, const [Integer](#) &n)  
*Multiplication (inplace) res\*=n.*
- static giv\_all\_inlined [Integer](#) & [mulin](#) ([Integer](#) &res, const int64\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [mulin](#) ([Integer](#) &res, const uint64\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [mulin](#) ([Integer](#) &res, const int32\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [mulin](#) ([Integer](#) &res, const uint32\_t n)  
*Addition (inplace) res+=n.*
- static giv\_all\_inlined [Integer](#) & [mul](#) ([Integer](#) &res, const [Integer](#) &n1, const [Integer](#) &n2)

- Multiplication*  $res = n1 * n2$ .
- static giv\_all\_inlined **Integer** & **mul** (**Integer** &res, const **Integer** &n1, const int64\_t n2)  
*Addition (inplace)*  $res += n$ .
- static giv\_all\_inlined **Integer** & **mul** (**Integer** &res, const **Integer** &n1, const uint64\_t n2)  
*Addition (inplace)*  $res += n$ .
- static giv\_all\_inlined **Integer** & **mul** (**Integer** &res, const **Integer** &n1, const int32\_t n2)  
*Addition (inplace)*  $res += n$ .
- static giv\_all\_inlined **Integer** & **mul** (**Integer** &res, const **Integer** &n1, const uint32\_t n2)  
*Addition (inplace)*  $res += n$ .

### fused add-multiply

Groups a multiplication and an addition/division in a single function.

This is usually faster than doing the two operations separately (and preferable to using operators).

- static giv\_all\_inlined **Integer** & **axy** (**Integer** &res, const **Integer** &a, const **Integer** &x, const **Integer** &y)  
 $axy\ res = ax + y$ .
- static giv\_all\_inlined **Integer** & **axy** (**Integer** &res, const **Integer** &a, const uint64\_t x, const **Integer** &y)  
 $axy\ res = ax + y$ .
- static giv\_all\_inlined **Integer** & **axyin** (**Integer** &res, const **Integer** &a, const **Integer** &x)  
*axyin (inplace)*  $res += ax$ .
- static giv\_all\_inlined **Integer** & **axyin** (**Integer** &res, const **Integer** &a, const uint64\_t x)  
 $axy\ res = ax + y$ .
- static giv\_all\_inlined **Integer** & **maxpy** (**Integer** &res, const **Integer** &a, const **Integer** &x, const **Integer** &y)  
 $maxpy\ res = y - ax$ .
- static giv\_all\_inlined **Integer** & **maxpy** (**Integer** &res, const **Integer** &a, const uint64\_t x, const **Integer** &y)  
 $axy\ res = ax + y$ .
- static giv\_all\_inlined **Integer** & **maxpyin** (**Integer** &res, const **Integer** &a, const **Integer** &x)  
*maxpyin (inplace)*  $res -= ax$ .
- static giv\_all\_inlined **Integer** & **maxpyin** (**Integer** &res, const **Integer** &a, const uint64\_t x)  
 $axy\ res = ax + y$ .
- static giv\_all\_inlined **Integer** & **axmy** (**Integer** &res, const **Integer** &a, const **Integer** &x, const **Integer** &y)  
 $axmy\ res = ax - y$ .
- static giv\_all\_inlined **Integer** & **axmy** (**Integer** &res, const **Integer** &a, const uint64\_t x, const **Integer** &y)  
 $axy\ res = ax + y$ .
- static giv\_all\_inlined **Integer** & **axmyin** (**Integer** &res, const **Integer** &a, const **Integer** &x)  
*axmyin (in place)*  $res = ax - res$ .
- static giv\_all\_inlined **Integer** & **axmyin** (**Integer** &res, const **Integer** &a, const uint64\_t x)  
 $axy\ res = ax + y$ .

### Random numbers functions

Other stuff gmp has (temporary)

- static void **seeding** (uint64\_t s)  
*Random numbers (no doc)*
- static void **seeding** (const **Integer** &s)  
*Random numbers (no doc)*
- static void **seeding** ()  
*Random numbers (no doc)*
- static bool **RandBool** ()  
*Random numbers (no doc)*
- template<bool ALWAYSPOSITIVE>  
static **Integer** & **random\_lesssthan** (**Integer** &r, const **Integer** &m)  
*returns a random integer  $r$  in the interval  $[[x, m-1]]$  where  $x = 0$  or  $-(m-1)$  according to ALWAYSPOSITIVE*
- static **Integer** & **random\_lesssthan** (**Integer** &r, const **Integer** &m)  
*Random numbers (no doc)*
- template<bool ALWAYSPOSITIVE>  
static **Integer** & **random\_lesssthan\_2exp** (**Integer** &r, const uint64\_t &m)

- returns a random integer  $x$  in the interval  $[[x, 2^{m-1}]]$  where  $x = 0$  or  $-(2^{m-1})$  according to `ALWAYSPOSITIVE` returns a random integer  $x$  of at most  $m$  bits
- static `Integer & random_less-than_2exp (Integer &r, const uint64_t &m)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE>  
static `Integer random_less-than_2exp (const uint64_t &m)`  
Random numbers (no doc)
  - static `Integer random_less-than_2exp (const uint64_t &m)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE>  
static `Integer & random_less-than (Integer &r, const uint64_t &m)`  
Random numbers (no doc)
  - static `Integer & random_less-than (Integer &r, const uint64_t &m)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE, class T >  
static `Integer random_less-than (const T &m)`  
Random numbers (no doc)
  - template<class T >  
static `Integer random_less-than (const T &m)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE>  
static `Integer & random_exact_2exp (Integer &r, const uint64_t &m)`  
returns a reference to a random number  $x$  of the size  $m$  bits, exactly.
  - static `Integer & random_exact_2exp (Integer &r, const uint64_t &m)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE>  
static `Integer & random_exact (Integer &r, const Integer &s)`  
returns a reference to a random number  $x$  of the size of  $s$ , exactly.
  - static `Integer & random_exact (Integer &r, const Integer &s)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE>  
static `Integer & random_exact (Integer &r, const uint64_t &m)`  
Random numbers (no doc)
  - static `Integer & random_exact (Integer &r, const uint64_t &m)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE, class T >  
static `Integer & random_exact (Integer &r, const T &m)`  
Random numbers (no doc)
  - template<class T >  
static `Integer & random_exact (Integer &r, const T &m)`  
Random numbers (no doc)
  - template<bool ALWAYSPOSITIVE, class T >  
static `Integer random_exact (const T &s)`  
Random numbers (no doc)
  - template<class T >  
static `Integer random_exact (const T &s)`  
Random numbers (no doc)
  - static `Integer & random_between (Integer &r, const Integer &m, const Integer &M)`  
Random numbers (no doc)
  - static `Integer random_between (const Integer &m, const Integer &M)`  
Random numbers (no doc)
  - static `Integer & random_between_2exp (Integer &r, const uint64_t &m, const uint64_t &M)`  
Random numbers (no doc)
  - static `Integer & random_between (Integer &r, const uint64_t &m, const uint64_t &M)`  
Random numbers (no doc)
  - static `Integer random_between_2exp (const uint64_t &m, const uint64_t &M)`  
Random numbers (no doc)
  - static `Integer random_between (const uint64_t &m, const uint64_t &M)`  
Random numbers (no doc)

- `template<class R >`  
static `Integer random_between` (const R &m, const R &M)  
*Random numbers (no doc)*
- `template<class R >`  
static `Integer & random_between` (`Integer` &r, const R &m, const R &M)  
*Random numbers (no doc)*
- `template<bool ALWAYSPOSITIVE, class T >`  
static `Integer & random` (`Integer` &r, const T &m)  
*returns a random integer less than...*
- `template<class T >`  
static `Integer & random` (`Integer` &r, const T &m)  
*Random numbers (no doc)*
- `template<bool ALWAYSPOSITIVE, class T >`  
static `Integer random` (const T &sz)  
*returns a random integer less than...*
- `template<class T >`  
static `Integer random` (const T &sz)  
*Random numbers (no doc)*
- `template<bool ALWAYSPOSITIVE>`  
static `Integer random` ()  
*Random numbers (no doc)*
- static `Integer random` ()  
*Random numbers (no doc)*
- `template<bool ALWAYSPOSITIVE, class T >`  
static `Integer nonzerorandom` (const T &sz)  
*Random numbers (no doc)*
- `template<bool ALWAYSPOSITIVE, class T >`  
static `Integer & nonzerorandom` (`Integer` &r, const T &size)  
*Random numbers (no doc)*
- `template<class T >`  
static `Integer nonzerorandom` (const T &sz)  
*Random numbers (no doc)*
- `template<class T >`  
static `Integer & nonzerorandom` (`Integer` &r, const T &size)  
*Random numbers (no doc)*
- static `Integer nonzerorandom` ()  
*Random numbers (no doc)*

## Static Public Attributes

- static const `Integer zero`  
*zero (0)*
- static const `Integer one`  
*one (1)*
- static const `Integer mOne`  
*minus one (-1)*

## Friends

- `giv_all_inlined Integer & powmod` (`Integer` &Res, const `Integer` &n, const `uint64_t` e, const `Integer` &m)  
*modular pow. return  $n^e \bmod m$ .*
- `giv_all_inlined Integer powmod` (const `Integer` &n, const `uint64_t` e, const `Integer` &m)  
*modular pow.*
- `giv_all_inlined Integer fact` (`uint64_t` l)

- fact*
- `giv_all_inlined Integer sqrt` (const `Integer` &p)
- (square) roots*
- `giv_all_inlined Integer & sqrt` (`Integer` &r, const `Integer` &p)
- (square) roots*
- `giv_all_inlined Integer sqrtrem` (const `Integer` &p, `Integer` &rem)
- (square) roots*
- `giv_all_inlined Integer & sqrtrem` (`Integer` &r, const `Integer` &p, `Integer` &rem)
- (square) roots*
- `giv_all_inlined bool root` (`Integer` &q, const `Integer` &a, `uint32_t` n)
- (square) roots*
- `giv_all_inlined int64_t logp` (const `Integer` &a, const `Integer` &p)
- logs*
- `giv_all_inlined double logtwo` (const `Integer` &a)
- logs*
- `giv_all_inlined double naturallog` (const `Integer` &a)
- logs*

### Arithmetic functions

- `giv_all_inlined Integer gcd` (const `Integer` &a, const `Integer` &b)
- gcd.*
- `giv_all_inlined Integer gcd` (`Integer` &u, `Integer` &v, const `Integer` &a, const `Integer` &b)
- gcd.*
- `giv_all_inlined Integer & gcd` (`Integer` &g, const `Integer` &a, const `Integer` &b)
- gcd.*
- `giv_all_inlined Integer & gcd` (`Integer` &g, `Integer` &u, `Integer` &v, const `Integer` &a, const `Integer` &b)
- gcd.*
- `giv_all_inlined Integer & inv` (`Integer` &u, const `Integer` &a, const `Integer` &b)
- Inverse.*
- `giv_all_inlined Integer & invin` (`Integer` &u, const `Integer` &b)
- Compute the inverse inplace u = u/b.*
- `giv_all_inlined Integer pp` (const `Integer` &P, const `Integer` &Q)
- pp*
- `giv_all_inlined Integer & lcm` (`Integer` &g, const `Integer` &a, const `Integer` &b)
- lcm*
- `giv_all_inlined Integer lcm` (const `Integer` &a, const `Integer` &b)
- lcm*
- `giv_all_inlined Integer & pow` (`Integer` &Res, const `Integer` &n, const `int64_t` l)
- pow.*
- `giv_all_inlined Integer & pow` (`Integer` &Res, const `uint64_t` n, const `uint64_t` l)
- gcd.*
- `giv_all_inlined Integer & pow` (`Integer` &Res, const `Integer` &n, const `uint64_t` l)
- gcd.*
- `giv_all_inlined Integer & pow` (`Integer` &Res, const `Integer` &n, const `int32_t` l)
- gcd.*
- `giv_all_inlined Integer & pow` (`Integer` &Res, const `Integer` &n, const `uint32_t` l)
- gcd.*
- `giv_all_inlined Integer pow` (const `Integer` &n, const `int64_t` l)
- pow.*
- `giv_all_inlined Integer pow` (const `Integer` &n, const `uint64_t` l)
- gcd.*
- `giv_all_inlined Integer pow` (const `Integer` &n, const `int32_t` l)
- gcd.*
- `giv_all_inlined Integer pow` (const `Integer` &n, const `uint32_t` l)
- gcd.*

## primes

- `giv_all_inlined Integer & Protected::prevprime (Integer &, const Integer &p)`
- `giv_all_inlined Integer & Protected::nextprime (Integer &, const Integer &p)`
- `giv_all_inlined int32_t Protected::probab_prime (const Integer &p, int32_t r)`
- `giv_all_inlined int32_t jacobi (const Integer &u, const Integer &v)`
- `giv_all_inlined int32_t legendre (const Integer &u, const Integer &v)`

## Comparison operators.

Compare with operators.

- `giv_all_inlined int32_t operator>= (const Integer &l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator>= (const int32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator>= (const int64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator>= (const uint64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator>= (const uint32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator>= (const double l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator>= (const float l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator<= (const Integer &l) const`  
*less or equal*
- `giv_all_inlined int32_t operator<= (const int32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator<= (const int64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator<= (const uint64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator<= (const uint32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator<= (const double l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator<= (const float l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator!= (const Integer &l) const`  
*operator != (not equal)*
- `giv_all_inlined int32_t operator!= (const int32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator!= (const int64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator!= (const uint64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator!= (const uint32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator!= (const double l) const`

- greater or equal.*
- `giv_all_inlined int32_t operator!= (const float l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator== (const Integer &l) const`  
*Equality.*
- `giv_all_inlined int32_t operator== (const int32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator== (const int64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator== (const uint64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator== (const uint32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator== (const double l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator== (const float l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator> (const Integer &l) const`  
*greater (strict)*
- `giv_all_inlined int32_t operator> (const int32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator> (const int64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator> (const uint64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator> (const uint32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator> (const double l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator> (const float l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator< (const Integer &l) const`  
*less (strict)*
- `giv_all_inlined int32_t operator< (const int32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator< (const int64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator< (const uint64_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator< (const uint32_t l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator< (const double l) const`  
*greater or equal.*
- `giv_all_inlined int32_t operator< (const float l) const`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator>= (uint32_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator>= (float l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator>= (double l, const Integer &n)`  
*greater or equal.*

- `giv_all_inlined friend int32_t operator>= (int32_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator>= (int64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator>= (uint64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator<= (uint32_t l, const Integer &n)`  
*less or equal*
- `giv_all_inlined friend int32_t operator<= (float l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator<= (double l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator<= (int32_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator<= (int64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator<= (uint64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator!= (uint32_t l, const Integer &n)`  
*operator != (not equal)*
- `giv_all_inlined friend int32_t operator!= (float l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator!= (double l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator!= (int32_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator!= (int64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator!= (uint64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator== (uint32_t l, const Integer &n)`  
*Equality.*
- `giv_all_inlined friend int32_t operator== (float l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator== (double l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator== (int32_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator== (int64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator== (uint64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator> (uint32_t l, const Integer &n)`  
*greater (strict)*
- `giv_all_inlined friend int32_t operator> (float l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator> (double l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator> (int32_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator> (int64_t l, const Integer &n)`

- greater or equal.*
- `giv_all_inlined friend int32_t operator> (uint64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator< (uint32_t l, const Integer &n)`  
*less (strict)*
- `giv_all_inlined friend int32_t operator< (float l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator< (double l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator< (int32_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator< (int64_t l, const Integer &n)`  
*greater or equal.*
- `giv_all_inlined friend int32_t operator< (uint64_t l, const Integer &n)`  
*greater or equal.*

### Addition, subtraction, multiplication operators

- `giv_all_inlined Integer operator+ (const Integer &n) const`  
*operator +.*
- `giv_all_inlined Integer operator+ (const uint64_t n) const`  
*operator +.*
- `giv_all_inlined Integer operator+ (const int64_t n) const`  
*operator +.*
- `giv_all_inlined Integer operator+ (const uint32_t n) const`  
*operator +.*
- `giv_all_inlined Integer operator+ (const int32_t n) const`  
*operator +.*
- `giv_all_inlined Integer & operator+= (const Integer &n)`  
*operator += .*
- `giv_all_inlined Integer & operator+= (const uint64_t n)`  
*operator +.*
- `giv_all_inlined Integer & operator+= (const int64_t n)`  
*operator +.*
- `giv_all_inlined Integer & operator+= (const uint32_t n)`  
*operator +.*
- `giv_all_inlined Integer & operator+= (const int32_t n)`  
*operator +.*
- `template<class XXX >`  
`Integer & operator+= (const XXX &n)`  
*operator +.*
- `giv_all_inlined Integer operator- (const Integer &n) const`  
*operator -.*
- `giv_all_inlined Integer operator- (const uint64_t n) const`  
*operator +.*
- `giv_all_inlined Integer operator- (const int64_t n) const`  
*operator +.*
- `giv_all_inlined Integer operator- (const uint32_t n) const`  
*operator +.*
- `giv_all_inlined Integer operator- (const int32_t n) const`  
*operator +.*

- operator +.*
- `giv_all_inlined Integer & operator-= (const Integer &n)`
- operator -=.*
- `giv_all_inlined Integer & operator-= (const uint64_t n)`
- operator +.*
- `giv_all_inlined Integer & operator-= (const int64_t n)`
- operator +.*
- `giv_all_inlined Integer & operator-= (const uint32_t n)`
- operator +.*
- `giv_all_inlined Integer & operator-= (const int32_t n)`
- operator +.*
- `template<class XXX >`  
`Integer & operator-= (const XXX &n)`
- operator +.*
- `giv_all_inlined Integer operator- () const`
- Opposite.*
- `giv_all_inlined Integer operator* (const Integer &n) const`
- operator \*.*
- `giv_all_inlined Integer operator* (const uint64_t n) const`
- operator +.*
- `giv_all_inlined Integer operator* (const int64_t n) const`
- operator +.*
- `giv_all_inlined Integer operator* (const uint32_t n) const`
- operator +.*
- `giv_all_inlined Integer operator* (const int32_t n) const`
- operator +.*
- `giv_all_inlined Integer & operator*= (const Integer &n)`
- operator \* =.*
- `giv_all_inlined Integer & operator*= (const uint64_t n)`
- operator +.*
- `giv_all_inlined Integer & operator*= (const int64_t n)`
- operator +.*
- `giv_all_inlined Integer & operator*= (const uint32_t n)`
- operator +.*
- `giv_all_inlined Integer & operator*= (const int32_t n)`
- operator +.*
- `template<class XXX >`  
`Integer & operator*= (const XXX &n)`
- operator +.*
- `giv_all_inlined Integer operator+ (const int32_t l, const Integer &n)`
- operator +.*
- `giv_all_inlined Integer operator+ (const uint32_t l, const Integer &n)`
- operator +.*
- `giv_all_inlined Integer operator+ (const int64_t l, const Integer &n)`
- operator +.*
- `giv_all_inlined Integer operator+ (const uint64_t l, const Integer &n)`
- operator +.*
- `giv_all_inlined Integer operator- (const int32_t l, const Integer &n)`
- operator -.*
- `giv_all_inlined Integer operator- (const uint32_t l, const Integer &n)`
- operator +.*

- `giv_all_inlined Integer operator-` (const int64\_t l, const Integer &n)  
*operator +.*
- `giv_all_inlined Integer operator-` (const uint64\_t l, const Integer &n)  
*operator +.*
- `giv_all_inlined Integer operator*` (const int32\_t l, const Integer &n)  
*operator \**
- `giv_all_inlined Integer operator*` (const uint32\_t l, const Integer &n)  
*operator +.*
- `giv_all_inlined Integer operator*` (const int64\_t l, const Integer &n)  
*operator +.*
- `giv_all_inlined Integer operator*` (const uint64\_t l, const Integer &n)  
*operator +.*

## Division/euclidean division/modulo

The convention for rounding are the following :

- $q = a/b$ , or equivalent operations with the name `div` or `divin`, return  $q$  rounded towards 0, in the same manner as C's '/' (truncated division).
- $r = a \% b$  behaves like C %. The modulo function % rounds towards 0 and the sign of the dividend is preserved. This is :

$$a = bq + r, \text{ with } |r| < |b| \text{ and } ar \geq 0$$

- $r = a \bmod b$  or similar functions have the same behaviour as GMP `mpz_mod`, that is the remainder is always positive ( $\geq 0$ ). This is the division algorithm convention that is used (see `divmod`). In a formula :

$$a = bq + r, \text{ with } 0 \leq r < |b|$$

### Warning

if  $q=a/b$  and  $r= a \% b$  then  $a = b q + r$  is always true (with in addition  $0 \leq |r| < |b|$ ). This is also true for `divmod(q, a, b, r)` (and  $0 \leq r < |b|$ ). However, one should not mix the two conventions and expect equalities (except if  $a \geq 0$ ) .

- `giv_all_inlined Integer operator/` (const Integer &d) const  
*Division operator.*
- `giv_all_inlined Integer operator/` (const uint64\_t d) const  
*Division operator.*
- `giv_all_inlined Integer operator/` (const int64\_t d) const  
*Division operator.*
- `giv_all_inlined Integer operator/` (const uint32\_t d) const  
*Division operator.*
- `giv_all_inlined Integer operator/` (const int32\_t d) const  
*Division operator.*
- `giv_all_inlined Integer & operator/=` (const Integer &d)  
*Division operator (inplace).*
- `giv_all_inlined Integer & operator/=` (const uint64\_t d)  
*Division operator.*
- `giv_all_inlined Integer & operator/=` (const int64\_t d)  
*Division operator.*

- `giv_all_inlined Integer & operator/= (const uint32_t d)`  
*Division operator.*
- `giv_all_inlined Integer & operator/= (const int32_t d)`  
*Division operator.*
- `template<class XXX >`  
`Integer & operator/= (const XXX &d)`  
*Division operator.*
- `giv_all_inlined Integer operator% (const Integer &n) const`  
*Modulo operator.*
- `giv_all_inlined int64_t operator% (const uint64_t n) const`  
*Division operator.*
- `giv_all_inlined int64_t operator% (const int64_t n) const`  
*Division operator.*
- `giv_all_inlined int32_t operator% (const uint32_t n) const`  
*Division operator.*
- `giv_all_inlined int32_t operator% (const int32_t n) const`  
*Division operator.*
- `giv_all_inlined double operator% (const double n) const`  
*Division operator.*
- `int16_t operator% (const uint16_t n) const`  
*Division operator.*
- `template<class XXX >`  
`XXX operator% (const XXX &n) const`  
*Division operator.*
- `giv_all_inlined Integer & operator%=(const Integer &n)`  
*Modulo operator (inplace).*
- `giv_all_inlined Integer & operator%=(const uint64_t n)`  
*Division operator.*
- `giv_all_inlined Integer & operator%=(const int64_t n)`  
*Division operator.*
- `giv_all_inlined Integer & operator%=(const uint32_t n)`  
*Division operator.*
- `giv_all_inlined Integer & operator%=(const int32_t n)`  
*Division operator.*
- `template<class XXX >`  
`Integer & operator%=(const XXX &n)`  
*Division operator.*
- `giv_all_inlined Integer operator/ (const int32_t l, const Integer &n)`  
*operator /*
- `giv_all_inlined Integer operator/ (const int64_t l, const Integer &n)`  
*Division operator.*
- `giv_all_inlined Integer operator/ (const uint32_t l, const Integer &n)`  
*operator /*
- `giv_all_inlined Integer operator/ (const uint64_t l, const Integer &n)`  
*Division operator.*
- `giv_all_inlined Integer operator% (const int64_t l, const Integer &n)`  
*operator %*
- `giv_all_inlined Integer operator% (const uint64_t l, const Integer &n)`  
*Division operator.*
- `giv_all_inlined Integer operator% (const int32_t l, const Integer &n)`  
*Division operator.*

- `giv_all_inlined Integer operator%` (`const uint32_t l`, `const Integer &n`)  
*Division operator.*
- `static giv_all_inlined Integer &divin` (`Integer &q`, `const Integer &d`)  
*Division  $q/=d$ .*
- `static giv_all_inlined Integer &divin` (`Integer &q`, `const int64_t d`)  
*Division operator.*
- `static giv_all_inlined Integer &divin` (`Integer &q`, `const uint64_t d`)  
*Division operator.*
- `static giv_all_inlined Integer &div` (`Integer &q`, `const Integer &n`, `const Integer &d`)  
*Division  $q=n/d$ .*
- `static giv_all_inlined Integer &div` (`Integer &q`, `const Integer &n`, `const int64_t d`)  
*Division operator.*
- `static giv_all_inlined Integer &div` (`Integer &q`, `const Integer &n`, `const int32_t d`)  
*Division operator.*
- `static giv_all_inlined Integer &div` (`Integer &q`, `const Integer &n`, `const uint64_t d`)  
*Division operator.*
- `static giv_all_inlined Integer &divexact` (`Integer &q`, `const Integer &n`, `const Integer &d`)  
*Division when  $d$  divides  $n$ .*
- `static giv_all_inlined Integer &divexact` (`Integer &q`, `const Integer &n`, `const uint64_t &d`)  
*Division operator.*
- `static giv_all_inlined Integer &divexact` (`Integer &q`, `const Integer &n`, `const int64_t &d`)  
*Division operator.*
- `static giv_all_inlined Integer divexact` (`const Integer &n`, `const Integer &d`)  
*Division when  $d$  divides  $n$ .*
- `static giv_all_inlined Integer divexact` (`const Integer &n`, `const uint64_t &d`)  
*Division operator.*
- `static giv_all_inlined Integer divexact` (`const Integer &n`, `const int64_t &d`)  
*Division operator.*
- `static giv_all_inlined Integer &trem` (`Integer &r`, `const Integer &n`, `const Integer &d`)  
*Stuff.*
- `static giv_all_inlined Integer &crem` (`Integer &r`, `const Integer &n`, `const Integer &d`)  
*Division operator.*
- `static giv_all_inlined Integer &frem` (`Integer &r`, `const Integer &n`, `const Integer &d`)  
*Division operator.*
- `static giv_all_inlined Integer &trem` (`Integer &r`, `const Integer &n`, `const uint64_t &d`)  
*Stuff.*
- `static giv_all_inlined Integer &crem` (`Integer &r`, `const Integer &n`, `const uint64_t &d`)  
*Division operator.*
- `static giv_all_inlined Integer &frem` (`Integer &r`, `const Integer &n`, `const uint64_t &d`)  
*Division operator.*
- `static giv_all_inlined uint64_t trem` (`const Integer &n`, `const uint64_t &d`)  
*Stuff.*
- `static giv_all_inlined uint64_t crem` (`const Integer &n`, `const uint64_t &d`)  
*Division operator.*
- `static giv_all_inlined uint64_t frem` (`const Integer &n`, `const uint64_t &d`)  
*Division operator.*
- `static giv_all_inlined Integer &modin` (`Integer &r`, `const Integer &n`)  
*Function  $\text{mod}$  (inplace).*
- `static giv_all_inlined Integer &modin` (`Integer &r`, `const int64_t n`)  
*Division operator.*
- `static giv_all_inlined Integer &modin` (`Integer &r`, `const uint64_t n`)

*Division operator.*

- static giv\_all\_inlined `Integer & mod (Integer &r, const Integer &n, const Integer &d)`

*Function mod.*

- static giv\_all\_inlined `Integer & mod (Integer &r, const Integer &n, const int64_t d)`

*Division operator.*

- static giv\_all\_inlined `Integer & mod (Integer &r, const Integer &n, const uint64_t d)`

*Division operator.*

- static giv\_all\_inlined `Integer & mod (Integer &r, const Integer &n, const int32_t d)`

*Division operator.*

- static giv\_all\_inlined `Integer & mod (Integer &r, const Integer &n, const uint32_t d)`

*Division operator.*

- static giv\_all\_inlined `Integer & divmod (Integer &q, Integer &r, const Integer &n, const Integer &d)`

*Euclidean division.*

- static giv\_all\_inlined `Integer & divmod (Integer &q, int64_t &r, const Integer &n, const int64_t d)`

*Division operator.*

- static giv\_all\_inlined `Integer & divmod (Integer &q, uint64_t &r, const Integer &n, const uint64_t d)`

*Division operator.*

- static giv\_all\_inlined `Integer & ceil (Integer &res, const Integer &n, const Integer &d)`

*rounding functions.*

- static giv\_all\_inlined `Integer & floor (Integer &res, const Integer &n, const Integer &d)`
- static giv\_all\_inlined `Integer & trunc (Integer &res, const Integer &n, const Integer &d)`
- static giv\_all\_inlined `Integer ceil (const Integer &n, const Integer &d)`

*rounding functions.*

- static giv\_all\_inlined `Integer floor (const Integer &n, const Integer &d)`
- static giv\_all\_inlined `Integer trunc (const Integer &n, const Integer &d)`

## Miscellaneous.

- int32\_t **sign** () const

*sign*

- int32\_t **priv\_sign** () const

*private sign*

- giv\_all\_inlined void **swap** (Integer &a, Integer &b)
- int32\_t **sign** (const Integer &a)

*sign*

## representation

get representation

- mpz\_ptr **get\_mpz** ()

*get representation (constant)*

- mpz\_srcptr **get\_mpz** () const

*get representation (constant)*

- mpz\_srcptr **get\_mpz\_const** () const

*get representation (constant)*

- giv\_all\_inlined size\_t **size** () const

*returns the number of machine words used to store \*this*

- `giv_all_inlined size_t size_in_base (int32_t B) const`  
*returns  $\text{ceil}(\log_{\text{BASE}}(*this))$ .*
- `giv_all_inlined size_t bitsize () const`  
*returns  $\text{ceil}(\log_2(*this))$ .*
- `giv_all_inlined uint64_t operator[] (size_t i) const`  
*return the i-th word of the integer.*
- `giv_all_inlined uint64_t length (const Integer &a)`  
*returns the number of bytes used to store \*this*
- `giv_all_inlined int32_t isperfectpower (const Integer &n)`  
*perfect power*
- `giv_all_inlined Integer abs (const Integer &n)`  
*absolute value*
- `giv_all_inlined bool isOdd (const Integer &)`  
*parity of an integer*

## I/O

- `giv_all_inlined std::ostream & print (std::ostream &o) const`  
*print32\_t integer.*
- `giv_all_inlined std::istream & operator>> (std::istream &i, Integer &n)`  
*Input/Output of Integers.*
- `giv_all_inlined std::ostream & operator<< (std::ostream &o, const Integer &n)`  
*out operator.*
- `giv_all_inlined std::ostream & absOutput (std::ostream &o, const Integer &n)`  
*nodoc*
- `giv_all_inlined void Protected::importWords (Integer &x, size_t count, int32_t order, int32_t size, int32_t endian, size_t nails, const void *op)`  
*nodoc*

## Comparisons functions.

- `giv_all_inlined friend int32_t compare (const Integer &a, const Integer &b)`  
*Compares two integers.*
- `giv_all_inlined friend int32_t absCompare (const Integer &a, const Integer &b)`  
*Compare the norm of two integers.*
- `giv_all_inlined friend int32_t absCompare (const Integer &a, const double b)`  
*Compares two integers.*
- `giv_all_inlined friend int32_t absCompare (const Integer &a, const float b)`  
*Compares two integers.*
- `giv_all_inlined friend int32_t absCompare (const Integer &a, const uint64_t b)`  
*Compares two integers.*
- `giv_all_inlined friend int32_t absCompare (const Integer &a, const unsigned b)`  
*Compares two integers.*
- `giv_all_inlined friend int32_t absCompare (const Integer &a, const int64_t b)`  
*Compares two integers.*
- `giv_all_inlined friend int32_t absCompare (const Integer &a, const int32_t b)`  
*Compares two integers.*
- `template<class T >`  
`giv_all_inlined friend int32_t absCompare (const T a, const Integer &b)`

- Compares two integers.*
- `giv_all_inlined int32_t isOne (const Integer &a)`  
*name compare to 1 and 0*
- `giv_all_inlined int32_t isMOne (const Integer &a)`  
*Compares two integers.*
- `giv_all_inlined int32_t nonZero (const Integer &a)`  
*name compare to 1 and 0*
- `giv_all_inlined int32_t isZero (const Integer &a)`  
*name compare to 1 and 0*
- `giv_all_inlined int32_t isZero (const int16_t a)`  
*Compares two integers.*
- `giv_all_inlined int32_t isZero (const int32_t a)`  
*Compares two integers.*
- `giv_all_inlined int32_t isZero (const int64_t a)`  
*Compares two integers.*
- `giv_all_inlined int32_t isZero (const uint16_t a)`  
*Compares two integers.*
- `giv_all_inlined int32_t isZero (const uint32_t a)`  
*Compares two integers.*
- `giv_all_inlined int32_t isZero (const uint64_t a)`  
*Compares two integers.*
- `template<class A , class B >`  
`static giv_all_inlined bool isleq (const A &a, const B &b)`  
*isleq*

### 17.47.1 Detailed Description

This is the [Integer](#) class.

An [Integer](#) is represented as a GMP integer. This class provides arithmetic on Integers.

#### Examples

[examples/FiniteField/GF128.C](#), [examples/FiniteField/Test\\_Extension.C](#), [examples/FiniteField/all\\_field.C](#), [examples/FiniteField/domain\\_to\\_operatorstyle.C](#), [examples/Integer/ModularSquareRoot.C](#), [examples/Integer/ProbLucas.C](#), [examples/Integer/RSA\\_breaking.C](#), [examples/Integer/RSA\\_decipher.C](#), [examples/Integer/RSA\\_encipher.C](#), [examples/Integer/iexponentiation.C](#), [examples/Integer/ifactor.C](#), [examples/Integer/ifactor\\_lenstra.C](#), [examples/Integer/igcd.C](#), [examples/Integer/igcdext.C](#), [examples/Integer/ilcm.C](#), [examples/Integer/ispower.C](#), [examples/Integer/isprime.C](#), [examples/Integer/isroot.C](#), [examples/Integer/lambda.C](#), [examples/Integer/lambda\\_inv.C](#), [examples/Integer/nb\\_primes.C](#), [examples/Integer/nextprime.C](#), [examples/Integer/order.C](#), [examples/Integer/phi.C](#), [examples/Integer/prevprime.C](#), [examples/Integer/primitiveelement.C](#), [examples/Integer/primitiveroot.C](#), [examples/Integer/probable\\_primroot.C](#), [examples/Polynomial/PolynomialCRT.C](#), [examples/Rational/iratecon.C](#), and [examples/Rational/polydouble.C](#).

### 17.47.2 Constructor & Destructor Documentation

#### 17.47.2.1 Integer() [1/13]

```
Integer (
    int32_t n = 0 )
```

Constructor form a known type.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.2 Integer()** [2/13]

```
Integer (
    int64_t n )
```

Constructor form a known type.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.3 Integer()** [3/13]

```
Integer (
    unsigned char n )
```

Constructor form a known type.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.4 Integer()** [4/13]

```
Integer (
    uint32_t n )
```

Constructor form a known type.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.5 Integer()** [5/13]

```
Integer (
    uint64_t n )
```

Constructor form a known type.

**Parameters**

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.6 Integer()** [6/13]

```
Integer (
    double n )
```

Constructor form a known type.

**Parameters**

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.7 Integer()** [7/13]

```
Integer (
    const char * n )
```

Constructor form a known type.

**Parameters**

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.8 Integer()** [8/13]

```
giv_all_inlined Integer (
    const mpz_class & a ) [inline]
```

Constructor form a known type.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.9 Integer()** [9/13]

```
Integer (
    const RecInt::ruint< K > & n ) [inline]
```

Constructor form a known type.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.10 Integer()** [10/13]

```
Integer (
    const RecInt::rint< K > & n ) [inline]
```

Constructor form a known type.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.11 Integer()** [11/13]

```
Integer (
    const Integer & n )
```

Copy constructor.

## Parameters

<i>n</i>	input to be constructed from
----------	------------------------------

**17.47.2.12 Integer()** [12/13]

```
giv_all_inlined Integer (
    uint64_t * d,
    int64_t sz )
```

Creates a new [Integer](#) from pointers.

**Parameters**

<i>d</i>	array
<i>sz</i>	size

**17.47.2.13 Integer()** [13/13]

```
Integer (
    const vect_t & v )
```

Creates a new Integers for a vector of limbs.

**Parameters**

<i>v</i>	vector of limbs
----------	-----------------

**17.47.3 Member Function Documentation****17.47.3.1 operator=()**

```
Integer & operator= (
    const Integer & n )
```

copy from an integer.

**Parameters**

<i>n</i>	integer to copy.
----------	------------------

**17.47.3.2 logcpy()**

```
Integer & logcpy (
    const Integer & n )
```

copy from an integer.

#### Parameters

<i>n</i>	integer to copy.
----------	------------------

### 17.47.3.3 copy()

```
Integer & copy (
    const Integer & n )
```

copy from an integer.

