Xint

Generated by Doxygen 1.9.5

1 Xint class implements operation on large integer numbers

If you need to perform precise operations on integer data that do not fit into common number types, you can successfully use objects of the Xint class. You can operate on numbers consisting of tens of thousands of digits.

1.1 Limitations

The only limitation on the size of the number are computer resources.

2 xint

Xint class implementation. A class that allows operations on large integer data

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Class Documentation

5.1 Xint Class Reference

A Xint class.

```
#include <xint.h>
```

Public Member Functions

- Xint ()

  Default constructor.
- Xint (unsigned long long n)

  Copy constructor.
- Xint (unsigned long n)

  Copy constructor.
- Xint (long long n)

  Copy constructor.
- Xint (long n)

  Copy constructor.
- Xint (unsigned int n)

  Copy constructor.
- Xint (int n)

  Copy constructor.
- Xint (string &)

  Copy constructor.
- Xint (const char *)

  Copy constructor.
- Xint (const Xint &)

  Copy constructor.
- Xint (vector< int8_t > &)

  Copy constructor. Construct a new object from vector<int8_t>.
- void mul10 ()
- void div10 ()
- void mul2 ()
- void div2 ()
- bool zero ()
- long long ll ()
- Xint & operator++ ()
- Xint operator++ (int temp)
- Xint & operator-- ()
- Xint operator-- (int temp)
- Xint & operator= (long long &b)
- Xint & operator= (unsigned long &b)
- Xint add (Xint &b)
- Xint sub (Xint &b)
- Xint mul (Xint &b)
- Xint div (Xint &b)
- long long to_long ()
- Xint power (long exponent)
- Xint power (Xint exponent)
- long log (long base)
- Xint root (long base)

  Calculates the root of a given degree.
- Xint abs ()

Private Member Functions

- Xint & inc_pos ()

  Increase by one the value represented by the "number" vector.
- Xint & dec_pos ()

  Decrease by one the value represented by the "number" vector.
- Xint & plus_assign (Xint &b)

  Sum of two vectors. Assigning to vector number the sum of itself and the vector number of parameter b.
- Xint & minus_assign (Xint &b)

  Difference of two vectors. Assigning to vector number the difference of itself and the vector number of parameter b.
- bool less (Xint &b)

  Compares the object's number vector value to the number vector value of the parameter.
- bool greater (Xint &b)

  Compares the object's number vector value to the number vector value of the parameter.
- pair< vector< int8_t >, vector< int8_t > > divide (Xint &b)

  Divide with remainder by given number. Divides vector numbers by parameter `b`.

Private Attributes

- vector< int8_t > number
- short sign
- int len

Friends

- Xint & operator+= (Xint &, Xint &)
- Xint & operator-= (Xint &, Xint &)
- bool operator== (const Xint &x1, const Xint &x2)

  Determine if the Xint object is equivalent to the other.
- bool operator!= (const Xint &x1, const Xint &x2)

  Determine if the Xint object is not equivalent to the other.
- Xint & operator/= (Xint &, Xint &)
- Xint & operator%= (Xint &, Xint &)
- bool operator< (Xint &a, Xint &b)

  Comparing the values of two Xint objects.
- bool operator< (Xint &a, long long b)
- bool operator< (long long a, Xint b)
- bool operator> (Xint &a, long long b)
- bool operator> (long long a, Xint b)
- bool operator<= (Xint &a, long long b)
- bool operator<= (long long a, Xint b)
- bool operator>= (Xint &a, long long b)
- bool operator>= (long long a, Xint b)
- bool operator< (vector< int8_t > number, Xint &b)
- Xint & operator*= (Xint &a, Xint &b)
- Xint operator+ (Xint a, Xint b)
- Xint operator- (Xint a, Xint b)
- Xint operator* (Xint a, Xint b)
- bool operator> (Xint &a, Xint &b)

  Comparing the values of two Xint objects.
- bool operator>= (Xint &a, Xint &b)

Comparing the values of two Xint objects.

- bool operator<= (Xint &a, Xint &b)

  Comparing the values of two Xint objects.
- Xint operator% (Xint &a, Xint &b)
- Xint operator/ (Xint a, Xint b)
- ostream & operator<< (ostream &, const Xint &)
- istream & operator>> (istream &, Xint &)

5.1.1 Detailed Description

A Xint class.

Exact operation on large integers.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Xint() [1/11] `Xint::Xint ( )`

Default constructor.

Create a new Xint object with value 0.

See also

Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long long n);
Xint(long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(Xint &);
Xint(vector<int8_t> &);

5.1.2.2 Xint() [2/11] `Xint::Xint ( unsigned long long n )`

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

n	unsigned long long.