#### Parameters

<i>n</i>	integer to copy.
----------	------------------

#### Examples

[examples/Integer/ProbLucas.C.](#)

### 17.47.3.4 isleq()

```
static giv_all_inlined bool isleq (
    const A & a,
    const B & b ) [inline], [static]
```

isleq

#### Parameters

<i>a,b</i>	
------------	--

### 17.47.3.5 operator>=() [1/7]

```
int32_t operator>= (
    const Integer & l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.6 operator>=()** [2/7]

```
int32_t operator>= (
    const int32_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.7 operator>=()** [3/7]

```
int32_t operator>= (
    const int64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.8 operator>=()** [4/7]

```
int32_t operator>= (
    const uint64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.9 operator>=()** [5/7]

```
int32_t operator>= (
    const uint32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.10 operator>=()** [6/7]

```
int32_t operator>= (
    const double l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.11 operator>=()** [7/7]

```
int32_t operator>= (
    const float l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.12 operator<=()** [1/7]

```
int32_t operator<= (
    const Integer & l ) const
```

less or equal

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.13 operator<=()** [2/7]

```
int32_t operator<= (
    const int32_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.14 operator<=()** [3/7]

```
int32_t operator<= (
    const int64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.15 operator<=()** [4/7]

```
int32_t operator<= (
    const uint64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.16 operator<=()** [5/7]

```
int32_t operator<= (
    const uint32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.17 operator<=()** [6/7]

```
int32_t operator<= (
    const double l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.18 operator<=()** [7/7]

```
int32_t operator<= (
    const float l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.19 operator"!=()** [1/7]

```
int32_t operator!= (
    const Integer & l ) const
```

operator != (not equal)

**Parameters**

/	integer
---	---------

**Returns**

1 iff `l == this`

**17.47.3.20 operator"!=() [2/7]**

```
int32_t operator!= (
    const int32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.21 operator"!=() [3/7]**

```
int32_t operator!= (
    const int64_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.22 operator"!=() [4/7]**

```
int32_t operator!= (
    const uint64_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.23 operator"!="() [5/7]**

```
int32_t operator!= (
    const uint32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.24 operator"!="() [6/7]**

```
int32_t operator!= (
    const double l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.25 operator"!="() [7/7]**

```
int32_t operator!= (
    const float l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.26 operator==( ) [1/7]**

```
int32_t operator==(
    const Integer & l ) const
```

Equality.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.27 operator==()** [2/7]

```
int32_t operator== (
    const int32_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.28 operator==()** [3/7]

```
int32_t operator== (
    const int64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.29 operator==()** [4/7]

```
int32_t operator== (
    const uint64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.30 operator==( ) [5/7]**

```
int32_t operator== (
    const uint32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.31 operator==( ) [6/7]**

```
int32_t operator== (
    const double l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.32 operator==( ) [7/7]**

```
int32_t operator== (
    const float l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.33 operator>( ) [1/7]**

```
int32_t operator> (
    const Integer & l ) const
```

greater (strict)

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.34 operator>() [2/7]**

```
int32_t operator> (  
    const int32_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.35 operator>() [3/7]**

```
int32_t operator> (  
    const int64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.36 operator>() [4/7]**

```
int32_t operator> (  
    const uint64_t l ) const
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.3.37 operator>() [5/7]**

```
int32_t operator> (
    const uint32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.38 operator>() [6/7]**

```
int32_t operator> (
    const double l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.39 operator>() [7/7]**

```
int32_t operator> (
    const float l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.40 operator<() [1/7]**

```
int32_t operator< (
    const Integer & l ) const
```

less (strict)

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.41 operator<>() [2/7]**

```
int32_t operator< (  
    const int32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.42 operator<>() [3/7]**

```
int32_t operator< (  
    const int64_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.43 operator<>() [4/7]**

```
int32_t operator< (  
    const uint64_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.44 operator<>() [5/7]**

```
int32_t operator< (
    const uint32_t l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.45 operator<>() [6/7]**

```
int32_t operator< (
    const double l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.46 operator<>() [7/7]**

```
int32_t operator< (
    const float l ) const
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.3.47 operator^() [1/3]**

```
Integer operator^ (
    const Integer & a ) const
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.48 operator^() [2/3]**

```
Integer operator^ (
    const uint64_t & a ) const
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.49 operator^() [3/3]**

```
Integer operator^ (
    const uint32_t & a ) const
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.50 operator^=() [1/3]**

```
Integer & operator^= (
    const Integer & a )
```

XOR inplace (^=)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.51 operator^=()** [2/3]

```
Integer & operator^= (
    const uint64_t & a )
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.52 operator^=()** [3/3]

```
Integer & operator^= (
    const uint32_t & a )
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.53 operator" |()** [1/3]

```
Integer operator| (
    const Integer & a ) const
```

OR (|)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.54 operator" |()** [2/3]

```
Integer operator| (
    const uint64_t & a ) const
```

XOR (^)

**Parameters**

<i>a</i>	integer
----------	---------

**17.47.3.55 operator" |() [3/3]**

```
Integer operator| (
    const uint32_t & a ) const
```

XOR (^)

**Parameters**

<i>a</i>	integer
----------	---------

**17.47.3.56 operator" |=() [1/3]**

```
Integer & operator|= (
    const Integer & a )
```

OR inplace (|=)

**Parameters**

<i>a</i>	integer
----------	---------

**17.47.3.57 operator" |=() [2/3]**

```
Integer & operator|= (
    const uint64_t & a )
```

XOR (^)

**Parameters**

<i>a</i>	integer
----------	---------

**17.47.3.58 operator" |="() [3/3]**

```
Integer & operator|=(  
    const uint32_t & a )
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.59 operator&() [1/3]**

```
Integer operator& (  
    const Integer & a ) const
```

AND (&amp;)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.60 operator&() [2/3]**

```
uint32_t operator& (  
    const uint32_t & a ) const
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.61 operator&() [3/3]**

```
uint64_t operator& (  
    const uint64_t & a ) const
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.62 operator&=()** [1/3]

```
Integer & operator&= (  
    const Integer & a )
```

AND inplace (&amp;=)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.63 operator&=()** [2/3]

```
Integer & operator&= (  
    const uint64_t & a )
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.64 operator&=()** [3/3]

```
Integer & operator&= (  
    const uint32_t & a )
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.65 operator<<() [1/4]**

```
Integer operator<< (
    int32_t l ) const
```

left shift (<<)

**Parameters**

/	shift
---	-------

**17.47.3.66 operator<<() [2/4]**

```
Integer operator<< (
    int64_t l ) const
```

XOR (^)

**Parameters**

a	integer
---	---------

**17.47.3.67 operator<<() [3/4]**

```
Integer operator<< (
    uint32_t l ) const
```

XOR (^)

**Parameters**

a	integer
---	---------

**17.47.3.68 operator<<() [4/4]**

```
Integer operator<< (
    uint64_t l ) const
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.69 operator<<=()** [1/4]

```
Integer & operator<<= (  
    int32_t l )
```

left shift inplace (<<=)

## Parameters

<i>l</i>	shift
----------	-------

**17.47.3.70 operator<<=()** [2/4]

```
Integer & operator<<= (  
    int64_t l )
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.71 operator<<=()** [3/4]

```
Integer & operator<<= (  
    uint32_t l )
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.72 operator<<=()** [4/4]

```
Integer & operator<<= (
    uint64_t l )
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.73 operator>>()** [1/4]

```
Integer operator>> (
    int32_t l ) const
```

right shift (&gt;&gt;)

Parameters

<i>l</i>	shift
----------	-------

**17.47.3.74 operator>>()** [2/4]

```
Integer operator>> (
    int64_t l ) const
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.75 operator>>()** [3/4]

```
Integer operator>> (
    uint32_t l ) const
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.76 operator>>() [4/4]**

```
Integer operator>> (
    uint64_t l ) const
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.77 operator>>=() [1/4]**

```
Integer & operator>>= (
    int32_t l )
```

right shift inplace (&gt;&gt;=)

## Parameters

<i>l</i>	shift
----------	-------

**17.47.3.78 operator>>=() [2/4]**

```
Integer & operator>>= (
    int64_t l )
```

XOR (^)

## Parameters

<i>a</i>	integer
----------	---------

**17.47.3.79 operator>>=()** [3/4]

```
Integer & operator>>= (
    uint32_t l )
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.80 operator>>=()** [4/4]

```
Integer & operator>>= (
    uint64_t l )
```

XOR (^)

Parameters

<i>a</i>	integer
----------	---------

**17.47.3.81 addin()** [1/5]

```
Integer & addin (
    Integer & res,
    const Integer & n ) [static]
```

Addition (inplace)  $res += n$ .

Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.82 addin()** [2/5]

```
Integer & addin (
    Integer & res,
    const int64_t n ) [static]
```

Addition (inplace)  $res += n$ .

## Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.83 addin()** [3/5]

```
Integer & addin (
    Integer & res,
    const uint64_t n ) [static]
```

Addition (inplace)  $res += n$ .

## Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.84 addin()** [4/5]

```
static giv_all_inlined Integer & addin (
    Integer & res,
    const int32_t n ) [inline], [static]
```

Addition (inplace)  $res += n$ .

## Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.85 addin()** [5/5]

```
static giv_all_inlined Integer & addin (
    Integer & res,
    const uint32_t n ) [inline], [static]
```

Addition (inplace)  $res += n$ .

## Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.86 add()** [1/5]

```
Integer & add (  
    Integer & res,  
    const Integer & n1,  
    const Integer & n2 ) [static]
```

Addition `res=n1+n2`.

**Parameters**

<i>res</i>	as in the formula
<i>n1</i>	as in the formula
<i>n2</i>	as in the formula

**17.47.3.87 add()** [2/5]

```
Integer & add (  
    Integer & res,  
    const Integer & n1,  
    const int64_t n2 ) [static]
```

Addition (inplace) `res+=n`.

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.88 add()** [3/5]

```
Integer & add (  
    Integer & res,  
    const Integer & n1,  
    const uint64_t n2 ) [static]
```

Addition (inplace) `res+=n`.

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.89 add()** [4/5]

```
static giv_all_inlined Integer & add (
    Integer & res,
    const Integer & n1,
    const int32_t n2 ) [inline], [static]
```

Addition (inplace)  $res+=n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.90 add()** [5/5]

```
static giv_all_inlined Integer & add (
    Integer & res,
    const Integer & n1,
    const uint32_t n2 ) [inline], [static]
```

Addition (inplace)  $res+=n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.91 subin()** [1/5]

```
Integer & subin (
    Integer & res,
    const Integer & n ) [static]
```

Substraction (inplace)  $res-=n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.92 subin()** [2/5]

```
Integer & subin (  
    Integer & res,  
    const int64_t n ) [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.93 subin()** [3/5]

```
Integer & subin (  
    Integer & res,  
    const uint64_t n ) [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.94 subin()** [4/5]

```
static giv_all_inlined Integer & subin (  
    Integer & res,  
    const int32_t n ) [inline], [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.95 subin()** [5/5]

```
static giv_all_inlined Integer & subin (  

```

```
Integer & res,
const uint32_t n ) [inline], [static]
```

Addition (inplace)  $res += n$ .

#### Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

#### 17.47.3.96 sub() [1/5]

```
Integer & sub (
    Integer & res,
    const Integer & n1,
    const Integer & n2 ) [static]
```

Substraction  $res = n1 - n2$ .

#### Parameters

<i>res</i>	as in the formula
<i>n1</i>	as in the formula
<i>n2</i>	as in the formula

#### 17.47.3.97 sub() [2/5]

```
Integer & sub (
    Integer & res,
    const Integer & n1,
    const int64_t n2 ) [static]
```

Addition (inplace)  $res += n$ .

#### Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

#### 17.47.3.98 sub() [3/5]

```
Integer & sub (
    Integer & res,
```

```
const Integer & n1,
const uint64_t n2 ) [static]
```

Addition (inplace)  $res += n$ .

#### Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

#### 17.47.3.99 sub() [4/5]

```
static giv_all_inlined Integer & sub (
    Integer & res,
    const Integer & n1,
    const int32_t n2 ) [inline], [static]
```

Addition (inplace)  $res += n$ .

#### Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

#### 17.47.3.100 sub() [5/5]

```
static giv_all_inlined Integer & sub (
    Integer & res,
    const Integer & n1,
    const uint32_t n2 ) [inline], [static]
```

Addition (inplace)  $res += n$ .

#### Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

#### 17.47.3.101 negin()

```
Integer & negin (
    Integer & res ) [static]
```

Negation (inplace)  $res = -res$ .

## Parameters

<i>res</i>	as in the formula
------------	-------------------

**17.47.3.102 neg()**

```
Integer & neg (
    Integer & res,
    const Integer & n ) [static]
```

Negation `res=-n`.

## Parameters

<i>n</i>	as in the formula
<i>res</i>	as in the formula

**17.47.3.103 mulin()** [1/5]

```
Integer & mulin (
    Integer & res,
    const Integer & n ) [static]
```

Multiplication (inplace) `res*=n`.

## Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.104 mulin()** [2/5]

```
Integer & mulin (
    Integer & res,
    const int64_t n ) [static]
```

Addition (inplace) `res+=n`.

## Parameters

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.105 mulin()** [3/5]

```
Integer & mulin (  
    Integer & res,  
    const uint64_t n ) [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.106 mulin()** [4/5]

```
static giv_all_inlined Integer & mulin (  
    Integer & res,  
    const int32_t n ) [inline], [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.107 mulin()** [5/5]

```
static giv_all_inlined Integer & mulin (  
    Integer & res,  
    const uint32_t n ) [inline], [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.108 mul()** [1/5]

```
Integer & mul (
    Integer & res,
    const Integer & n1,
    const Integer & n2 ) [static]
```

Multiplication `res=n1*n2`.

**Parameters**

<i>res</i>	as in the formula
<i>n1</i>	as in the formula
<i>n2</i>	as in the formula

**17.47.3.109 mul()** [2/5]

```
Integer & mul (
    Integer & res,
    const Integer & n1,
    const int64_t n2 ) [static]
```

Addition (inplace) `res+=n`.

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.110 mul()** [3/5]

```
Integer & mul (
    Integer & res,
    const Integer & n1,
    const uint64_t n2 ) [static]
```

Addition (inplace) `res+=n`.

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.111 mul()** [4/5]

```
static giv_all_inlined Integer & mul (  
    Integer & res,  
    const Integer & n1,  
    const int32_t n2 ) [inline], [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.112 mul()** [5/5]

```
static giv_all_inlined Integer & mul (  
    Integer & res,  
    const Integer & n1,  
    const uint32_t n2 ) [inline], [static]
```

Addition (inplace)  $res += n$ .

**Parameters**

<i>res</i>	as in the formula
<i>n</i>	as in the formula

**17.47.3.113 operator+()** [1/5]

```
Integer operator+ (  
    const Integer & n ) const
```

operator +.

**Returns**

$(*this) + n$

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.114 operator+()** [2/5]

```
Integer operator+ (  
    const uint64_t n ) const
```

operator +.

**Returns**

(*\*this*)+*n*

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.115 operator+()** [3/5]

```
Integer operator+ (  
    const int64_t n ) const
```

operator +.

**Returns**

(*\*this*)+*n*

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.116 operator+()** [4/5]

```
giv_all_inlined Integer operator+ (  
    const uint32_t n ) const [inline]
```

operator +.

**Returns**

(*\*this*)+*n*

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.117 operator+()** [5/5]

```
giv_all_inlined Integer operator+ (
    const int32_t n ) const [inline]
```

operator +.

**Returns**

(*\*this*) + *n*

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.118 operator+=()** [1/6]

```
Integer & operator+= (
    const Integer & n )
```

operator += .

**Parameters**

<i>n</i>	as friend in the formula.
----------	---------------------------

**Returns**

(*\*this*) += *n*.

**17.47.3.119 operator+=()** [2/6]

```
Integer & operator+= (
    const uint64_t n )
```

operator +.

**Returns**

(*\*this*) + *n*

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.3.120 operator+=() [3/6]**

```
Integer & operator+= (
    const int64_t n )
```

operator +.

**Returns**

(\*this)+n

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.3.121 operator+=() [4/6]**

```
giv_all_inlined Integer & operator+= (
    const uint32_t n ) [inline]
```

operator +.

**Returns**

(\*this)+n

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.3.122 operator+=() [5/6]**

```
giv_all_inlined Integer & operator+= (
    const int32_t n ) [inline]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.123 operator+=() [6/6]**

```
Integer & operator+= (  
    const XXX & n ) [inline]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.124 operator-() [1/6]**

```
Integer operator- (  
    const Integer & n ) const
```

operator −.

**Returns**

`(*this)-n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.125 operator-() [2/6]**

```
Integer operator- (  
    const uint64_t n ) const
```

operator +.

#### Returns

`(*this)+n`

#### Parameters

$n$	as in the formula.
-----	--------------------

### 17.47.3.126 operator-() [3/6]

```
Integer operator- (
    const int64_t n ) const
```

operator +.

#### Returns

`(*this)+n`

#### Parameters

$n$	as in the formula.
-----	--------------------

### 17.47.3.127 operator-() [4/6]

```
giv_all_inlined Integer operator- (
    const uint32_t n ) const [inline]
```

operator +.

#### Returns

`(*this)+n`

#### Parameters

$n$	as in the formula.
-----	--------------------

**17.47.3.128 operator-() [5/6]**

```
giv_all_inlined Integer operator- (
    const int32_t n ) const [inline]
```

operator +.

**Returns**

(\*this)+n

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.3.129 operator-=() [1/6]**

```
Integer & operator-= (
    const Integer & n )
```

operator -= .

**Parameters**

$n$	as in the formula.
-----	--------------------

**Returns**

(\*this) -= n.

**17.47.3.130 operator-=() [2/6]**

```
Integer & operator-= (
    const uint64_t n )
```

operator +.

**Returns**

(\*this)+n

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.3.131 operator-=()** [3/6]

```
Integer & operator-= (  
    const int64_t n )
```

operator +.

**Returns**

(\*this)+n

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.132 operator-=()** [4/6]

```
giv_all_inlined Integer & operator-= (  
    const uint32_t n ) [inline]
```

operator +.

**Returns**

(\*this)+n

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.133 operator-=()** [5/6]

```
giv_all_inlined Integer & operator-= (  
    const int32_t n ) [inline]
```

operator +.

**Returns**

(\*this)+n

## Parameters

$n$	as in the formula.
-----	--------------------

**17.47.3.134 operator-=( ) [6/6]**

```
Integer & operator-= (
    const XXX & n ) [inline]
```

operator +.

## Returns

(*\*this*) + *n*

## Parameters

$n$	as in the formula.
-----	--------------------

**17.47.3.135 operator-( ) [6/6]**

```
Integer operator- ( ) const
```

Opposite.

## Returns

-(*\*this*).

**17.47.3.136 operator\*( ) [1/5]**

```
Integer operator* (
    const Integer & n ) const
```

operator \*.

## Returns

(*\*this*) \* *n*

## Parameters

$n$	as in the formula.
-----	--------------------

**17.47.3.137 operator\*() [2/5]**

```
Integer operator* (
    const uint64_t n ) const
```

operator +.

## Returns

(\*this)+n

## Parameters

$n$	as in the formula.
-----	--------------------

**17.47.3.138 operator\*() [3/5]**

```
Integer operator* (
    const int64_t n ) const
```

operator +.

## Returns

(\*this)+n

## Parameters

$n$	as in the formula.
-----	--------------------

**17.47.3.139 operator\*() [4/5]**

```
giv_all_inlined Integer operator* (
    const uint32_t n ) const [inline]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.140 operator\*() [5/5]**

```
giv_all_inlined Integer operator* (
    const int32_t n ) const [inline]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.141 operator\*=( ) [1/6]**

```
Integer & operator*= (
    const Integer & n )
```

operator \* = .

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**Returns**

`(*this) *= n.`

**17.47.3.142 operator\*=( ) [2/6]**

```
Integer & operator*= (
    const uint64_t n )
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.143 operator\*=( ) [3/6]**

```
Integer & operator*= (
    const int64_t n )
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.144 operator\*=( ) [4/6]**

```
giv_all_inlined Integer & operator*= (
    const uint32_t n ) [inline]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.3.145 operator\*=( ) [5/6]**

```
giv_all_inlined Integer & operator*= (
    const int32_t n ) [inline]
```

operator +.

#### Returns

`(*this)+n`

#### Parameters

<i>n</i>	as in the formula.
----------	--------------------

### 17.47.3.146 operator\*=( ) [6/6]

```
Integer & operator*= (
    const XXX & n ) [inline]
```

operator +.

#### Returns

`(*this)+n`

#### Parameters

<i>n</i>	as in the formula.
----------	--------------------

### 17.47.3.147 axpy() [1/2]

```
Integer & axpy (
    Integer & res,
    const Integer & a,
    const Integer & x,
    const Integer & y ) [static]
```

`axpy res = ax+y.`

#### Parameters

<i>res</i>	Integers as in the formula
<i>a</i>	Integers as in the formula
<i>x</i>	Integers as in the formula
<i>y</i>	Integers as in the formula

**17.47.3.148 axpy()** [2/2]

```
Integer & axpy (
    Integer & res,
    const Integer & a,
    const uint64_t x,
    const Integer & y ) [static]
```

axpy res = ax+y.

**Parameters**

<i>res</i>	Integers as in the formula
<i>a</i>	Integers as in the formula
<i>x</i>	Integers as in the formula
<i>y</i>	Integers as in the formula

**17.47.3.149 axpyin()** [1/2]

```
Integer & axpyin (
    Integer & res,
    const Integer & a,
    const Integer & x ) [static]
```

axpyin (inplace) res += ax.

**Parameters**

<i>res</i>	Integers as in the formula.
<i>a</i>	Integers as in the formula.
<i>x</i>	Integers as in the formula.

**17.47.3.150 axpyin()** [2/2]

```
Integer & axpyin (
    Integer & res,
    const Integer & a,
    const uint64_t x ) [static]
```

axpy res = ax+y.

**Parameters**

<i>res</i>	Integers as in the formula
<i>a</i>	Integers as in the formula
<i>x</i>	Integers as in the formula
<i>y</i>	Integers as in the formula

**17.47.3.151 maxpy()** [1/2]

```
Integer & maxpy (
    Integer & res,
    const Integer & a,
    const Integer & x,
    const Integer & y ) [static]
```

maxpy res = y - ax.

**Parameters**

<i>res</i>	Integers as in the formula.
<i>a</i>	Integers as in the formula.
<i>x</i>	Integers as in the formula.
<i>y</i>	Integers as in the formula.

**17.47.3.152 maxpy()** [2/2]

```
Integer & maxpy (
    Integer & res,
    const Integer & a,
    const uint64_t x,
    const Integer & y ) [static]
```

axy res = ax+y.

**Parameters**

<i>res</i>	Integers as in the formula
<i>a</i>	Integers as in the formula
<i>x</i>	Integers as in the formula
<i>y</i>	Integers as in the formula

**17.47.3.153 maxpyin()** [1/2]

```
Integer & maxpyin (
    Integer & res,
    const Integer & a,
    const Integer & x ) [static]
```

maxpyin res -= ax.

## Parameters

<i>res</i>	Integers as in the formula.
<i>a</i>	Integers as in the formula.
<i>x</i>	Integers as in the formula.

**17.47.3.154 maxpyin()** [2/2]

```
Integer & maxpyin (
    Integer & res,
    const Integer & a,
    const uint64_t x ) [static]
```

axpy res = ax+y.

## Parameters

<i>res</i>	Integers as in the formula
<i>a</i>	Integers as in the formula
<i>x</i>	Integers as in the formula
<i>y</i>	Integers as in the formula

**17.47.3.155 axmy()** [1/2]

```
Integer & axmy (
    Integer & res,
    const Integer & a,
    const Integer & x,
    const Integer & y ) [static]
```

axmy res = ax - y.

## Parameters

<i>res</i>	Integers as in the formula.
<i>a</i>	Integers as in the formula.
<i>x</i>	Integers as in the formula.
<i>y</i>	Integers as in the formula.

**17.47.3.156 axmy()** [2/2]

```
Integer & axmy (
```

```
Integer & res,
const Integer & a,
const uint64_t x,
const Integer & y ) [static]
```

axpy res = ax+y.

#### Parameters

<i>res</i>	Integers as in the formula
<i>a</i>	Integers as in the formula
<i>x</i>	Integers as in the formula
<i>y</i>	Integers as in the formula

#### 17.47.3.157 axmyin() [1/2]

```
Integer & axmyin (
    Integer & res,
    const Integer & a,
    const Integer & x ) [static]
```

axmyin (in place) res = ax - res.

#### Parameters

<i>res</i>	Integers as in the formula.
<i>a</i>	Integers as in the formula.
<i>x</i>	Integers as in the formula.

#### 17.47.3.158 axmyin() [2/2]

```
Integer & axmyin (
    Integer & res,
    const Integer & a,
    const uint64_t x ) [static]
```

axpy res = ax+y.

#### Parameters

<i>res</i>	Integers as in the formula
<i>a</i>	Integers as in the formula
<i>x</i>	Integers as in the formula
<i>y</i>	Integers as in the formula

**17.47.3.159 divin()** [1/3]

```
Integer & divin (
    Integer & q,
    const Integer & d ) [static]
```

Division  $q/=d$ .

**Parameters**

$q$	quotient
$d$	divisor.

**Returns**

$q$

**17.47.3.160 divin()** [2/3]

```
Integer & divin (
    Integer & q,
    const int64_t d ) [static]
```

Division operator.

**Parameters**

$d$	divisor
-----	---------

**17.47.3.161 divin()** [3/3]

```
Integer & divin (
    Integer & q,
    const uint64_t d ) [static]
```

Division operator.

**Parameters**

$d$	divisor
-----	---------

**17.47.3.162 div()** [1/4]

```
Integer & div (  
    Integer & q,  
    const Integer & n,  
    const Integer & d ) [static]
```

Division  $q=n/d$ .

**Parameters**

$q$	quotient
$n$	dividend.
$d$	divisor

**Returns**

$q$

**17.47.3.163 div()** [2/4]

```
Integer & div (  
    Integer & q,  
    const Integer & n,  
    const int64_t d ) [static]
```

Division operator.

**Parameters**

$d$	divisor
-----	---------

**17.47.3.164 div()** [3/4]

```
Integer & div (  
    Integer & q,  
    const Integer & n,  
    const int32_t d ) [static]
```

Division operator.

**Parameters**

$d$	divisor
-----	---------

**17.47.3.165 div()** [4/4]

```
Integer & div (
    Integer & q,
    const Integer & n,
    const uint64_t d ) [static]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.166 divexact()** [1/6]

```
Integer & divexact (
    Integer & q,
    const Integer & n,
    const Integer & d ) [static]
```

Division when d divides n.

**Parameters**

<i>q</i>	exact quotient
<i>n</i>	dividend
<i>d</i>	divisor

**Warning**

if quotient is not exact, the result is not predictable.

**17.47.3.167 divexact()** [2/6]

```
Integer & divexact (
    Integer & q,
    const Integer & n,
    const uint64_t & d ) [static]
```

Division operator.

## Parameters

$d$	divisor
-----	---------

**17.47.3.168 divexact() [3/6]**

```
Integer & divexact (  
    Integer & q,  
    const Integer & n,  
    const int64_t & d ) [static]
```

Division operator.

## Parameters

$d$	divisor
-----	---------

**17.47.3.169 divexact() [4/6]**

```
Integer divexact (  
    const Integer & n,  
    const Integer & d ) [static]
```

Division when  $d$  divides  $n$ .

## Parameters

$n$	dividend
$d$	divisor

## Returns

exact quotient  $n/d$

## Warning

if quotient is not exact, the result is not predictable.

**17.47.3.170 divexact() [5/6]**

```
Integer divexact (  
    const Integer & n,  
    const uint64_t & d ) [static]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.171 divexact()** [6/6]

```
Integer divexact (  
    const Integer & n,  
    const int64_t & d ) [static]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.172 crem()** [1/3]

```
Integer & crem (  
    Integer & r,  
    const Integer & n,  
    const Integer & d ) [static]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.173 frem()** [1/3]

```
Integer & frem (  
    Integer & r,  
    const Integer & n,  
    const Integer & d ) [static]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.174 crem()** [2/3]

```
Integer & crem (  
    Integer & r,  
    const Integer & n,  
    const uint64_t & d ) [static]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.175 frem()** [2/3]

```
Integer & frem (  
    Integer & r,  
    const Integer & n,  
    const uint64_t & d ) [static]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.176 crem()** [3/3]

```
uint64_t crem (  
    const Integer & n,  
    const uint64_t & d ) [static]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.177 frem()** [3/3]

```
uint64_t frem (  

```

```
const Integer & n,  
const uint64_t & d ) [static]
```

Division operator.

#### Parameters

<i>d</i>	divisor
----------	---------

### 17.47.3.178 operator/() [1/5]

```
Integer operator/ (  
    const Integer & d ) const
```

Division operator.

#### Parameters

<i>d</i>	divisor
----------	---------

### 17.47.3.179 operator/() [2/5]

```
Integer operator/ (  
    const uint64_t d ) const
```

Division operator.

#### Parameters

<i>d</i>	divisor
----------	---------

### 17.47.3.180 operator/() [3/5]

```
Integer operator/ (  
    const int64_t d ) const
```

Division operator.

#### Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.181 operator/()** [4/5]

```
giv_all_inlined Integer operator/ (
    const uint32_t d ) const [inline]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.182 operator/()** [5/5]

```
giv_all_inlined Integer operator/ (
    const int32_t d ) const [inline]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.183 operator/=()** [1/6]

```
Integer & operator/= (
    const Integer & d )
```

Division operator (inplace).

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.184 operator/=()** [2/6]

```
Integer & operator/= (
    const uint64_t d )
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.185 operator/=() [3/6]**

```
Integer & operator/= (
    const int64_t d )
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.186 operator/=() [4/6]**

```
giv_all_inlined Integer & operator/= (
    const uint32_t d ) [inline]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.187 operator/=() [5/6]**

```
giv_all_inlined Integer & operator/= (
    const int32_t d ) [inline]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.188 operator/=( ) [6/6]**

```
Integer & operator/= (
    const XXX & d ) [inline]
```

Division operator.

Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.189 modin() [1/3]**

```
Integer & modin (
    Integer & r,
    const Integer & n ) [static]
```

Function mod (inplace).

$$r \leftarrow r \bmod n$$

Parameters

<i>r</i>	remainder
<i>n</i>	modulus

**17.47.3.190 modin() [2/3]**

```
Integer & modin (
    Integer & r,
    const int64_t n ) [static]
```

Division operator.

Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.191 modin() [3/3]**

```
Integer & modin (
    Integer & r,
    const uint64_t n ) [static]
```

Division operator.

#### Parameters

$d$	divisor
-----	---------

### 17.47.3.192 mod() [1/5]

```
Integer & mod (
    Integer & r,
    const Integer & n,
    const Integer & d ) [static]
```

Function mod.

$$r \leftarrow n \bmod d$$

#### Parameters

$r$	remainder
$n$	integer
$d$	modulus

### 17.47.3.193 mod() [2/5]

```
Integer & mod (
    Integer & r,
    const Integer & n,
    const int64_t d ) [static]
```

Division operator.

#### Parameters

$d$	divisor
-----	---------

### 17.47.3.194 mod() [3/5]

```
Integer & mod (
    Integer & r,
    const Integer & n,
    const uint64_t d ) [static]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.195 mod()** [4/5]

```
static giv_all_inlined Integer & mod (
    Integer & r,
    const Integer & n,
    const int32_t d ) [inline], [static]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.196 mod()** [5/5]

```
static giv_all_inlined Integer & mod (
    Integer & r,
    const Integer & n,
    const uint32_t d ) [inline], [static]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.197 divmod()** [1/3]

```
Integer & divmod (
    Integer & q,
    Integer & r,
    const Integer & n,
    const Integer & d ) [static]
```

Euclidean division.

$n = d \cdot q + r$  . Computes both the quotient and the residue (as in quorem).

## Parameters

$q$	as in the formula
$r$	as in the formula
$n$	as in the formula
$d$	as in the formula

## Returns

the quotient  $q$

**17.47.3.198 divmod()** [2/3]

```
Integer & divmod (
    Integer & q,
    int64_t & r,
    const Integer & n,
    const int64_t d ) [static]
```

Division operator.

## Parameters

$d$	divisor
-----	---------

**17.47.3.199 divmod()** [3/3]

```
Integer & divmod (
    Integer & q,
    uint64_t & r,
    const Integer & n,
    const uint64_t d ) [static]
```

Division operator.

## Parameters

$d$	divisor
-----	---------

**17.47.3.200 ceil()** [1/2]

```
Integer & ceil (
    Integer & res,
```

```
const Integer & n,  
const Integer & d ) [static]
```

rounding functions.

these are the same as the STL ones, except for the signature.

#### Parameters

<i>res</i>	the result
<i>n</i>	the numerator
<i>d</i>	the demominator

same as `std::ceil (n/d)`

#### 17.47.3.201 floor() [1/2]

```
Integer & floor (  
    Integer & res,  
    const Integer & n,  
    const Integer & d ) [static]
```

same as `std::floor(n/d)`

#### 17.47.3.202 trunc() [1/2]

```
Integer & trunc (  
    Integer & res,  
    const Integer & n,  
    const Integer & d ) [static]
```

same as `std::trunc(n/d)`

#### 17.47.3.203 ceil() [2/2]

```
Integer ceil (  
    const Integer & n,  
    const Integer & d ) [static]
```

rounding functions.

these are the same as the STL ones, except for the signature.

#### Parameters

<i>n</i>	the numerator
<i>d</i>	the demominator

**Returns**

n/d rounded.

same as std::ceil (n/d)

**17.47.3.204 floor()** [2/2]

```
Integer floor (  
    const Integer & n,  
    const Integer & d ) [static]
```

same as std::floor(n/d)

**17.47.3.205 trunc()** [2/2]

```
Integer trunc (  
    const Integer & n,  
    const Integer & d ) [static]
```

same as std::trunc(n/d)

**17.47.3.206 operator%()** [1/8]

```
Integer operator% (  
    const Integer & n ) const
```

Modulo operator.

**Parameters**

<i>n</i>	modulus
----------	---------

**Returns**

remainder (\*this) mod n

**17.47.3.207 operator%()** [2/8]

```
int64_t operator% (  
    const uint64_t n ) const
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.208 operator%() [3/8]**

```
int64_t operator% (  
    const int64_t n ) const
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.209 operator%() [4/8]**

```
giv_all_inlined int32_t operator% (  
    const uint32_t n ) const [inline]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.210 operator%() [5/8]**

```
giv_all_inlined int32_t operator% (  
    const int32_t n ) const [inline]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.211 operator%() [6/8]**

```
double operator% (  
    const double n ) const
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.212 operator%() [7/8]**

```
int16_t operator% (  
    const uint16_t n ) const [inline]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.213 operator%() [8/8]**

```
XXX operator% (  
    const XXX & n ) const [inline]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.3.214 operator%=( ) [1/6]**

```
Integer & operator%= (  
    const Integer & n )
```

Modulo operator (inplace).

## Parameters

<i>n</i>	modulus
----------	---------

## Returns

```
remainder (*this) <- (*this) mod n
```

**17.47.3.215 operator%=( ) [2/6]**

```
Integer & operator%= (
    const uint64_t n )
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.216 operator%=( ) [3/6]**

```
Integer & operator%= (
    const int64_t n )
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.217 operator%=( ) [4/6]**

```
giv_all_inlined Integer & operator%= (
    const uint32_t n ) [inline]
```

Division operator.

**Parameters**

<i>d</i>	divisor
----------	---------

**17.47.3.218 operator%=( ) [5/6]**

```
giv_all_inlined Integer & operator%= (
    const int32_t n ) [inline]
```

Division operator.

## Parameters

$d$	divisor
-----	---------

**17.47.3.219 operator%=( ) [6/6]**

```
Integer & operator%= (
    const XXX & n ) [inline]
```

Division operator.

## Parameters

$d$	divisor
-----	---------

**17.47.3.220 size\_in\_base()**

```
size_t size_in_base (
    int32_t B ) const
```

returns `ceil(log_BASE(*this))`.

**17.47.3.221 bitsize()**

```
size_t bitsize ( ) const
```

returns `ceil(log_2(*this))` .

**17.47.3.222 operator[]()**

```
uint64_t operator[] (
    size_t i ) const
```

return the i-th word of the integer.

Word 0 is lowest word.

**17.47.3.223 random\_less-than()**

```
Integer & random_less-than (
    Integer & r,
    const Integer & m ) [inline], [static]
```

returns a random integer  $r$  in the interval  $[x, m-1]$  where  $x = 0$  or  $-(m-1)$  according to ALWAYSPOSITIVE

**Bug**  $m$  has to be an integer here.

**17.47.3.224 print()**

```
std::ostream & print (
    std::ostream & o ) const
```

print32\_t integer.

**Parameters**

$o$	output stream.
-----	----------------

**17.47.4 Friends And Related Function Documentation****17.47.4.1 compare**

```
giv_all_inlined friend int32_t compare (
    const Integer & a,
    const Integer & b ) [friend]
```

Compares two integers.

**Parameters**

$a$	integer
$b$	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.2 absCompare [1/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const Integer & a,  
    const Integer & b ) [friend]
```

Compare the norm of two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $|a| > |b|$ , 0 if  $|a| = |b|$  and -1 otherwise.

**17.47.4.3 absCompare [2/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const Integer & a,  
    const double b ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.4 absCompare [3/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const Integer & a,  
    const float b ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.5 absCompare [4/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const Integer & a,  
    const uint64_t b ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.6 absCompare [5/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const Integer & a,  
    const unsigned b ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.7 absCompare [6/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const Integer & a,  
    const int64_t b ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.8 absCompare [7/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const Integer & a,  
    const int32_t b ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.9 absCompare [8/8]**

```
giv_all_inlined friend int32_t absCompare (  
    const T a,  
    const Integer & b ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

#### 17.47.4.10 isOne

```
giv_all_inlined int32_t isOne (  
    const Integer & a ) [friend]
```

name compare to 1 and 0

##### Parameters

<i>a</i>	
----------	--

#### 17.47.4.11 isMOne

```
giv_all_inlined int32_t isMOne (  
    const Integer & a ) [friend]
```

Compares two integers.

##### Parameters

<i>a</i>	integer
<i>b</i>	integer

##### Returns

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

#### 17.47.4.12 nonZero

```
giv_all_inlined int32_t nonZero (  
    const Integer & a ) [friend]
```

name compare to 1 and 0

##### Parameters

<i>a</i>	
----------	--

#### 17.47.4.13 isZero [1/7]

```
giv_all_inlined int32_t isZero (  
    const Integer & a ) [friend]
```

name compare to 1 and 0

#### Parameters

<i>a</i>	
----------	--

#### 17.47.4.14 isZero [2/7]

```
giv_all_inlined int32_t isZero (  
    const int16_t a ) [friend]
```

Compares two integers.

#### Parameters

<i>a</i>	integer
<i>b</i>	integer

#### Returns

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

#### 17.47.4.15 isZero [3/7]

```
giv_all_inlined int32_t isZero (  
    const int32_t a ) [friend]
```

Compares two integers.

#### Parameters

<i>a</i>	integer
<i>b</i>	integer

#### Returns

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

#### 17.47.4.16 isZero [4/7]

```
giv_all_inlined int32_t isZero (  
    const int64_t a ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.17 isZero [5/7]**

```
giv_all_inlined int32_t isZero (  
    const uint16_t a ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.18 isZero [6/7]**

```
giv_all_inlined int32_t isZero (  
    const uint32_t a ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.19 isZero** [7/7]

```
giv_all_inlined int32_t isZero (
    const uint64_t a ) [friend]
```

Compares two integers.

**Parameters**

<i>a</i>	integer
<i>b</i>	integer

**Returns**

1 if  $a > b$ , 0 if  $a = b$  and -1 otherwise.

**17.47.4.20 operator>=** [1/6]

```
giv_all_inlined friend int32_t operator>= (
    uint32_t l,
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

<i>l, n</i>	integers to compare
-------------	---------------------

**17.47.4.21 operator>=** [2/6]

```
giv_all_inlined friend int32_t operator>= (
    float l,
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

<i>l</i>	integer to be compared to
----------	---------------------------

**17.47.4.22 operator>= [3/6]**

```
giv_all_inlined friend int32_t operator>= (  
    double l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.23 operator>= [4/6]**

```
giv_all_inlined friend int32_t operator>= (  
    int32_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.24 operator>= [5/6]**

```
giv_all_inlined friend int32_t operator>= (  
    int64_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.25 operator>= [6/6]**

```
giv_all_inlined friend int32_t operator>= (  
    uint64_t l,  
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.26 operator<= [1/6]**

```
giv_all_inlined friend int32_t operator<= (  
    uint32_t l,  
    const Integer & n ) [friend]
```

less or equal

## Parameters

<i>l,n</i>	integers to compare
------------	---------------------

**17.47.4.27 operator<= [2/6]**

```
giv_all_inlined friend int32_t operator<= (  
    float l,  
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.28 operator<= [3/6]**

```
giv_all_inlined friend int32_t operator<= (  
    double l,  
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.29 operator<= [4/6]**

```
giv_all_inlined friend int32_t operator<= (
    int32_t l,
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.30 operator<= [5/6]**

```
giv_all_inlined friend int32_t operator<= (
    int64_t l,
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.31 operator<= [6/6]**

```
giv_all_inlined friend int32_t operator<= (
    uint64_t l,
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.32 operator"!= [1/6]**

```
giv_all_inlined friend int32_t operator!= (
    uint32_t l,
    const Integer & n ) [friend]
```

operator != (not equal)

**Parameters**

$l, n$	integer
--------	---------

**Returns**

1 iff  $l == n$

**17.47.4.33 operator"!= [2/6]**

```
giv_all_inlined friend int32_t operator!= (
    float l,
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.34 operator"!= [3/6]**

```
giv_all_inlined friend int32_t operator!= (
    double l,
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.35 operator"!= [4/6]**

```
giv_all_inlined friend int32_t operator!= (
    int32_t l,
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.36 operator">=" [5/6]**

```
giv_all_inlined friend int32_t operator!= (
    int64_t l,
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.37 operator">=" [6/6]**

```
giv_all_inlined friend int32_t operator!= (
    uint64_t l,
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.38 operator==" [1/6]**

```
giv_all_inlined friend int32_t operator== (
    uint32_t l,
    const Integer & n ) [friend]
```

Equality.

## Parameters

<i>l,n</i>	integers to compare
------------	---------------------

**17.47.4.39 operator== [2/6]**

```
giv_all_inlined friend int32_t operator== (  
    float l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.40 operator== [3/6]**

```
giv_all_inlined friend int32_t operator== (  
    double l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.41 operator== [4/6]**

```
giv_all_inlined friend int32_t operator== (  
    int32_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.42 operator== [5/6]**

```
giv_all_inlined friend int32_t operator== (  
    int64_t l,  
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.43 operator==** [6/6]

```
giv_all_inlined friend int32_t operator== (
    uint64_t l,
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.44 operator>** [1/6]

```
giv_all_inlined friend int32_t operator> (
    uint32_t l,
    const Integer & n ) [friend]
```

greater (strict)

## Parameters

<i>l,n</i>	integers to compare
------------	---------------------

**17.47.4.45 operator>=** [2/6]

```
giv_all_inlined friend int32_t operator>= (
    float l,
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.46 operator> [3/6]**

```
giv_all_inlined friend int32_t operator> (  
    double l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.47 operator> [4/6]**

```
giv_all_inlined friend int32_t operator> (  
    int32_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.48 operator> [5/6]**

```
giv_all_inlined friend int32_t operator> (  
    int64_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.49 operator> [6/6]**

```
giv_all_inlined friend int32_t operator> (  
    uint64_t l,  
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.50 operator< [1/6]**

```
giv_all_inlined friend int32_t operator< (  
    uint32_t l,  
    const Integer & n ) [friend]
```

less (strict)

## Parameters

<i>l,n</i>	integers to compare
------------	---------------------

**17.47.4.51 operator< [2/6]**

```
giv_all_inlined friend int32_t operator< (  
    float l,  
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.52 operator< [3/6]**

```
giv_all_inlined friend int32_t operator< (  
    double l,  
    const Integer & n ) [friend]
```

greater or equal.

## Parameters

/	integer to be compared to
---	---------------------------

**17.47.4.53 operator< [4/6]**

```
giv_all_inlined friend int32_t operator< (  
    int32_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.54 operator< [5/6]**

```
giv_all_inlined friend int32_t operator< (  
    int64_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.55 operator< [6/6]**

```
giv_all_inlined friend int32_t operator< (  
    uint64_t l,  
    const Integer & n ) [friend]
```

greater or equal.

**Parameters**

/	integer to be compared to
---	---------------------------

**17.47.4.56 operator+ [1/4]**

```
giv_all_inlined Integer operator+ (  
    const int32_t l,  
    const Integer & n ) [friend]
```

operator +.

## Parameters

$l, n$	to be added
--------	-------------

**17.47.4.57 operator+ [2/4]**

```
giv_all_inlined Integer operator+ (  
    const uint32_t l,  
    const Integer & n ) [friend]
```

operator +.

## Returns

(*\*this*)+*n*

## Parameters

$n$	as in the formula.
-----	--------------------

**17.47.4.58 operator+ [3/4]**

```
giv_all_inlined Integer operator+ (  
    const int64_t l,  
    const Integer & n ) [friend]
```

operator +.

## Returns

(*\*this*)+*n*

## Parameters

$n$	as in the formula.
-----	--------------------

**17.47.4.59 operator+ [4/4]**

```
giv_all_inlined Integer operator+ (  
    const uint64_t l,  
    const Integer & n ) [friend]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<code>n</code>	as in the formula.
----------------	--------------------

**17.47.4.60 operator- [1/4]**

```
giv_all_inlined Integer operator- (
    const int32_t l,
    const Integer & n ) [friend]
```

operator -

**Parameters**

<code>l,n</code>	to be subtracted
------------------	------------------

**17.47.4.61 operator- [2/4]**

```
giv_all_inlined Integer operator- (
    const uint32_t l,
    const Integer & n ) [friend]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<code>n</code>	as in the formula.
----------------	--------------------

**17.47.4.62 operator- [3/4]**

```
giv_all_inlined Integer operator- (
    const int64_t l,
    const Integer & n ) [friend]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.4.63 operator- [4/4]**

```
giv_all_inlined Integer operator- (
    const uint64_t l,
    const Integer & n ) [friend]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

<i>n</i>	as in the formula.
----------	--------------------

**17.47.4.64 operator\* [1/4]**

```
giv_all_inlined Integer operator* (
    const int32_t l,
    const Integer & n ) [friend]
```

operator \*

**Parameters**

<i>l,n</i>	to be multpct
------------	---------------

**17.47.4.65 operator\* [2/4]**

```
giv_all_inlined Integer operator* (
    const uint32_t l,
    const Integer & n ) [friend]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.4.66 operator\* [3/4]**

```
giv_all_inlined Integer operator* (  
    const int64_t l,  
    const Integer & n ) [friend]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.4.67 operator\* [4/4]**

```
giv_all_inlined Integer operator* (  
    const uint64_t l,  
    const Integer & n ) [friend]
```

operator +.

**Returns**

`(*this)+n`

**Parameters**

$n$	as in the formula.
-----	--------------------

**17.47.4.68 operator/ [1/2]**

```
giv_all_inlined Integer operator/ (
```

```
const int64_t l,
const Integer & n ) [friend]
```

Division operator.

#### Parameters

<i>d</i>	divisor
----------	---------

### 17.47.4.69 operator/ [2/2]

```
giv_all_inlined Integer operator/ (
    const uint64_t l,
    const Integer & n ) [friend]
```

Division operator.

#### Parameters

<i>d</i>	divisor
----------	---------

### 17.47.4.70 operator% [1/4]

```
giv_all_inlined Integer operator% (
    const int64_t l,
    const Integer & n ) [friend]
```

operator %

#### Parameters

<i>l</i>	
<i>n</i>	

#### Returns

nl

### 17.47.4.71 operator% [2/4]

```
giv_all_inlined Integer operator% (
    const uint64_t l,
    const Integer & n ) [friend]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.4.72 operator% [3/4]**

```
giv_all_inlined Integer operator% (  
    const int32_t l,  
    const Integer & n ) [friend]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.4.73 operator% [4/4]**

```
giv_all_inlined Integer operator% (  
    const uint32_t l,  
    const Integer & n ) [friend]
```

Division operator.

## Parameters

<i>d</i>	divisor
----------	---------

**17.47.4.74 gcd [1/4]**

```
gcd (  
    const Integer & a,  
    const Integer & b ) [friend]
```

gcd.

## Parameters

<i>a,b</i>	integers
------------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.75 gcd [2/4]**

```
giv_all_inlined Integer gcd (  
    Integer & u,  
    Integer & v,  
    const Integer & a,  
    const Integer & b ) [friend]
```

gcd.

## Parameters

<code>a,b</code>	integers
------------------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.76 gcd [3/4]**

```
giv_all_inlined Integer & gcd (  
    Integer & g,  
    const Integer & a,  
    const Integer & b ) [friend]
```

gcd.

## Parameters

<code>a,b</code>	integers
------------------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.77 gcd [4/4]**

```

giv_all_inlined Integer & gcd (
    Integer & g,
    Integer & u,
    Integer & v,
    const Integer & a,
    const Integer & b ) [friend]

```

gcd.

**Parameters**

$a, b$	integers
--------	----------

**Returns**

gcd(a,b)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.78 inv**

```

giv_all_inlined Integer & inv (
    Integer & u,
    const Integer & a,
    const Integer & b ) [friend]

```

Inverse.

Compute the inverse  $u = a/b$ .

**Parameters**

	$u$	
	$a$	
	$b$	
	$a$	
	$b$	
out	$u$	is set to $a^{-1}$ modulo b

**17.47.4.79 invin**

```

giv_all_inlined Integer & invin (
    Integer & u,
    const Integer & b ) [friend]

```

Compute the inverse inplace  $u = u/b$ .

## Parameters

$u$	
$b$	

**17.47.4.80 pp**

```
giv_all_inlined Integer pp (
    const Integer & P,
    const Integer & Q ) [friend]
```

pp

## Parameters

$P, Q$	params
--------	--------

**17.47.4.81 lcm [1/2]**

```
giv_all_inlined Integer & lcm (
    Integer & g,
    const Integer & a,
    const Integer & b ) [friend]
```

lcm

## Parameters

$g, a, b$	
-----------	--

## Returns

 $g = \text{lcm}(a, b)$ **17.47.4.82 lcm [2/2]**

```
giv_all_inlined Integer lcm (
    const Integer & a,
    const Integer & b ) [friend]
```

lcm

## Parameters

$a, b$	
--------	--

**17.47.4.83 pow [1/9]**

```
giv_all_inlined Integer & pow (
    Integer & Res,
    const Integer & n,
    const int64_t l ) [friend]
```

pow.

return  $n^l$

## Parameters

$Res, n, l$	
-------------	--

**17.47.4.84 pow [2/9]**

```
giv_all_inlined Integer & pow (
    Integer & Res,
    const uint64_t n,
    const uint64_t l ) [friend]
```

gcd.

## Parameters

$a, b$	integers
--------	----------

## Returns

[gcd\(a,b\)](#)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.85 pow [3/9]**

```
giv_all_inlined Integer & pow (
    Integer & Res,
    const Integer & n,
    const uint64_t l ) [friend]
```

gcd.

## Parameters

$a, b$	integers
--------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.86 pow [4/9]**

```
giv_all_inlined Integer & pow (  
    Integer & Res,  
    const Integer & n,  
    const int32_t l ) [friend]
```

`gcd.`

## Parameters

$a, b$	integers
--------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.87 pow [5/9]**

```
giv_all_inlined Integer & pow (  
    Integer & Res,  
    const Integer & n,  
    const uint32_t l ) [friend]
```

`gcd.`

## Parameters

$a, b$	integers
--------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.88 pow [6/9]**

```
giv_all_inlined Integer pow (
    const Integer & n,
    const int64_t l ) [friend]
```

pow.

return  $n^l$

**Parameters**

$n, l$	
--------	--

**17.47.4.89 pow [7/9]**

```
giv_all_inlined Integer pow (
    const Integer & n,
    const uint64_t l ) [friend]
```

gcd.

**Parameters**

$a, b$	integers
--------	----------

**Returns**

$\text{gcd}(a, b)$

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.90 pow [8/9]**

```
giv_all_inlined Integer pow (
    const Integer & n,
    const int32_t l ) [friend]
```

gcd.

**Parameters**

$a, b$	integers
--------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.91 pow [9/9]**

```
giv_all_inlined Integer pow (
    const Integer & n,
    const uint32_t l ) [friend]
```

gcd.

## Parameters

<code>a,b</code>	integers
------------------	----------

## Returns

`gcd(a,b)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

**17.47.4.92 powmod**

```
giv_all_inlined Integer powmod (
    const Integer & n,
    const uint64_t e,
    const Integer & m ) [friend]
```

modular pow.

return  $n^e \bmod m$ .

## Parameters

<code>n,e,m</code>	
--------------------	--

**17.47.4.93 fact**

```
giv_all_inlined Integer fact (
    uint64_t l ) [friend]
```

fact

## Parameters

$i$	
-----	--

**17.47.4.94 sqrt [1/2]**

```
giv_all_inlined Integer sqrt (  
    const Integer & p ) [friend]
```

(square) roots

## Parameters

$p$	
-----	--

**17.47.4.95 sqrt [2/2]**

```
giv_all_inlined Integer & sqrt (  
    Integer & r,  
    const Integer & p ) [friend]
```

(square) roots

## Parameters

$r, p$	
--------	--

**17.47.4.96 sqrtrem [1/2]**

```
giv_all_inlined Integer sqrtrem (  
    const Integer & p,  
    Integer & rem ) [friend]
```

(square) roots

## Parameters

$p, rem$	
----------	--

**17.47.4.97 sqrtrem [2/2]**

```

giv_all_inlined Integer & sqrtrem (
    Integer & r,
    const Integer & p,
    Integer & rem ) [friend]

```

(square) roots

**Parameters**

<i>r,p,rem</i>	
----------------	--

**17.47.4.98 root**

```

giv_all_inlined bool root (
    Integer & q,
    const Integer & a,
    uint32_t n ) [friend]

```

(square) roots

**Parameters**

<i>q,a,n</i>	
--------------	--

**17.47.4.99 logp**

```

giv_all_inlined int64_t logp (
    const Integer & a,
    const Integer & p ) [friend]

```

logs

**Parameters**

<i>a,p</i>	
------------	--

**17.47.4.100 logtwo**

```

giv_all_inlined double logtwo (
    const Integer & a ) [friend]

```

logs

## Parameters

<i>a</i>	
----------	--

**17.47.4.101 naturallog**

```
giv_all_inlined double naturallog (  
    const Integer & a ) [friend]
```

logs

## Parameters

<i>a</i>	
----------	--

**17.47.4.102 swap**

```
giv_all_inlined void swap (  
    Integer & a,  
    Integer & b ) [friend]
```

swap

## Parameters

<i>a,b</i>	
------------	--

**17.47.4.103 sign**

```
int32_t sign (  
    const Integer & a ) [friend]
```

sign

## Parameters

<i>a</i>	
----------	--

#### 17.47.4.104 length

```
giv_all_inlined uint64_t length (  
    const Integer & a ) [friend]
```

returns the number of bytes used to store \*this

##### Parameters

<i>a</i>	
----------	--

**Bug** JGD 23.04.2012: shouldn't it be "mp\_limb\_t" instead of "uint64\_t"?

#### 17.47.4.105 isOdd

```
giv_all_inlined bool isOdd (  
    const Integer & a ) [friend]
```

parity of an integer

##### Parameters

<i>a</i>	integer
----------	---------

##### Returns

1 if odd, 0 if even

#### 17.47.4.106 operator>>

```
giv_all_inlined std::istream & operator>> (  
    std::istream & i,  
    Integer & n ) [friend]
```

Input/Output of Integers.

in operator.

##### Parameters

<i>i</i>	input stream
<i>n</i>	integer to be built

#### 17.47.4.107 operator<<

```
giv_all_inlined std::ostream & operator<< (  
    std::ostream & o,  
    const Integer & n ) [friend]
```

out operator.

##### Parameters

<i>o</i>	output stream
<i>n</i>	integer to be printed

#### 17.47.4.108 absOutput

```
giv_all_inlined std::ostream & absOutput (  
    std::ostream & o,  
    const Integer & n ) [friend]
```

nodoc

##### Parameters

<i>o</i>	output
<i>n</i>	integer

#### 17.47.4.109 Protected::importWords

```
giv_all_inlined void Protected::importWords (  
    Integer & x,  
    size_t count,  
    int32_t order,  
    int32_t size,  
    int32_t endian,  
    size_t nails,  
    const void * op ) [friend]
```

nodoc

##### Parameters

<i>x</i>	x
<i>count</i>	x
<i>order</i>	x

## Parameters

<i>size</i>	x
<i>endian</i>	x
<i>nails</i>	x
<i>op</i>	x

The documentation for this class was generated from the following files:

- [gmp++\\_int.h](#) (4.1.1)
- [gmp++\\_int\\_add.C](#) (4.1.1)
- [gmp++\\_int\\_compare.C](#) (4.1.1)
- [gmp++\\_int\\_cstor.C](#) (4.1.1)
- [gmp++\\_int\\_div.C](#) (4.1.1)
- [gmp++\\_int\\_io.C](#) (4.1.1)
- [gmp++\\_int\\_lib.C](#) (4.1.1)
- [gmp++\\_int\\_misc.C](#) (4.1.1)
- [gmp++\\_int\\_mod.C](#) (4.1.1)
- [gmp++\\_int\\_mul.C](#) (4.1.1)
- [gmp++\\_int\\_rand.inl](#) (4.1.1)
- [gmp++\\_int\\_sub.C](#) (4.1.1)

## 17.48 IntegerDom Class Reference

[Integer](#) Domain.

```
#include <givinteger.h>
```

Inheritance diagram for IntegerDom:

Collaboration diagram for IntegerDom:

### Public Member Functions

- bool **isUnit** (const [Rep](#) &x) const  
*isUnit*
- bool **isDivisor** (const [Element](#) &a, const [Element](#) &b) const  
*isDivisor (a, b) Test if  $b \mid a$ .*

### 17.48.1 Detailed Description

[Integer](#) Domain.

#### Examples

[examples/Integer/igcd.C](#), [examples/Integer/igcdext.C](#), [examples/Integer/ilcm.C](#), [examples/Integer/lambda.C](#), [examples/Integer/lambda\\_inv.C](#), [examples/Integer/phi.C](#), [examples/Integer/primitiveelement.C](#), and [examples/Integer/probable\\_primroot.C](#).

The documentation for this class was generated from the following file:

- [givinteger.h](#) (4.1.1)

## 17.49 Interpolation< Domain, REDUCE > Struct Template Reference

[Interpolation.](#)

```
#include <givinterp.h>
```

Inheritance diagram for Interpolation< Domain, REDUCE >:

Collaboration diagram for Interpolation< Domain, REDUCE >:

### Public Member Functions

- Rep & [setdegree](#) (Rep &P) const  
*Compute the degree of P.*
- size\_t & [sqrfree](#) (size\_t &Nfact, Rep \*Fact, const Rep &P) const  
*Sqrfree decomposition.*

### 17.49.1 Detailed Description

```
template<class Domain, bool REDUCE = true>
struct Givaro::Interpolation< Domain, REDUCE >
```

[Interpolation.](#)

Examples

[examples/Polynomial/interpolate.C.](#)

### 17.49.2 Member Function Documentation

#### 17.49.2.1 setdegree()

```
Poly1Dom< Domain, Dense >::Rep & setdegree (
    Rep & P ) const [inline], [inherited]
```

Compute the degree of P.

**Warning**

this is an infamous function that may not leave P constant !!

**Parameters**

$P$	polynomial
-----	------------

### 17.49.2.2 sqrfree()

```
size_t & sqrfree (
    size_t & Nfact,
    Rep * Fact,
    const Rep & P ) const [inherited]
```

Sqrfree decomposition.

Decompose  $P$  such that:  $P = \text{Fact}[0]^0 * \text{Fact}[1]^1 * \dots * \text{Fact}[P.\text{degree}()]^{(P.\text{degree}())}$ , with  $\text{Fact}[0]$  the leading coefficient. The array  $\text{Fact}$  must be allocated before calling the function. The size of  $\text{Fact}$  must be  $\text{deg}P+1$  is all factors should be computed. For more readable version of the algorithm, see Geddes, p342.

#### Parameters

<i>Nfact</i>	[in] the size of <i>Fact</i>
<i>Fact</i>	[in] an array of dimension <i>Nfact</i>
<i>Nfact</i>	[out] is the number of factor in the sqrfree decomposition
<i>Fact</i>	[out] contains at most <i>Nfact</i> factors of the decomposition.
<i>P</i>	rep.

The documentation for this struct was generated from the following file:

- [givinterp.h \(4.1.1\)](#)

## 17.50 IntFactorDom< MyRandIter > Class Template Reference

[Integer](#) Factor Domain.

```
#include <givintfactor.h>
```

Inheritance diagram for IntFactorDom< MyRandIter >:

Collaboration diagram for IntFactorDom< MyRandIter >:

### Public Member Functions

- `template<class Container1 , class Container2 >`  
`bool set (Container1 &setint, Container2 &setpwd, const Rep &a, unsigned long loops=0) const`  
*Factors with primes.*
- `Rep & Erathostene (Rep &, const Rep &p) const`  
*returns a small factor*
- `bool isUnit (const Rep &x) const`  
*isUnit*
- `bool isDivisor (const Element &a, const Element &b) const`  
*isDivisor (a, b) Test if  $b \mid a$ .*

### 17.50.1 Detailed Description

```
template<class MyRandlter = GivRandom>
class Givaro::IntFactorDom< MyRandlter >
```

[Integer](#) Factor Domain.

Examples

[examples/Integer/ProbLucas.C](#), [examples/Integer/ifactor.C](#), and [examples/Integer/ifactor\\_lenstra.C](#).

The documentation for this class was generated from the following files:

- [givintfactor.h](#) (4.1.1)
- [givintfactor.inl](#) (4.1.1)

## 17.51 IntNumTheoDom< MyRandlter > Class Template Reference

Num theory Domain.

```
#include <givintnumtheo.h>
```

Inheritance diagram for IntNumTheoDom< MyRandlter >:

Collaboration diagram for IntNumTheoDom< MyRandlter >:

### Public Member Functions

- template<template< class, class > class Container, template< class > class Alloc>  
Rep & **phi** (Rep &res, const Container< Rep, Alloc< Rep > > &Lf, const Rep &n) const  
*Euler's phi function.*
- Rep & **prim\_root** (Rep &, const Rep &) const  
*Primitive Root.*
- template<class Array >  
Rep & **prim\_root\_of\_prime** (Rep &A, const Array &Lf, const Rep &phin, const Rep &n) const  
*Add Jacobi for quadratic nonresidue.*
- Rep & **probable\_prim\_root** (Rep &, double &, const Rep &n, const uint64\_t L=100000000\_ui64) const  
*Polynomial-time generation of primitive roots.*
- Rep & **probable\_prim\_root** (Rep &, double &, const Rep &n, const double epsilon) const  
*Here L is computed so that the error is close to epsilon.*
- Rep & **prim\_inv** (Rep &, const Rep &) const  
*Generalization of primitive roots for any modulus Primitive means maximal order Primitive Element, Primitive invertible Both functions concides except for m=8.*
- template<template< class, class > class Container, template< class > class Alloc>  
short **mobius** (const Container< Rep, Alloc< Rep > > &lpow) const  
*Mbius function.*
- short **mobius** (const Rep &a) const  
*Mbius function.*
- template<class Container1 , class Container2 >  
bool **set** (Container1 &setint, Container2 &setpwd, const Rep &a, unsigned long loops=0) const  
*Factors with primes.*
- Rep & **Erathostene** (Rep &, const Rep &p) const  
*returns a small factor*
- bool **isUnit** (const Rep &x) const  
*isUnit*
- bool **isDivisor** (const [Element](#) &a, const [Element](#) &b) const  
*isDivisor (a, b) Test if b | a.*

### 17.51.1 Detailed Description

```
template<class MyRandIter = GivRandom>
class Givaro::IntNumTheoDom< MyRandIter >
```

Num theory Domain.

#### Examples

[examples/Integer/isproof.C](#), [examples/Integer/lambda.C](#), [examples/Integer/lambda\\_inv.C](#), [examples/Integer/order.C](#), [examples/Integer/phi.C](#), [examples/Integer/primitiveelement.C](#), [examples/Integer/primitiveroot.C](#), and [examples/Integer/probable\\_primroot.C](#).

### 17.51.2 Member Function Documentation

#### 17.51.2.1 probable\_prim\_root()

```
IntNumTheoDom< MyRandIter >::Rep & probable_prim_root (
    Rep & primroot,
    double & error,
    const Rep & n,
    const uint64_t L = 10000000_ui64 ) const
```

Polynomial-time generation of primitive roots.

*L* is number of loops of Pollard partial factorization of  $n-1$  10,000,000 gives at least  $1-2^{-40}$  probability of success [Dubrois & Dumas, Industrial-strength primitive roots] Returns the probable primitive root and the probability of error.

#### 17.51.2.2 prim\_inv()

```
IntNumTheoDom< MyRandIter >::Rep & prim_inv (
    Rep & A,
    const Rep & n ) const
```

Generalization of primitive roots for any modulus Primitive means maximal order Primitive Element, Primitive invertible Both functions concides except for  $m=8$ .

Lambda Function : maximal orbit size lambda : Order of a primitive Element lambda\_inv : Order of an invertible primitive Element Both functions concides except for  $m=8$

The documentation for this class was generated from the following files:

- [givintnumtheo.h \(4.1.1\)](#)
- [givintnumtheo.inl \(4.1.1\)](#)

## 17.52 IntPrimeDom Class Reference

Primality tests.

```
#include <givintprime.h>
```

Inheritance diagram for IntPrimeDom:

Collaboration diagram for IntPrimeDom:

### Public Member Functions

- bool **isUnit** (const [Rep](#) &x) const  
*isUnit*
- bool **isDivisor** (const [Element](#) &a, const [Element](#) &b) const  
*isDivisor (a, b) Test if  $b \mid a$ .*

### 17.52.1 Detailed Description

Primality tests.

#### Examples

[examples/Integer/ispower.C](#), [examples/Integer/isprime.C](#), [examples/Integer/nb\\_primes.C](#), [examples/Integer/nextprime.C](#), [examples/Integer/prevprime.C](#), and [examples/Polynomial/PolynomialCRT.C](#).

The documentation for this class was generated from the following files:

- [givintprime.h \(4.1.1\)](#)
- [givintprime.C \(4.1.1\)](#)
- [givintprime.inl \(4.1.1\)](#)

## 17.53 IntRNSsystem< Container, Alloc > Class Template Reference

RNS system class. No doc.

```
#include <givintrns.h>
```

Inheritance diagram for IntRNSsystem< Container, Alloc >:

Collaboration diagram for IntRNSsystem< Container, Alloc >:

### Public Member Functions

- bool **isUnit** (const [Rep](#) &x) const  
*isUnit*
- bool **isDivisor** (const [Element](#) &a, const [Element](#) &b) const  
*isDivisor (a, b) Test if  $b \mid a$ .*

### 17.53.1 Detailed Description

```
template<template< class, class > class Container, template< class > class Alloc>
class Givaro::IntRNSsystem< Container, Alloc >
```

RNS system class. No doc.

The documentation for this class was generated from the following files:

- [givintrns.h \(4.1.1\)](#)
- [givintrns\\_convert.inl \(4.1.1\)](#)
- [givintrns\\_cstor.inl \(4.1.1\)](#)

## 17.54 IntRSADom< MyRandlter > Class Template Reference

RSA domain.

```
#include <givintrsa.h>
```

Inheritance diagram for IntRSADom< MyRandlter >:

Collaboration diagram for IntRSADom< MyRandlter >:

### Public Member Functions

- **IntRSADom** (bool fi=false, MyRandlter g=MyRandlter())  
*Constructors.*
- const **Element** & **getn** () const  
*Accesses.*
- std::ostream & **encipher** (std::ostream &, std::istream &) const  
*Text conversions.*
- **Element** & **strong\_prime** (random\_generator &g, int64\_t psize, **Element** &p) const  
*Strong Primes.*
- void **keys\_gen** (random\_generator &g, int64\_t psize, int64\_t qsize, **Element** &n, **Element** &e, **Element** &d, **Element** &p, **Element** &q) const  
*Key gen.*
- int64\_t **log** (const **Element** &n, const int64\_t t=10) const  
*log[10]*
- std::ostream & **ecriture\_str** (std::ostream &, const **Element** &) const  
*Text conversions.*
- **Element** & **point\_break** (**Element** &u)  
*Breaking codes : finding u knowing only m an k ...*
- template<class Container1 , class Container2 >  
bool **set** (Container1 &setint, Container2 &setpwd, const **Rep** &a, unsigned long loops=0) const  
*Factors with primes.*
- **Rep** & **Erathostene** (**Rep** &, const **Rep** &p) const  
*returns a small factor*
- bool **isUnit** (const **Rep** &x) const  
*isUnit*
- bool **isDivisor** (const **Element** &a, const **Element** &b) const  
*isDivisor (a, b) Test if b | a.*

## Protected Attributes

- `bool _fast_impl`  
*Fast implementation.*

### 17.54.1 Detailed Description

```
template<class MyRandIter = GivRandom>
class Givaro::IntRSADom< MyRandIter >
```

RSA domain.

#### Examples

[examples/Integer/RSA\\_breaking.C](#), [examples/Integer/RSA\\_decipher.C](#), [examples/Integer/RSA\\_encipher.C](#),  
and [examples/Integer/RSA\\_keys\\_generator.C](#).

### 17.54.2 Member Function Documentation

#### 17.54.2.1 strong\_prime()

```
IntRSADom< MyRandIter >::Element & strong_prime (
    random_generator & g,
    int64_t psize,
    Element & p ) const
```

Strong Primes.

**Bibliography** • J. Gordon, *Strong Primes Are Easy to Find*, EUROCRYPT'84, LNCS 209.

#### 17.54.2.2 keys\_gen()

```
void keys_gen (
    random_generator & g,
    int64_t psize,
    int64_t qsize,
    Element & n,
    Element & e,
    Element & d,
    Element & p,
    Element & q ) const
```

**Key** gen.

Here  $m = p \cdot q$   $p$  and  $q$  are prime numbers of respective sizes  $psize$ ,  $qsize$  Moreover  $p-1$  and  $q-1$  have one prime factor of respective size  $2/3$  since  $k.u = 1 \bmod (p-1)(q-1)$

### 17.54.3 Field Documentation

#### 17.54.3.1 `_fast_impl`

```
bool _fast_impl [protected]
```

Fast implementation.

Means simple enciphering key, and deciphering via chinese remaindering.

#### Warning

this means less security !

The documentation for this class was generated from the following files:

- [givintrsa.h \(4.1.1\)](#)
- [givintrsa.inl \(4.1.1\)](#)

## 17.55 IntSqrtModDom< MyRandIter > Class Template Reference

[Modular](#) square roots.

```
#include <givintsqrootmod.h>
```

Inheritance diagram for IntSqrtModDom< MyRandIter >:

Collaboration diagram for IntSqrtModDom< MyRandIter >:

### Public Member Functions

- `template<class Container1 , class Container2 >`  
`bool set (Container1 &setint, Container2 &setpwd, const Rep &a, unsigned long loops=0) const`  
*Factors with primes.*
- `Rep & Erathostene (Rep &, const Rep &p) const`  
*returns a small factor*
- `bool isUnit (const Rep &x) const`  
*isUnit*
- `bool isDivisor (const Element &a, const Element &b) const`  
*isDivisor (a, b) Test if b | a.*

### 17.55.1 Detailed Description

```
template<class MyRandlter = GivRandom>
class Givaro::IntSqrtModDom< MyRandlter >
```

[Modular](#) square roots.

Examples

[examples/Integer/ModularSquareRoot.C.](#)

The documentation for this class was generated from the following files:

- [givintsqrootmod.h \(4.1.1\)](#)
- [givintsqrootmod.inl \(4.1.1\)](#)

## 17.56 Key< T > Class Template Reference

The class [Key](#).

```
#include <givhashtable.h>
```

### 17.56.1 Detailed Description

```
template<class T>
class Givaro::Key< T >
```

The class [Key](#).

must have :

- default ctor
- operator== : (const [Key](#)&, const [Key](#)&) -> int
- godel : void -> int

The documentation for this class was generated from the following file:

- [givhashtable.h \(4.1.1\)](#)

## 17.57 List0< T > Class Template Reference

ListO.

```
#include <givlist0.h>
```

### 17.57.1 Detailed Description

```
template<class T>
class Givaro::List0< T >
```

ListO.

The documentation for this class was generated from the following files:

- [givlist0.h \(4.1.1\)](#)
- [givlist0.inl \(4.1.1\)](#)

## 17.58 Modular< IntType, \_Compute\_t, Enable > Class Template Reference

Forward declaration for [Givaro::Modular](#).

```
#include <modular-inttype.h>
```

Inheritance diagram for Modular< IntType, \_Compute\_t, Enable >:

Collaboration diagram for Modular< IntType, \_Compute\_t, Enable >:

### 17.58.1 Detailed Description

```
template<typename IntType, typename _Compute_t, typename Enable>
class Givaro::Modular< IntType, _Compute_t, Enable >
```

Forward declaration for [Givaro::Modular](#).

This class implement the standard arithmetic with Modulo Elements.

Elements will be stored in the storage type. While arithmetics will occur on the unsigned version of COMP type.  
Example: Modular<int32\_t, uint64\_t>

- The representation of an integer a in Z<sub>p</sub> is the value a % p

#### Examples

[examples/FiniteField/all\\_field.C](#), [examples/FiniteField/exponentiation.C](#), [examples/FiniteField/ff\\_arith.C](#),  
[examples/FiniteField/zpz\\_atomic.C](#), [examples/Integer/ModularSquareRoot.C](#), [examples/Integer/ieponentiation.C](#),  
[examples/Polynomial/PolynomialCRT.C](#), and [examples/Polynomial/interpolate.C](#).

The documentation for this class was generated from the following files:

- [modular-general.h \(4.1.1\)](#)
- [modular-inttype.h \(4.1.1\)](#)
- [modular-floating.inl \(4.1.1\)](#)
- [modular-integral.inl \(4.1.1\)](#)
- [modular-inttype.inl \(4.1.1\)](#)
- [modular-ruint.inl \(4.1.1\)](#)

## 17.59 Modular< \_Storage\_t, \_Compute\_t, typename std::enable\_if< is\_same\_ruint< \_Storage\_t, \_Compute\_t >::value||is\_smaller\_ruint< \_Storage\_t, \_Compute\_t >::value >::type > Class Template Reference

The standard arithmetic in modular rings using fixed size precision.

```
#include <modular-ruint.h>
```

Inheritance diagram for Modular< \_Storage\_t, \_Compute\_t, typename std::enable\_if< is\_same\_ruint< \_Storage\_t, \_Compute\_t >::value||is\_smaller\_ruint< \_Storage\_t, \_Compute\_t >::value >::type >:

Collaboration diagram for Modular< \_Storage\_t, \_Compute\_t, typename std::enable\_if< is\_same\_ruint< \_Storage\_t, \_Compute\_t >::value||is\_smaller\_ruint< \_Storage\_t, \_Compute\_t >::value >::type >:

### 17.59.1 Detailed Description

```
template<typename _Storage_t, typename _Compute_t>
class Givaro::Modular< _Storage_t, _Compute_t, typename std::enable_if< is_same_ruint< _Storage_t, _Compute_t >::value||is_smaller_ruint< _Storage_t, _Compute_t >::value >::type >
```

The standard arithmetic in modular rings using fixed size precision.

The documentation for this class was generated from the following file:

- modular-ruint.h (4.1.1)

## 17.60 Modular< Integer > Class Reference

This class implement the standard arithmetic with Modulo Elements.

```
#include <modular-integer.h>
```

Inheritance diagram for Modular< Integer >:

Collaboration diagram for Modular< Integer >:

### 17.60.1 Detailed Description

This class implement the standard arithmetic with Modulo Elements.

- The representation of an integer  $a$  in  $Z_p$  is the value  $a \% p$

The documentation for this class was generated from the following files:

- modular-integer.h (4.1.1)
- modular-integer.inl (4.1.1)

## 17.61 Modular< Log16 > Class Reference

This class implement the standard arithmetic with Modulo Elements.

```
#include <modular-log16.h>
```

### 17.61.1 Detailed Description

This class implement the standard arithmetic with Modulo Elements.

- The representation of an integer  $a$  in  $Z_p$  is the value  $a \% p$
- $p$  max is 16381
- $p$  si supposed to be prime

The documentation for this class was generated from the following files:

- modular-log16.h (4.1.1)
- modular-log16.inl (4.1.1)

## 17.62 Modular\_implem< \_Storage\_t, \_Compute\_t, \_Residu\_t > Class Template Reference

This class implement the standard arithmetic with Modulo Elements.

```
#include <modular-implem.h>
```

Inheritance diagram for Modular\_implem< \_Storage\_t, \_Compute\_t, \_Residu\_t >:

### 17.62.1 Detailed Description

```
template<typename _Storage_t, typename _Compute_t, typename _Residu_t>  
class Givaro::Modular_implem< _Storage_t, _Compute_t, _Residu_t >
```

This class implement the standard arithmetic with Modulo Elements.

- The representation of an integer  $a$  in  $Z_p$  is the value  $a \% p$
- $m$  max is 46341
- $p$  max is 46337

The documentation for this class was generated from the following file:

- [modular-implem.h \(4.1.1\)](#)

## 17.63 ModularRandIter< Ring > Class Template Reference

Random ring Element generator.

```
#include <givranditer.h>
```

### Common Object Interface.

These methods are required of all LinBox random ring Element generators.

- typedef Ring::Element [Element](#)  
*Ring Element type.*
- [ModularRandIter](#) (const Ring &F, const size\_t &size=0, const uint64\_t &seed=0)  
*Constructor from ring, sampling size, and seed.*
- [ModularRandIter](#) (const [ModularRandIter](#) &R)  
*Copy constructor.*
- [~ModularRandIter](#) (void)  
*Destructor.*
- [ModularRandIter](#)< Ring > & [operator=](#) (const [ModularRandIter](#)< Ring > &R)  
*Assignment operator.*
- [Element](#) & [operator\(\)](#) ([Element](#) &elt) const  
*Random ring Element creator with assignement.*
- [Element](#) & [random](#) ([Element](#) &elt) const  
*Ring Element type.*
- [Element](#) [operator\(\)](#) () const  
*Ring Element type.*
- [Element](#) [random](#) () const  
*Ring Element type.*

### Implementation-Specific Methods.

- const Ring & [ring](#) () const  
*Random generator.*

#### 17.63.1 Detailed Description

```
template<class Ring>
class Givaro::ModularRandIter< Ring >
```

Random ring Element generator.

This class defines a ring Element generator for all givaro modular rings (Gfq and [Modular](#)) throught a template argument as a ring. The random generator used is the givrandom.

#### 17.63.2 Member Typedef Documentation

### 17.63.2.1 Element

```
typedef Ring::Element Element
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

## 17.63.3 Constructor & Destructor Documentation

### 17.63.3.1 ModularRandIter() [1/2]

```
ModularRandIter (
    const Ring & F,
    const size_t & size = 0,
    const uint64_t & seed = 0 ) [inline]
```

Constructor from ring, sampling size, and seed.

The random ring Element iterator works in the ring  $F$ , is seeded by  $seed$ , and it returns any one Element with probability no more than  $1/\min(F.\text{cardinality}())$ .  $size$  has no meaning in [ModularRandIter](#). A seed of zero means to use some arbitrary seed for the generator.

#### Parameters

$F$	LinBox ring archetype object in which to do arithmetic
$seed$	constant integer reference from which to seed random number generator (default = 0)

### 17.63.3.2 ModularRandIter() [2/2]

```
ModularRandIter (
    const ModularRandIter< Ring > & R ) [inline]
```

Copy constructor.

Constructs ALP\_randIter object by copying the random ring Element generator. This is required to allow generator objects to be passed by value into functions. In this implementation, this means copying the random ring Element generator to which  $R.\text{randIter\_ptr}$  points.

#### Parameters

$R$	ALP_randIter object.
-----	----------------------

### 17.63.3.3 ~ModularRandIter()

```
~ModularRandIter (
    void ) [inline]
```

Destructor.

This destructs the random ring Element generator object. In this implementation, this destroys the generator by deleting the random generator object to which `_randIter_ptr` points.

## 17.63.4 Member Function Documentation

### 17.63.4.1 operator=()

```
ModularRandIter< Ring > & operator= (
    const ModularRandIter< Ring > & R ) [inline]
```

Assignment operator.

Assigns ALP\_randIter object R to generator. In this implementation, this means copying the generator to which `R._randIter_ptr` points.

Parameters

<i>R</i>	ALP_randIter object.
----------	----------------------

### 17.63.4.2 operator>() [1/2]

```
Element & operator() (
    Element & elt ) const [inline]
```

Random ring Element creator with assignment.

This returns a random ring Element from the information supplied at the creation of the generator.

Returns

random ring Element

**17.63.4.3 random()** [1/2]

```
Element & random (
    Element & elt ) const [inline]
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

**17.63.4.4 operator>()** [2/2]

```
Element operator() ( ) const [inline]
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

**17.63.4.5 random()** [2/2]

```
Element random ( ) const [inline]
```

Ring Element type.

The ring Element must contain a default constructor, a copy constructor, a destructor, and an assignment operator.

The documentation for this class was generated from the following file:

- [givranditer.h \(4.1.1\)](#)

**17.64 Montgomery< int32\_t > Class Reference**

This class implements the standard arithmetic with Modulo Elements.

```
#include <montgomery-int32.h>
```

**17.64.1 Detailed Description**

This class implements the standard arithmetic with Modulo Elements.

Reduction is made through Montgomery's reduction. Representation of a is by storing (aB).

- We must have  $\gcd(p,2)=1$
- We must have  $(p-1)^2 + p(B-1) < B^2$ , i.e.  $2 < p \leq 40504$  for  $B = 2^{16}$ .
- m max is 40503
- p max is 40499

The documentation for this class was generated from the following files:

- [montgomery-int32.h \(4.1.1\)](#)
- [montgomery-int32.inl \(4.1.1\)](#)

## 17.65 Montgomery< RecInt::ruint< K > > Class Template Reference

The recint-based Montgomery ring.

```
#include <montgomery-ruint.h>
```

### 17.65.1 Detailed Description

```
template<size_t K>
class Givaro::Montgomery< RecInt::ruint< K > >
```

The recint-based Montgomery ring.

Only odd moduli allowed An integer  $(a \bmod p)$  is stored as  $(a * r \bmod 2^{\{2^K\}})$  with  $(r = 2^{\{2^K\}} \bmod p)$ .

The documentation for this class was generated from the following files:

- [montgomery-ruint.h \(4.1.1\)](#)
- [montgomery-ruint.inl \(4.1.1\)](#)

## 17.66 Neutral Class Reference

[Neutral](#) type.

```
#include <givbasictype.h>
```

Collaboration diagram for Neutral:

### 17.66.1 Detailed Description

[Neutral](#) type.

definition of zero and one

The documentation for this class was generated from the following files:

- [givbasictype.h \(4.1.1\)](#)
- [givbasictype.C \(4.1.1\)](#)

## 17.67 NewtonInterpGeom< Domain, REDUCE > Struct Template Reference

Newton.

```
#include <givinterpgeom.h>
```

Inherits [TruncDom< Domain >](#).

## Public Member Functions

- Rep & [setdegree](#) (Rep &P) const  
*Compute the degree of P.*
- size\_t & [sqrfree](#) (size\_t &Nfact, Rep \*Fact, const Rep &P) const  
*Sqrfree decomposition.*

### 17.67.1 Detailed Description

```
template<class Domain, bool REDUCE = true>
struct Givaro::NewtonInterpGeom< Domain, REDUCE >
```

Newton.

### 17.67.2 Member Function Documentation

#### 17.67.2.1 setdegree()

```
Poly1Dom< Domain, Dense >::Rep & setdegree (
    Rep & P ) const [inline], [inherited]
```

Compute the degree of P.

#### Warning

this is an infamous function that may not leave P constant !!

#### Parameters

$P$	polynomial
-----	------------

#### 17.67.2.2 sqrfree()

```
size_t & sqrfree (
    size_t & Nfact,
    Rep * Fact,
    const Rep & P ) const [inherited]
```

Sqrfree decomposition.

Decompose P such that:  $P = \text{Fact}[0]^0 * \text{Fact}[1]^1 * \dots * \text{Fact}[P.\text{degree}()]^{(P.\text{degree}())}$ , with Fact[0] the leading coefficient. The array Fact must be allocated before calling the function. The size of Fact must be degP+1 is all factors should be computed. For more readable version of the algorithm, see Geddes, p342.

## Parameters

<i>Nfact</i>	[in] the size of Fact
<i>Fact</i>	[in] an array of dimension Nfact
<i>Nfact</i>	[out] is the number of factor in the sqrfree decomposition
<i>Fact</i>	[out] contains at most Nfact factors of the decomposition.
<i>P</i>	rep.

The documentation for this struct was generated from the following file:

- [givinterpgeom.h \(4.1.1\)](#)

## 17.68 NewtonInterpGeomMultip< Domain, REDUCE > Struct Template Reference

Newton (multip)

```
#include <givinterpgeom-multip.h>
```

Inherits TruncDom< Domain >.

### Public Member Functions

- Rep & [setdegree](#) (Rep &P) const  
*Compute the degree of P.*
- size\_t & [sqrfree](#) (size\_t &Nfact, Rep \*Fact, const Rep &P) const  
*Sqrfree decomposition.*

### 17.68.1 Detailed Description

```
template<class Domain, bool REDUCE = true>
struct Givaro::NewtonInterpGeomMultip< Domain, REDUCE >
```

Newton (multip)

### 17.68.2 Member Function Documentation

#### 17.68.2.1 setdegree()

```
PolylDom< Domain, Dense >::Rep & setdegree (
    Rep & P ) const [inline], [inherited]
```

Compute the degree of P.

#### Warning

this is an infamous function that may not leave P constant !!

## Parameters

$P$	polynomial
-----	------------

**17.68.2.2 sqrfree()**

```
size_t & sqrfree (
    size_t & Nfact,
    Rep * Fact,
    const Rep & P ) const [inherited]
```

Sqrfree decomposition.

Decompose  $P$  such that:  $P = \text{Fact}[0]^0 * \text{Fact}[1]^1 * \dots * \text{Fact}[P.\text{degree}()]^{(P.\text{degree}())}$ , with  $\text{Fact}[0]$  the leading coefficient. The array  $\text{Fact}$  must be allocated before calling the function. The size of  $\text{Fact}$  must be  $\text{deg}P+1$  is all factors should be computed. For more readable version of the algorithm, see Geddes, p342.

## Parameters

<i>Nfact</i>	[in] the size of Fact
<i>Fact</i>	[in] an array of dimension Nfact
<i>Nfact</i>	[out] is the number of factor in the sqrfree decomposition
<i>Fact</i>	[out] contains at most Nfact factors of the decomposition.
<i>P</i>	rep.