See also

Xint();
Xint(unsigned long n);
Xint(long long n);
Xint(long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(Xint &);
Xint(vector<int8_t> &);

5.1.2.3 Xint() [3/11] Xint::Xint (unsigned long *n*)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

n	unsigned long.

See also

Xint();
Xint(unsigned long long n);
Xint(long long n);
Xint(long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(Xint &);
Xint(vector<int8_t> &);

5.1.2.4 Xint() [4/11] Xint::Xint (long long *n*)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

in	*n*	long long.

See also

Xint();
Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(Xint &);
Xint(vector<int8_t> &);

5.1.2.5 Xint() [5/11] Xint::Xint (long *n*)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

n	long.

See also

Xint();
Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(Xint &);
Xint(vector<int8_t> &);

5.1.2.6 Xint() [6/11] Xint::Xint (unsigned int *n*)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

n	unsigned int.

See also

Xint();
Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long long n);
Xint(long n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(Xint &);
Xint(vector<int8_t> &);

5.1.2.7 Xint() [7/11] Xint::Xint (int *n*)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

n	int.

See also

Xint();
Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long long n);
Xint(long n);
Xint(unsigned int n);

Xint(string &);

Xint(const char *);

Xint(Xint &);

Xint(vector<int8_t> &);

5.1.2.8 Xint() [8/11] Xint::Xint (string & *s*)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

s	string.

See also

Xint();

Xint(unsigned long long n);

Xint(unsigned long n);

Xint(long long n);

Xint(long n);

Xint(unsigned int n);

Xint(int n);

Xint(const char *);

Xint(Xint &);

Xint(vector<int8_t> &);

5.1.2.9 Xint() [9/11] Xint::Xint (const char * *s*)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

s	*char.

See also

Xint();
Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long long n);
Xint(long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(Xint &);
Xint(vector<int8_t> &);

5.1.2.10 Xint() [10/11] Xint::Xint (const Xint & x)

Copy constructor.

Create a new Xint object from parmeter value.

Parameters

n	Xint.

See also

Xint();
Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long long n);
Xint(long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(vector<int8_t> &);

5.1.2.11 Xint() [11/11] Xint::Xint (vector< int8_t > & n)

Copy constructor. Construct a new object from vector<int8_t>.

Create a new Xint object from parmeter value.

Parameters

n	vector<int8_t>.

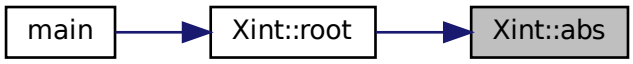
See also

Xint();
Xint(unsigned long long n);
Xint(unsigned long n);
Xint(long long n);
Xint(long n);
Xint(unsigned int n);
Xint(int n);
Xint(string &);
Xint(const char *);
Xint(Xint &);

5.1.3 Member Function Documentation

5.1.3.1 abs() `Xint Xint::abs ( )`

Here is the caller graph for this function:



5.1.3.2 add() `Xint Xint::add ( Xint & b )`

Here is the caller graph for this function:

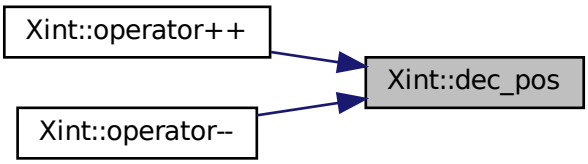
5.1.3.3 dec_pos() `Xint & Xint::dec_pos ( ) [private]`

Decrease by one the value represented by the "number" vector.

Returns

A reference to itself

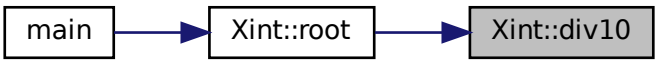
Here is the caller graph for this function:



5.1.3.4 div() `Xint Xint::div ( Xint & b )`

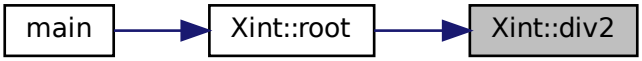
5.1.3.5 div10() `void Xint::div10 ( )`

Here is the caller graph for this function:



5.1.3.6 div2() `void Xint::div2 ( )`

Here is the caller graph for this function:



5.1.3.7 divide() `pair< vector< int8_t >, vector< int8_t > > Xint::divide ( Xint & b ) [private]`

Divide with remainder by given number. Divides vector numbers by parameter `b`.

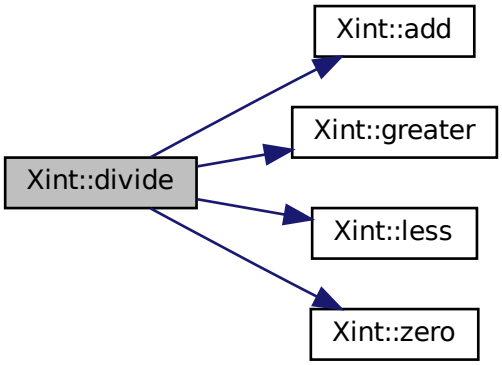
Parameters

in	*b*	– Divider.