The documentation for this struct was generated from the following file:

- [givinterpgeom-multip.h \(4.1.1\)](#)

**17.69 ObjectInit Class Reference**

[GivModule](#).

```
#include <givmodule.h>
```

**17.69.1 Detailed Description**

[GivModule](#).

Purpose: definition of object to be initialized after all modules initialization

The documentation for this class was generated from the following files:

- [givmodule.h \(4.1.1\)](#)
- [givmodule.C \(4.1.1\)](#)

## 17.70 OMPTimer Struct Reference

OMP timer.

```
#include <givomptimer.h>
```

### 17.70.1 Detailed Description

OMP timer.

The documentation for this struct was generated from the following file:

- [givomptimer.h \(4.1.1\)](#)

## 17.71 Pair< T1, T2 > Struct Template Reference

[Pair](#).

```
#include <givelem.h>
```

### 17.71.1 Detailed Description

```
template<class T1, class T2>  
struct Givaro::Pair< T1, T2 >
```

[Pair](#).

The documentation for this struct was generated from the following file:

- [givelem.h \(4.1.1\)](#)

## 17.72 Poly1CRT< Field > Class Template Reference

Poly1 CRT.

```
#include <givpoly1crt.h>
```

Collaboration diagram for Poly1CRT< Field >:

### 17.72.1 Detailed Description

```
template<class Field>
class Givaro::Poly1CRT< Field >
```

Poly1 CRT.

Examples

[examples/Polynomial/PolynomialCRT.C.](#)

The documentation for this class was generated from the following files:

- [givpoly1crt.h \(4.1.1\)](#)
- [givpoly1crtconvert.inl \(4.1.1\)](#)
- [givpoly1crtcstor.inl \(4.1.1\)](#)

## 17.73 Poly1Dom< Domain, Dense > Class Template Reference

Class Poly1Dom.

```
#include <givpolyldense.h>
```

Inheritance diagram for Poly1Dom< Domain, Dense >:

Collaboration diagram for Poly1Dom< Domain, Dense >:

### Public Member Functions

- Rep & [setdegree](#) (Rep &P) const  
*Compute the degree of P.*
- size\_t & [sqrfree](#) (size\_t &Nfact, Rep \*Fact, const Rep &P) const  
*Sqrfree decomposition.*

### 17.73.1 Detailed Description

```
template<class Domain>
class Givaro::Poly1Dom< Domain, Dense >
```

Class Poly1Dom.

### 17.73.2 Member Function Documentation

#### 17.73.2.1 setdegree()

```
Poly1Dom< Domain, Dense >::Rep & setdegree (
    Rep & P ) const [inline]
```

Compute the degree of P.

Warning

this is an infamous function that may not leave P constant !!

## Parameters

$P$	polynomial
-----	------------

**17.73.2.2 sqrfree()**

```
size_t & sqrfree (
    size_t & Nfact,
    Rep * Fact,
    const Rep & P ) const
```

Sqrfree decomposition.

Decompose  $P$  such that:  $P = \text{Fact}[0]^{\wedge}0 * \text{Fact}[1]^{\wedge}1 * \dots * \text{Fact}[P.\text{degree}()]^{\wedge}(P.\text{degree}())$ , with  $\text{Fact}[0]$  the leading coefficient. The array  $\text{Fact}$  must be allocated before calling the function. The size of  $\text{Fact}$  must be  $\text{deg}P+1$  is all factors should be computed. For more readable version of the algorithm, see Geddes, p342.

## Parameters

<i>Nfact</i>	[in] the size of <i>Fact</i>
<i>Fact</i>	[in] an array of dimension <i>Nfact</i>
<i>Nfact</i>	[out] is the number of factor in the sqrfree decomposition
<i>Fact</i>	[out] contains at most <i>Nfact</i> factors of the decomposition.
<i>P</i>	rep.

The documentation for this class was generated from the following files:

- [givpoly1dense.h \(4.1.1\)](#)
- [givpoly1addsub.inl \(4.1.1\)](#)
- [givpoly1axy.inl \(4.1.1\)](#)
- [givpoly1cstor.inl \(4.1.1\)](#)
- [givpoly1cyclo.inl \(4.1.1\)](#)
- [givpoly1gcd.inl \(4.1.1\)](#)
- [givpoly1io.inl \(4.1.1\)](#)
- [givpoly1kara.inl \(4.1.1\)](#)
- [givpoly1misc.inl \(4.1.1\)](#)
- [givpoly1muldiv.inl \(4.1.1\)](#)
- [givpoly1ratrecon.inl \(4.1.1\)](#)
- [givpoly1sqrfree.inl \(4.1.1\)](#)

## 17.74 Poly1FactorDom< Domain, Tag, RandomIterator > Class Template Reference

[Poly1FactorDom](#).

```
#include <givpoly1factor.h>
```

Inherits [Poly1Dom< Domain, StorageTag >](#).

Collaboration diagram for [Poly1FactorDom< Domain, Tag, RandomIterator >](#):

## Public Member Functions

- [Poly1FactorDom](#) (const Domain &d, const [Indeter](#) &X=[Indeter](#)(), const RandomIterator &g=RandomIterator())
- Element & **random\_irreducible** (Element &P, [Degree](#) n) const  
*random irreducible polynomial*
- Element & **creux\_random\_irreducible** (Element &P, [Degree](#) n) const  
*random irreducible polynomial tries to be sparse*
- Element & **ixe\_irreducible** (Element &R, [Degree](#) n) const  
*random irreducible polynomial with X as primitive root*
- Element & **ixe\_irreducible2** (Element &R, [Degree](#) n) const  
*random irreducible polynomial with X as primitive root*

### 17.74.1 Detailed Description

```
template<class Domain, class Tag = Dense, class RandomIterator = GivRandom>
class Givaro::Poly1FactorDom< Domain, Tag, RandomIterator >
```

[Poly1FactorDom](#).

#### Examples

[examples/Integer/ModularSquareRoot.C](#), [examples/Polynomial/isirred.C](#), [examples/Polynomial/isprimitive.C](#), [examples/Polynomial/pol\\_eval.C](#), and [examples/Polynomial/pol\\_factor.C](#).

### 17.74.2 Constructor & Destructor Documentation

#### 17.74.2.1 Poly1FactorDom()

```
Poly1FactorDom (
    const Domain & d,
    const Indeter & X = Indeter(),
    const RandomIterator & g = RandomIterator() ) [inline]
```

#### Warning

there is a copy of the random Iterator ...

The documentation for this class was generated from the following files:

- [givpoly1factor.h](#) (4.1.1)
- [givpoly1factor.inl](#) (4.1.1)
- [givpoly1proot.inl](#) (4.1.1)

## 17.75 Poly1PadicDom< Domain, Dense > Class Template Reference

Poly1 p-adic.

```
#include <givpoly1padic.h>
```

Inheritance diagram for Poly1PadicDom< Domain, Dense >:

Collaboration diagram for Poly1PadicDom< Domain, Dense >:

### Public Member Functions

- Rep & [setdegree](#) (Rep &P) const  
*Compute the degree of P.*
- size\_t & [sqrfree](#) (size\_t &Nfact, Rep \*Fact, const Rep &P) const  
*Sqrfree decomposition.*
- bool **isUnit** (const Rep &x) const  
*isUnit*
- bool **isDivisor** (const Element &a, const Element &b) const  
*isDivisor (a, b) Test if  $b \mid a$ .*

### 17.75.1 Detailed Description

```
template<class Domain>
class Givaro::Poly1PadicDom< Domain, Dense >
```

Poly1 p-adic.

### 17.75.2 Member Function Documentation

#### 17.75.2.1 setdegree()

```
Poly1Dom< Domain, Dense >::Rep & setdegree (
    Rep & P ) const [inline], [inherited]
```

Compute the degree of P.

#### Warning

this is an infamous function that may not leave  $P$  constant !!

#### Parameters

$P$	polynomial
-----	------------

### 17.75.2.2 sqrfree()

```
size_t & sqrfree (
    size_t & Nfact,
    Rep * Fact,
    const Rep & P ) const [inherited]
```

Sqrfree decomposition.

Decompose  $P$  such that:  $P = \text{Fact}[0]^0 * \text{Fact}[1]^1 * \dots * \text{Fact}[P.\text{degree}()]^{(P.\text{degree}())}$ , with  $\text{Fact}[0]$  the leading coefficient. The array  $\text{Fact}$  must be allocated before calling the function. The size of  $\text{Fact}$  must be  $\text{deg}P+1$  if all factors should be computed. For more readable version of the algorithm, see Geddes, p342.

#### Parameters

<i>Nfact</i>	[in] the size of <i>Fact</i>
<i>Fact</i>	[in] an array of dimension <i>Nfact</i>
<i>Nfact</i>	[out] is the number of factor in the sqrfree decomposition
<i>Fact</i>	[out] contains at most <i>Nfact</i> factors of the decomposition.
<i>P</i>	rep.

The documentation for this class was generated from the following file:

- [givpoly1padic.h \(4.1.1\)](#)

## 17.76 Primes16 Class Reference

class [Primes16](#)

```
#include <givprimes16.h>
```

### 17.76.1 Detailed Description

class [Primes16](#)

The documentation for this class was generated from the following files:

- [givprimes16.h \(4.1.1\)](#)
- [givprimes16.C \(4.1.1\)](#)

## 17.77 QField< Rational > Class Reference

[Rational](#) Domain.

```
#include <givrational.h>
```

Inherits [FieldInterface< \\_Element >](#).

Collaboration diagram for [QField< Rational >](#):

### 17.77.1 Detailed Description

[Rational](#) Domain.

The documentation for this class was generated from the following file:

- [givrational.h \(4.1.1\)](#)

## 17.78 RandomIntegerIterator< \_Unsigned, \_Exact\_Size > Class Template Reference

Random [Integer](#) Iterator.

```
#include <random-integer.h>
```

Collaboration diagram for RandomIntegerIterator< \_Unsigned, \_Exact\_Size >:

### Public Member Functions

- [RandomIntegerIterator](#) (const [Integer\\_Domain](#) &D, size\_t bits=30, uint64\_t seed=0)  
*Constructor.*
- [RandomIntegerIterator](#) (const [RandomIntegerIterator](#) &R)  
*copy constructor.*
- [RandomIntegerIterator](#) & operator= (const [RandomIntegerIterator](#) &R)  
*copy.*
- [RandomIntegerIterator](#) & operator++ ()  
*operator++() creates a new random integer.*
- const [Integer\\_Type](#) & operator\* () const  
*get the random integer.*
- const [Integer\\_Type](#) & randomInteger () const  
*get the random integer.*

### Static Public Member Functions

- static void [setSeed](#) (uint64\_t ul)  
*Sets the seed.*

### Protected Attributes

- size\_t [\\_bits](#)  
*common length of all integers*
- [Integer\\_Type](#) [\\_integer](#)  
*the generated integer.*

### 17.78.1 Detailed Description

```
template<bool _Unsigned = true, bool _Exact_Size = false>
class Givaro::RandomIntegerIterator< _Unsigned, _Exact_Size >
```

Random [Integer](#) Iterator.

Generates integers of specified length.

## Template Parameters

<i>_Unsigned</i>	if <code>true</code> , then only non negative integers are generated, if <code>false</code> , their sign is random.
<i>_Exact_Size</i>	if <code>true</code> , then random integers have exactly the number of required bits, if <code>false</code> , they have less than the required number of bits

## 17.78.2 Constructor &amp; Destructor Documentation

## 17.78.2.1 RandomIntegerIterator() [1/2]

```
RandomIntegerIterator (
    const Integer_Domain & D,
    size_t bits = 30,
    uint64_t seed = 0 ) [inline]
```

Constructor.

## Parameters

<i>bits</i>	size of integers (in bits)
<i>seed</i>	if 0 a seed will be generated, otherwise, the provided seed will be use.

## 17.78.2.2 RandomIntegerIterator() [2/2]

```
RandomIntegerIterator (
    const RandomIntegerIterator< _Unsigned, _Exact_Size > & R ) [inline]
```

copy constructor.

## Parameters

<i>R</i>	random iterator to be copied.
----------	-------------------------------

## 17.78.3 Member Function Documentation

## 17.78.3.1 operator=()

```
RandomIntegerIterator & operator= (
    const RandomIntegerIterator< _Unsigned, _Exact_Size > & R ) [inline]
```

copy.

## Parameters

<i>R</i>	random iterator to be copied.
----------	-------------------------------

**17.78.3.2 operator\*()**

```
const Integer_Type & operator* ( ) const [inline]
```

get the random integer.

returns the actual integer.

**17.78.3.3 randomInteger()**

```
const Integer_Type & randomInteger ( ) const [inline]
```

get the random integer.

returns the actual integer.

## Warning

a new integer is not generated.

**17.78.3.4 setSeed()**

```
static void setSeed (
    uint64_t ul ) [inline], [static]
```

Sets the seed.

Set the random seed to be ul.

## Parameters

<i>ul</i>	the new seed.
-----------	---------------

The documentation for this class was generated from the following file:

- random-integer.h (4.1.1)

**17.79 Rational Class Reference**

Rationals. No doc.

```
#include <givrational.h>
```

Collaboration diagram for Rational:

## Public Member Functions

- [Rational](#) (const [Integer](#) &f, const [Integer](#) &m, const [Integer](#) &k, bool recurs=false)  
*[Rational](#) number reconstruction.*

## 17.79.1 Detailed Description

Rationals. No doc.

### Examples

[examples/Rational/iratrecon.C](#).

## 17.79.2 Constructor & Destructor Documentation

### 17.79.2.1 Rational()

```
Rational (
    const Integer & f,
    const Integer & m,
    const Integer & k,
    bool recurs = false )
```

[Rational](#) number reconstruction.

$num/den \equiv f \pmod{m}$ , with  $|num| < k$  and  $0 < |den| \leq f/kf$

**Bibliography**    • von zur Gathen & Gerhard *Modern Computer Algebra*, 5.10, Cambridge Univ. Press 1999]

The documentation for this class was generated from the following files:

- [givrational.h](#) (4.1.1)
- [givataddsub.C](#) (4.1.1)
- [givatcpy.C](#) (4.1.1)
- [givatcstor.C](#) (4.1.1)
- [givratio.C](#) (4.1.1)
- [givrational.inl](#) (4.1.1)
- [givatmisc.C](#) (4.1.1)
- [givatmuldiv.C](#) (4.1.1)
- [givatreconstruct.C](#) (4.1.1)

## 17.80 RealTimer Class Reference

Real timer.

```
#include <givtimer.h>
```

Inheritance diagram for RealTimer:

Collaboration diagram for RealTimer:

### 17.80.1 Detailed Description

Real timer.

The documentation for this class was generated from the following files:

- [givtimer.h \(4.1.1\)](#)
- [givtimer.C \(4.1.1\)](#)

## 17.81 RefCounter Class Reference

Ref counter.

```
#include <givref_count.h>
```

### 17.81.1 Detailed Description

Ref counter.

The documentation for this class was generated from the following file:

- [givref\\_count.h \(4.1.1\)](#)

## 17.82 RefCountPtr< T > Class Template Reference

RefCount Pointer.

```
#include <givpointer.h>
```

### 17.82.1 Detailed Description

```
template<class T>  
class Givaro::RefCountPtr< T >
```

RefCount Pointer.

The documentation for this class was generated from the following file:

- [givpointer.h \(4.1.1\)](#)

## 17.83 RNSsystem< RING, Domain > Class Template Reference

class [RNSsystem](#).

```
#include <givrnns.h>
```

Collaboration diagram for RNSsystem< RING, Domain >:

### Public Member Functions

- const [array](#) & **Reciprocals** () const  
*– Returns an array of the reciprocal  $ck = \left( \prod_{j=0..k-1} p_j \right)^{-1} \bmod pk$*

### 17.83.1 Detailed Description

```
template<class RING, class Domain>
class Givaro::RNSsystem< RING, Domain >
```

class [RNSsystem](#).

Structure which manages list of domains in order to convert integer to/from RNS number system using a mixed radix form. This class is parameterized by the type of `RING` and of `Domain`. The ring should have:

- Ring( Modulo ) and Domain.init(Ring) conversions
- operator \*= (Ring&, const Modulo&)
- operator += (Ring&, const Modulo&)

The documentation for this class was generated from the following files:

- [givrnns.h](#) (4.1.1)
- [givrnconvert.inl](#) (4.1.1)
- [givrnscstor.inl](#) (4.1.1)

## 17.84 RNSsystemFixed< Ints > Class Template Reference

NO DOC.

```
#include <givrnnsfixed.h>
```

Collaboration diagram for RNSsystemFixed< Ints >:

### 17.84.1 Detailed Description

```
template<class Ints>
class Givaro::RNSsystemFixed< Ints >
```

NO DOC.

The documentation for this class was generated from the following files:

- [givrnsgfixed.h \(4.1.1\)](#)
- [givrnsgfixed.inl \(4.1.1\)](#)

## 17.85 Stack< **THING** > Class Template Reference

[Stack.](#)

```
#include <givstack.h>
```

### 17.85.1 Detailed Description

```
template<class THING>
class Givaro::Stack< THING >
```

[Stack.](#)

The documentation for this class was generated from the following files:

- [givstack.h \(4.1.1\)](#)
- [givstack.inl \(4.1.1\)](#)

## 17.86 StaticElement< **DomainStyle** > Struct Template Reference

Static Element.

```
#include <StaticElement.h>
```

### 17.86.1 Detailed Description

```
template<class DomainStyle>
struct Givaro::StaticElement< DomainStyle >
```

Static Element.

Examples

[examples/FiniteField/all\\_field.C.](#)

The documentation for this struct was generated from the following files:

- [StaticElement.h \(4.1.1\)](#)
- [all\\_field.C \(4.1.1\)](#)

## 17.87 SysTimer Class Reference

Sys timer.

```
#include <givtimer.h>
```

Inheritance diagram for SysTimer:

Collaboration diagram for SysTimer:

### 17.87.1 Detailed Description

Sys timer.

The documentation for this class was generated from the following files:

- [givtimer.h \(4.1.1\)](#)
- [givtimer.C \(4.1.1\)](#)

## 17.88 Timer Class Reference

[Timer](#).

```
#include <givtimer.h>
```

### Public Member Functions

- void [clear](#) ()  
*Clear timer.*
- void [start](#) ()  
*Start timer.*
- void [stop](#) ()  
*Stop timer.*
- double [usertime](#) () const  
*total amount of second spent in user mode.*
- double [systime](#) () const  
*total amount of second spent in system mode.*
- double [realtime](#) () const  
*real total amount of second spent.*
- double [userElapsedTime](#) ()  
*User mode time spent since start.*
- double [sysElapsedTime](#) ()  
*System mode time spent since start.*
- double [realElapsedTime](#) ()  
*real total amount of second spent since start.*

## 17.88.1 Detailed Description

Timer.

### Examples

examples/FiniteField/gfq\_atomic.C, examples/FiniteField/zpz\_atomic.C, examples/Integer/ModularSquareRoot.C, examples/Integer/ProbLucas.C, examples/Integer/RSA\_breaking.C, examples/Integer/RSA\_decipher.C, examples/Integer/RSA\_encipher.C, examples/Integer/RSA\_keys\_generator.C, examples/Integer/iexponentiation.C, examples/Integer/ifactor.C, examples/Integer/ifactor\_lenstra.C, examples/Integer/igcd.C, examples/Integer/igcdext.C, examples/Integer/ilcm.C, examples/Integer/ispower.C, examples/Integer/isprime.C, examples/Integer/isproot.C, examples/Integer/lambda.C, examples/Integer/lambda\_inv.C, examples/Integer/nb\_primes.C, examples/Integer/nextprime.C, examples/Integer/order.C, examples/Integer/phi.C, examples/Integer/prevprime.C, examples/Integer/primitiveelement.C, examples/Integer/primitiveroot.C, examples/Integer/probable\_primroot.C, examples/Polynomial/PolynomialCRT.C, examples/Polynomial/highorder.C, examples/Polynomial/interpolate.C, examples/Polynomial/isirred.C, examples/Polynomial/isprimitive.C, examples/Polynomial/pol\_eval.C, examples/Polynomial/pol\_factor.C, and examples/Rational/iratrecon.C.

## 17.88.2 Member Function Documentation

### 17.88.2.1 clear()

```
void clear ( )
```

Clear timer.

Everything reset to 0. This need not be called before the first start since the constructor does it.

### Examples

examples/FiniteField/gfq\_atomic.C, examples/FiniteField/zpz\_atomic.C, examples/Integer/ProbLucas.C, examples/Integer/RSA\_breaking.C, examples/Integer/RSA\_decipher.C, examples/Integer/RSA\_encipher.C, examples/Integer/RSA\_keys\_generator.C, examples/Integer/iexponentiation.C, examples/Integer/ifactor.C, examples/Integer/ifactor\_lenstra.C, examples/Integer/igcd.C, examples/Integer/igcdext.C, examples/Integer/ilcm.C, examples/Integer/ispower.C, examples/Integer/isprime.C, examples/Integer/isproot.C, examples/Integer/lambda.C, examples/Integer/lambda\_inv.C, examples/Integer/nb\_primes.C, examples/Integer/nextprime.C, examples/Integer/order.C, examples/Integer/phi.C, examples/Integer/prevprime.C, examples/Integer/primitiveelement.C, examples/Integer/primitiveroot.C, examples/Integer/probable\_primroot.C, examples/Polynomial/PolynomialCRT.C, examples/Polynomial/highorder.C, examples/Polynomial/interpolate.C, examples/Polynomial/isirred.C, examples/Polynomial/isprimitive.C, examples/Polynomial/pol\_eval.C, examples/Polynomial/pol\_factor.C, and examples/Rational/iratrecon.C.

**17.88.2.2 start()**

```
void start ( )
```

Start timer.

Starts the timer. If called after another `start()` or a `stop()`, it sets the timer to a totally fresh new start.

**Examples**

[examples/FiniteField/gfq\\_atomic.C](#), [examples/FiniteField/zpz\\_atomic.C](#), [examples/Integer/ModularSquareRoot.C](#), [examples/Integer/ProbLucas.C](#), [examples/Integer/RSA\\_breaking.C](#), [examples/Integer/RSA\\_decipher.C](#), [examples/Integer/RSA\\_encipher.C](#), [examples/Integer/RSA\\_keys\\_generator.C](#), [examples/Integer/iexponentiation.C](#), [examples/Integer/ifactor.C](#), [examples/Integer/ifactor\\_lenstra.C](#), [examples/Integer/igcd.C](#), [examples/Integer/igcdext.C](#), [examples/Integer/ilcm.C](#), [examples/Integer/ispower.C](#), [examples/Integer/isprime.C](#), [examples/Integer/isproot.C](#), [examples/Integer/lambda.C](#), [examples/Integer/lambda\\_inv.C](#), [examples/Integer/nb\\_primes.C](#), [examples/Integer/nextprime.C](#), [examples/Integer/order.C](#), [examples/Integer/phi.C](#), [examples/Integer/prevprime.C](#), [examples/Integer/primitiveelement.C](#), [examples/Integer/primitiveroot.C](#), [examples/Integer/probable\\_primroot.C](#), [examples/Polynomial/PolynomialCRT.C](#), [examples/Polynomial/highorder.C](#), [examples/Polynomial/interpolate.C](#), [examples/Polynomial/isirred.C](#), [examples/Polynomial/isprimitive.C](#), [examples/Polynomial/pol\\_eval.C](#), [examples/Polynomial/pol\\_factor.C](#), and [examples/Rational/iratrecon.C](#).

**17.88.2.3 stop()**

```
void stop ( )
```

Stop timer.

Stops the timer. The time since the previous `start()` is stored. If called again, `stop()` will store the time since the previous `start()` again, acting as a `pause()`.

**Precondition**

`start()` should have been called before...

**Examples**

[examples/FiniteField/gfq\\_atomic.C](#), [examples/FiniteField/zpz\\_atomic.C](#), [examples/Integer/ModularSquareRoot.C](#), [examples/Integer/ProbLucas.C](#), [examples/Integer/RSA\\_breaking.C](#), [examples/Integer/RSA\\_decipher.C](#), [examples/Integer/RSA\\_encipher.C](#), [examples/Integer/RSA\\_keys\\_generator.C](#), [examples/Integer/iexponentiation.C](#), [examples/Integer/ifactor.C](#), [examples/Integer/ifactor\\_lenstra.C](#), [examples/Integer/igcd.C](#), [examples/Integer/igcdext.C](#), [examples/Integer/ilcm.C](#), [examples/Integer/ispower.C](#), [examples/Integer/isprime.C](#), [examples/Integer/isproot.C](#), [examples/Integer/lambda.C](#), [examples/Integer/lambda\\_inv.C](#), [examples/Integer/nb\\_primes.C](#), [examples/Integer/nextprime.C](#), [examples/Integer/order.C](#), [examples/Integer/phi.C](#), [examples/Integer/prevprime.C](#), [examples/Integer/primitiveelement.C](#), [examples/Integer/primitiveroot.C](#), [examples/Integer/probable\\_primroot.C](#), [examples/Polynomial/PolynomialCRT.C](#), [examples/Polynomial/highorder.C](#), [examples/Polynomial/interpolate.C](#), [examples/Polynomial/isirred.C](#), [examples/Polynomial/isprimitive.C](#), [examples/Polynomial/pol\\_eval.C](#), [examples/Polynomial/pol\\_factor.C](#), and [examples/Rational/iratrecon.C](#).

#### 17.88.2.4 usertime()

```
double usertime ( ) const [inline]
```

total amount of second spent in user mode.

##### Returns

the user time elapsed between the latest `start()` and the latest `stop()`.

##### Precondition

`stop()` is called before.

##### Examples

[examples/FiniteField/gfq\\_atomic.C](#), [examples/FiniteField/zpz\\_atomic.C](#), and [examples/Polynomial/pol\\_factor.C](#).

#### 17.88.2.5 systime()

```
double systime ( ) const [inline]
```

total amount of second spent in system mode.

##### Returns

the system time elapsed between the latest `start()` and the latest `stop()`.

##### Precondition

`stop()` is called before.

#### 17.88.2.6 realtime()

```
double realtime ( ) const [inline]
```

real total amount of second spent.

##### Returns

the real total time elapsed between the latest `start()` and the latest `stop()`.

##### Precondition

`stop()` is called before.

### 17.88.2.7 userElapsedTime()

```
double userElapsedTime ( ) [inline]
```

User mode time spent since start.

A call to `stop()` is useless.

#### Returns

elapsed time (in seconds) since `start()` in user mode.

### 17.88.2.8 sysElapsedTime()

```
double sysElapsedTime ( ) [inline]
```

System mode time spent since start.

A call to `stop()` is useless.

#### Returns

elapsed time (in seconds) since `start()` in system mode.

### 17.88.2.9 realElapsedTime()

```
double realElapsedTime ( ) [inline]
```

real total amount of second spent since start.

A call to `stop()` is useless.

#### Returns

elapsed time (in seconds) since `start()`.

The documentation for this class was generated from the following files:

- [givtimer.h \(4.1.1\)](#)
- [givtimer.C \(4.1.1\)](#)

## 17.89 UserTimer Class Reference

User timer.

```
#include <givtimer.h>
```

Inheritance diagram for UserTimer:

Collaboration diagram for UserTimer:

### 17.89.1 Detailed Description

User timer.

The documentation for this class was generated from the following files:

- [givtimer.h \(4.1.1\)](#)
- [givtimer.C \(4.1.1\)](#)

## 17.90 VectorDom< Domain, StorageTag > Class Template Reference

VectorDom<Domain,StorageTag>

```
#include <givvector.h>
```

### 17.90.1 Detailed Description

```
template<class Domain, class StorageTag>  
class Givaro::VectorDom< Domain, StorageTag >
```

VectorDom<Domain,StorageTag>

The documentation for this class was generated from the following file:

- [givvector.h \(4.1.1\)](#)

## 17.91 ZRing< \_Element > Class Template Reference

Class [ZRing](#).

```
#include <zring.h>
```

Inherits UnparametricOperations< \_Element >.

## Public Member Functions

### Comparison Predicates

- bool **areEqual** (const Element &x, const Element &y) const  
 $x == y$

### Arithmetic Operations

*The first argument is set and is also the return value.*

- Element & **add** (Element &x, const Element &y, const Element &z) const  
 $x := y + z$
- Element & **sub** (Element &x, const Element &y, const Element &z) const  
 $x := y - z$
- Element & **mul** (Element &x, const Element &y, const Element &z) const  
 $x := y * z$
- Element & **div** (Element &x, const Element &y, const Element &z) const  
 $x := y / z$
- Element & **mod** (Element &x, const Element &y, const Element &z) const  
 $x := y \bmod z$
- Element & **neg** (Element &x, const Element &y) const  
 $x := -y$
- Element & **inv** (Element &x, const Element &y) const  
 $x := 1/y$
- Element & **axpy** (Element &z, const Element &a, const Element &x, const Element &y) const  
 $z := a * x + y$
- Element & **axpyin** (Element &z, const Element &a, const Element &x) const  
 $z := a * x + z$
- Element & **axmy** (Element &z, const Element &a, const Element &x, const Element &y) const  
 $z := a * x - y$
- Element & **axmyin** (Element &z, const Element &a, const Element &x) const  
 $z := a * x - z$
- Element & **maxpy** (Element &z, const Element &a, const Element &x, const Element &y) const  
 $z := y - a * x$
- Element & **maxpyin** (Element &z, const Element &a, const Element &x) const  
 $z := z - a * x$

### Inplace Arithmetic Operations

*The first argument is modified and the result is the return value.*

- Element & **addin** (Element &x, const Element &y) const  
 $x := x + y$
- Element & **subin** (Element &x, const Element &y) const  
 $x := x - y$
- Element & **mulin** (Element &x, const Element &y) const  
 $x := x * y$
- Element & **divin** (Element &x, const Element &y) const  
 $x := x / y$
- Element & **modin** (Element &x, const Element &y) const  
 $x := x \bmod y$
- Element & **negin** (Element &x) const  
 $x := -x$
- Element & **invin** (Element &x) const  
 $x := 1/x$

### Input/Output Operations

- std::ostream & **write** (std::ostream &os, std::string F) const  
*Print field.*
- std::istream & **read** (std::istream &is) const  
*Read field.*

## Protected Member Functions

- void [RationalReconstruction](#) (Element &a, Element &b, const Element &f, const Element &m, const Element &k, bool forcedreduce, bool recursive) const  
[Rational](#) number reconstruction.

### 17.91.1 Detailed Description

```
template<class _Element>
class Givaro::ZRing<_Element>
```

Class [ZRing](#).

Ring of integers, using the `_Element` base type.

### 17.91.2 Member Function Documentation

#### 17.91.2.1 RationalReconstruction()

```
void RationalReconstruction (
    Element & a,
    Element & b,
    const Element & f,
    const Element & m,
    const Element & k,
    bool forcedreduce,
    bool recursive ) const [inline], [protected]
```

[Rational](#) number reconstruction.

$\frac{n}{d} \equiv f \pmod{m}$ , with  $|n| < k$  and  $0 < |d| \leq \frac{f}{k}$ .

**Bibliography** • von zur Gathen & Gerhard, *Modern Computer Algebra*, 5.10, Cambridge Univ. Press 1999

#### 17.91.2.2 write()

```
std::ostream & write (
    std::ostream & os,
    std::string F ) const [inline], [inherited]
```

Print field.

#### Returns

output stream to which field is written.

**Parameters**

<i>os</i>	output stream to which field is written.
-----------	--

**17.91.2.3 read()**

```
std::istream & read (  
    std::istream & is ) const [inline], [inherited]
```

Read field.

**Returns**

input stream from which field is read.

**Parameters**

<i>is</i>	input stream from which field is read.
-----------	--

The documentation for this class was generated from the following files:

- random-integer.h (4.1.1)
- zring.h (4.1.1)

## Chapter 18

# File Documentation

### 18.1 all\_field.C File Reference

```
#include <iostream>
#include <givaro/gfq.h>
#include <givaro/montgomery.h>
#include <givaro/modular.h>
#include <givaro/StaticElement.h>
```

Include dependency graph for all\_field.C:

#### Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.2 domain\_to\_operatorstyle.C File Reference

```
#include <iostream>
#include <givaro/gfq.h>
#include <givaro/StaticElement.h>
```

Include dependency graph for domain\_to\_operatorstyle.C:

#### Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.3 exponentiation.C File Reference

```
#include <iostream>
#include <givaro/givpower.h>
#include <givaro/modular.h>
#include <givaro/gfq.h>
```

Include dependency graph for exponentiation.C:

## 18.4 ff\_arith.C File Reference

```
#include <iostream>
#include <givaro/modular.h>
#include <givaro/montgomery.h>
#include <givaro/gfq.h>
#include <givaro/gfqext.h>
#include <sys/time.h>
#include <sys/resource.h>
Include dependency graph for ff_arith.C:
```

## 18.5 GF128.C File Reference

```
#include <givaro/gfq.h>
#include <givaro/givtimer.h>
Include dependency graph for GF128.C:
```

## 18.6 GFirreducible.C File Reference

```
#include <givaro/gfq.h>
#include <givaro/givpower.h>
#include <givaro/givtimer.h>
Include dependency graph for GFirreducible.C:
```

## 18.7 gfq\_atomic.C File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <givaro/givtimer.h>
#include <givaro/givrandom.h>
#include <givaro/gfq.h>
Include dependency graph for gfq_atomic.C:
```

## 18.8 Test\_Extension.C File Reference

```
#include "givaro/givpoly1.h"
#include "givaro/extension.h"
Include dependency graph for Test_Extension.C:
```

## 18.9 zpz\_atomic.C File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <givaro/givtimer.h>
#include <givaro/givrandom.h>
#include <givaro/modular.h>
Include dependency graph for zpz_atomic.C:
```

## 18.10 iexponentiation.C File Reference

```
#include <iostream>
#include <givaro/modular-integer.h>
#include <givaro/givpower.h>
#include <givaro/givtimer.h>
Include dependency graph for iexponentiation.C:
```

## 18.11 ifactor.C File Reference

```
#include <iostream>
#include <givaro/givinit.h>
#include <givaro/givintfactor.h>
#include <givaro/givtimer.h>
Include dependency graph for ifactor.C:
```

## 18.12 ifactor\_lenstra.C File Reference

```
#include <iostream>
#include <givaro/givinit.h>
#include <givaro/givintfactor.h>
#include <givaro/givtimer.h>
Include dependency graph for ifactor_lenstra.C:
```

## 18.13 igcd.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
Include dependency graph for igcd.C:
```

## 18.14 igcdext.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
Include dependency graph for igcdext.C:
```

## 18.15 ilcm.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
Include dependency graph for ilcm.C:
```

## 18.16 ispower.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givinteger.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
Include dependency graph for ispower.C:
```

## 18.17 isroot.C File Reference

```
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
Include dependency graph for isroot.C:
```

## 18.18 lambda.C File Reference

```
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
Include dependency graph for lambda.C:
```

## 18.19 lambda\_inv.C File Reference

```
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
Include dependency graph for lambda_inv.C:
```

## 18.20 ModularSquareRoot.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintsqrootmod.h>
#include <givaro/givtimer.h>
#include <givaro/givpolylfactor.h>
#include <givaro/modular-integer.h>
Include dependency graph for ModularSquareRoot.C:
```

## 18.21 nb\_primes.C File Reference

```
#include <iostream>
#include "givaro/givintprime.h"
#include "givaro/givtimer.h"
Include dependency graph for nb_primes.C:
```

## 18.22 nextprime.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
Include dependency graph for nextprime.C:
```

## 18.23 order.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
Include dependency graph for order.C:
```

## 18.24 phi.C File Reference

```
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
Include dependency graph for phi.C:
```

## 18.25 prevprime.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
Include dependency graph for prevprime.C:
```

## 18.26 primitiveelement.C File Reference

```
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
Include dependency graph for primitiveelement.C:
```

## 18.27 primitiveroot.C File Reference

```
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
Include dependency graph for primitiveroot.C:
```

## 18.28 probable\_primroot.C File Reference

```
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
#include <cmath>
Include dependency graph for probable_primroot.C:
```

## 18.29 ProbLucas.C File Reference

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
#include <cmath>
#include <givaro/givintprime.h>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
#include <givaro/modular-integer.h>
#include <givaro/givinteger.h>
#include <givaro/givrandom.h>
Include dependency graph for ProbLucas.C:
```

## 18.30 RSA\_breaking.C File Reference

```
#include <iostream>
#include "givaro/givintrsa.h"
#include "givaro/givtimer.h"
Include dependency graph for RSA_breaking.C:
```

## 18.31 RSA\_decipher.C File Reference

```
#include <iostream>
#include <fstream>
#include "givaro/givintrsa.h"
#include "givaro/givrandom.h"
#include "givaro/givtimer.h"
Include dependency graph for RSA_decipher.C:
```

## 18.32 RSA\_encipher.C File Reference

```
#include <iostream>
#include <fstream>
#include "givaro/givintrsa.h"
#include "givaro/givrandom.h"
#include "givaro/givtimer.h"
Include dependency graph for RSA_encipher.C:
```

## 18.33 RSA\_keys\_generator.C File Reference

```
#include <iostream>
#include "givaro/givintrsa.h"
#include "givaro/givrandom.h"
#include "givaro/givtimer.h"
Include dependency graph for RSA_keys_generator.C:
```

## 18.34 highorder.C File Reference

```
#include <iostream>
#include <givaro/givrandom.h>
#include <givaro/givtimer.h>
#include <givaro/gfq.h>
#include <givaro/givpoly1.h>
#include <givaro/givtruncdomain.h>
#include <givaro/givhighorder.h>
Include dependency graph for highorder.C:
```

## 18.35 interpolate.C File Reference

```
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <givaro/modular.h>
#include <givaro/givpoly1factor.h>
#include <givaro/givtimer.h>
#include <givaro/givinterp.h>
Include dependency graph for interpolate.C:
```

## 18.36 isirred.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpoly1factor.h>
#include <givaro/givtimer.h>
Include dependency graph for isirred.C:
```

## 18.37 isprimitive.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpolylfactor.h>
#include <givaro/givtimer.h>
```

Include dependency graph for isprimitive.C:

## 18.38 pol\_arith.C File Reference

```
#include <iostream>
#include <givaro/gfq.h>
#include <givaro/givpoly1.h>
```

Include dependency graph for pol\_arith.C:

## 18.39 pol\_eval.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpolylfactor.h>
#include <givaro/givtimer.h>
```

Include dependency graph for pol\_eval.C:

## 18.40 pol\_factor.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpolylfactor.h>
#include <givaro/givtimer.h>
```

Include dependency graph for pol\_factor.C:

## 18.41 PolynomialCRT.C File Reference

```
#include <iostream>
#include <algorithm>
#include <givaro/givtimer.h>
#include <givaro/givpolylcrt.h>
#include <givaro/givintprime.h>
#include <givaro/montgomery.h>
#include <givaro/extension.h>
#include <givaro/modular.h>
#include <givaro/gfq.h>
#include <givaro/chineseremainder.h>
#include <givaro/givrns.h>
#include <givaro/givrandom.h>
#include <givaro/givrational.h>
```

Include dependency graph for PolynomialCRT.C:

## 18.42 trunc\_arith.C File Reference

```
#include <iostream>
#include <givaro/givrandom.h>
#include <givaro/givtimer.h>
#include <givaro/gfq.h>
#include <givaro/givpoly1.h>
#include <givaro/givtruncdomain.h>
Include dependency graph for trunc_arith.C:
```

## 18.43 iratrecon.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givrational.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
Include dependency graph for iratrecon.C:
```

## 18.44 polydouble.C File Reference

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givpoly1.h>
#include <givaro/givrational.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
#include <givaro/givprint.h>
Include dependency graph for polydouble.C:
```

## 18.45 givarray0.h File Reference

Array of type T with reference mechanism.

```
#include <stddef.h>
#include "givaro/givaromm.h"
#include "givaro/givperf.h"
#include "givaro/giverror.h"
#include "givaro/givarray0.inl"
Include dependency graph for givarray0.h:
```

### Data Structures

- struct [\\_perfArray0< T >](#)  
defined by marco GIVARO\_PERF\_DEFCLASS. ref counting and stuff.
- class [Array0< T >](#)  
NODOC.

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.45.1 Detailed Description

Array of type T with reference mechanism.

## 18.46 givarrayallocator.h File Reference

NO DOC.

## Data Structures

- class [ArrayAllocatort< T, Tag >](#)  
*ArrayAllocator: class for allocation of arrays.*
- class [Array0Tag](#)  
*Array0Tag.*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.46.1 Detailed Description

NO DOC.

## 18.47 givarrayfixed.h File Reference

ArrayFixed of type T with fixed dimension.

```
#include <stddef.h>
#include "givaro/givaromm.h"
#include "givaro/giverror.h"
#include "givaro/givperf.h"
Include dependency graph for givarrayfixed.h:
```

## Data Structures

- class [ArrayFixed< T, SIZE >](#)  
*ArrayFixed.*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.47.1 Detailed Description

ArrayFixed of type T with fixed dimension.

## 18.48 givbits.h File Reference

field of n bits, for any n

```
#include <iostream>
#include "givaro/givaromm.h"
#include "givaro/givararray0.h"
#include "givaro/givbits.inl"
Include dependency graph for givbits.h:
```

## Data Structures

- class [Bits](#)  
*[Bits](#).*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.48.1 Detailed Description

field of n bits, for any n

## 18.49 givelem.h File Reference

definition of a reference to an object.

## Data Structures

- struct [ElemRef< T >](#)  
*[Elem Ref](#).*
- struct [ElemConstRef< T >](#)  
*[Elem const Ref](#).*
- struct [Pair< T1, T2 >](#)  
*[Pair](#).*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- `template<class T1 , class T2 >`  
`std::ostream & operator<< (std::ostream &o, const Pair< T1, T2 > &p)`  
*IO.*
- `template<class T1 , class T2 >`  
`std::istream & operator>> (std::istream &fin, Pair< T1, T2 > &p)`  
*IO.*

### 18.49.1 Detailed Description

definition of a reference to an object.

## 18.50 givhashtable.h File Reference

hash table

```
#include "givaro/givhashtable.inl"
Include dependency graph for givhashtable.h:
```

## Data Structures

- class [Key< T >](#)  
*The class [Key](#).*
- class [HashTable< T, Key >](#)  
*Hash table.*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.50.1 Detailed Description

hash table

## 18.51 givlist0.h File Reference

List of type T with double link and various insert/get/rmv method.

```
#include "givaro/givbasictype.h"
#include "givaro/giverror.h"
#include "givaro/givlist0.inl"
Include dependency graph for givlist0.h:
```

### Data Structures

- class [List0< T >](#)  
*ListO.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

#### 18.51.1 Detailed Description

List of type T with double link and various insert/get/rmv method.

Used reference counting on each node of the list.

## 18.52 givstack.h File Reference

no doc.

```
#include "givaro/givstack.inl"
Include dependency graph for givstack.h:
```

### Data Structures

- class [Stack< THING >](#)  
*Stack.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

#### 18.52.1 Detailed Description

no doc.

## 18.53 chineseremainder.h File Reference

Chinese Remainder Algorithm for 2 Elements.

```
#include "givaro/givconfig.h"
#include "givaro/giverror.h"
Include dependency graph for chineseremainder.h:
```

### Data Structures

- struct [ChineseRemainder< Ring, Domain, REDUCE >](#)  
*CRA.*
- struct [ChineseRemainder< Ring, Domain, false >](#)  
*CRA2.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.53.1 Detailed Description

Chinese Remainder Algorithm for 2 Elements.

See also

For any number of moduli see [zpz/givrns.h](#) or [zpz/givrnsfixed.h](#)

## 18.54 extension.h File Reference

NO DOX.

```
#include <gmp.h>
#include <givaro/gfq.h>
#include <givaro/givconfig.h>
#include <givaro/givpoly1.h>
#include <givaro/givpoly1factor.h>
#include <givaro/givpoly1padic.h>
#include "givaro/givtablelimits.h"
Include dependency graph for extension.h:
```

### Data Structures

- class [Extension< BFT >](#)  
*Extension.*
- class [GIV\\_ExtensionrandIter< ExtensionField, Type >](#)  
*Extension rand iters.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- template<class Rt >  
Rt **FF\_EXPONENT\_MAX** (const Rt p, const Rt maxe=21)  
XXX.
- template<class Rt >  
Rt **FF\_SUBEXPONENT\_MAX** (const Rt p, const Rt e)  
XXX.
- template<typename Field >  
int64\_t **Exponent\_Trait** (const Field &F)  
XXX.
- template<> int64\_t **Exponent\_Trait** (const GFqDom< int64\_t > &F)  
XXX.
- template<typename BaseField >  
int64\_t **Exponent\_Trait** (const Extension< BaseField > &F)  
XXX.

### 18.54.1 Detailed Description

NO DOX.

## 18.55 gfq.h File Reference

Arithmetic on  $\text{GF}(p^k)$ , with  $p$  a prime number less than  $2^{16}$ .

```
#include "givaro/givconfig.h"
#include "givaro/givinteger.h"
#include "givaro/givranditer.h"
#include "givaro/givpolylfactor.h"
#include <string>
#include <vector>
#include "givaro/gfq.inl"
Include dependency graph for gfq.h:
```

## Data Structures

- class [GFqDom< TT >](#)  
*class [GFqDom](#)*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.55.1 Detailed Description

Arithmetic on  $\text{GF}(p^k)$ , with  $p$  a prime number less than  $2^{16}$ .

## 18.56 gfqext.h File Reference

Arithmetic on  $\text{GF}(p^k)$ , with  $p$  a prime number less than  $2^{15}$ .

```
#include "givaro/gfq.h"
#include "givaro/givpower.h"
#include <limits>
#include <deque>
Include dependency graph for gfqext.h:
```

### Data Structures

- class [GFqExtFast< TT >](#)  
*GFq Ext.*
- class [GFqExt< TT >](#)  
*GFq Ext (other)*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.56.1 Detailed Description

Arithmetic on  $\text{GF}(p^k)$ , with  $p$  a prime number less than  $2^{15}$ .

Specialized for fast conversions to floating point numbers. Main difference in interface is init/convert.

**Bibliography** • JG Dumas, *Q-adic Transform Revisited*, ISSAC 2008

#### Warning

$k$  strictly greater than 1

## 18.57 gfqkronecker.h File Reference

Arithmetic on  $\text{GF}(p^k)$ , with dynamic Kronecker substitution.

```
#include "givaro/givzpz.h"
#include "givaro/givzpzInt.h"
#include "givaro/gfq.h"
#include "givaro/givpower.h"
#include <limits>
#include <vector>
#include <deque>
Include dependency graph for gfqkronecker.h:
```

## Data Structures

- struct [GFqKronecker](#)< TT, Ints >  
[GFqKronecker](#).

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.57.1 Detailed Description

Arithmetic on  $\text{GF}(p^k)$ , with dynamic Kronecker substitution.

#### Precondition

$k(p-1)^2 < \text{word size}$

## 18.58 givprimes16.h File Reference

set of primes less than  $2^{16}$

```
#include <stddef.h>
```

Include dependency graph for givprimes16.h:

## Data Structures

- class [Primes16](#)  
*class [Primes16](#)*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.58.1 Detailed Description

set of primes less than  $2^{16}$

## 18.59 givrns.h File Reference

Modular arithmetic for GIVARO.

```
#include "givaro/givconfig.h"
#include "givaro/giverror.h"
#include "givaro/givarrray0.h"
#include "givaro/givrnsstor.inl"
#include "givaro/givrnsconvert.inl"
Include dependency graph for givrns.h:
```

## Data Structures

- class [RNSsystem](#)< [RING](#), [Domain](#) >  
*class [RNSsystem](#).*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.59.1 Detailed Description

Modular arithmetic for GIVARO.

Here is defined arithmetic functions on rns representation and interface between RNS and Integer, all is done via the Chinese Remainder Algorithm.

## 18.60 givrnsfixed.h File Reference

Chinese Remainder Algorithm.

```
#include "givaro/givrns.h"
#include "givaro/givrandom.h"
#include "givaro/givintprime.h"
#include "givaro/modular-integer.h"
#include <vector>
#include "givaro/givrnsfixed.inl"
Include dependency graph for givrnsfixed.h:
```

## Data Structures

- class [RNSsystemFixed](#)< [Ints](#) >  
*NO DOC.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.60.1 Detailed Description

Chinese Remainder Algorithm.

## 18.61 StaticElement.h File Reference

NO DOC.

```
#include <iostream>
#include <gmp++/gmp++.h>
Include dependency graph for StaticElement.h:
```

### Data Structures

- struct [StaticElement< DomainStyle >](#)  
*Static Element.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

#### 18.61.1 Detailed Description

NO DOC.

## 18.62 gmp++\_int.h File Reference

Core gmp++\_int.h.

```
#include <vector>
#include <list>
#include <string>
#include <assert.h>
#include "recint/ruconvert.h"
#include "recint/rconvert.h"
#include <givaro/givcaster.h>
#include "gmp++/gmp++_int_rand.inl"
Include dependency graph for gmp++_int.h:
```

### Data Structures

- class [Integer](#)  
*This is the [Integer](#) class.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*
- namespace [std](#)  
*STL namespace.*

## Functions

- Integer & [inv](#) (Integer &u, const Integer &a, const Integer &b)  
*Modular inverse.*
- Integer & [invin](#) (Integer &u, const Integer &b)
- Integer [gcd](#) (const Integer &a, const Integer &b)
- Integer [pp](#) (const Integer &P, const Integer &Q)
- Integer & [lcm](#) (Integer &g, const Integer &a, const Integer &b)
- Integer [lcm](#) (const Integer &a, const Integer &b)
- Integer & [pow](#) (Integer &Res, const Integer &n, const int64\_t l)
- Integer [pow](#) (const Integer &n, const int64\_t l)
- Integer [powmod](#) (const Integer &n, const uint64\_t e, const Integer &m)
- int32\_t [sign](#) (const Integer &a)
- int32\_t [compare](#) (const Integer &a, const Integer &b)
- int32\_t [absCompare](#) (const Integer &a, const Integer &b)
- int32\_t [isZero](#) (const Integer &a)
- int32\_t [nonZero](#) (const Integer &a)
- int32\_t [isOne](#) (const Integer &a)
- Integer [fact](#) (uint64\_t l)
- Integer [sqrt](#) (const Integer &p)
- Integer [sqrtrem](#) (const Integer &p, Integer &rem)
- Integer & [sqrt](#) (Integer &r, const Integer &p)
- Integer & [sqrtrem](#) (Integer &r, const Integer &p, Integer &rem)
- bool [root](#) (Integer &q, const Integer &, uint32\_t n)
- int64\_t [logp](#) (const Integer &a, const Integer &p)
- double [logtwo](#) (const Integer &a)
- double [naturallog](#) (const Integer &a)
- void [swap](#) (Integer &, Integer &)
- bool [isOdd](#) (const Integer &a)  
*Tests parity of an integer.*
- uint64\_t [length](#) (const Integer &a)
- std::istream & [operator>>](#) (std::istream &i, Integer &n)
- std::ostream & [operator<<](#) (std::ostream &o, const Integer &n)
- std::ostream & [absOutput](#) (std::ostream &o, const Integer &n)

### 18.62.1 Detailed Description

Core gmp++\_int.h.

## 18.63 gmp++\_int\_add.C File Reference

adding stuff.

```
#include "gmp++/gmp++.h"
```

Include dependency graph for gmp++\_int\_add.C:

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- Integer [operator+](#) (const int32\_t l, const Integer &n)

### 18.63.1 Detailed Description

adding stuff.

## 18.64 gmp++\_int\_compare.C File Reference

routines to compare integers.

```
#include "gmp++/gmp++.h"
#include <cstdlib>
Include dependency graph for gmp++_int_compare.C:
```

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- int32\_t [compare](#) (const Integer &a, const Integer &b)
- int32\_t [absCompare](#) (const Integer &a, const Integer &b)
- int32\_t [operator!=](#) (uint32\_t l, const Integer &n)
- int32\_t [operator==](#) (uint32\_t l, const Integer &n)
- int32\_t [operator>](#) (uint32\_t l, const Integer &n)
- int32\_t [operator<](#) (uint32\_t l, const Integer &n)
- int32\_t [operator>=](#) (uint32\_t l, const Integer &n)
- int32\_t [operator<=](#) (uint32\_t l, const Integer &n)
- int32\_t [isOne](#) (const Integer &a)
- int32\_t [nonZero](#) (const Integer &a)
- int32\_t [isZero](#) (const Integer &a)

### 18.64.1 Detailed Description

routines to compare integers.

## 18.65 gmp++\_int\_cstor.C File Reference

cstoring stuff.

```
#include <iostream>
#include "gmp++/gmp++.h"
Include dependency graph for gmp++_int_cstor.C:
```

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.65.1 Detailed Description

cstoring stuff.

## 18.66 gmp++\_int\_div.C File Reference

diving stuff.

```
#include "gmp++/gmp++.h"
#include <cstdlib>
Include dependency graph for gmp++_int_div.C:
```

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.66.1 Detailed Description

diving stuff.

## 18.67 gmp++\_int\_gcd.C File Reference

gcding stuff.

```
#include "gmp++/gmp++.h"
#include "givaro/giverror.h"
Include dependency graph for gmp++_int_gcd.C:
```

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- Integer [lcm](#) (const Integer &a, const Integer &b)
- Integer & [lcm](#) (Integer &g, const Integer &a, const Integer &b)
- Integer [gcd](#) (const Integer &a, const Integer &b)
- Integer & [inv](#) (Integer &u, const Integer &a, const Integer &b)  
*[Modular](#) inverse.*
- Integer & [invin](#) (Integer &u, const Integer &b)
- Integer [pp](#) (const Integer &P, const Integer &Q)

### 18.67.1 Detailed Description

goding stuff.

## 18.68 gmp++\_int\_io.C File Reference

ioing stuff.

```
#include <iostream>
#include <stdlib.h>
#include "gmp++/gmp++.h"
#include <sstream>
Include dependency graph for gmp++_int_io.C:
```

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### Functions

- std::ostream & [absOutput](#) (std::ostream &o, const Integer &n)
- std::istream & [operator>>](#) (std::istream &i, Integer &n)
- std::ostream & [operator<<](#) (std::ostream &o, const Integer &n)

### 18.68.1 Detailed Description

ioing stuff.

## 18.69 gmp++\_int\_lib.C File Reference

libing stuff.

```
#include "gmp++/gmp++.h"
Include dependency graph for gmp++_int_lib.C:
```

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.69.1 Detailed Description

libing stuff.

## 18.70 gmp++\_int\_misc.C File Reference

miscing stuff.

```
#include <iostream>
#include <cmath>
#include "gmp++/gmp++.h"
#include <sstream>
Include dependency graph for gmp++_int_misc.C:
```

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### Functions

- Integer [fact](#) (uint64\_t l)
- Integer & [sqrt](#) (Integer &r, const Integer &p)
- Integer & [sqrtrem](#) (Integer &r, const Integer &p, Integer &rem)
- Integer [sqrt](#) (const Integer &p)
- Integer [sqrtrem](#) (const Integer &p, Integer &rem)
- bool [root](#) (Integer &q, const Integer &, uint32\_t n)
- void [swap](#) (Integer &, Integer &)
- double [naturallog](#) (const Integer &a)
- bool [isOdd](#) (const Integer &a)  
*Tests parity of an integer.*
- int64\_t [logp](#) (const Integer &a, const Integer &p)
- double [logtwo](#) (const Integer &a)
- uint64\_t [length](#) (const Integer &a)

### 18.70.1 Detailed Description

miscing stuff.

## 18.71 gmp++\_int\_mod.C File Reference

moding stuff.

```
#include "gmp++/gmp++.h"
#include <cstdlib>
Include dependency graph for gmp++_int_mod.C:
```

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- Integer [operator%](#) (const int64\_t l, const Integer &n)

### 18.71.1 Detailed Description

moding stuff.

## 18.72 gmp++\_int\_mul.C File Reference

muling stuff.

```
#include "gmp++/gmp++.h"
Include dependency graph for gmp++_int_mul.C:
```

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- Integer [operator\\*](#) (const int32\_t l, const Integer &n)

### 18.72.1 Detailed Description

muling stuff.

## 18.73 gmp++\_int\_pow.C File Reference

powing stuff.

```
#include "gmp++/gmp++.h"
#include <cstdlib>
Include dependency graph for gmp++_int_pow.C:
```

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- Integer & [pow](#) (Integer &Res, const Integer &n, const int64\_t l)
- Integer [pow](#) (const Integer &n, const int64\_t l)
- Integer [powmod](#) (const Integer &n, const uint64\_t e, const Integer &m)

### 18.73.1 Detailed Description

powing stuff.

## 18.74 gmp++\_int\_rand.inl File Reference

randing stuff.

```
#include <givaro/givtimer.h>
#include <givaro/givrandom.h>
#include <givaro/udl.h>
#include <assert.h>
Include dependency graph for gmp++_int_rand.inl:
```

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.74.1 Detailed Description

randing stuff.

## 18.75 gmp++\_int\_sub.C File Reference

subing stuff.

```
#include "gmp++/gmp++.h"
Include dependency graph for gmp++_int_sub.C:
```

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- Integer [operator-](#) (const int32\_t l, const Integer &n)

### 18.75.1 Detailed Description

subing stuff.

## 18.76 givinteger.h File Reference

Integer Domain class definition.

```
#include "givaro-config.h"
#include "gmp++/gmp++.h"
#include "givaro/givbasictype.h"
#include "givaro/givinit.h"
#include "givaro/giverror.h"
#include "givaro/givranditer.h"
#include "givaro/random-integer.h"
#include "givaro/ring-interface.h"
#include <string>
```

Include dependency graph for givinteger.h:

### Data Structures

- class [IntegerDom](#)  
*Integer Domain.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.76.1 Detailed Description

Integer Domain class definition.

## 18.77 givintfactor.h File Reference

factorisation

```
#include <iostream>
#include "givaro/givinteger.h"
#include "givaro/givintprime.h"
#include "givaro/givrandom.h"
#include "givaro/givintfactor.inl"
```

Include dependency graph for givintfactor.h:

## Data Structures

- class `IntFactorDom< MyRandIter >`  
*Integer Factor Domain.*

## Namespaces

- namespace `Givaro`  
*Namespace in which the whole `Givaro` library resides.*

### 18.77.1 Detailed Description

factorisation

- Prime numbers
  - Factor sets :
  - Pollard's rho method for factorization
  - Elliptic curves factorization by Lenstra

## 18.78 givintnumtheo.h File Reference

num theory.

```
#include <iostream>
#include "givaro/givinteger.h"
#include "givaro/givintprime.h"
#include "givaro/givintfactor.h"
#include "givaro/givrandom.h"
#include "givaro/udl.h"
#include "givaro/givintnumtheo.inl"
Include dependency graph for givintnumtheo.h:
```

## Data Structures

- class `IntNumTheoDom< MyRandIter >`  
*Num theory Domain.*

## Namespaces

- namespace `Givaro`  
*Namespace in which the whole `Givaro` library resides.*

### 18.78.1 Detailed Description

num theory.

- Euler's phi function
- Primitive roots
- RSA scheme.

## 18.79 givintprime.h File Reference

primes

```
#include "givaro/givinteger.h"
#include "givaro/givintprime.inl"
Include dependency graph for givintprime.h:
```

### Data Structures

- class [FermatDom](#)  
*Fermat numbers.*
- class [IntPrimeDom](#)  
*Primality tests.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.79.1 Detailed Description

primes

- Prime numbers
- Modular powering,
- Fermat numbers,
- Primality tests
- Factorization : (There are parameters to fix)

## 18.80 givintrns.h File Reference

arithmetic for RNS representations.

```
#include "givaro/givconfig.h"
#include "givaro/givinteger.h"
#include "givaro/givintrns_cstor.inl"
#include "givaro/givintrns_convert.inl"
Include dependency graph for givintrns.h:
```

### Data Structures

- class [IntRNSsystem](#)< [Container](#), [Alloc](#) >  
*RNS system class. No doc.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

#### 18.80.1 Detailed Description

arithmetic for RNS representations.

Modular arithmetic for GIVARO. Here is defined arithmetic functions on rns representation with [Givaro](#) Integers.

## 18.81 givintrsa.h File Reference

RSA scheme.

```
#include <iostream>
#include "givaro/givinteger.h"
#include "givaro/givintprime.h"
#include "givaro/givintfactor.h"
#include "givaro/givrandom.h"
#include "givaro/givintrsa.inl"
Include dependency graph for givintrsa.h:
```

### Data Structures

- class [IntRSADom](#)< [MyRandIter](#) >  
*RSA domain.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.81.1 Detailed Description

RSA scheme.

## 18.82 givintsqrootmod.h File Reference

Modular square roots.

```
#include <iostream>
#include "givaro/givinteger.h"
#include "givaro/givintprime.h"
#include "givaro/givintfactor.h"
#include "givaro/givintrns.h"
#include "givaro/givrandom.h"
#include "givaro/givpower.h"
#include <cmath>
#include "givaro/givintsqrootmod.inl"
Include dependency graph for givintsqrootmod.h:
```

### Data Structures

- class [IntSqrtModDom< MyRandIter >](#)  
*Modular square roots.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.82.1 Detailed Description

Modular square roots.

## 18.83 givaromm.h File Reference

Memory management in [Givaro](#) two memory managers:

```
#include <cstring>
#include <stddef.h>
#include <iostream>
#include <new>
#include "givaro/givmodule.h"
#include "givaro/givbasictype.h"
#include <givaro/givconfig.h>
Include dependency graph for givaromm.h:
```

## Data Structures

- class [GivMMInfo](#)  
*Static informations of memory allocation.*
- class [BlocFreeList](#)  
*Data structure of a bloc.*
- class [GivMMFreeList](#)  
*Implementation of a memory manager with free-lists.*
- class [GivMMRefCount](#)  
*Memory management with reference counter on allocated data.*
- class [GivaroMM< T >](#)  
*Memory manager that allocates array of object of type T for.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- `std::ostream & operator<< (std::ostream &o, const GivMMInfo &T)`  
*IO.*

### 18.83.1 Detailed Description

Memory management in [Givaro](#) two memory managers:

- the first one handle a set on free-list of blocs ;
- the second one implement a reference mecanism on the bloc.

The latter used method of the former.

## 18.84 givpointer.h File Reference

auto ptr management

```
#include "givaro/givaromm.h"
Include dependency graph for givpointer.h:
```

## Data Structures

- class [RefCountPtr< T >](#)  
*RefCount Pointer.*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.84.1 Detailed Description

auto ptr management

## 18.85 givref\_count.h File Reference

Definition of the Counter class, Counter.

```
#include <stddef.h>
```

Include dependency graph for givref\_count.h:

## Data Structures

- class [RefCounter](#)

*Ref counter.*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.85.1 Detailed Description

Definition of the Counter class, Counter.

This class definition objects to handle reference counter for memory allocation (eg array0).

## 18.86 givrational.h File Reference

Rationals (and domain), composed of an integer (numerator), and a positive integer (denominator) NO DOC.

```
#include "givaro/givinteger.h"
```

```
#include "givaro/givmodule.h"
```

```
#include "givaro/ring-interface.h"
```

```
#include "givaro/givrational.inl"
```

Include dependency graph for givrational.h:

## Data Structures

- class [Rational](#)  
*Rationals. No doc.*
- class [QField< Rational >](#)  
*Rational Domain.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.86.1 Detailed Description

Rationals (and domain), composed of an integer (numerator), and a positive integer (denominator) NO DOC.

## 18.87 modular-implem.h File Reference

Generic implementation of Modular.

```
#include "givaro/givinteger.h"
#include "givaro/givcaster.h"
#include "givaro/givranditer.h"
#include "givaro/ring-interface.h"
#include "givaro/modular-general.h"
Include dependency graph for modular-implem.h:
```

## Data Structures

- class [Modular\\_implem< \\_Storage\\_t, \\_Compute\\_t, \\_Residu\\_t >](#)  
*This class implement the standard arithmetic with Modulo Elements.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.87.1 Detailed Description

Generic implementation of Modular.

## 18.88 modular-integral.h File Reference

representation of  $\mathbb{Z}/m\mathbb{Z}$  over int types.

```
#include "givaro/givinteger.h"
#include "givaro/givcaster.h"
#include "givaro/givranditer.h"
#include "givaro/ring-interface.h"
#include "givaro/modular-general.h"
#include "givaro/modular-implem.h"
#include "modular-mulprecomp.inl"
#include "modular-integral.inl"
```

Include dependency graph for modular-integral.h:

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.88.1 Detailed Description

representation of  $\mathbb{Z}/m\mathbb{Z}$  over int types.

## 18.89 modular.h File Reference

Family of arithmetics over  $\mathbb{Z}_p$  ( $\mathbb{Z}/p\mathbb{Z}$ ).

```
#include <givaro/givconfig.h>
#include "givaro/modular-integral.h"
#include "givaro/modular-floating.h"
#include "givaro/modular-integer.h"
#include "givaro/modular-inttype.h"
#include "givaro/modular-log16.h"
#include "givaro/modular-ruint.h"
```

Include dependency graph for modular.h:

### 18.89.1 Detailed Description

Family of arithmetics over  $\mathbb{Z}_p$  ( $\mathbb{Z}/p\mathbb{Z}$ ).

## 18.90 montgomery.h File Reference

Family of arithmetics over  $\mathbb{Z}_p$  ( $\mathbb{Z}/p\mathbb{Z}$ ).

```
#include "givaro/montgomery-int32.h"
#include "givaro/montgomery-ruint.h"
```

Include dependency graph for montgomery.h:

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.90.1 Detailed Description

Family of arithmetics over  $\mathbb{Z}_p$  ( $\mathbb{Z}/p\mathbb{Z}$ ).

## 18.91 givbasictype.h File Reference

NO DOC.

```
#include "givaro/givconfig.h"
#include <stdlib.h>
#include <sys/types.h>
Include dependency graph for givbasictype.h:
```

## Data Structures

- class [Neutral](#)  
*[Neutral](#) type.*
- class [givNoInit](#)  
*Used to build no initialized object as static object.*
- class [givNoCopy](#)  
*Used to call ctor without copy.*
- class [givWithCopy](#)  
*Used to call ctor with copy.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.91.1 Detailed Description

NO DOC.

## 18.92 givcaster.h File Reference

NO DOC.

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.92.1 Detailed Description

NO DOC.

## 18.93 givconfig.h File Reference

configuration file for [Givaro](#)

```
#include <givaro-config.h>
#include <limits>
Include dependency graph for givconfig.h:
```

### 18.93.1 Detailed Description

configuration file for [Givaro](#)

## 18.94 giverror.h File Reference

error exception.

```
#include <iostream>
Include dependency graph for giverror.h:
```

## Data Structures

- class [GivError](#)  
*Base class for exeception handling in [Givaro](#).*
- class [GivMathError](#)  
*Math error.*
- class [GivBadFormat](#)  
*Exception thrown in input of data structure.*
- class [GivMathDivZero](#)  
*Div by 0.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.94.1 Detailed Description

error exception.

## 18.95 givgenarith.h File Reference

Domain definition for basic type of the language.

```
#include "givaro/givbasictype.h"
Include dependency graph for givgenarith.h:
```

### Data Structures

- struct [\\_\\_givdom\\_trait\\_name< T >](#)  
*give a name for /read/write*
- class [BaseDomain< T >](#)  
*Base Domain.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### Typedefs

- typedef BaseDomain< char > **CharDom**  
*char dom*
- typedef BaseDomain< short > **ShortDom**  
*short dom*
- typedef BaseDomain< int > **IntDom**  
*int dom*
- typedef BaseDomain< long > **LongDom**  
*long dom*
- typedef BaseDomain< float > **FloatDom**  
*float dom*
- typedef BaseDomain< double > **DoubleDom**  
*double dom*

### 18.95.1 Detailed Description

Domain definition for basic type of the language.

## 18.96 givinit.h File Reference

NO DOC.

```
#include "givaro/givconfig.h"
```

```
#include <iostream>
```

Include dependency graph for givinit.h:

### Data Structures

- class [GivaroMain](#)  
*Initialisation of GIVARO .*
- class [GivaroAppli](#)  
*Main application class Could be not used.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

#### 18.96.1 Detailed Description

NO DOC.

## 18.97 givmodule.h File Reference

NO DOC.

### Data Structures

- class [GivaroNoInit](#)  
*GivaroNoInit.*
- class [InitAfter](#)  
*InitAfter.*
- class [GivModule](#)  
*GivModule.*
- class [ObjectInit](#)  
*GivModule.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.97.1 Detailed Description

NO DOC.

## 18.98 givperf.h File Reference

performance analysis

### 18.98.1 Detailed Description

performance analysis

## 18.99 givpower.h File Reference

NO DOC.

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### Functions

- `template<typename T >`  
`unsigned GIVINTLOG (const T &a)`  
*[Integer](#) log.*
- `template<class TT , class UU >`  
`TT power (const TT n, const UU l)`  
*Powering.*
- `template<class D , class TT >`  
`TT & dom_power (TT &res, const TT &n, long l, const D &F)`  
*dom\_power*

### 18.99.1 Detailed Description

NO DOC.

## 18.100 givprint.h File Reference

helper print for containers

```
#include <iostream>
#include <iterator>
#include <vector>
#include <list>
#include <set>
```

Include dependency graph for givprint.h:

## Namespaces

- namespace [std](#)  
*STL namespace.*

## Functions

- template<class T , typename A = std::allocator<T>>  
std::ostream & [operator<<](#) (std::ostream &out, const std::vector< T, A > &V)  
*Prints a vector on output.*
- template<class S , class T >  
std::ostream & [operator<<](#) (std::ostream &out, const std::pair< S, T > &C)  
*Prints a pair.*
- template<class T , typename A = std::allocator<T>>  
std::ostream & [operator<<](#) (std::ostream &out, const std::list< T, A > &L)  
*Prints a list.*
- template<class T , typename A = std::allocator<T>>  
std::ostream & [operator<<](#) (std::ostream &out, const std::set< T, A > &S)  
*Prints a set.*

### 18.100.1 Detailed Description

helper print for containers

## 18.101 givranditer.h File Reference

NO DOC [Givaro](#) ring Elements generator.

```
#include "givaro/givconfig.h"
#include "givaro/givrandom.h"
#include "givaro/givtimer.h"
#include <sys/time.h>
#include <stdlib.h>
#include <limits>
```

Include dependency graph for givranditer.h:

## Data Structures

- class [GIV\\_randIter< Ring, Type >](#)  
*Random ring Element generator.*
- class [ModularRandIter< Ring >](#)  
*Random ring Element generator.*
- class [GeneralRingRandIter< Ring >](#)  
*UnparametricRandIter.*
- class [GeneralRingNonZeroRandIter< Ring, RandIter >](#)  
*Random iterator for nonzero random numbers.*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.101.1 Detailed Description

NO DOC [Givaro](#) ring Elements generator.

## 18.102 givrandom.h File Reference

NO DOC.

```
#include <givaro/givconfig.h>
#include <givaro/udl.h>
#include <givaro/givtimer.h>
#include <sys/time.h>
#include <sys/resource.h>
```

Include dependency graph for givrandom.h:

## Data Structures

- class [GivRandom](#)

*[GivRandom](#).*

## Namespaces

- namespace [Givaro](#)

*Namespace in which the whole [Givaro](#) library resides.*

### 18.102.1 Detailed Description

NO DOC.

- Bibliography**
- Fishman, GS *Multiplicative congruential random number generators...* Math. Comp. 54:331-344 (1990).

## 18.103 givtimer.h File Reference

timer

```
#include <givaro/givconfig.h>
#include <iostream>
```

Include dependency graph for givtimer.h:

## Data Structures

- class [BaseTimer](#)  
*base for class [RealTimer](#); class [SysTimer](#); class [UserTimer](#);*
- class [RealTimer](#)  
*Real timer.*
- class [UserTimer](#)  
*User timer.*
- class [SysTimer](#)  
*Sys timer.*
- class [Timer](#)  
*Timer.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- `std::ostream & operator<< (std::ostream &o, const BaseTimer &BT)`  
*I/O.*
- `std::ostream & operator<< (std::ostream &o, const Timer &T)`  
*I/O.*

### 18.103.1 Detailed Description

timer

## 18.104 givdegree.h File Reference

NO DOC opaque class for Degree of polynomial.

```
#include <cstdint>
```

```
#include <iostream>
```

Include dependency graph for givdegree.h:

## Data Structures

- class [Degree](#)  
*[Degree](#) type for polynomials.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- `int64_t value` (const Degree &d)  
*value*

### 18.104.1 Detailed Description

NO DOC opaque class for Degree of polynomial.

Degree of polynomial 0 is Degree::deginfy with value DEGPOLYZERO.

## 18.105 givindeter.h File Reference

indeterminates for polynomial manipulation

```
#include <iostream>
#include <string>
Include dependency graph for givindeter.h:
```

## Data Structures

- class [Indeter](#)  
*Indeterminate.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Functions

- int [operator==](#) (const Indeter &i1, const Indeter &i2)

### 18.105.1 Detailed Description

indeterminates for polynomial manipulation

## 18.106 givinterp.h File Reference

NO DOC.

```
#include "givaro/givconfig.h"
#include "givaro/giverror.h"
#include "givaro/givpoly1.h"
Include dependency graph for givinterp.h:
```

## Data Structures

- struct [Interpolation](#)< Domain, REDUCE >  
*Interpolation.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.106.1 Detailed Description

NO DOC.

## 18.107 givinterpgeom-multip.h File Reference

Interpolation at geometric points.

```
#include "givaro/givconfig.h"
#include "givaro/giverror.h"
#include "givaro/givpoly1.h"
#include <givaro/givtruncdomain.h>
#include <vector>
Include dependency graph for givinterpgeom-multip.h:
```

## Data Structures

- struct [NewtonInterpGeomMultip](#)< Domain, REDUCE >  
*Newton (multip)*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.107.1 Detailed Description

Interpolation at geometric points.

- Bibliography**
- A Bostan and E Schost, *Polynomial evaluation and interpolation on special sets of points*, Journal of Complexity 21(4): 420-446, 2005.

## 18.108 givinterpgeom.h File Reference

Interpolation at geometric points.

```
#include "givaro/givconfig.h"
#include "givaro/giverror.h"
#include "givaro/givpoly1.h"
#include <givaro/givtruncdomain.h>
Include dependency graph for givinterpgeom.h:
```

### Data Structures

- struct [NewtonInterpGeom](#)< Domain, REDUCE >  
*Newton.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

#### 18.108.1 Detailed Description

Interpolation at geometric points.

- Bibliography**
- A Bostan and E Schost, *Polynomial evaluation and interpolation on special sets of points*, Journal of Complexity 21(4): 420-446, 2005.

## 18.109 givpoly1.h File Reference

NO DOC.

```
#include "givaro/giverror.h"
#include "givaro/givarray0.h"
#include "givaro/givcategory.h"
#include "givaro/giviterator.h"
#include "givaro/givops.h"
#include "givaro/givpoly1dense.h"
#include "givaro/givpoly1denseops.inl"
Include dependency graph for givpoly1.h:
```

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.109.1 Detailed Description

NO DOC.

## 18.110 givpoly1crt.h File Reference

Polynomial Chinese Remaindering of degree 1.

```
#include <givaro/givpoly1.h>
#include <givaro/givindeter.h>
#include <vector>
#include "givaro/givpoly1crtcstor.inl"
#include "givaro/givpoly1crtconvert.inl"
Include dependency graph for givpoly1crt.h:
```

### Data Structures

- class [Poly1CRT< Field >](#)  
*Poly1 CRT.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.110.1 Detailed Description

Polynomial Chinese Remaindering of degree 1.

## 18.111 givpoly1dense.h File Reference

univariate polynomial over T.

```
#include <iostream>
#include "givaro/givdegree.h"
#include "givaro/givindeter.h"
#include "givaro/givinteger.h"
#include "givaro/givrandom.h"
#include <vector>
Include dependency graph for givpoly1dense.h:
```

### Data Structures

- class [Poly1Dom< Domain, Dense >](#)  
*Class Poly1Dom.*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

## Typedefs

- `template<typename T, typename A = std::allocator<T>>`  
`using givvector = std::vector< T, A >`  
*givvector*

### 18.111.1 Detailed Description

univariate polynomial over T.

we assume that T is a ring  $(0,1,+,*)$

## 18.112 givpoly1factor.h File Reference

NO DOC.

```
#include <givaro/givrandom.h>
#include <givaro/givpoly1.h>
#include "givaro/givpoly1factor.inl"
#include "givaro/givpoly1proot.inl"
Include dependency graph for givpoly1factor.h:
```

## Data Structures

- class [Poly1FactorDom< Domain, Tag, RandomIterator >](#)  
*[Poly1FactorDom](#).*

## Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

### 18.112.1 Detailed Description

NO DOC.

- Irreducibility test
- Factorisations de Polynomes dans  $\mathbb{F}_p[X]$  :
- Distinct Degree
- Cantor-Zassenhaus
- Berlekamp: moved in LinBox

## 18.113 givpoly1padic.h File Reference

NO DOC.

```
#include <givaro/givinteger.h>
#include <givaro/givpoly1.h>
#include <cmath>
#include <iostream>
Include dependency graph for givpoly1padic.h:
```

### Data Structures

- class [Poly1PadicDom](#)< [Domain](#), [Dense](#) >  
*Poly1 p-adic.*

### Namespaces

- namespace [Givaro](#)  
*Namespace in which the whole [Givaro](#) library resides.*

#### 18.113.1 Detailed Description

NO DOC.

## 18.114 test-crt.C File Reference

NO DOC.

```
#include <iostream>
#include <givaro/givintprime.h>
#include <givaro/montgomery.h>
#include <givaro/modular.h>
#include <givaro/gfq.h>
#include <givaro/chineseremainder.h>
#include <givaro/givrns.h>
#include <givaro/givrnsfixed.h>
#include <givaro/givtimer.h>
#include <givaro/givrandom.h>
Include dependency graph for test-crt.C:
```

#### 18.114.1 Detailed Description

NO DOC.

## 18.115 test-integer.C File Reference

tests integer.h fuctions not tested elsewhere.

```
#include <stdlib.h>
#include "gmp++/gmp++.h"
#include <typeinfo>
#include <iostream>
Include dependency graph for test-integer.C:
```

### Functions

- int [main](#) (int argc, char \*\*argv)

#### 18.115.1 Detailed Description

tests integer.h fuctions not tested elsewhere.

**Test** tests integer.h fuctions not tested elsewhere.

#### 18.115.2 Function Documentation

##### 18.115.2.1 main()

```
int main (
    int argc,
    char ** argv )
```

**Todo** test gcd...

## 18.116 test-modsqroot.C File Reference

NO DOC.

```
#include <iostream>
#include <stdlib.h>
#include <givaro/givintsqrootmod.h>
#include <givaro/givtimer.h>
Include dependency graph for test-modsqroot.C:
```

### 18.116.1 Detailed Description

NO DOC.

See also

[examples/Integer/ModularSquareRoot.C](#)

## 18.117 test-random.C File Reference

we test bounds for random Integers

```
#include <iostream>
#include <givaro/givinteger.h>
#include <givaro/modular.h>
#include <givaro/montgomery.h>
#include <givaro/modular-ruint.h>
#include <givaro/modular-balanced.h>
Include dependency graph for test-random.C:
```

### Functions

- int **test1** ()  
*tests ret= .func (arg, arg); ...*
- int **test2** ()  
*tests*
- int **test3** ()  
*tests*
- int **test4** ()  
*test possibly < 0 random numbers*
- int **test5** ()  
*tests standard interface*

### 18.117.1 Detailed Description

we test bounds for random Integers

**Test** we test bounds for random Integers

### 18.117.2 Function Documentation

**18.117.2.1 test2()**

```
int test2 ( )
```

tests

```
ret= ::func(arg,arg);
```

**18.117.2.2 test3()**

```
int test3 ( )
```

tests

```
::func(ret,arg,arg);
```

## Chapter 19

# Example Documentation

### 19.1 examples/FiniteField/all\_field.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/gfq.h>
#include <givaro/montgomery.h>
#include <givaro/modular.h>
#include <givaro/StaticElement.h>
using namespace Givaro;
namespace Givaro {
    // Domain kind
    typedef Modular<uint32_t> Field1; typedef StaticElement< Field1 > Element1;    template<> Field1
    Element1::_domain(2);
    typedef GFqDom<int64_t> Field2; typedef StaticElement< Field2 > Element2;    template<> Field2
    Element2::_domain(2);
    typedef Montgomery<int32_t> Field3; typedef StaticElement< Field3 > Element3;    template<> Field3
    Element3::_domain(2);
    typedef Modular<Integer> Field4; typedef StaticElement< Field4 > Element4;    template<> Field4
    Element4::_domain(2);
    typedef Modular<int32_t> Field5; typedef StaticElement< Field5 > Element5;    template<> Field5
    Element5::_domain(2);
    typedef Modular<int16_t> Field6; typedef StaticElement< Field6 > Element6;    template<> Field6
    Element6::_domain(2);
    typedef Modular<Log16> Field7; typedef StaticElement< Field7 > Element7;    template<> Field7
    Element7::_domain(2);
#ifdef GIVARO_USE_SIXTYFOUR
    typedef Modular<int64_t> Field8; typedef StaticElement< Field8 > Element8;    template<>
    Field8 Element8::_domain(2);
#endif
}
template<class Field, class Element>
void TestField()
{
    uint64_t P = 251;
    // Initialization of static member
    Element::setDomain( Field( typename Field::Residu_t(P) ) );
    // Initialisations of elements
    Element a(2),b(-29.8),c(33),d(Integer("123456789012345678901234567890")),e(0);
    e += (a = b);    std::cout << a << " = " << b << " mod " << P << "; " << std::endl;
    e += (a = b + c);    std::cout << a << " = " << b << " + " << c << " mod " << P << "; " << std::endl;
    e += (a = b - c);    std::cout << a << " = " << b << " - " << c << " mod " << P << "; " << std::endl;
    e += (a = b * c);    std::cout << a << " = " << b << " * " << c << " mod " << P << "; " << std::endl;
    e += (a = b / c);    std::cout << a << " = " << b << " / " << c << " mod " << P << "; " << std::endl;
    std::cout << d << " + " << a << " mod " << P << " = "; e += (d += a); std::cout << d << "; " << std::endl;
    std::cout << d << " - " << a << " mod " << P << " = "; e += (d -= a); std::cout << d << "; " << std::endl;
    std::cout << d << " * " << a << " mod " << P << " = "; e += (d *= a); std::cout << d << "; " << std::endl;
    std::cout << d << " / " << a << " mod " << P << " = "; e += (d /= a); std::cout << d << "; " << std::endl;
    std::cout << a << " is non zero ? " << (a != Element(0) ) << std::endl;
    a = 0; std::cout << a << " is zero ? " << (a == Element(0) ) << std::endl;
}
```

```

        double dd(0.0); dd += (double)(e); dd += (float)e; dd += (uint32_t)e;
        Element::getDomain().write( std::cerr << "Test: " << dd << " within " << std::endl;
    }
int main(int argc, char ** argv) {
    TestField<Field1, Element1>();
    TestField<Field2, Element2>();
    TestField<Field3, Element3>();
    TestField<Field4, Element4>();
    TestField<Field5, Element5>();
    TestField<Field6, Element6>();
    TestField<Field7, Element7>();
#ifdef GIVARO_USE_SIXTYFOUR
    TestField<Field8, Element8>();
#endif
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.2 examples/FiniteField/domain\_to\_operatorstyle.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/gfq.h>
#include <givaro/StaticElement.h>
using namespace Givaro;
namespace Givaro
{
    // Finite Field with Domain style
    typedef GFqDom<int64_t> Field;
    // Wrapper to give an operator style to the elements
    typedef StaticElement< Field > Element;
    // Mandatory declaration (because of static template)
    // and an actual constructed field is mandatory (the "(2)") for g++ 3.4
    template<>
    Field Element::_domain(2);
}
int main(int argc, char ** argv) {
    uint64_t P = (argc>1 ? (uint64_t)atoi(argv[1]) : 5009U);
    // Initialization of static member
    Element::setDomain( Field(P) );
    // Initialisations of elements
    Element a(2),b(-29.8),c(33),d(Integer("123456789012345678901234567890"));
    // Computations
    a = b;      std::cerr << a << " = " << b << " mod " << P << "; " << std::endl;
    a = b + c;  std::cerr << a << " = " << b << " + " << c << " mod " << P << "; " << std::endl;
    a = b - c;  std::cerr << a << " = " << b << " - " << c << " mod " << P << "; " << std::endl;
    a = b * c;  std::cerr << a << " = " << b << " * " << c << " mod " << P << "; " << std::endl;
    a = b / c;  std::cerr << a << " = " << b << " / " << c << " mod " << P << "; " << std::endl;
    // Computations in place
    std::cerr << d << " + " << a << " mod " << P << " = ";
    d += a; std::cerr << d << "; " << std::endl;
    std::cerr << d << " - " << a << " mod " << P << " = ";
    d -= a; std::cerr << d << "; " << std::endl;
    std::cerr << d << " * " << a << " mod " << P << " = ";
    d *= a; std::cerr << d << "; " << std::endl;
    std::cerr << d << " / " << a << " mod " << P << " = ";
    d /= a; std::cerr << d << "; " << std::endl;
    // Tests
    const Element zero(0);
    std::cerr << a << " is non zero is " << (a != zero) << std::endl;
    std::cerr << a << " is non zero is " << (! Element::isZero(a)) << std::endl;
    std::cerr << a << " is non zero is " << (! a.isZero()) << std::endl;
    a = 0; std::cerr << a << " is zero is " << (a == zero) << std::endl;
    std::cerr << a << " is zero is " << Element::isZero(a) << std::endl;
    std::cerr << a << " is zero is " << a.isZero() << std::endl;
    // Access to Field object
    const Field & F = Element::getDomain();
    F.write( std::cerr << "Test: within " << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.3 examples/FiniteField/exponentiation.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givpower.h>
#include <givaro/modular.h>
#include <givaro/gfq.h>
using namespace Givaro;
int main(int argc, char ** argv) {
    {
        Modular<int32_t> Z13(13); // modulo 13 over 32 bits
        Modular<int32_t>::Element a, c;
        Z13.init(a, 7);
        long l = 29;
        dom_power(c, a, l, Z13); // c = 7^29 modulo 13 by squaring
        std::cerr << "Within ";
        Z13.write( std::cerr );
        std::cerr << " : " << std::flush;
        // Separate output writing
        Z13.write( std::cout, a << " ^ " << l << " = " << std::flush;
        Z13.write( std::cerr, c << std::endl;
    }
    {
        typedef GFqDom<int>::Residu_t TT;
        int Mod = 13; int exponent = 1;
        GFqDom<int> GF13( (TT) Mod, (TT) exponent ); // finite field with 13 elements
        GFqDom<int>::Element a, c;
        GF13.init(a, 7); // 7 modulo 13
        long l = 29;
        dom_power(c, a, l, GF13); // c = 7^29 modulo 13 by squaring
        // Writing all outputs in a single command line
        GF13.write( std::cerr << "Within " ) << " : " << std::flush;
        GF13.write( GF13.write(
            std::cout, a << " ^ " << l << " = ", c << std::endl;
        )
    }
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:src:cin=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.4 examples/FiniteField/ff\_arith.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/modular.h>
#include <givaro/montgomery.h>
#include <givaro/gfq.h>
#include <givaro/gfqext.h>
using namespace Givaro;
template<class Field>
void TestField(const Field& F) {
    std::cerr << "Within " ;
    F.write(std::cerr );
    std::cerr << " : " << std::flush;
    typename Field::Element a, b, c, d;
    F.init(a, 7U);
    F.init(b, -29.3);
    F.init(c); // empty constructor
    F.init(d); // empty constructor
    F.add(c, a, b); // c = a+b
    // Separate output writing
    F.write( std::cout, a << " + " << std::flush;
    F.write( std::cout, b << " = " << std::flush;
```

```

F.write( std::cerr, c) << std::endl;
F.mul(c, a, b); // c = a*b
F.axpy(d, a, b, c); // d = a*b + c;
// Writing all outputs in a single command line
F.write( std::cerr << "Within " ) << " : " << std::flush;
F.write( F.write( F.write( F.write(
    std::cout, c) << " + ", a) << " * ", b) << " = ", d) << std::endl;
{
    typename Field::Element e;
    F.init(e); F.assign(e,d);
    F.maxpy(e, a, b, d); // e = d-a*b
    // Writing all outputs in a single command line
    F.write( std::cerr << "Within " ) << " : " << std::flush;
    F.write( F.write( F.write( F.write(
        std::cout, d) << " - ", a) << " * ", b) << " = ", e) << std::endl;
}
{
    typename Field::Element e;
    F.init(e); F.assign(e,d);
    F.maxpyin(e, a, b); // e = d - a*b;
    // Writing all outputs in a single command line
    F.write( std::cerr << "Within " ) << " : " << std::flush;
    F.write( F.write( F.write( F.write(
        std::cout, d) << " - ", a) << " * ", b) << " = ", e) << std::endl;
}
{
    typename Field::Element e;
    F.init(e); F.assign(e,d);
    F.axmy(e, a, b, d); // e = a*b -d;
    // Writing all outputs in a single command line
    F.write( std::cerr << "Within " ) << " : " << std::flush;
    F.write( F.write( F.write( F.write(
        std::cout, a) << " * ", b) << " - ", d) << " = ", e) << std::endl;
}
{
    typename Field::Element e;
    F.init(e); F.assign(e,d);
    F.maxpyin(e, a, b); // e = d - a*b;
    // Writing all outputs in a single command line
    F.write( std::cerr << "Within " ) << " : " << std::flush;
    F.write( F.write( F.write( F.write(
        std::cout, d) << " - ", a) << " * ", b) << " = ", e) << std::endl;
}
// Four operations
F.write( F.write( std::cout, a) << " += ", b) << " is " ;
F.write( std::cout, F.addin(a, b) ) << " ; ";
F.write( F.write( std::cout, a) << " -= ", b) << " is " ;
F.write( std::cout, F.subin(a, b) ) << " ; ";
F.write( F.write( std::cout, a) << " *= ", b) << " is " ;
F.write( std::cout, F.mulin(a, b) ) << " ; ";
F.write( F.write( std::cout, a) << " /= ", b) << " is " ;
F.write( std::cout, F.divin(a, b) ) << std::endl;
F.init(a,22996);
F.inv(b,a);
F.write( F.write( std::cout << "1/", a) << " is ", b) << std::endl;
F.mul(c,b,a);
F.write( std::cout << "1 is ", c) << std::endl;
F.init(a,22996);
F.init(b,22996);
F.write( std::cout << "1/", a) << " is ";
F.invin(a);
F.write( std::cout, a) << std::endl;
F.mulin(a,b);
F.write( std::cout << "1 is ", a) << std::endl;
F.init(a,37403);
F.inv(b,a);
F.write( F.write( std::cout << "1/", a) << " is ", b) << std::endl;
F.mul(c,b,a);
F.write( std::cout << "1 is ", c) << std::endl;
F.init(a,37403);
F.init(b,37403);
F.write( std::cout << "1/", a) << " is ";
F.invin(a);
F.write( std::cout, a) << std::endl;
F.mulin(a,b);
F.write( std::cout << "1 is ", a) << std::endl;
}
extern "C" {
# include <sys/time.h>
# include <sys/resource.h>
}
int main(int argc, char ** argv) {
    // modulo 13 over 16 bits
    Modular<int16_t> C13(13); TestField( C13 );
    // modulo 13 over 32 bits
    Modular<int32_t> Z13(13); TestField( Z13 );
    // modulo 13 over unsigned 32 bits

```

```

Modular<uint32_t> U13(13); TestField( U13 );
#ifdef __USE_Givaro_SIXTYFOUR__
// modulo 13 over 64 bits
Modular<int64_t> LL13(13U); TestField( LL13 );
#endif
// modulo 13 fully tabulated
Modular<Log16> L13(13); TestField( L13 );
// modulo 13 over 32 bits with Montgomery reduction
Montgomery<int32_t> M13(13); TestField( M13 );
Montgomery<int32_t> M3(39989); TestField( M3 );
// modulo 13 with primitive root representation
GFqDom<int> GF13( 13 ); TestField( GF13 );
// modulo 13 over arbitrary size
Modular<Integer> IntZ13(13); TestField( IntZ13 );
// Zech log finite field with 5^4 elements
GFqDom<int> GF625( 5, 4 ); TestField( GF625 );
// Zech log finite field with 256 elements
// and prescribed irreducible polynomial
std::vector< GFqDom<int64_t>::Residu_t > Irred(9);
Irred[0] = 1; Irred[1] = 1; Irred[2] = 0; Irred[3] = 1;
Irred[4] = 1; Irred[5] = 0; Irred[6] = 0; Irred[7] = 0;
Irred[8] = 1;
GFqDom<int64_t> F256(2,8, Irred); TestField( F256 );
// Zech log finite field with 3^4 elements
// Using the Q-adic Transform
GFqExt<int32_t> GF81( 3, 4 ); TestField( GF81 );
// Zech log finite field with 2Mb tables
struct rusage tmp1 ;
getrusage (RUSAGE_SELF, &tmp1) ;
// user time
double tim = (double) tmp1.ru_utime.tv_sec + ((double) tmp1.ru_utime.tv_usec)/ ( 1000000.0 ) ;
;
getrusage (RUSAGE_SELF, &tmp1) ;
tim = (double) tmp1.ru_utime.tv_sec + ((double) tmp1.ru_utime.tv_usec)/ (1000000.0) - tim;
std::cerr << "Initialization took " << tim << " cpu seconds and : " << std::endl;
std::cerr
<< tmp1.ru_maxrss << " maximum resident set size" << std::endl
<< tmp1.ru_ixrss << " integral shared memory size" << std::endl
<< tmp1.ru_idrss << " integral unshared data size" << std::endl
<< tmp1.ru_isrss << " integral unshared stack size" << std::endl
<< tmp1.ru_minflt << " page reclaims" << std::endl
<< tmp1.ru_majflt << " page faults" << std::endl
<< tmp1.ru_nswap << " swaps" << std::endl
<< tmp1.ru_inblock << " block input operations" << std::endl
<< tmp1.ru_oublock << " block output operations" << std::endl;
return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.5 examples/FiniteField/GF128.C

NO DOC.

NO DOC

```

// =====
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
// Time-stamp: <12 Jun 15 18:38:55 Jean-Guillaume.Dumas@imag.fr>
// Thanks to Dieter Schuster
// =====
#include <givaro/gfq.h>
#include <givaro/givtimer.h>
using namespace Givaro;
int main(int argc, char** argv)
{
    GFqDom<int64_t> GF128(2, 7);
    GFqDom<int64_t>::Element b, c, gen;
    GF128.init(b, 5U);
    GF128.init(c, 3);
    GF128.write(std::cout, b) << std::endl;
    GF128.write(std::cout, c) << std::endl;
    GFqDom<int64_t>::Element f,g,h,j;
    GFqDom<int64_t> F2(2);
    Poly1Dom< GFqDom<int64_t>, Dense> Pol2(F2);
    Poly1Dom< GFqDom<int64_t>, Dense>::Element P, Q, R;
    Pol2.init(P, Degree(1));

```

```

F2.init(P[0],1);
F2.init(P[1],1);
GF128.init(f, P);
GF128.write(std::cout << "2-adic representation of 1+X is: ", f)
<< std::endl
<< " ... while its internal representation is: "
<< f << std::endl;
GF128.write(std::cout << "Indeed, we are in ") <<std::endl;
GF128.generator(gen);
GF128.write(std::cout <<
    "In this field, the generator used is (in 2-adic): ", gen)
<< std::endl
<< " whose internal representation is "
<< gen << std::endl;
Poly1PadicDom< GFqDom<int64_t>, Dense > Padic2(Pol2);
//
std::cout << "Irreducible (in 2-adic): "
<< GF128.irreducible() << std::endl;
GF128.init(g, Padic2.radix( Q, Integer(5) ));
GF128.write(std::cout
    << "2-adic representation of 1+X^2 is: ", g)
<< std::endl;
GF128.init(h);
GF128.add(h, g, f);
GF128.write(std::cout
    << "2-adic representation of X+X^2 is: ", h)
<< std::endl;
GF128.mul(h, g, f);
GF128.write(std::cout
    << "2-adic representation of 1+X+X^2+X^3 is: ", h)
<< std::endl;
GF128.div(h, g, f);
GF128.write(std::cout
    << "2-adic representation of 1+X is: ", h)
<< std::endl;
GF128.init(j, Padic2.radix( Q, Integer(213) ));
GF128.write(std::cout
    << "2-adic representation of the moding out of X^7+X^6+X^4+X^2+1 by the irreducible is: ", j)
<< std::endl;
return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.6 examples/FiniteField/GFirreducible.C

NO DOC.

NO DOC

```

// =====
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
// Time-stamp: <12 Jun 15 18:47:22 Jean-Guillaume.Dumas@imag.fr>
// Check of irreducible polynomial and generators of GFq
// =====
//
#include <givaro/gfq.h>
#include <givaro/givpower.h>
#include <givaro/givtimer.h>
using namespace Givaro;
int main(int argc, char** argv)
{
    int64_t p = (argc>1?atoi(argv[1]):5);
    int64_t e = (argc>2?atoi(argv[2]):3);
    typedef GFqDom<int64_t>::Residu_t TT;
    GFqDom<int64_t> GFq((TT)p, (TT)e);
    GFqDom<int64_t> PrimeField((TT)p,1);
    std::cout << "Working in GF(" << p << '^' << e << ') ' << std::endl;
    std::cout << "Elements are polynomials in X modulo " << p << std::endl;
    Poly1Dom< GFqDom<int64_t>, Dense > Pdom( PrimeField, Indeter("X") );
    // First get the irreducible polynomial irred
    // via irred = X^e - mQ
    // and X^e = X^(e-1) * X
    GFqDom<int64_t>::Element temo, t, tmp;
    Poly1Dom< GFqDom<int64_t>, Dense >::Element G, H, J, mQ, irred, modP;
    Pdom.init(G, Degree((int64_t)GFq.exponent()-1)); // X^(e-1)
    GFq.init(temo,G); // internal representation

```

```

Pdom.init(H, Degree(1) ); // X
GFq.init(t,H); // internal representation
GFq.init(tmp);
GFq.mul(tmp, temo, t); // internal representation of X^e
GFq.negin(tmp); // internal representation of -X^e, i.e. of the irreducible polynomial without
    the largest monomial, X^e
int64_t lowerpart;
// p-adic value of the lower part of the irreducible polynomial
GFq.convert(lowerpart, tmp);
std::cout << ' ' << p << "-adic value of the lower part of the irreducible : " << lowerpart << std::endl;
int64_t ptoe = power(p,e); // p-adic value of X^e
std::cout << ' ' << p << '^' << e << " is : " << ptoe << std::endl;
std::cout << " --> Computed irreducible: " << ptoe+lowerpart << std::endl;
std::cout << "Stored      irreducible: " << GFq.irreducible() << std::endl;
Poly1PadicDom< GFqDom<int64_t>, Dense > PAD(Pdom);
Poly1PadicDom< GFqDom<int64_t>, Dense >::Element Polynomial;
PAD.radix(Polynomial, GFq.irreducible());
std::cout << "Irreducible polynomial coefficients: ";
for(Poly1PadicDom< GFqDom<int64_t>, Dense >::Element::iterator it = Polynomial.begin(); it !=
    Polynomial.end(); ++it)
    PrimeField.write(std::cout << ' ', *it);
std::cout << std::endl;
PAD.write(std::cout << "The latter " << GFq.irreducible() << " represents: ", Polynomial)
<< " in " << p << "-adic"
<< std::endl;
return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.7 examples/FiniteField/gfq\_atomic.C

NO DOC.

NO DOC

```

// Copyright(c)1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <stdio.h>
#include <stdlib.h>
#include <givaro/givtimer.h>
#include <givaro/givrandom.h>
#include <givaro/gfq.h>
using namespace Givaro;
#ifndef GIVMIN
#define GIVMIN(a,b) (((a)<(b))?(a):(b))
#endif
typedef GFqDom<int64_t> Domain;
typedef GFqDom<int64_t>::Element Modulo;
// NB: number of iterations, TAILLE: vector size
#ifndef NB
#define NB 1000000
#endif
#ifndef TAILLE
#define TAILLE 256
#endif
int main(int argc, char ** argv)
{
    GivRandom generator;
    int64_t P (65521); // argv[1] : characteristic
    int64_t expo(1); // argv[2] : exponent
    int offset = 0;
    if (argc > ++offset) P = atoi( argv[ offset ] );
    if (argc > ++offset) expo = atoi( argv[ offset ] );
    Timer inver;
    inver.clear();
    inver.start();
    Domain GFq((uint64_t)P, (uint64_t)expo); // Buiding of finite field with P^expo Elements
    Modulo * z1 = new Modulo[TAILLE], * z2 = new Modulo[TAILLE], * z23 = new Modulo[TAILLE], * z3 = new
        Modulo[TAILLE];
    // int64_t seuil = GIVMIN(P*2,TAILLE);
    std::cout << "." << std::flush;
    for(int i=0; i<TAILLE; ++i){
        GFq.random(generator,z1[i]) ;
        GFq.nonzerorandom(generator,z2[i]) ;
        GFq.random(generator,z23[i]) ;
    }
    std::cout << "." << std::flush;
}

```

```

inver.stop();
std::cout << "Init et Tableaux des modulo " << P << " :\n" << inver << std::endl << std::flush;
double coef = (double)NB*TAILLE / 1e6;
Timer tim;tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.mul(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Mul: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.add(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Add: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axpyin(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd IN place: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axpy(z3[i], z1[i], z2[i], z23[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.sub(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Sub: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
#endif
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.div(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Div: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
#endif
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.neg(z3[i], z1[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Neg: " << (coef / tim.usertime())
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
for(int i=0; i<TAILLE; ++i){
    GFq.random(generator,z1[i]) ;
    GFq.nonzerorandom(generator,z2[i]) ;
    GFq.random(generator,z23[i]) ;
}
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.add(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Add: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.sub(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Sub: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axpyin(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd IN place: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axpy(z3[i], z1[i], z2[i], z23[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.neg(z3[i], z1[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Neg: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.8 examples/FiniteField/Test\_Extension.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include "givaro/givpoly1.h"
#include "givaro/extension.h"
using namespace Givaro;
template<class FField> void FaireEssai(const FField & F) {
    F.write( std::cout << "Working in : " ) << std::endl;
    typename FField::Element a, b, r;
    std::cout << "Enter an Element of this field: "; F.read( std::cin , a );
    // F.init( a, "1+3*X+5*X^2" );
    F.init( b, (Integer)23 );
    F.add(r, a, b);
    F.write( F.write( F.write( F.write(std::cout, a) << " + " , b ) << " = " , r ) << " with " ) << std::endl ;
}
template void FaireEssai< Extension<> >(const Extension<> & F) ;
template void FaireEssai< GFqDom<int64_t> >(const GFqDom<int64_t> & F) ;
int main (int argc, char * * argv) {
    uint64_t q = (argc>1?(uint64_t)atoi(argv[1]):13);
    uint64_t expo = (argc>2?(uint64_t)atoi(argv[2]):8);
    /*
        GFqDom<int64_t> Toto(q,1);
        Toto.write( std::cout << "This is " ) << std::endl ;
        FaireEssai( Toto );
    */
    std::cerr << "Exponent max for zech logs with characteristic " << q << " : " << FF_EXPONENT_MAX(q,expo) <<
        std::endl;
    std::cerr << "Sub-Exponent max for zech logs " << q << "^" << expo << " : " << FF_SUBEXPONENT_MAX(q,expo) <<
        std::endl;
    std::cerr << "NEED polynomial representation : " << NEED_POLYNOMIAL_REPRESENTATION(q,expo) << std::endl;
    if ( NEED_POLYNOMIAL_REPRESENTATION(q,expo) )
        FaireEssai( Extension<>(q, expo) );
    else
        FaireEssai( GFqDom<int64_t>(q, expo) );
    FaireEssai( EXTENSION(q, expo) );
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.9 examples/FiniteField/zpz\_atomic.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <stdio.h>
#include <stdlib.h>
#include <givaro/givtimer.h>
#include <givaro/givrandom.h>
#include <givaro/modular.h>
using namespace Givaro;
#ifndef GIVMIN
#define GIVMIN(a,b) (((a)<(b))?(a):(b))
#endif
typedef Modular<int32_t> Domain;
typedef Domain::Element Modulo;
#ifndef NB
#define NB 1000000
#endif
#ifndef TAILLE
#define TAILLE 256
#endif
int main(int argc, char ** argv)
{
    GivRandom generator;
```

```

long P (32749);      // argv[1] : Modulus
// int TAILLE (256); // argv[2] : Vector size
// long NB (500000); // argv[3] : Number of iterations
int offset = 0;
if (argc > ++offset) P = atoi( argv[ offset ] );
// if (argc > ++offset) TAILLE = atoi( argv[ offset ] );
// if (argc > ++offset) NB = atoi( argv[ offset ] );
Timer inver;
inver.clear();
inver.start();
Domain GFq((Domain::Residu_t)P);
std::cout << "." << std::flush;
Modulo * z1 = new Modulo[TAILLE], * z2 = new Modulo[TAILLE], * z23 = new Modulo[TAILLE], * z3 = new
Modulo[TAILLE];
// long seuil = GIVMIN(P*2,TAILLE);
std::cout << "." << std::flush;
for(int i=0; i<TAILLE; ++i){
    GFq.random(generator,z1[i]) ;
    GFq.nonzerorandom(generator,z2[i]) ;
    GFq.random(generator,z23[i]) ;
}
std::cout << "." << std::flush;
inver.stop();
std::cout << "Init et Tableaux des modulo " << P << " :\n" << inver << std::endl << std::flush;
double coef = (double)NB*TAILLE / 1e6;
Timer tim;tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.mul(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Mul: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.add(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Add: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axpyin(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd IN place: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axpy(z3[i], z1[i], z2[i], z23[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.sub(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Sub: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
#endif
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.div(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Div: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
#endif
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.neg(z3[i], z1[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Neg: " << (coef / tim.usertime())
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
for(int i=0; i<TAILLE; ++i){
    GFq.random(generator,z1[i]) ;
    GFq.nonzerorandom(generator,z2[i]) ;
    GFq.random(generator,z23[i]) ;
}
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.add(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Add: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.sub(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Sub: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();

```

```

for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axyin(z3[i], z1[i], z2[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd IN place: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.axy(z3[i], z1[i], z2[i], z23[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " MulAdd: " << 2*coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
tim.clear();tim.start();
for (int k=0; k<NB; ++k) for(int i=0; i<TAILLE; ++i)
    GFq.neg(z3[i], z1[i]);
tim.stop();
std::cout << NB << " * " << TAILLE << " Neg: " << coef / tim.usertime()
<< "Mop/s\n" << tim << ", ex: " << z3[0] << std::endl << std::flush;
return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.10 examples/Integer/iexponentiation.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/modular-integer.h>
#include <givaro/givpower.h>
#include <givaro/givtimer.h>
using namespace Givaro;
// Modular exponentiation
// argv[1] : a
// argv[2] : e
// argv[3] : p
// res ----> a^e % p
int main(int argc, char ** argv) {
    {
        Integer p(argv[3]);
        Modular<Integer> Zp( p );
        Modular<Integer>::Element a, b;
        unsigned long e = (unsigned long)atoi(argv[2]) ;
        Zp.init(a, Integer(argv[1]));
        Zp.init(b);
        Timer tim;tim.clear();tim.start();
        dom_power(b, a, (long)e, Zp);
        tim.stop();
        Zp.write( std::cout << '(' , a) << '^' << e << ") % " << p << '=' << std::flush;
        Zp.write( std::cerr, b) << std::endl;
        std::cerr << tim << std::endl;
    }
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.11 examples/Integer/ifactor.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>

```

```

#include <givaro/givinit.h>
#include <givaro/givintfactor.h>
#include <givaro/givtimer.h>
using namespace std;
using namespace Givaro;
int main(int argc, char** argv)
{
    IntFactorDom<> IP;
    Integer m;
    if (argc > 1)
        m = Integer(argv[1]);
    else
        cin >> m;
    if (IP.islt(m,0) ) {
        cerr << "-";
        IP.negin(m);
    }
    if (IP.islt(m,4))
        IP.write(cerr,m) << endl;
    else {
        Timer tim; tim.clear(); tim.start();
        IP.write(cerr,m) << endl;
        tim.stop();
        cerr << tim << endl;
    }
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.12 examples/Integer/ifactor\_lenstra.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#define GIVARO_LENSTRA
#include <iostream>
using namespace std;
#include <givaro/givinit.h>
#include <givaro/givintfactor.h>
#include <givaro/givtimer.h>
using namespace Givaro;
int main(int argc, char** argv)
{
    IntFactorDom<> IP;
#ifdef __GIVARO_GMP_NO_CXX
    IP.seeding();
    // std::cerr << "Seeding..." << std::endl;
#endif
    Integer m;
    if (argc > 1)
        m = Integer(argv[1]);
    else
        cin >> m;
    if (IP.islt(m,0) ) {
        cerr << "-";
        IP.negin(m);
    }
    if (IP.islt(m,4))
        IP.write(cerr,m) << endl;
    else {
        Timer tim; tim.clear(); tim.start();
        IP.write(cerr,m) << endl;
        tim.stop();
        cerr << tim << endl;
    }
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.13 examples/Integer/igcd.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>           // Givaro initialization
using namespace Givaro;
int main(int argc, char** argv)
{
    // Givaro::Init(&argc, &argv);
    IntegerDom IP;
    IntegerDom::Element GG, g,a,b;
    int offset = 0;
    if (argc > ++offset) a = Integer(argv[offset]); else cin >> a;
    if (argc > ++offset) b = Integer(argv[offset]); else cin >> b;
    Timer tim; tim.clear(); tim.start();
    IP.gcd(GG,a,b);
    for ( ; argc > ++offset; ) {
        a = Integer(argv[offset]);
        IP.gcd(g, GG, a);
        GG = g;
    }
    tim.stop();
    cout << GG << endl;
    cerr << tim << endl;
    // Givaro::End();
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.14 examples/Integer/igcdext.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>           // Givaro initialization
using namespace Givaro;
int main(int argc, char** argv)
{
    // Givaro::Init(&argc, &argv);
    IntegerDom IP;
    IntegerDom::Element g,a,b,u,v;
    if (argc > 1) a = Integer(argv[1]); else cin >> a;
    if (argc > 2) b = Integer(argv[2]); else cin >> b;
    Timer tim; tim.clear(); tim.start();
    IP.gcd(g,u,v,a,b);
    tim.stop();
    cout << g << " = " << u << " * " << a << " + " << v << " * " << b << endl;
    cerr << tim << endl;
    // Givaro::End();
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.15 examples/Integer/ilcm.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>           // Givaro initialization
using namespace Givaro;
int main(int argc, char** argv)
{
    // Givaro::Init(&argc, &argv);
    IntegerDom IP;
    IntegerDom::Element CM, g,a,b, c;
    int offset = 0;
    if (argc > ++offset) a = Integer(argv[offset]); else cin >> a;
    if (argc > ++offset) b = Integer(argv[offset]); else cin >> b;
    Timer tim; tim.clear(); tim.start();
    IP.gcd(g,a,b);
    IP.mul(CM, a, b);
    IP.divin(CM, g);
    for ( ; argc > ++offset; ) {
        c = Integer(argv[offset]);
        IP.gcd(g, CM, c);
        IP.divin(CM, g);
        IP.mulin(CM, c);
    }
    tim.stop();
    cout << CM << endl;
    cerr << "gcd+mul: " << tim << endl;
    tim.clear(); tim.start();
    IP.lcm(CM,a,b);
    // cerr << "lcm(" << a << ", " << b << ") = " << CM << endl;
    for ( offset = 2; argc > ++offset; ) {
        c = Integer(argv[offset]);
        // cerr << "lcm(" << c << ", " << CM << ") = " ;
        IP.lcmin(CM, c);
        // cerr << CM << endl;
    }
    tim.stop();
    cout << CM << endl;
    cerr << "lcm: " << tim << endl;
    // Givaro::End();
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:src:cin=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.16 examples/Integer/ispower.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <stdlib.h>
#include <givaro/givinteger.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
using namespace Givaro;
int main(int argc, char** argv)
{
    Integer m, p;
    if (argc > 1) m = Integer(argv[1]);
    IntPrimeDom IP;
```

```

{
    Timer tim; tim.clear(); tim.start();
    int a = isperfectpower(m);
    tim.stop();
    std::cout << a << std::endl;
    std::cerr << tim << std::endl;
}
{
    Timer tim; tim.clear(); tim.start();
    int a = (int)IP.isprimepower(p, m);
    tim.stop();
    if (a) std::cout << "is " << p << "^" << a << std::endl;
    else std::cout << "not a prime power" << std::endl;
    std::cerr << tim << std::endl;
}
return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,{0,\:0,t0,+0,=s

```

## 19.17 examples/Integer/isprime.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h> // Givaro initialization
using namespace Givaro;
int main(int argc, char** argv)
{
    // Givaro::Init(&argc, &argv);
    IntPrimeDom IP;
    IntPrimeDom::Element m;
    if (argc > 1) m = Integer(argv[1]);
    unsigned int r = argc > 2 ? (unsigned int)atoi(argv[2]) : 5;
    Timer tim; tim.clear(); tim.start();
    bool a = IP.isprime(m, (int)r);
    tim.stop();
    cout << (a?"true":"false") << endl;
    cerr << tim << endl;
    // Givaro::End();
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,{0,\:0,t0,+0,=s

```

## 19.18 examples/Integer/isproot.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
using namespace Givaro;
//-----
// Deterministic, non-polynomial (factor b-1 for the order),
// test for primitive roots.
//-----

```

```

int main(int argc, char** argv)
{
    IntNumTheoDom<> IP;
    IntNumTheoDom<>::Element a,b;
    if (argc > 1) a = Integer(argv[1]); else std::cin >> a;
    if (argc > 2) b = Integer(argv[2]); else std::cin >> b;
    Timer tim; tim.clear(); tim.start();
    bool f = IP.is_prim_root(a,b);
    tim.stop();
    std::cout << f << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.19 examples/Integer/lambda.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
using namespace Givaro;
// Lambda function : order of a primitive Element
//                  (Element of maximal orbit size)
//
int main(int argc, char** argv)
{
    IntNumTheoDom<> IP;
    IntNumTheoDom<>::Element a,pr;
    if (argc > 1) a = IntNumTheoDom<>::Element(argv[1]); else std::cin >> a;
    Timer tim; tim.clear(); tim.start();
    IP.lambda(pr, a);
    tim.stop();
    IntegerDom().write( std::cout, pr ) << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.20 examples/Integer/lambda\_inv.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
using namespace Givaro;
// Lambda function : order of a primitive invertible
//                  (invertible Element of maximal orbit size)
//
int main(int argc, char** argv)
{
    IntNumTheoDom<> IP;
    IntNumTheoDom<>::Element a,pr;
    if (argc > 1) a = IntNumTheoDom<>::Element(argv[1]); else std::cin >> a;
    Timer tim; tim.clear(); tim.start();
    IP.lambda_inv(pr, a);
    tim.stop();
    IntegerDom().write( std::cout, pr ) << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.21 examples/Integer/ModularSquareRoot.C

NO DOC.

NO DOC

```
// ===== //
// Copyright (c) 1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
// Time-stamp: <21 Feb 12 17:02:32 Jean-Guillaume.Dumas@imag.fr>
// Givaro : Modular square roots
// ===== //
#include <iostream>
#include <stdlib.h>
#include <givaro/givintsqrootmod.h>
#include <givaro/givtimer.h>
#include <givaro/givpolylfactor.h>
#include <givaro/modular-integer.h>
using namespace Givaro;
// Algorithm 3.34 (Square Root Mod p) of
// Handbook of Applied Cryptography
// by Menezes, van Oorschot, Vanstone
int main(int argc, char** argv)
{
    Integer a(argv[1]), n(argv[2]);
    IntSqrtModDom<> ISM;
    {
        std::cerr << "a: " << a << std::endl;
        std::cerr << "n: " << n << std::endl;
        Integer::seeding ( (unsigned long)BaseTimer::seed ());
        Integer r;
        Timer chrono; chrono.start();
        ISM.sqrootmod(r,a,n);
        chrono.stop();
        std::cout << r << std::endl;
        std::cerr << chrono << std::endl;
        std::cerr << "Check, (" << r << ")^2 mod " << n << " = " << ( (r*r)%n) << std::endl;
    }
    if (ISM.isprime(n)) {
        std::cout << "Using polynomial factorization : " << std::endl;
        typedef Modular<Integer> Field;
        typedef Poly1FactorDom<Field,Dense> Polys;
        typedef Polys::Element Polynomial;
        Field F(n); Polys Pol(F, "X");
        Polynomial quad, root;
        Pol.init(quad, Degree(2));
        F.init(quad[0],a); F.negin(quad[0]);
        Timer chrono; chrono.start();
        if (Pol.is_irreducible(quad)) {
            std::cerr << "a << " is not a quadratic residue mod " << n << std::endl;
            chrono.stop();
            std::cerr << chrono << std::endl;
            return 0;
        }
        Pol.SplitFactor(root, quad, Degree(1));
        chrono.stop();
        Pol.divin(root, root[1]);
        Pol.write(std::cout, root) << std::endl;
        std::cerr << chrono << std::endl;
        std::cerr << "Check, (" << root[0] << ")^2 mod " << n << " = " << ( (root[0]*root[0])%n) << std::endl;
    }
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.22 examples/Integer/nb\_primes.C

NO DOC.

NO DOC

```
// Copyright (c) 1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
```

```
// see the COPYRIGHT file for more details.
#include <iostream>
#include "givaro/givintprime.h"
#include "givaro/givtimer.h"
using namespace Givaro;
inline Integer GIVMAX(const Integer& a, const Integer& b) { return (a<b?b:a); }
int main(int argc, char** argv)
{
    IntPrimeDom IPD;
    Integer m, tp = 2, ttp=1, np;
    if (argc > 1)
        np = Integer(argv[1]);
    else
        std::cin >> np;
    if (argc > 2)
        m = GIVMAX(1, Integer(argv[2]));
    else
        m = 2;
    Integer nf = m;
    unsigned long long nb = (IPD.isprime(m)?1:0);
    Timer tim; tim.clear(); tim.start();
    for (; m < np; tp *= 2) {
        std::cout << nb << " primes between " << nf << " and " << m << "\t\t == nextprime(" << (ttp+nf-1) << ")" <<
        std::endl; ttp = tp;
        for (; (m < np) && (m < (nf+tp)); ++nb)
            IPD.nextprimein(m);
    }
    tim.stop();
    std::cout << (m>np?nb-1:nb) << " primes between " << nf << " and " << np << std::endl << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.23 examples/Integer/nextprime.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h> // Givaro initialization
using namespace Givaro;
int main(int argc, char** argv)
{
    // Givaro::Init(&argc, &argv);
    IntPrimeDom IP;
    IntPrimeDom::Element m, ff;
    if (argc > 1) m = Integer(argv[1]);
    else std::cin >> m;
    Timer tim; tim.clear(); tim.start();
    IP.nextprimein(m,1);
    tim.stop();
    cout << m << endl;
    cerr << tim << endl;
    // Givaro::End();
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.24 examples/Integer/order.C

NO DOC.

NO DOC

```
// ===== //
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
// Time-stamp: <04 Sep 02 18:10:22 Jean-Guillaume.Dumas@imag.fr>
// ===== //
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
using namespace Givaro;
int main(int argc, char ** argv)
{
    IntNumTheoDom<> IP;
    IntNumTheoDom<>::Element a,q,o;
    if (argc > 1) a = Integer(argv[1]); else cin >> a;
    if (argc > 2) q = Integer(argv[2]); else cin >> q;
    Timer tim; tim.clear(); tim.start();
    // Ordre de a dans GF(q)
    IP.order(o, a, q);
    tim.stop();
    cout << o << endl;
    cerr << tim << endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.25 examples/Integer/phi.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
// Euler's phi function (totient)
//
using namespace Givaro;
int main(int argc, char** argv)
{
    IntNumTheoDom<> IP;
    IntNumTheoDom<>::Element a,pr;
    if (argc > 1) a = IntNumTheoDom<>::Element(argv[1]); else std::cin >> a;
    Timer tim; tim.clear(); tim.start();
    IP.phi(pr, a);
    tim.stop();
    IntegerDom().write( std::cout, pr ) << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.26 examples/Integer/prevprime.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
```

```

using namespace std;
#include <stdlib.h>
#include <givaro/givintprime.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>           // Givaro initialization
using namespace Givaro;
int main(int argc, char** argv)
{
    // Givaro::Init(&argc, &argv);
    IntPrimeDom IP;
    IntPrimeDom::Element m, ff;
    if (argc > 1) m = Integer(argv[1]);
    else std::cin >> m;
    Timer tim; tim.clear(); tim.start();
    IP.pvprimein(m);
    tim.stop();
    cout << m << endl;
    cerr << tim << endl;
    // Givaro::End();
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.27 examples/Integer/primitiveelement.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
using namespace Givaro;
int main(int argc, char** argv)
{
    IntNumTheoDom<> IP;
    IntNumTheoDom<>::Element a,pr;
    if (argc > 1) a = IntNumTheoDom<>::Element(argv[1]); else std::cin >> a;
    Timer tim; tim.clear(); tim.start();
    IP.prim_elem(pr, a);
    tim.stop();
    IntegerDom().write( std::cout, pr ) << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.28 examples/Integer/primitiveroot.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#define GIVARO_LENSTRA
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
#ifdef TIMING
#define TIMING 1
#endif
using namespace Givaro;
int main(int argc, char** argv)
{

```

```

    IntNumTheoDom<> IP;
#ifdef __GIVARO_GMP_NO_CXX
    IP.seeding();
#endif
    IntNumTheoDom<>::Element a,pr;
    if (argc > 1) a = IntNumTheoDom<>::Element(argv[1]); else std::cin >> a;
    uint64_t runs;
    Timer tim; tim.clear();
    if (IP.isprime(a)) {
        Integer phin; IP.sub(phin,a,IP.one);
        std::vector<Integer> Lf;
        IP.write(std::cout << "Totient : ", Lf,phin) << std::endl;
        tim.start();
        for(uint64_t i = 0; i < TIMING; ++i)
            IP.prim_root_of_prime(pr, a);
        tim.stop();
        IP.write( std::cout << "Deterministic   : ", pr ) << std::endl;
        std::cerr << tim << std::endl;
    }
    tim.start();
    for(uint64_t i = 0; i < TIMING; ++i)
        IP.prim_root(pr, runs, a);
    tim.stop();
    IP.write( std::cout << "Random : ", pr ) << std::endl;
    std::cerr << tim << " (" << runs << " runs)" << std::endl;
    tim.start();
    for(uint64_t i = 0; i < TIMING; ++i)
        IP.lowest_prim_root(pr, a);
    tim.stop();
    IP.write( std::cout << "Lowest : ", pr ) << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.29 examples/Integer/probable\_primroot.C

NO DOC.

NO DOC

```

// Copyright(c) 1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
#include <cmath>
using namespace Givaro;
// Polynomial-time generation of primitive roots
// L is number of loops of Pollard partial factorization of n-1
// 10,000,000 gives at least 1-2^{-40} probability of success
// [Dubrois & Dumas, Industrial-strength primitive roots]
// Returns the probable primitive root and the probability of error.
int main(int argc, char** argv)
{
    IntNumTheoDom<> IP;
#ifdef __GMP_PLUSPLUS__
    IP.seeding( (unsigned long)BaseTimer::seed() );
#endif
    double error;
    IntNumTheoDom<>::Element a,pr,g;
    if (argc > 1) a = IntNumTheoDom<>::Element(argv[1]); else std::cin >> a;
    bool comp ; if ( (comp=! IP.isprime(a))) ) std::cerr << a << " is not prime, primitive root will have no
        sense and may loop forever ..." << std::endl;
    double epsilon = argc > 2 ? atof(argv[2]) : 0.0000001;
    Timer tim; tim.clear();
    tim.start();
    //=====
    // Default is partial factorization up to factors of at least 12 digits.
    // with probability of error much less than 2^{-40}
    // IP.probable_prim_root(pr, error, a );
    //=====
    // Choosing L to be O(log^2(p))
    // gives the best probability with O(log^4(p)) complexity
    // Probability of error is approximately O(1/log^4(p))
    // IP.probable_prim_root(pr, error, a, (unsigned long)power(logtwo(a),2));
    //=====

```

```

// Choosing L to be O( \sqrt(epsilon) )
// gives probability of error at most epsilon
// Newton-Raphson iteration is used for
// 1-epsilon = (1+2/(p-1))*(1-1/B)^(ln( (p-1)/2 )/ln(B))
// So that no factor less than B can be avoided
// With Pollard's rho factorization, L is chosen to be sqrt(B)
// Might not be polynomial if epsilon is too big
#define GIVARO_POLLARD
IP.probable_prim_root(pr, error, a, epsilon );
tim.stop();
if (comp) std::cerr << IP.gcd(g,a,pr) << " is a factor of " << a << std::endl;
IntegerDom().write( std::cout << "Prim root    : ", pr );
if (error > 0) {
    std::cout << ", correct with probability at least : 1-" << error << std::endl;
    std::cerr << tim << std::endl;
    std::cerr << "Now checking primitivity, this may take some time (complete factorization of n-1) ...";
#define GIVARO_LENSTRA
Timer verif; verif.clear(); verif.start();
if ( IP.isorder(a-1, pr, a) ) {
    verif.stop();
    std::cerr << "... Pimitivity checked" << std::endl;
    std::cerr << verif << std::endl;
}
else {
    verif.stop();
    std::cerr << "... WARNING : FAILURE" << std::endl;
    std::cerr << verif << std::endl;
}
} else {
    std::cout << ", deterministically correct" << std::endl;
    std::cerr << tim << std::endl;
}
return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.30 examples/Integer/ProbLucas.C

NO DOC.

NO DOC

```

// ===== //
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
// Contributor: Jack Dubrois < Jacques.Dubrois@imag.fr>
// Time-stamp: <12 Jun 15 18:24:19 Jean-Guillaume.Dumas@imag.fr>
//
// Primality check using Probabilistic Lucas ////////////////
// i.e. Primitive Root with choosen probability ////////////////
// ===== //
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <stdio.h>
using namespace std;
#include <stdlib.h>
#include <cmath>
#include <givaro/givintprime.h>
#include <givaro/givintnumtheo.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>
#include <givaro/modular-integer.h>
#include <givaro/givinteger.h>
#include <givaro/givrandom.h>
#include <givaro/givinteger.h>
using namespace Givaro;
IntFactorDom<> IP;
#define GIVABSDIV(a,b) ((a)<(b)?((a)/(b)):((b)/(a)))
#ifndef GIVMIN
#define GIVMIN(a,b) ((a)<(b)?(a):(b))
#endif
#define ProbLucas_factor_first_primes(tmp,n) (tmp = IP.isZero(IP.mod(tmp,n,23)) ? 23: (
    IP.isZero(IP.mod(tmp,n,19)) ? 19: ( IP.isZero(IP.mod(tmp,n,17)) ? 17: ( IP.isZero(IP.mod(tmp,n,2)) ? 2: (
    IP.isZero(IP.mod(tmp,n,3)) ? 3: ( IP.isZero(IP.mod(tmp,n,5)) ? 5: ( IP.isZero(IP.mod(tmp,n,7)) ? 7: (
    IP.isZero(IP.mod(tmp,n,11)) ? 11: 13 ))))))))

```

```

#define ProbLucas_factor_second_primes(tmp,n) (tmp = IP.isZero(IP.mod(tmp,n,31)) ? 31 : (
    IP.isZero(IP.mod(tmp,n,29)) ? 29 : ( IP.isZero(IP.mod(tmp,n,37)) ? 37 : ( IP.isZero(IP.mod(tmp,n,41)) ? 41 : (
    IP.isZero(IP.mod(tmp,n,43)) ? 43 : ( IP.isZero(IP.mod(tmp,n,71)) ? 71 : ( IP.isZero(IP.mod(tmp,n,67)) ? 67 : (
    IP.isZero(IP.mod(tmp,n,61)) ? 61 : ( IP.isZero(IP.mod(tmp,n,59)) ? 59 : ( IP.isZero(IP.mod(tmp,n,53)) ? 53 : (
    IP.isZero(IP.mod(tmp,n,47)) ? 47 : ( IP.isZero(IP.mod(tmp,n,97)) ? 97 : ( IP.isZero(IP.mod(tmp,n,89)) ? 89 : (
    IP.isZero(IP.mod(tmp,n,83)) ? 83 : ( IP.isZero(IP.mod(tmp,n,79)) ? 79 : 73))))))))))
Integer& MyPollard(GivRandom& gen, Integer& g, const Integer& n, const unsigned long threshold)
{
    g=1;
    Integer m(0U), x, y, t;
    static Integer PROD_first_primes(223092870);
    static Integer PROD_second_primes("10334565887047481278774629361");
    if (isOne(gcd(y,n,PROD_first_primes))) {
        if (isOne(gcd(y,n,PROD_second_primes))) {
            IP.random(gen, y, n);
            unsigned long p(1);
            for(unsigned long c = 0; isOne(g) && (++c < threshold); ) {
                if( p == c ) {
                    x=y;
                    p <<= 1;
                }
                // Pollard fctin
                IP.mulin(y,y);
                IP.addin(y,1U);
                IP.modin(y,n);
                gcd(g,IP.sub(t,y,x),n);
            }
            return g;
        } else {
            return ProbLucas_factor_second_primes(g,n);
        }
    } else {
        return ProbLucas_factor_first_primes(g,n);
    }
}

unsigned long Revert(const Integer p, const double epsilon, double firstguess)
{
    // unsigned long L;
    double t1, t4, t8, t34(firstguess), dL;
    t1 = (double)p-1.0;
    t4 = 2.0/t1+1.0;
    t8 = logtwo(p)*log(2.0)-log(2.0); // log( p/2 )
    do {
        double t3, t5, t7, t9, t11, t12, t18, t22, t23 ;
        // std::cerr << "dL: " << t34 << std::endl;
        dL = t34;
        t3 = 1.0/dL;
        t5 = t3*t3;
        t7 = 1.0-t5;
        t9 = 2.0*log(dL);
        t11 = t8/t9;
        t12 = ::pow(t7,t11);
        t18 = t9*t9;
        t22 = log(t7);
        t23 = (-t8/t18*t3*t22+t11*t5*t3/t7);
        t34 = dL-( 1.0 - (1.0-epsilon)/(t4*t12))/(2.0*t23);
    } while( (GIVABSDIV(t34,dL) < 0.95) && (dL<1048576.0) );
    return /* L =*/ (unsigned long)GIVMIN(dL,1048576.0);
}

unsigned long Revert(const Integer p, const double epsilon)
{
    return Revert(p, epsilon, ::sqrt(1.2/epsilon));
}

bool ProbLucas(const Integer n, const double orig_epsilon)
{
#ifdef __GMP_PLUSPLUS__
    Integer::seeding( (unsigned long)BaseTimer::seed() );
#endif
    GivRandom generator;
    Integer Q=n-1,a,q,tmp(1);
    Integer nmu=Q;
    double P = 1.0;
    double epsilon = orig_epsilon;
    // Miller-Rabin
    unsigned long s=0;
    for( ; !( (int)Q & 0x1) ; Q*=1, ++s) { }
    for(unsigned int i=0; ; ) {
        IP.random(generator,a,n);
        IP.powmod(tmp,a,Q,n);
        if (tmp!=1) {
            if (tmp != nmu) {
                for(i=(unsigned int) s; --i>0;) {
                    tmp *= tmp;
                    tmp %= n;
                    if ((tmp == 1) || (tmp == nmu))
                        break;
                }
            }
        }
    }
}

```

```

        if (tmp != nmu)
            return 0;
        if (i == 0)
            break;
    }
    else {
        if (s == 1) break;
    }
}
epsilon *= 4.0;
P /= 2.0;
}
unsigned long L = Revert(Q,epsilon);
Integer trn, r, b, expo; root( trn, n, 3); trn *= trn;
q = 2;
expo = nmu;
MyPollard(generator, q, Q, L);
if (q == 1) {
    q = Q;
    expo = nmu; expo /= q;
    IP.random(generator, a, n);
    IP.powmod(tmp,a,expo,n);
    if (tmp == 1) {
        P /= (double)(q);
        if (P < epsilon) {
            std::cerr << "Composite with probability > 1-" << P << std::endl;
            return 0;
        }
    }
    Q = 1;
} else {
    expo = nmu; expo /= q;
    r=0;
    Integer::divexact(b, Q, q);
    while( isZero(r) ) {
        Q.copy(b);
        Integer::divmod( b, r, Q, q );
    }
    L = Revert(Q,epsilon, (double)L);
}
while ( Q > trn ) {
    IP.random(generator, a, n);
    IP.powmod(tmp,a,expo,n);
    if (tmp == 1) {
        P /= (double)(q);
        if (P < epsilon) {
            std::cerr << "Composite with probability > 1-" << P << std::endl;
            return 0;
        }
    }
    else {
        if (IP.gcd(q,(tmp-1U),n) != 1) {
            std::cerr << "Factor found : " << q << std::endl;
            return 0;
        }
        MyPollard(generator, q, Q, L);
        if (q == 1) {
            q = Q;
            expo = nmu; expo /= q;
            IP.random(generator, a, n);
            IP.powmod(tmp,a,expo,n);
            if (tmp == 1) {
                P /= (double)(q);
                if (P < epsilon) {
                    std::cerr << "Composite with probability > 1-" << P << std::endl;
                    return 0;
                }
            }
            break;
        }
        else {
            expo = nmu; expo /= q;
            r=0;
            Integer::divexact(b, Q, q);
            while( isZero(r) ) {
                Q.copy(b);
                Integer::divmod( b, r, Q, q );
            }
            L = Revert(Q,epsilon, (double)L);
        }
    }
}
if (! IP.isprime(q))
    std::cerr << "Prime with probability > 1-" << orig_epsilon << std::endl;
return 1;
}
int main (int argc, char * * argv)
{
    Integer P;

```

```

if (argc > 1) P = Integer(argv[1]); else std::cin >> P;
double epsilon = argc > 2 ? atof(argv[2]) : 0.000001;
unsigned int NB = argc > 3 ? (unsigned int)atoi(argv[3]) : 1U;
//      std::cerr << "P: " << P << " "; proba: " << epsilon << std::endl;
bool al(true);
Timer tim; tim.clear();
for(unsigned int i = 0; i < NB; ++i) {
    Timer tt; tt.clear(); tt.start();
    al = ProbLucas(P, epsilon);
    tt.stop();
    //      std::cout << al << std::endl;
    //      std::cerr << tt << std::endl;
    tim += tt;
}
std::cout << (al?"prime":"composite") << endl;
std::cerr << tim << std::endl;
return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.31 examples/Integer/RSA\_breaking.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
using namespace std;
#define GIVARO_LENSTRA
#include "givaro/givintrsa.h"
#include "givaro/givtimer.h"
using namespace Givaro;
int main(int argc, char** argv)
{
    Timer tim;
    tim.clear();
    IntRSADom<>::Element m,k,u;
    if (argc > 1)
        m = IntRSADom<>::Element( argv[1] );
    else
        cin >> m;
    if (argc > 2)
        k = IntRSADom<>::Element( argv[2] );
    else
        cin >> k;
    IntRSADom<> IR(m,k);
    tim.start();
    IR.point_break(u);
    tim.stop();
    /* For a factored output :

        IR.write( cerr << "m=pq: ", IR.getm() ) ;
        IR.write( cerr << " , cipher k: " , IR.getk() ) << " ----> deciphering key: " ;
        IR.write( cout, u ) << endl;

    */
    // Unfactored output
    cerr << "n=pq: " << IR.getn() << " , cipher key: " << IR.getk() << " ----> deciphering key: ";
    cout << u << endl;
    cerr << tim << endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.32 examples/Integer/RSA\_decipher.C

NO DOC.

## NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <fstream>
#include "givaro/givintrsa.h"
#include "givaro/givrandom.h"
#include "givaro/givtimer.h"
// RSA, in CBC mode, deciphering of files
using namespace Givaro;
int main(int argc, char** argv)
{
    Timer tim;
    tim.clear();
    IntrSADom<GivRandom>::Rep n,e,d;
    if (argc > 3)
        n = Integer( argv[3] );
    else
        std::cin >> e;
    if (argc > 4)
        e = Integer( argv[4] );
    else
        std::cin >> e;
    if (argc > 5)
        d = Integer( argv[5] );
    else
        std::cin >> d;
    IntrSADom<GivRandom> IR(n,e,d);
    std::ifstream TXT(argv[1]);
    if (!TXT) { std::cerr << "Error opening input file: " << argv[1] << std::endl; return -1; }
    std::ofstream OUT(argv[2]);
    if (!OUT) { std::cerr << "Error opening output file: " << argv[2] << std::endl; return -1; }
    tim.start();
    IR.decipher( OUT, TXT );
    OUT.close();
    TXT.close();
    tim.stop();
    std::cerr << tim << std::endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.33 examples/Integer/RSA\_encipher.C

## NO DOC.

## NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <fstream>
#include "givaro/givintrsa.h"
#include "givaro/givrandom.h"
#include "givaro/givtimer.h"
// RSA, in CBC mode, enciphering of files
using namespace Givaro;
int main(int argc, char** argv)
{
    Timer tim;
    tim.clear();
    IntrSADom<GivRandom>::Rep n,e;
    if (argc > 3)
        n = Integer( argv[3] );
    else
        std::cin >> n;
    if (argc > 4)
        e = Integer( argv[4] );
    else
        std::cin >> e;
    IntrSADom<GivRandom> IR(n,e);
    std::ifstream TXT(argv[1]);
    if (!TXT) { std::cerr << "Error opening input file: " << argv[1] << std::endl; return -1; }
    std::ofstream OUT(argv[2]);
```

```

    if (!OUT) { std::cerr << "Error opening output file: " << argv[2] << std::endl; return -1; }
    tim.start();
    IR.encipher( OUT, TXT );
    OUT.close();
    TXT.close();
    tim.stop();
    std::cerr << tim << std::endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.34 examples/Integer/RSA\_keys\_generator.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include "givaro/givintrsa.h"
#include "givaro/givrandom.h"
#include "givaro/givtimer.h"
using namespace Givaro;
int main(int argc, char** argv)
{
    Timer tim;
    tim.clear();
    // m will be p times q
    // Sizes are in number of int to compose the big integer
    long keysize;
    if (argc > 1)
        keysize = atoi( argv[1] );
    else
        std::cin >> keysize;
    GivRandom generator;
    Integer::seeding(generator.seed());
    tim.start();
    IntRSADom<GivRandom> IR(keysize,true,generator);
    tim.stop();
    std::cout << IR.getn() << " " << IR.gete() << " " << IR.getd() << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.35 examples/Polynomial/pol\_arith.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/gfq.h>
#include <givaro/givpoly1.h>
using namespace Givaro;
int main(int argc, char ** argv) {
    {
        GFqDom<int> Z13( 13, 1 ); // integers modulo 13
        GFqDom<int>::Element tmp;
        // Polynomials over Z13, with X as indeterminate
        Poly1Dom< GFqDom<int>, Dense > DP13( Z13, Indeter("X") );
        Poly1Dom< GFqDom<int>, Dense>::Element P, Q, R, monomial;
        // DP13.read( std::cin, P); // would read P as a succession of integers :
        // deg leadcoeff (lead-1)coeff ... unitcoeff
    }
}

```

```

DP13.init(P, {5,-33,12}); // P is now 5-33*X+12*X^2
DP13.write( std::cout << "P: " , P )<< std::endl;
DP13.assign( Q, Z13.init(tmp,6) ); // Q is degree 0 polynomial : 6 modulo 13
DP13.write( std::cout << "Q: " , Q )<< std::endl;
DP13.init( monomial, Degree(4), 3U);
DP13.write( std::cout << "m: " , monomial )<< std::endl;
DP13.addin( Q, monomial) ;
DP13.write( std::cout << "Q: " , Q )<< std::endl;
DP13.init( monomial, Degree(1), 75U);
DP13.write( std::cout << "m: " , monomial )<< std::endl;
DP13.addin( Q, monomial) ;
DP13.write( std::cout << "Q: " , Q )<< std::endl;
DP13.init( monomial, Degree(3), 45U);
DP13.write( std::cout << "m: " , monomial )<< std::endl;
DP13.subin( Q, monomial) ;
DP13.write( std::cout << "Q: " , Q )<< std::endl;
// Q is now 3*X^4+75*X-45*X^3+6
DP13.mul ( R, P, Q); // R = P*Q;
DP13.write( DP13.write(
    std::cout << "( " , P ) << " ) * ( " , Q ) << " )";
DP13.write(std::cout << " = " , R) << std::endl;
DP13.gcd ( R, P, Q); //
DP13.write( DP13.write( DP13.write(
    std::cout << "gcd(" , P ) << " , " , Q ) << " ) = " , R) << std::endl;

DP13.lcm ( R, P, Q); //
DP13.write( DP13.write( DP13.write(
    std::cout << "lcm(" , P ) << " , " , Q ) << " ) = " , R) << std::endl;

DP13.lcm ( R, Q, P); //
DP13.write( DP13.write( DP13.write(
    std::cout << "lcm(" , Q ) << " , " , P ) << " ) = " , R) << std::endl;
}
return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:src:cin=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.36 examples/Polynomial/highorder.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#define GIVARO_DEBUG 1
#include <givaro/givrandom.h>
#include <givaro/givtimer.h>
#include <givaro/gfq.h>
#include <givaro/givpoly1.h>
#include <givaro/givtruncdomain.h>
#include <givaro/givhighorder.h>
using namespace Givaro;
typedef GFqDom<int> Field;
typedef Poly1Dom< Field, Dense> Polys;
typedef HighOrder< Field > HighOrders;
long long TFDcount = 0;
long long FiDcount = 0;
Timer Ttaylor, Thighorder, Tfiduccia;
bool TestFracDevel(const HighOrders& HO101, const Polys::Element P, const Polys::Element oQ, Degree a,
    Degree b) {
    bool success = false, successF=false;
    try {
        ++TFDcount;
        // std::cerr << "-----" << TFDcount
        <<std::endl;
        HighOrders::Truncated TQ;
        Degree dp; HO101.getpoldom().degree(dp, P);
        HO101.gettruncdom().assign(TQ, oQ,0,dp-1);
        Polys::Element Q;
        HO101.gettruncdom().convert(Q, TQ);
        //HO101.write(std::cout << "Q:=", TQ) << ' '; << std::endl;
        HighOrders::Element Fra;
        Fra._num = Q;
        Fra._den = P;
        Polys::Element TTy;
        Timer tt; tt.clear(); tt.start();
        HO101.taylor(TTy, Fra, b);
    }
}

```

```

tt.stop();
Ttaylor+=tt;
HighOrders::Truncated TTy17;
HO101.gettruncdom().assign(TTy17, TTy, a, b);
//HO101.write(std::cout << "TTy17:=", TTy17) << ';' << std::endl;
HighOrders::Truncated TTy17b;
tt.clear(); tt.start();
HO101.FracDevel(TTy17b, Fra, a, b);
tt.stop();
Thighorder+=tt;
//HO101.write(std::cout << "TTy17b:=", TTy17b) << ';' << std::endl;
success = HO101.gettruncdom().areEqual(TTy17, TTy17b);
if (! success) {
    std::cerr << "----- BEG ERROR -----" << std::endl;
    HO101.getpoldom().write(std::cerr << "Q:=", Q) << ';' << std::endl;
    HO101.getpoldom().write(std::cerr << "P:=", P) << ';' << std::endl;
    HO101.getpoldom().write(std::cerr << "TTy:=", TTy) << ';' << std::endl;
    //      Fra._num = HO101.getpoldom().one;
    //      Polys::Element Tay;
    //      HO101.taylor(Tay, Fra, b);
    //      HO101.getpoldom().write(std::cerr << "Tay:=", Tay) << ';' << std::endl;
    HO101.write(std::cerr << "TTy17:=", TTy17) << ';' << std::endl;
    HO101.write(std::cerr << "TTy17b:=", TTy17b) << ';' << std::endl;
    std::cerr << "----- END ERROR -----" << std::endl;
}
HighOrders::Truncated TTy17c; HO101.gettruncdom().init(TTy17c);
HighOrders::Truncated TTy17d; HO101.gettruncdom().init(TTy17d);
tt.clear(); tt.start();
//      HO101.Fiduccia(TTy17c, Fra, b);
//      for(Degree d=a; d<b; ++d) {
//          HO101.Fiduccia(TTy17d, Fra, d);
//          HO101.gettruncdom().addn(TTy17c, TTy17d);
//      }
HO101.Fiduccia(TTy17c, Fra, a, b);
tt.stop();
Tfiduccia+=tt;
//HO101.write(std::cout << "TTy17b:=", TTy17b) << ';' << std::endl;
successF = HO101.gettruncdom().areEqual(TTy17, TTy17c);
if (! successF) {
    std::cerr << "----- BEG ERROR -----" << std::endl;
    HO101.getpoldom().write(std::cerr << "Q:=", Q) << ';' << std::endl;
    HO101.getpoldom().write(std::cerr << "P:=", P) << ';' << std::endl;
    HO101.getpoldom().write(std::cerr << "TTy:=", TTy) << ';' << std::endl;
    //      Fra._num = HO101.getpoldom().one;
    //      Polys::Element Tay;
    //      HO101.taylor(Tay, Fra, b);
    //      HO101.getpoldom().write(std::cerr << "Tay:=", Tay) << ';' << std::endl;
    HO101.write(std::cerr << "TTy17:=", TTy17) << ';' << std::endl;
    HO101.write(std::cerr << "TTy17c:=", TTy17c) << ';' << std::endl;
    std::cerr << "----- END ERROR -----" << std::endl;
} else {
    ++Fidcount;
}
} catch (GivError &e) {
    std::cerr << e << std::endl;
}
return success && successF;
}

int main(int argc, char ** argv) {
    long numb = (argc>1?atoi(argv[1]):200);
    std::cerr << "numb: " << numb << std::endl;
    long ttnn = (argc>2?atoi(argv[2]):100);
    std::cerr << "ttnn: " << ttnn << std::endl;
    long seed = (argc>3?atoi(argv[3]):BaseTimer::seed());
    std::cerr << "seed: " << seed << std::endl;
    Field Z101( 101, 1 ); // integers modulo 101
    // Polynomials over Z101, with X as indeterminate
    Polys DP101( Z101, Indeter("X") );
    Polys::Element P, Q, R, monomial;
    GivRandom generator((unsigned long)seed);
    long deg1 = (long) generator() % 6;
    long deg2 = (long) generator() % 6;
    long deg3 = (long) generator() % 155;
    // long v1 = generator() % 195;
    // long v2 = v1 + (generator() % 5);
    DP101.random(generator, P, Degree(deg1) );
    DP101.random(generator, Q, Degree(deg2) );
    DP101.random(generator, R, Degree(deg3) );
    Degree dP; DP101.degree(dP,P);
    Degree dQ; DP101.degree(dQ,Q);
    Degree dR; DP101.degree(dR,R);
    DP101.write(std::cout << "P:=", P) << ';' << std::endl;
    HighOrders HO101( Z101, Indeter("X") );
    HighOrders::Element F;
    F._num = DP101.one;
    F._den = P;
    Polys::Element Tay;

```

```

HO101.taylor(Tay, F, 128);
DP101.write(std::cout << "Tay:=", Tay) << ';' << std::endl;
Polys::Element S; Degree dS;
size_t e = 0 ; // initialisé à quoi ? dans GammaId, k0 est const...
HighOrders::Truncated G0;
HO101.GammaId(G0, S, dS, (long)e, P, dP);
std::cout << "e:=" << e << ';' << std::endl;
HO101.write(std::cout << "G0:=", G0) << ';' << std::endl;
DP101.write(std::cout << "S:=", S) << ';' << std::endl;
std::vector<HighOrders::Truncated> Gam, T;
std::vector<Degree> D;
Polys::Element nTay; Degree dT;
HO101.highorder(Gam, T, D, nTay, dT, Degree(970), Degree(1000), P, dP);
HO101.write(std::cout << "Gam:=", Gam.back()) << ';' << std::endl;
HighOrders::Truncated TP; HO101.gettruncdom().assign(TP, P);
HighOrders::Truncated I;
HO101.Inverse(I, 113, 127, TP, dP, nTay, dT, Gam, T, D);
HO101.write(std::cout << "I]_155^175:=", I) << ';' << std::endl;
Ttaylor.clear();
Thighorder.clear();
bool success=true;
success &= TestFracDevel(HO101, P, Q, ttn-(17*ttn)/100,ttn);
success &= TestFracDevel(HO101, P, Q, ttn-(27*ttn)/100,ttn);
success &= TestFracDevel(HO101, P, Q, ttn-(87*ttn)/100,ttn);
success &= TestFracDevel(HO101, P, Q, 0,ttn);
success &= TestFracDevel(HO101, P, Q, 1,ttn);
success &= TestFracDevel(HO101, P, Q, 2,ttn);
success &= TestFracDevel(HO101, P, Q, ttn,ttn);
success &= TestFracDevel(HO101, P, Q, ttn-1,ttn);
success &= TestFracDevel(HO101, P, Q, ttn-2,ttn);
for(long i=0; i<numb; ++i) {
    long Deg1 = (long)generator() % ((66*ttn)/100);
    long Deg2 = (long)generator() % ((65*ttn)/100);
    long v1 = (long)generator() % ((19195*ttn)/100);
    long v2 = v1 + ((long)generator() % ((45*ttn)/100));
    DP101.random(generator, P, Degree(Deg1) );
    DP101.random(generator, Q, Degree(Deg2) );
    success &= TestFracDevel(HO101, P, Q, v1, v2);
}
std::cerr << "Taylor: " << Ttaylor << std::endl;
std::cerr << "HighOrder: " << Thighorder << std::endl;
std::cerr << "Fiduccia: " << Tfiduccia << std::endl;
if (! success) {
    std::cerr << "Error: " << seed << std::endl;
} else {
    std::cerr << "Success:" << TFDcount << std::endl;
}
std::cerr << "Success F:" << FiDcount << std::endl;
return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.37 examples/Polynomial/interpolate.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <givaro/modular.h>
#include <givaro/givpolyfactor.h>
#include <givaro/givtimer.h>
#include <givaro/givinterp.h>
using namespace std;
using namespace Givaro;
int main(int argc, char** argv)
{
    typedef Modular<int32_t>::Residu_t UT ;
    UT MOD;
    if (argc > 2)
        MOD = (UT) atoi(argv[2]);
    else
        std::cin >> MOD;

```

```

Modular<int32_t> F(MOD);
Interpolation< Modular<int32_t> > FD(F, Indeter("X"));
Interpolation< Modular<int32_t> >::Element nouv, prec;
int EarlyTerm = 0, Bound = 5;
Modular<int32_t>::Element x, f;
std::ifstream input (argv[1]);
F.read(input, x);
F.read(input, f);
FD(x,f);
prec = FD.interpolator();
Timer tim; tim.clear(); tim.start();
while(! F.read(input, x).eof() ) {
    F.read(input, f);
    FD(x,f);
    nouv = FD.interpolator();
    if (FD.areEqual(nouv, prec)) {
        if (++EarlyTerm > Bound) { std::cerr << "EarlyTerminated" << std::endl ; break; }
    } else
        EarlyTerm = 0;
    prec = nouv;
}
tim.stop();
FD.write( std::cout, FD.interpolator() ) << std::endl;
std::cerr << tim << std::endl;
return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.38 examples/Polynomial/isprimitive.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpolylfactor.h>
#include <givaro/givtimer.h>
// using namespace std;
using namespace Givaro;
int main(int argc, char** argv)
{
    typedef GFqDom<int64_t>::Residu_t UT;
    UT MOD;
    if (argc > 1)
        MOD = (UT)atoi(argv[1]);
    else
        std::cin >> MOD;
    uint64_t expo = 1;
    if (argc > 2) expo = (uint64_t)atoi(argv[2]);
    GFqDom<int64_t> F(MOD, expo);
    Poly1FactorDom<GFqDom<int64_t>, Dense> FD(F, Indeter("X"));
    Poly1FactorDom<GFqDom<int64_t>, Dense>::Element P, IXE;
    FD.init(IXE, Degree(1), F.one);
    FD.read( std::cin, P );
    Timer tim; tim.clear(); tim.start();
    bool f = FD.is_prim_root(IXE, P );
    tim.stop();
    F.write( FD.write( std::cout, P ) << " is " << (f?"":"not ") << "primitive in " ) << std::endl;
    // std::cout << f << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.39 examples/Polynomial/isirred.C

NO DOC.

## NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpolylfactor.h>
#include <givaro/givtimer.h>
using namespace std;
using namespace Givaro;
int main(int argc, char** argv)
{
    typedef GFqDom<int64_t>::Residu_t UT ;
    UT MOD;
    if (argc > 1)
        MOD = (UT) (atoi(argv[1]));
    else
        std::cin >> MOD;
    uint64_t expo = 1;
    if (argc > 2) expo = (uint64_t)atoi(argv[2]);
    GFqDom<int64_t> F(MOD, expo);
    PolyLFactorDom<GFqDom<int64_t>, Dense> FD(F, Indeter("X"));
    PolyLFactorDom<GFqDom<int64_t>, Dense>::Element P;
    FD.read( cin, P );
    Timer tim; tim.clear(); tim.start();
    bool f = FD.is_irreducible( P );
    tim.stop();
    F.write( FD.write( cout, P ) << " is " << (f?"":"not ") << "irreducible in " ) << endl;
    // std::cout << f << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.40 examples/Polynomial/pol\_eval.C

## NO DOC.

## NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpolylfactor.h>
#include <givaro/givtimer.h>
using namespace Givaro;
using namespace std;
int main(int argc, char** argv)
{
    GFqDom<int64_t>::Residu_t MOD;
    if (argc > 1)
        MOD = (GFqDom<int64_t>::Residu_t) atoi(argv[1]);
    else
        std::cin >> MOD;
    uint64_t expo = 1;
    if (argc > 2) expo = (uint64_t)atoi(argv[2]);
    GFqDom<int64_t> F(MOD, expo);
    PolyLFactorDom<GFqDom<int64_t>, Dense> FD(F, Indeter("X"));
    PolyLFactorDom<GFqDom<int64_t>, Dense>::Element P;
    FD.read( cin, P );
    GFqDom<int64_t>::Element res, val;
    F.read( cin, val );
    Timer tim; tim.clear(); tim.start();
    FD.eval(res, P, val );
    tim.stop();
    F.write( F.write( FD.write( cout, P ) << " is ", res ) << " at ", val ) << endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s
```

## 19.41 examples/Polynomial/pol\_factor.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <stdlib.h>
#include <givaro/gfq.h>
#include <givaro/givpolylfactor.h>
#include <givaro/givtimer.h>
using namespace std;
using namespace Givaro;
int main(int argc, char** argv)
{
    GFqDom<int64_t>::Residu_t MOD;
    if (argc > 1)
        MOD = (GFqDom<int64_t>::Residu_t) atoi(argv[1]);
    else
        std::cin >> MOD;
    uint64_t expo = 1;
    if (argc > 2) expo = (uint64_t)atoi(argv[2]);
    Timer tim2; tim2.clear(); tim2.start();
    GFqDom<int64_t> F(MOD, expo);
    tim2.stop();
    std::cerr<<"init field -> "<<tim2.usertime()<<std::endl;
    Poly1FactorDom<GFqDom<int64_t>, Dense> FD(F, Indeter("X"));
    typedef Poly1FactorDom<GFqDom<int64_t>, Dense>::Element Polys ;
    Polys P;
    FD.read( cin, P );
    std::vector<Polys> Lf;
    std::vector<uint64_t> Le;
    Timer tim; tim.clear(); tim.start();
    FD.CZfactor(Lf, Le, P);
    tim.stop();
    FD.write( cout, P ) << " is 1";
    std::vector<uint64_t>::const_iterator e = Le.begin();
    for(std::vector<Polys>::const_iterator i = Lf.begin(); i != Lf.end(); ++i, ++e) {
        FD.write(cout << " * (" , *i) << ")";
        if (*e > 1) cout << "^" << *e;
    }
    std::cout << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,{0,\:0,t0,+0,=s
```

## 19.42 examples/Polynomial/PolynomialCRT.C

NO DOC.

NO DOC

```
// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <algorithm>
#include <givaro/givtimer.h>
#include <givaro/givpolylcrt.h>
#include <givaro/givintprime.h>
#include <givaro/montgomery.h>
#include <givaro/extension.h>
#include <givaro/modular.h>
#include <givaro/gfq.h>
#include <givaro/chineseremainder.h> // Chinese Remainder of two elements
#include <givaro/givrns.h> // Chinese Remainder of an array of elements
#include <givaro/givrandom.h>
#include <givaro/givrational.h>
using namespace Givaro;
typedef GFqDom<int64_t> Field1;
```

```

typedef Modular<int16_t>      Field2;
typedef Modular<Log16>       Field3;
typedef Modular<int32_t>     Field4;
typedef Modular<int64_t>     Field5;
typedef Modular<uint32_t>    Field6;
typedef Montgomery<int32_t>  Field7;
typedef QField<Rational>     Field8;
typedef Modular<Integer>     Field9;
typedef Extension<>          Field10;
template <typename Field>
bool tmain(int argc, char ** argv, GivRandom& generator) {
    bool pass = true;
    typedef Poly1CRT< Field > CRTSystem;
    typedef typename CRTSystem::Element Poly;
    // typedef typename CRTSystem::Type_t    Scal;
    // typedef typename CRTSystem::array_E   VPoly;
    typedef typename CRTSystem::array_T VScal;
    IntPrimeDom ID;
    Integer a( generator() »(argc>2?atoi(argv[2]):17) );
    Field F(ID.nextprimein( a ));
    VScal Primes( argc>1 ? (size_t)atoi(argv[1]):15);
    VScal Moduli( Primes.size() );
    typename VScal::iterator i = Primes.begin();
    typename VScal::iterator e = Moduli.begin();
    for( ; i != Primes.end(); ++i, ++e) {
        do {
            F.init(*i, generator());
        } while ( (std::find(Primes.begin(), i, *i) != i) || (F.isZero(*i)) );
        F.init(*e, generator());
    }
    // for(typename VScal::const_iterator it=Primes.begin(); it!=Primes.end();++it)
    //     F.write(std::cout, *it) « std::endl;
    // for(typename VScal::const_iterator it=Moduli.begin(); it!=Moduli.end();++it)
    //     F.write(std::cout, *it) « std::endl;
    CRTSystem CRT( F, Primes, "X" );
    Poly res;
    Timer tim; tim.clear(); tim.start();
    CRT.RnsToRing( res, Moduli );
    tim.stop();
    F.write( std::cerr « tim « " using " » std::endl;
    if (Primes.size() < 14) {
        i = Primes.begin();
        e = Moduli.begin();
        for( ; i != Primes.end(); ++i, ++e)
            if (F.characteristic()>0)
                F.write(CRT.getpolydom().write(F.write(std::cout « "subs(X=", *i) « ",", res) « ") mod " «
                    F.characteristic() « " = ", *e) « ';' « std::endl;
            else
                F.write(CRT.getpolydom().write(F.write(std::cout « "subs(X=", *i) « ",", res) « ") = ", *e)
                    « ';' « std::endl;
    }
    VScal Verifs( Primes.size() );
    CRT.RingToRns( Verifs, res );
    typename VScal::const_iterator v = Verifs.begin();
    e = Moduli.begin();
    for( ; e != Moduli.end(); ++e, ++v)
        if (! F.areEqual(*e, *v) ) {
            F.write(std::cerr « "incoherency within " » std::endl;
            F.write(std::cerr « "e: ", *e ) « std::endl;
            F.write(std::cerr « "v: ", *v ) « std::endl;
            pass = false;
            break;
        }
    CRT.getpolydom().random(generator, res, Degree((int64_t)Primes.size()-1));
    CRT.RingToRns( Verifs, res );
    Poly nres;
    tim.clear(); tim.start();
    CRT.RnsToRing( nres, Verifs );
    tim.stop();
    if (! CRT.getpolydom().areEqual(res,nres) ) {
        CRT.getpolydom().write(std::cerr « "incoherency within " » std::endl;
        CRT.getpolydom().write(std::cerr « "r: ", res ) « std::endl;
        CRT.getpolydom().write(std::cerr « "n: ", nres ) « std::endl;
        pass = false;
    }
    F.write( std::cerr « tim « " using " » std::endl;
    return pass;
}
template <typename Field>
bool tmainext(int argc, char ** argv, GivRandom& generator) {
    bool pass = true;
    typedef Poly1CRT< Field > CRTSystem;
    typedef typename CRTSystem::Element Poly;
    // typedef typename CRTSystem::Type_t    Scal;
    // typedef typename CRTSystem::array_E   VPoly;
    typedef typename CRTSystem::array_T VScal;
    IntPrimeDom ID;

```

```

Integer a( generator() »(argc>2?atoi(argv[2]):17) );
Field F(ID.nextprimein( a ),2);
VScal Primes( argc>1 ? (size_t)atoi(argv[1]):15);
VScal Moduli( Primes.size() );
typename VScal::iterator i = Primes.begin();
typename VScal::iterator e = Moduli.begin();
for( ; i != Primes.end(); ++i, ++e) {
    F.init(*i); F.init(*e);
    do {
        F.random(generator,*i);
    } while ( (std::find(Primes.begin(), i, *i) != i) || (F.isZero(*i)) );
    F.random(generator,*e);
}
for(typename VScal::const_iterator it=Primes.begin(); it!=Primes.end();++it)
    F.write(std::cout, *it) << std::endl;
for(typename VScal::const_iterator it=Moduli.begin(); it!=Moduli.end();++it)
    F.write(std::cout, *it) << std::endl;
CRTSystem CRT( F, Primes, "X" );
Poly res;
Timer tim; tim.clear(); tim.start();
CRT.RnsToRing( res, Moduli );
tim.stop();
F.write( std::cerr << tim << " using " << std::endl;
if (Primes.size() < 14) {
    i = Primes.begin();
    e = Moduli.begin();
    for( ; i != Primes.end(); ++i, ++e)
        if (F.characteristic()>0)
            F.write(CRT.getpolydom().write(F.write(std::cout << "subs(X=", *i) << ", ", res) << ") mod " <<
F.characteristic() << " = ", *e) << ';' << std::endl;
        else
            F.write(CRT.getpolydom().write(F.write(std::cout << "subs(X=", *i) << ", ", res) << ") = ", *e)
<< ';' << std::endl;
}
VScal Verifs( Primes.size() );
CRT.RingToRns( Verifs, res );
typename VScal::const_iterator v = Verifs.begin();
e = Moduli.begin();
for( ; e != Moduli.end(); ++e, ++v)
    if (! F.areEqual(*e, *v) ) {
        F.write(std::cerr << "incoherency within " << std::endl;
        F.write(std::cerr << "e: ", *e ) << std::endl;
        F.write(std::cerr << "v: ", *v ) << std::endl;
        pass = false;
        break;
    }
CRT.getpolydom().random(generator, res, Degree((int64_t)Primes.size()-1));
CRT.RingToRns( Verifs, res );
Poly nres;
tim.clear(); tim.start();
CRT.RnsToRing( nres, Verifs );
tim.stop();
if (! CRT.getpolydom().areEqual(res,nres) ) {
    CRT.getpolydom().write(std::cerr << "incoherency within " << std::endl;
    CRT.getpolydom().write(std::cerr << "r: ", res ) << std::endl;
    CRT.getpolydom().write(std::cerr << "n: ", nres ) << std::endl;
    pass = false;
}
F.write( std::cerr << tim << " using " << std::endl;
return pass;
}
int main(int argc, char ** argv) {
    // argv[1] : number of primes
    // argv[2] : 2^{32-j} is size of primes
    // argv[3] : seed for generator
    GivRandom seedor( argc>3 ? (uint64_t)atoi(argv[3]): (uint64_t)BaseTimer::seed() );
    uint64_t seed = seedor.seed();
    std::cerr << "seed: " << seed << std::endl;
    Integer::seeding(seed);
    return
    tmain<Field1>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field2>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field3>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field4>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field5>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field6>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field7>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field8>(argc, argv, *( new GivRandom(seed)))
    && tmain<Field9>(argc, argv, *( new GivRandom(seed)))
    && tmainext<Field10>(argc, argv, *( new GivRandom(seed)))
    ;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.43 examples/Polynomial/trunc\_arith.C

NO DOC.

NO DOC

```
// Copyright(c) 1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <givaro/givrandom.h>
#include <givaro/givtimer.h>
#include <givaro/gfq.h>
#include <givaro/givpoly1.h>
#include <givaro/givtruncdomain.h>
using namespace Givaro;
long long TTcount = 0;
bool TestAdd(const TruncDom< GFqDom<int> >& DP, const TruncDom< GFqDom<int> >::Element& P, const TruncDom<
    GFqDom<int> >::Element& Q, size_t d1, size_t d2)
{
    ++TTcount;
    TruncDom< GFqDom<int> >::Element R, T, V;
    DP.add ( R, P, Q, (int64_t)d1, (int64_t)d2); // R = P*Q;
    // DP.write( DP.write(
    //     std::cout << "[(" , P ) << ") + (" , Q) << ") ]_" << d1 << '^' << d2 ;
    //     DP.write(std::cout << " = " , R) << std::endl;
    DP.add (T, P, Q);
    V=T;
    DP.truncin(V, (int64_t)d1, (int64_t)d2);
    if( DP.areNEqual( V, R) ) {
        std::cerr << "ERROR ADD:" << TTcount << std::endl;
        DP.write(std::cout << " R: " , R) << std::endl;
        DP.write(std::cout << " T: " , T) << std::endl;
        DP.write(std::cout << " V: " , V) << std::endl;
        Degree dP; DP.degree(dP,P);
        Degree dQ; DP.degree(dQ,Q);
        Degree vP; DP.val(vP,P);
        Degree vQ; DP.val(vQ,Q);
        std::cerr << "vR: " << vP << ", dR: " << dP << ", vP: " << vQ << ", dP: " << dQ << ", v: " << d1 << ", d: " <<
            d2 << std::endl;
        return false;
    }
    return true;
}

bool TestSub(const TruncDom< GFqDom<int> >& DP, const TruncDom< GFqDom<int> >::Element& P, const TruncDom<
    GFqDom<int> >::Element& Q, size_t d1, size_t d2)
{
    ++TTcount;
    TruncDom< GFqDom<int> >::Element R, T, V;
    DP.sub ( R, P, Q, (int64_t)d1, (int64_t)d2); // R = P*Q;
    // DP.write( DP.write(
    //     std::cout << "[(" , P ) << ") + (" , Q) << ") ]_" << d1 << '^' << d2 ;
    //     DP.write(std::cout << " = " , R) << std::endl;
    DP.sub (T, P, Q);
    V=T;
    DP.truncin(V, (int64_t)d1, (int64_t)d2);
    if( DP.areNEqual( V, R) ) {
        std::cerr << "ERROR SUB:" << TTcount << std::endl;
        DP.write(std::cout << " R: " , R) << std::endl;
        DP.write(std::cout << " T: " , T) << std::endl;
        DP.write(std::cout << " V: " , V) << std::endl;
        Degree dP; DP.degree(dP,P);
        Degree dQ; DP.degree(dQ,Q);
        Degree vP; DP.val(vP,P);
        Degree vQ; DP.val(vQ,Q);
        std::cerr << "vR: " << vP << ", dR: " << dP << ", vP: " << vQ << ", dP: " << dQ << ", v: " << d1 << ", d: " <<
            d2 << std::endl;
        return false;
    }
    return true;
}

bool TestMul(const TruncDom< GFqDom<int> >& DP, const TruncDom< GFqDom<int> >::Element& P, const TruncDom<
    GFqDom<int> >::Element& Q, size_t d1, size_t d2)
{
    ++TTcount;
    TruncDom< GFqDom<int> >::Element R, T, V;
    DP.mul ( R, P, Q, (int64_t)d1, (int64_t)d2); // R = P*Q;
    // DP.write( DP.write(
    //     std::cout << "[(" , P ) << ") + (" , Q) << ") ]_" << d1 << '^' << d2 ;
    //     DP.write(std::cout << " = " , R) << std::endl;
    DP.mul (T, P, Q);
    V=T;
    DP.truncin(V, (int64_t)d1, (int64_t)d2);
```

```

    if( DP.areNEqual( V, R ) ) {
        std::cerr << "ERROR MUL:" << TTcount << std::endl;
        DP.write(std::cout << " R: " , R) << std::endl;
        DP.write(std::cout << " T: " , T) << std::endl;
        DP.write(std::cout << " V: " , V) << std::endl;
        Degree dP; DP.degree(dP,P);
        Degree dQ; DP.degree(dQ,Q);
        Degree vP; DP.val(vP,P);
        Degree vQ; DP.val(vQ,Q);
        std::cerr << "vR: " << vP << ", dR: " << dP << ", vP: " << vQ << ", dP: " << dQ << ", v: " << d1 << ", d: " <<
        d2 << std::endl;
        return false;
    }
    return true;
}

bool TestAxy(const TruncDom< GFqDom<int> >& DP, const TruncDom< GFqDom<int> >::Element& P, const TruncDom<
    GFqDom<int> >::Element& Q, const TruncDom< GFqDom<int> >::Element& G, size_t d1, size_t d2)
{
    ++TTcount;
    TruncDom< GFqDom<int> >::Element R, T, V;
    DP.axy ( R, P, Q, G, (int64_t)d1, (int64_t)d2); // R = P*Q;
    // DP.write( DP.write(
    //     std::cout << "[( " , P ) << " ) + ( " , Q ) << " ]_" << d1 << '^' << d2 ;
    //     DP.write(std::cout << " = " , R) << std::endl;
    DP.axy ( T, P, Q, G);
    V=T;
    DP.truncin(V, (int64_t)d1, (int64_t)d2);
    if( DP.areNEqual( V, R ) ) {
        std::cerr << "ERROR Axy:" << std::endl;
        DP.write(std::cout << " P: " , P) << std::endl;
        DP.write(std::cout << " Q: " , Q) << std::endl;
        DP.write(std::cout << " G: " , G) << std::endl;
        DP.write(std::cout << " R: " , R) << std::endl;
        DP.write(std::cout << " T: " , T) << std::endl;
        DP.write(std::cout << " V: " , V) << std::endl;
        Degree dP; DP.degree(dP,P);
        Degree dQ; DP.degree(dQ,Q);
        Degree dG; DP.degree(dG,G);
        Degree vP; DP.val(vP,P);
        Degree vQ; DP.val(vQ,Q);
        Degree vG; DP.val(vG,G);
        std::cerr << "vP: " << vP << ", dP: " << dP << ", vQ: " << vQ << ", dQ: " << dQ << ", vG: " << vG << ", dG: " <<
        dG << ", v: " << d1 << ", d: " << d2 << std::endl;
        return false;
    }
    return true;
}

bool TestAxy(const TruncDom< GFqDom<int> >& DP, const TruncDom< GFqDom<int> >::Element& P, const TruncDom<
    GFqDom<int> >::Element& Q, const TruncDom< GFqDom<int> >::Element& G, size_t d1, size_t d2)
{
    ++TTcount;
    TruncDom< GFqDom<int> >::Element R, T, V;
    DP.axmy ( R, P, Q, G, (int64_t)d1, (int64_t)d2); // R = P*Q;
    // DP.write( DP.write(
    //     std::cout << "[( " , P ) << " ) + ( " , Q ) << " ]_" << d1 << '^' << d2 ;
    //     DP.write(std::cout << " = " , R) << std::endl;
    DP.axmy ( T, P, Q, G);
    V=T;
    DP.truncin(V, (int64_t)d1, (int64_t)d2);
    if( DP.areNEqual( V, R ) ) {
        std::cerr << "ERROR Axmy:" << std::endl;
        DP.write(std::cout << " R: " , R) << std::endl;
        DP.write(std::cout << " T: " , T) << std::endl;
        DP.write(std::cout << " V: " , V) << std::endl;
        Degree dP; DP.degree(dP,P);
        Degree dQ; DP.degree(dQ,Q);
        Degree vP; DP.val(vP,P);
        Degree vQ; DP.val(vQ,Q);
        std::cerr << "vR: " << vP << ", dR: " << dP << ", vP: " << vQ << ", dP: " << dQ << ", v: " << d1 << ", d: " <<
        d2 << std::endl;
        return false;
    }
    return true;
}

bool TestMaxpy(const TruncDom< GFqDom<int> >& DP, const TruncDom< GFqDom<int> >::Element& P, const TruncDom<
    GFqDom<int> >::Element& Q, const TruncDom< GFqDom<int> >::Element& G, size_t d1, size_t d2)
{
    ++TTcount;
    TruncDom< GFqDom<int> >::Element R, T, V;
    DP.maxpy ( R, P, Q, G, (int64_t)d1, (int64_t)d2); // R = P*Q;
    // DP.write( DP.write(
    //     std::cout << "[( " , P ) << " ) + ( " , Q ) << " ]_" << d1 << '^' << d2 ;
    //     DP.write(std::cout << " = " , R) << std::endl;
    DP.maxpy ( T, P, Q, G);
    V=T;
    DP.truncin(V, (int64_t)d1, (int64_t)d2);
    if( DP.areNEqual( V, R ) ) {

```

```

std::cerr << "ERROR Maxpy:" << std::endl;
DP.write(std::cout << " R: " , R) << std::endl;
DP.write(std::cout << " T: " , T) << std::endl;
DP.write(std::cout << " V: " , V) << std::endl;
Degree dP; DP.degree(dP,P);
Degree dQ; DP.degree(dQ,Q);
Degree vP; DP.val(vP,P);
Degree vQ; DP.val(vQ,Q);
std::cerr << "vR: " << vP << ", dR: " << dP << ", vP: " << vQ << ", dP: " << dQ << ", v: " << d1 << ", d: " <<
d2 << std::endl;
return false;
}
return true;
}
int main(int argc, char ** argv) {
{
    int64_t seed = (argc>1?atoi(argv[1]):BaseTimer::seed());
    std::cerr << "seed: " << seed << std::endl;
    GFqDom<int> Z101( 101, 1 ); // integers modulo 101
    // Polynomials over Z101, with X as indeterminate
    TruncDom< GFqDom<int> > DP101( Z101, Indeter("X") );
    TruncDom< GFqDom<int> >::Element P, Q, R, monomial;
    GFqDom<int>::Element tmp;
    DP101.assign( P, Z101.init(tmp,5) ); // P is degree 0 polynomial : 5 modulo 101
    DP101.write( std::cout << "P: " , P )<< std::endl;
    DP101.init( monomial, Degree(1), 33U) ; // -33 X
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    Degree deg,val;
    DP101.degree(deg,monomial);
    DP101.val(val,monomial);
    DP101.write( std::cout << "[m]_" << val << '^' << deg << ": " , monomial )<< std::endl;
    DP101.addin( P, monomial ); // P += monomial
    DP101.write( std::cout << "P: " , P )<< std::endl;
    DP101.init( monomial, Degree(2), 12U) ; // 12 X^2
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.addin( P, monomial ); // P is now 5-33X+12X^2
    DP101.write( std::cout << "P: " , P )<< std::endl;
    // // DP101.read( std::cin, P); // would read P as a succession of integers :
    // // deg leadcoeff (lead-1)coeff ... unitcoeff
    Q = P;
    DP101.write( std::cout << "P: " , P )<< std::endl;
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.init( Q, Degree(0), 6U );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.init( monomial, Degree(4), 3U);
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.addin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.init( monomial, Degree(1), 75U);
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.addin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.init( monomial, Degree(3), 45U);
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.subin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    // Q is now 3X^4+75X-45X^3+6
    DP101.mulin( Q, Degree(15) );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.mulin( monomial, Degree(32));
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.addin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.subin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.divin( Q, Degree(5) );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.divin( monomial, Degree(12));
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.addin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.subin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.divin( monomial, Degree(10));
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.addin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.subin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.divin( monomial, Degree(10));
    DP101.write( std::cout << "m: " , monomial )<< std::endl;
    DP101.addin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.subin( Q, monomial );
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
    DP101.setval(Q);
    DP101.write( std::cout << "Q: " , Q )<< std::endl;
}
}

```

```

DP101.mulin( P, Degree(15) );
DP101.mul ( R, P, Q); // R = P*Q;
DP101.write( DP101.write(
    std::cout << "(" , P ) << ")" * "(" , Q ) << ")"");
DP101.write(std::cout << " = " , R) << std::endl;
DP101.mul ( R, P, Q, Degree(28), Degree(30)); // R = P*Q;
DP101.write( DP101.write(
    std::cout << "[" ( " , P ) << "]" * "(" , Q ) << "]"_28^30";
DP101.write(std::cout << " = " , R) << std::endl;
DP101.mul ( R, P, Q, Degree(0), Degree(30)); // R = P*Q;
DP101.write( DP101.write(
    std::cout << "[" ( " , P ) << "]" * "(" , Q ) << "]"_0^30";
DP101.write(std::cout << " = " , R) << std::endl;
DP101.mul ( R, P, Q, Degree(28), Degree(100)); // R = P*Q;
DP101.write( DP101.write(
    std::cout << "[" ( " , P ) << "]" * "(" , Q ) << "]"_28^100";
DP101.write(std::cout << " = " , R) << std::endl;
DP101.mul ( R, P, Q, Degree(4), Degree(10)); // R = P*Q;
DP101.write( DP101.write(
    std::cout << "[" ( " , P ) << "]" * "(" , Q ) << "]"_4^10";
DP101.write(std::cout << " = " , R) << std::endl;
DP101.mul ( R, P, Q, Degree(75), Degree(100)); // R = P*Q;
DP101.write( DP101.write(
    std::cout << "[" ( " , P ) << "]" * "(" , Q ) << "]"_75^100";
DP101.write(std::cout << " = " , R) << std::endl;
DP101.setval(R);
DP101.write( std::cout << "R: " , R )<< std::endl;
DP101.mulin( Q, Degree(3) );
DP101.add ( R, P, Q); // R = P*Q;
DP101.write( DP101.write(
    std::cout << "(" , P ) << ")" + "(" , Q ) << ")"");
DP101.write(std::cout << " = " , R) << std::endl;
/*
    DP101.gcd ( R, P, Q); //

    DP101.write( DP101.write( DP101.write(
        std::cout << "gcd(" , P ) << " , Q ) << ") = " , R) << std::endl;

    DP101.lcm ( R, P, Q); //
    DP101.write( DP101.write( DP101.write(
        std::cout << "lcm(" , P ) << " , Q ) << ") = " , R) << std::endl;
    DP101.lcm ( R, Q, P); //
    DP101.write( DP101.write( DP101.write(
        std::cout << "lcm(" , Q ) << " , P ) << ") = " , R) << std::endl;

*/
}
return 0;
}
/* -*- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
// vim:sts=4:sw=4:ts=4:et:sr:cino=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.44 examples/Rational/iratrecon.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <stdlib.h>
#include <givaro/givrational.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h> // Givaro initialization
using namespace Givaro;
int main(int argc, char** argv)
{
    Integer f,m,k;
    if (argc > 1) f = Integer(argv[1]); else std::cin >> f;
    if (argc > 2) m = Integer(argv[2]); else std::cin >> m;
    QField<Rational> RD;
    Rational rec;
    Timer tim; tim.clear(); tim.start();
    if (argc > 3)
        RD.ratrecon(rec,f,m, Integer(argv[3]) );
    else
        RD.ratrecon(rec,f,m);
}

```

```

    tim.stop();
    std::cout << rec.ume() << "/" << rec.deno() << " - " << f << " mod " << m << " = 0;" << std::endl;
    std::cerr << tim << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:src:cin=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

## 19.45 examples/Rational/polydouble.C

NO DOC.

NO DOC

```

// Copyright(c)'1994-2009 by The Givaro group
// This file is part of Givaro.
// Givaro is governed by the CeCILL-B license under French law
// and abiding by the rules of distribution of free software.
// see the COPYRIGHT file for more details.
#include <iostream>
#include <stdlib.h>
#include <givaro/givpoly1.h>
#include <givaro/givrational.h>
#include <givaro/givtimer.h>
#include <givaro/givinit.h>           // Givaro initialization
#include <givaro/givprint.h>         // Givaro print utils
using namespace Givaro;
typedef Poly1Dom< QField<Rational>, Dense>::Element RatPoly;
typedef std::vector<double> DoublePoly;
int main(int argc, char** argv)
{
    srandom((unsigned int)BaseTimer::seed() );
    Integer f,m,k;
    QField<Rational> Q;
    Poly1Dom< QField<Rational>, Dense> PolQ(Q);
    DoublePoly D;
    RatPoly R;
    size_t n = (argc>1?(size_t)atoi(argv[1]):10);
    for(size_t i=0;i<n;++i)
        D.push_back( (double)(random()) / RAND_MAX );
    R.resize( D.size() );
    RatPoly::iterator it=R.begin();
    DoublePoly::const_iterator dit = D.begin();
    for( ; dit != D.end(); ++dit, ++it)
        *it = *dit;
    std::cout << "Double Poly : " << D << std::endl;
    std::cout << "REDUCED Rational " << R << std::endl;
    std::cout << "Approximations : ";
    RatPoly::const_iterator cit=R.begin();
    dit = D.begin();
    for( ; dit != D.end(); ++dit, ++cit)
        std::cout << std::endl << *cit << " is " << ((double)*cit) << " by " << ( (double)*cit - *dit ) << ' ';
    std::cout << std::endl;
    Rational::SetNoReduce();
    it=R.begin();
    dit = D.begin();
    for( ; dit != D.end(); ++dit, ++it)
        *it = *dit;
    std::cout << "Double Poly : " << D << std::endl;
    std::cout << "Unreduced Rational " << R << std::endl;
    std::cout << "Approximations : ";
    cit=R.begin();
    dit = D.begin();
    for( ; dit != D.end(); ++dit, ++cit)
        std::cout << std::endl << *cit << " is " << ((double)*cit) << " by " << ( (double)*cit - *dit ) << ' ';
    std::cout << std::endl;
    return 0;
}
/* -- mode: C++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -- */
// vim:sts=4:sw=4:ts=4:et:src:cin=>s,f0,{0,g0,(0,\:0,t0,+0,=s

```

# Index

`__giv_map_less_ith`< T, UNARYOP, ith >, 61  
`__givdom_trait_name`< T >, 61  
`_fast_impl`  
    IntRSADom< MyRandIter >, 253  
`_perfArray0`< T >, 62  
`_size`  
    Array0< T >, 65  
    ArrayAllocatort< T, Tag >, 68  
`~GIV_ExtensionrandIter`  
    GIV\_ExtensionrandIter< ExtensionField, Type >, 105  
`~GIV_randIter`  
    GIV\_randIter< Ring, Type >, 108  
`~ModularRandIter`  
    ModularRandIter< Ring >, 259  
  
`absCompare`  
    Givaro, 48  
    Integer, 209–212  
`absOutput`  
    Givaro, 53  
    Integer, 244  
`add`  
    GF2, 84, 85  
    Integer, 165, 166  
`addin`  
    GF2, 92, 93  
    Integer, 163, 164  
`all_field.C`(4.1.1), 289  
`allocate`  
    Array0< T >, 64  
    ArrayAllocatort< T, Tag >, 67  
    GivaroMM< T >, 111  
`areEqual`  
    GF2, 80  
`Array0`< T >, 62  
    \_size, 65  
    allocate, 64  
    copy, 64  
    reallocate, 64  
`Array0Tag`, 65  
`ArrayAllocatort`< T, Tag >, 65  
    \_size, 68  
    allocate, 67  
    copy, 67  
    reallocate, 67  
`ArrayFixed`< T, SIZE >, 68  
`assign`  
    GF2, 79  
`axmy`  
    GF2, 90, 91  
    Integer, 188  
`axmyin`  
    GF2, 98  
    Integer, 189  
`axpy`  
    GF2, 89, 90  
    Integer, 185  
`axpyin`  
    GF2, 97, 98  
    Integer, 186  
  
`BaseDomain`< T >, 69  
`BaseTimer`, 69  
`Bits`, 69  
`bitsize`  
    Integer, 208  
`BlocFreeList`, 70  
`bstruct`, 35  
  
`cardinality`  
    GF2, 80  
`ceil`  
    Integer, 202, 203  
`characteristic`  
    GF2, 80  
`ChineseRemainder`< Ring, Domain, false >, 71  
`ChineseRemainder`< Ring, Domain, REDUCE >, 70  
`chineseremainder.h`(4.1.1), 302  
`clear`  
    Timer, 281  
`compare`  
    Givaro, 48  
    Integer, 209  
`convert`  
    GF2, 79  
`copy`  
    Array0< T >, 64  
    ArrayAllocatort< T, Tag >, 67  
    Integer, 141  
`crem`  
    Integer, 194, 195  
  
`Degree`, 71  
`div`  
    GF2, 87  
    Integer, 190–192  
`divexact`  
    Integer, 192–194  
`divin`

- GF2, 95
- Integer, 190
- divmod
  - Integer, 201, 202
- domain\_to\_operatorstyle.C(4.1.1), 289
- ElemConstRef< T >, 71
- Element
  - GIV\_Extensionrandlter< ExtensionField, Type >, 103
  - GIV\_randlter< Ring, Type >, 107
  - ModularRandlter< Ring >, 258
- ElemRef< T >, 72
- exponentiation.C(4.1.1), 289
- Extension< BFT >, 72
- extension.h(4.1.1), 302
- fact
  - Givaro, 49
  - Integer, 239
- FermatDom, 73
- ff\_arith.C(4.1.1), 290
- floor
  - Integer, 203, 204
- frem
  - Integer, 194, 195
- gcd
  - Givaro, 45
  - Integer, 232, 233
- GeneralRingNonZeroRandlter< Ring, Randlter >, 73
- GeneralRingRandlter< Ring >, 74
- GF128.C(4.1.1), 290
- GF2, 74
  - add, 84, 85
  - addin, 92, 93
  - areEqual, 80
  - assign, 79
  - axmy, 90, 91
  - axmyin, 98
  - axpy, 89, 90
  - axpyin, 97, 98
  - cardinality, 80
  - characteristic, 80
  - convert, 79
  - div, 87
  - divin, 95
  - GF2, 78
  - init, 78
  - inv, 89
  - invin, 96, 97
  - isMOne, 82
  - isOne, 81
  - isUnit, 81
  - isZero, 81
  - maxCardinality, 100
  - maxpy, 91, 92
  - maxpyin, 99
  - mul, 86
  - mulin, 94
  - neg, 88
  - negin, 96
  - operator=, 78
  - read, 83, 84
  - sub, 85, 86
  - subin, 93
  - write, 82, 83
- GFirreducible.C(4.1.1), 290
- gfq.h(4.1.1), 303
- gfq\_atomic.C(4.1.1), 290
- GFqDom< TT >, 100
- GFqExt< TT >, 101
- gfqext.h(4.1.1), 304
- GFqExtFast< TT >, 101
- GFqKronecker< TT, Ints >, 102
- gfqkronecker.h(4.1.1), 304
- GIV\_Extensionrandlter
  - GIV\_Extensionrandlter< ExtensionField, Type >, 103, 105
- GIV\_Extensionrandlter< ExtensionField, Type >, 102
  - ~GIV\_Extensionrandlter, 105
  - Element, 103
  - GIV\_Extensionrandlter, 103, 105
  - operator(), 105, 106
  - random, 105
  - ring, 106
- GIV\_randlter
  - GIV\_randlter< Ring, Type >, 108
- GIV\_randlter< Ring, Type >, 106
  - ~GIV\_randlter, 108
  - Element, 107
  - GIV\_randlter, 108
  - operator(), 109
  - operator=, 109
  - random, 109
  - ring, 110
- Givaro, 35, 39
  - absCompare, 48
  - absOutput, 53
  - compare, 48
  - fact, 49
  - gcd, 45
  - inv, 45
  - invin, 45
  - isOdd, 52
  - isOne, 49
  - isZero, 48
  - lcm, 46
  - length, 52
  - logp, 51
  - logtwo, 51
  - naturallog, 51
  - nonZero, 49
  - operator!=, 54
  - operator<, 54
  - operator<=, 53
  - operator<=, 55

- operator>, 54
- operator>>, 52
- operator>=, 55
- operator\*, 56
- operator+, 53
- operator-, 56
- operator==, 54, 56
- operator%, 55
- pow, 47
- powmod, 47
- pp, 46
- root, 50
- sign, 48
- sqrt, 49, 50
- sqrtrem, 50
- swap, 51
- GivaroAppli, 110
- GivaroMain, 110
- GivaroMM< T >, 111
  - allocate, 111
- givaromm.h(4.1.1), 319
- GivaroNolnit, 112
- givarray0.h(4.1.1), 297
- givarrayallocator.h(4.1.1), 298
- givarrayfixed.h(4.1.1), 298
- GivBadFormat, 112
- givbasictype.h(4.1.1), 324
- givbits.h(4.1.1), 299
- givcaster.h(4.1.1), 324
- givconfig.h(4.1.1), 325
- givdegree.h(4.1.1), 331
- givelem.h(4.1.1), 299
- GivError, 112
- giverror.h(4.1.1), 325
- givgenarith.h(4.1.1), 326
- givhashtable.h(4.1.1), 300
- givindeter.h(4.1.1), 332
- givinit.h(4.1.1), 327
- givinteger.h(4.1.1), 315
- givinterp.h(4.1.1), 332
- givinterpgeom-multip.h(4.1.1), 333
- givinterpgeom.h(4.1.1), 334
- givintfactor.h(4.1.1), 315
- givintnumtheo.h(4.1.1), 316
- givintprime.h(4.1.1), 317
- givintrns.h(4.1.1), 318
- givintrsa.h(4.1.1), 318
- givintsqrootmod.h(4.1.1), 319
- givlist0.h(4.1.1), 301
- GivMathDivZero, 113
- GivMathError, 113
- GivMMFreeList, 114
- GivMMInfo, 114
- GivMMRefCount, 114
- GivModule, 115
- givmodule.h(4.1.1), 327
- givNoCopy, 115
- givNolnit, 116
- givperf.h(4.1.1), 328
- givpointer.h(4.1.1), 320
- givpoly1.h(4.1.1), 334
- givpoly1crt.h(4.1.1), 335
- givpoly1dense.h(4.1.1), 335
- givpoly1factor.h(4.1.1), 336
- givpoly1padic.h(4.1.1), 337
- givpower.h(4.1.1), 328
- givprimes16.h(4.1.1), 305
- givprint.h(4.1.1), 328
- givranditer.h(4.1.1), 329
- GivRandom, 116
- givrandom.h(4.1.1), 330
- givrational.h(4.1.1), 321
- givref\_count.h(4.1.1), 321
- givrns.h(4.1.1), 305
- givrnsfixed.h(4.1.1), 306
- givstack.h(4.1.1), 301
- givtimer.h(4.1.1), 330
- givWithCopy, 116
- GMP, 36
- gmp++\_int.h(4.1.1), 307
- gmp++\_int\_add.C(4.1.1), 308
- gmp++\_int\_compare.C(4.1.1), 309
- gmp++\_int\_cstor.C(4.1.1), 309
- gmp++\_int\_div.C(4.1.1), 310
- gmp++\_int\_gcd.C(4.1.1), 310
- gmp++\_int\_io.C(4.1.1), 311
- gmp++\_int\_lib.C(4.1.1), 311
- gmp++\_int\_misc.C(4.1.1), 312
- gmp++\_int\_mod.C(4.1.1), 312
- gmp++\_int\_mul.C(4.1.1), 313
- gmp++\_int\_pow.C(4.1.1), 313
- gmp++\_int\_rand.inl(4.1.1), 314
- gmp++\_int\_sub.C(4.1.1), 314
- HashTable< T, Key >, 117
- highorder.C(4.1.1), 295
- ixponentiation.C(4.1.1), 291
- ifactor.C(4.1.1), 291
- ifactor\_lenstra.C(4.1.1), 291
- igcd.C(4.1.1), 291
- igcdext.C(4.1.1), 291
- ilcm.C(4.1.1), 292
- Indeter, 117
- init
  - GF2, 78
- InitAfter, 117
- Integer, 36, 118
  - absCompare, 209–212
  - absOutput, 244
  - add, 165, 166
  - addin, 163, 164
  - axmy, 188
  - axmyin, 189
  - axpy, 185
  - axpyin, 186
  - bitsize, 208

- ceil, 202, 203
- compare, 209
- copy, 141
- crem, 194, 195
- div, 190–192
- divexact, 192–194
- divin, 190
- divmod, 201, 202
- fact, 239
- floor, 203, 204
- frem, 194, 195
- gcd, 232, 233
- Integer, 136–140
- inv, 234
- invin, 234
- isleq, 141
- isMOne, 213
- isOdd, 243
- isOne, 212
- isZero, 213–215
- lcm, 235
- length, 242
- logcpy, 140
- logp, 241
- logtwo, 241
- maxpy, 187
- maxpyin, 187, 188
- mod, 200, 201
- modin, 199
- mul, 171–173
- mulin, 170, 171
- naturallog, 242
- neg, 170
- negin, 169
- nonZero, 213
- operator!=, 145–147, 219–221
- operator<, 151–153, 225, 226
- operator<<, 158, 159, 244
- operator<=<=, 160
- operator<=, 143–145, 218, 219
- operator>, 149–151, 223, 224
- operator>>, 161, 162, 243
- operator>=>=, 162, 163
- operator>=, 141–143, 216, 217
- operator\*, 181–183, 229, 230
- operator\*=, 183–185
- operator^, 153, 154
- operator^=, 154, 155
- operator+, 173–175, 226, 227
- operator+=, 175–177
- operator-, 177, 178, 181, 228, 229
- operator-=, 179–181
- operator/, 196, 197, 230, 231
- operator/=, 197, 198
- operator=, 140
- operator==, 147–149, 221–223
- operator%, 204–206, 231, 232
- operator%=, 206–208
- operator&, 157
- operator&=, 158
- operator[], 208
- operator|, 155, 156
- operator|=, 156
- pow, 236–239
- powmod, 239
- pp, 235
- print, 209
- Protected::importWords, 244
- random\_lessthan, 208
- root, 241
- sign, 242
- size\_in\_base, 208
- sqrt, 240
- sqrtrem, 240
- sub, 168, 169
- subin, 166, 167
- swap, 242
- trunc, 203, 204
- IntegerDom, 245
- interpolate.C(4.1.1), 295
- Interpolation< Domain, REDUCE >, 246
  - setdegree, 246
  - sqrfree, 247
- IntFactorDom< MyRandIter >, 247
- IntNumTheoDom< MyRandIter >, 248
  - prim\_inv, 249
  - probable\_prim\_root, 249
- IntPrimeDom, 250
- IntRNSsystem< Container, Alloc >, 250
- IntRSADom< MyRandIter >, 251
  - \_fast\_impl, 253
  - keys\_gen, 252
  - strong\_prime, 252
- IntSqrtModDom< MyRandIter >, 253
- inv
  - GF2, 89
  - Givaro, 45
  - Integer, 234
- invin
  - GF2, 96, 97
  - Givaro, 45
  - Integer, 234
- iratrecon.C(4.1.1), 297
- isirred.C(4.1.1), 295
- isleq
  - Integer, 141
- isMOne
  - GF2, 82
  - Integer, 213
- isOdd
  - Givaro, 52
  - Integer, 243
- isOne
  - GF2, 81
  - Givaro, 49
  - Integer, 212

- ispower.C(4.1.1), [292](#)
- isprimitive.C(4.1.1), [296](#)
- isproot.C(4.1.1), [292](#)
- isUnit
  - GF2, [81](#)
- isZero
  - GF2, [81](#)
  - Givaro, [48](#)
  - Integer, [213–215](#)
- Key< T >, [254](#)
- keys\_gen
  - IntRSADom< MyRandIter >, [252](#)
- lambda.C(4.1.1), [292](#)
- lambda\_inv.C(4.1.1), [292](#)
- lcm
  - Givaro, [46](#)
  - Integer, [235](#)
- length
  - Givaro, [52](#)
  - Integer, [242](#)
- List0< T >, [254](#)
- logcpy
  - Integer, [140](#)
- logp
  - Givaro, [51](#)
  - Integer, [241](#)
- logtwo
  - Givaro, [51](#)
  - Integer, [241](#)
- main
  - test-integer.C, [338](#)
- maxCardinality
  - GF2, [100](#)
- maxpy
  - GF2, [91, 92](#)
  - Integer, [187](#)
- maxpyin
  - GF2, [99](#)
  - Integer, [187, 188](#)
- Memory, [36](#)
- mod
  - Integer, [200, 201](#)
- modin
  - Integer, [199](#)
- Modular< \_Storage\_t, \_Compute\_t, typename  
 std::enable\_if< is\_same\_ruint< \_Storage\_t,  
 \_Compute\_t >::value || is\_smaller\_ruint<  
 \_Storage\_t, \_Compute\_t >::value >::type >,  
[256](#)
- Modular< Integer >, [256](#)
- Modular< IntType, \_Compute\_t, Enable >, [255](#)
- Modular< Log16 >, [257](#)
- modular-implem.h(4.1.1), [322](#)
- modular-integral.h(4.1.1), [323](#)
- modular.h(4.1.1), [323](#)
- Modular\_implem< \_Storage\_t, \_Compute\_t, \_Residu\_t  
 >, [257](#)
- ModularRandIter
  - ModularRandIter< Ring >, [259](#)
- ModularRandIter< Ring >, [258](#)
  - ~ModularRandIter, [259](#)
  - Element, [258](#)
  - ModularRandIter, [259](#)
  - operator(), [260, 261](#)
  - operator=, [260](#)
  - random, [260, 261](#)
- ModularSquareRoot.C(4.1.1), [292](#)
- Montgomery< int32\_t >, [261](#)
- Montgomery< Reclnt::ruint< K > >, [262](#)
- montgomery.h(4.1.1), [323](#)
- mul
  - GF2, [86](#)
  - Integer, [171–173](#)
- mulin
  - GF2, [94](#)
  - Integer, [170, 171](#)
- naturallog
  - Givaro, [51](#)
  - Integer, [242](#)
- nb\_primes.C(4.1.1), [293](#)
- neg
  - GF2, [88](#)
  - Integer, [170](#)
- negin
  - GF2, [96](#)
  - Integer, [169](#)
- Neutral, [262](#)
- NewtonInterpGeom< Domain, REDUCE >, [262](#)
  - setdegree, [263](#)
  - sqrfree, [263](#)
- NewtonInterpGeomMultip< Domain, REDUCE >, [264](#)
  - setdegree, [264](#)
  - sqrfree, [265](#)
- nextprime.C(4.1.1), [293](#)
- nonZero
  - Givaro, [49](#)
  - Integer, [213](#)
- ObjectInit, [265](#)
- OMPTimer, [266](#)
- operator!=
  - Givaro, [54](#)
  - Integer, [145–147, 219–221](#)
- operator<
  - Givaro, [54](#)
  - Integer, [151–153, 225, 226](#)
- operator<<
  - Givaro, [53](#)
  - Integer, [158, 159, 244](#)
  - std, [57, 58](#)
- operator<=<=
  - Integer, [160](#)
- operator<=

- Givaro, 55
- Integer, 143–145, 218, 219
- operator>
  - Givaro, 54
  - Integer, 149–151, 223, 224
- operator>>
  - Givaro, 52
  - Integer, 161, 162, 243
- operator>>=
  - Integer, 162, 163
- operator>=
  - Givaro, 55
  - Integer, 141–143, 216, 217
- operator\*
  - Givaro, 56
  - Integer, 181–183, 229, 230
  - RandomIntegerIterator< \_Unsigned, \_Exact\_Size >, 275
- operator\*=
  - Integer, 183–185
- operator^
  - Integer, 153, 154
- operator^=
  - Integer, 154, 155
- operator()
  - GIV\_ExtensionRandIter< ExtensionField, Type >, 105, 106
  - GIV\_randIter< Ring, Type >, 109
  - ModularRandIter< Ring >, 260, 261
- operator+
  - Givaro, 53
  - Integer, 173–175, 226, 227
- operator+=
  - Integer, 175–177
- operator-
  - Givaro, 56
  - Integer, 177, 178, 181, 228, 229
- operator-=
  - Integer, 179–181
- operator/
  - Integer, 196, 197, 230, 231
- operator/=
  - Integer, 197, 198
- operator=
  - GF2, 78
  - GIV\_randIter< Ring, Type >, 109
  - Integer, 140
  - ModularRandIter< Ring >, 260
  - RandomIntegerIterator< \_Unsigned, \_Exact\_Size >, 273
- operator==
  - Givaro, 54, 56
  - Integer, 147–149, 221–223
- operator%
  - Givaro, 55
  - Integer, 204–206, 231, 232
- operator%=
  - Integer, 206–208
- operator&
  - Integer, 157
- operator&=
  - Integer, 158
- operator[]
  - Integer, 208
- operator|
  - Integer, 155, 156
- operator|=
  - Integer, 156
- order.C(4.1.1), 293
- Pair< T1, T2 >, 266
- phi.C(4.1.1), 293
- pol\_arith.C(4.1.1), 296
- pol\_eval.C(4.1.1), 296
- pol\_factor.C(4.1.1), 296
- Poly1CRT< Field >, 266
- Poly1Dom< Domain, Dense >, 267
  - setdegree, 267
  - sqrfree, 268
- Poly1FactorDom
  - Poly1FactorDom< Domain, Tag, RandomIterator >, 269
- Poly1FactorDom< Domain, Tag, RandomIterator >, 268
  - Poly1FactorDom, 269
- Poly1PadicDom< Domain, Dense >, 270
  - setdegree, 270
  - sqrfree, 271
- polydouble.C(4.1.1), 297
- PolynomialCRT.C(4.1.1), 296
- pow
  - Givaro, 47
  - Integer, 236–239
- powmod
  - Givaro, 47
  - Integer, 239
- pp
  - Givaro, 46
  - Integer, 235
- prevprime.C(4.1.1), 293
- prim\_inv
  - IntNumTheoDom< MyRandIter >, 249
- Primes16, 271
- primitiveelement.C(4.1.1), 293
- primitiveroot.C(4.1.1), 294
- print
  - Integer, 209
- probable\_prim\_root
  - IntNumTheoDom< MyRandIter >, 249
- probable\_primroot.C(4.1.1), 294
- ProbLucas.C(4.1.1), 294
- Protected::importWords
  - Integer, 244
- QField< Rational >, 271
- random

- GIV\_ExtensionrandIter< ExtensionField, Type >, 105
- GIV\_randIter< Ring, Type >, 109
- ModularRandIter< Ring >, 260, 261
- random\_lessThan
  - Integer, 208
- randomInteger
  - RandomIntegerIterator< \_Unsigned, \_Exact\_Size >, 275
- RandomIntegerIterator
  - RandomIntegerIterator< \_Unsigned, \_Exact\_Size >, 273
- RandomIntegerIterator< \_Unsigned, \_Exact\_Size >, 272
  - operator\*, 275
  - operator=, 273
  - randomInteger, 275
  - RandomIntegerIterator, 273
  - setSeed, 275
- Rational, 275
  - Rational, 276
- RationalReconstruction
  - ZRing< \_Element >, 287
- Rationals, 37
- read
  - GF2, 83, 84
  - ZRing< \_Element >, 288
- realElapsedTime
  - Timer, 284
- reallocate
  - Array0< T >, 64
  - ArrayAllocatort< T, Tag >, 67
- realtime
  - Timer, 283
- RealTimer, 277
- Reclnt, 56
- RefCounter, 277
- RefCountPtr< T >, 277
- ring
  - GIV\_ExtensionrandIter< ExtensionField, Type >, 106
  - GIV\_randIter< Ring, Type >, 110
- RNSSystem< RING, Domain >, 278
- RNSSystemFixed< Ints >, 278
- root
  - Givaro, 50
  - Integer, 241
- RSA\_breaking.C(4.1.1), 294
- RSA\_decipher.C(4.1.1), 294
- RSA\_encipher.C(4.1.1), 295
- RSA\_keys\_generator.C(4.1.1), 295
- setdegree
  - Interpolation< Domain, REDUCE >, 246
  - NewtonInterpGeom< Domain, REDUCE >, 263
  - NewtonInterpGeomMultip< Domain, REDUCE >, 264
  - Poly1Dom< Domain, Dense >, 267
  - Poly1PadicDom< Domain, Dense >, 270
- setSeed
  - RandomIntegerIterator< \_Unsigned, \_Exact\_Size >, 275
- sign
  - Givaro, 48
  - Integer, 242
- size\_in\_base
  - Integer, 208
- sqrfree
  - Interpolation< Domain, REDUCE >, 247
  - NewtonInterpGeom< Domain, REDUCE >, 263
  - NewtonInterpGeomMultip< Domain, REDUCE >, 265
  - Poly1Dom< Domain, Dense >, 268
  - Poly1PadicDom< Domain, Dense >, 271
- sqrt
  - Givaro, 49, 50
  - Integer, 240
- sqrtrem
  - Givaro, 50
  - Integer, 240
- Stack< THING >, 279
- start
  - Timer, 281
- StaticElement< DomainStyle >, 279
- StaticElement.h(4.1.1), 307
- std, 57
  - operator<<, 57, 58
- stop
  - Timer, 282
- strong\_prime
  - IntRSADom< MyRandIter >, 252
- sub
  - GF2, 85, 86
  - Integer, 168, 169
- subin
  - GF2, 93
  - Integer, 166, 167
- swap
  - Givaro, 51
  - Integer, 242
- sysElapsedTime
  - Timer, 284
- System, 37
- systemtime
  - Timer, 283
- SysTimer, 280
- test-crt.C(4.1.1), 337
- test-integer.C
  - main, 338
- test-integer.C(4.1.1), 338
- test-modsroot.C(4.1.1), 338
- test-random.C
  - test2, 339
  - test3, 340
- test-random.C(4.1.1), 339
- test2
  - test-random.C, 339

- test3
  - test-random.C, [340](#)
- Test\_Extension.C(4.1.1), [290](#)
- Timer, [280](#)
  - clear, [281](#)
  - realElapsedTime, [284](#)
  - realtime, [283](#)
  - start, [281](#)
  - stop, [282](#)
  - sysElapsedTime, [284](#)
  - sysTime, [283](#)
  - userElapsedTime, [283](#)
  - usertime, [282](#)
- trunc
  - Integer, [203](#), [204](#)
- trunc\_arith.C(4.1.1), [297](#)
- userElapsedTime
  - Timer, [283](#)
- usertime
  - Timer, [282](#)
- UserTimer, [285](#)
- VectorDom< Domain, StorageTag >, [285](#)
- write
  - GF2, [82](#), [83](#)
  - ZRing< \_Element >, [287](#)
- zpz\_atomic.C(4.1.1), [290](#)
- ZRing, [37](#)
- ZRing< \_Element >, [285](#)
  - RationalReconstruction, [287](#)
  - read, [288](#)
  - write, [287](#)