Returns

Value pair<vector<int8_t> ,vector<int8_t>>. <outcome, remainder>.

Here is the call graph for this function:



5.1.3.8 greater() `bool Xint::greater ( Xint & b ) [private]`

Compares the object's number vector value to the number vector value of the parameter.

Parameters

in	*b*	The object with the compared vector

Returns

True if the object vector is greater than the parameter vector. False otherwise.

Here is the caller graph for this function:

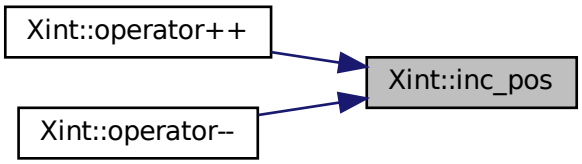
5.1.3.9 inc_pos() `Xint & Xint::inc_pos ( ) [private]`

Increase by one the value represented by the "number" vector.

Returns

A reference to itself

Here is the caller graph for this function:



5.1.3.10 less() `bool Xint::less ( Xint & b ) [private]`

Compares the object's number vector value to the number vector value of the parameter.

Parameters

in	*b*	The object with the compared vector

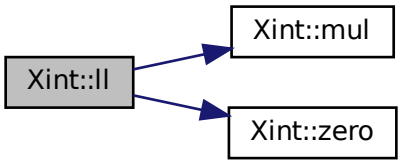
Returns

True if the object vector is smaller than the parameter vector. False otherwise.

Here is the caller graph for this function:

5.1.3.11 ll() `long long Xint::ll ( )`

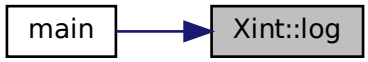
Here is the call graph for this function:



5.1.3.12 log() `long Xint::log ( long base )`

Here is the call graph for this function:

Here is the caller graph for this function:



5.1.3.13 minus_assign() `Xint & Xint::minus_assign ( Xint & b ) [private]`

Difference of two vectors. Assigning to vector number the difference of itself and the vector number of parameter b.

Parameters

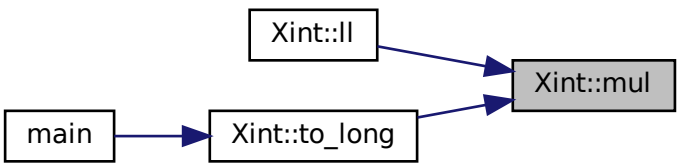
b	Xint &.

Returns

A reference to itself

5.1.3.14 mul() `Xint Xint::mul ( Xint & b )`

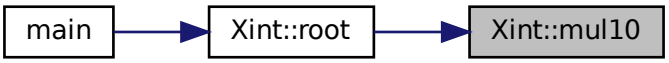
Here is the caller graph for this function:



5.1.3.15 mul10() `void Xint::mul10 ( )`

Here is the call graph for this function:

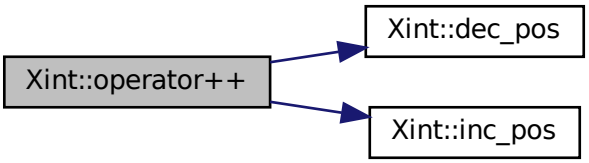
Here is the caller graph for this function:



5.1.3.16 mul2() `void Xint::mul2 ( )`

5.1.3.17 operator++() [1/2] `Xint & Xint::operator++ ( )`

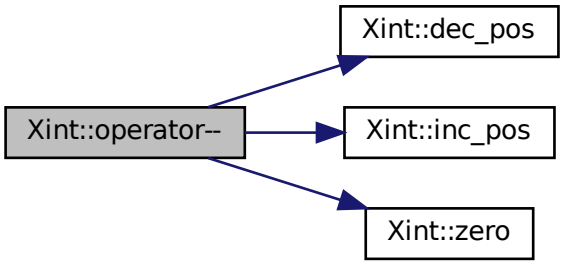
Here is the call graph for this function:



5.1.3.18 operator++() [2/2] Xint Xint::operator++ (int *temp*)

5.1.3.19 operator--() [1/2] Xint & Xint::operator-- ( )

Here is the call graph for this function:



5.1.3.20 operator--() [2/2] Xint Xint::operator-- (int *temp*)

5.1.3.21 operator=() [1/2] Xint & Xint::operator= (long long & *b*)

5.1.3.22 operator=() [2/2] Xint & Xint::operator= (unsigned long & *b*)

5.1.3.23 plus_assign() Xint & Xint::plus_assign (Xint & *b*) [private]

Sum of two vectors. Assigning to vector number the sum of itself and the vector number of parameter b.

Parameters

b	Xint &.

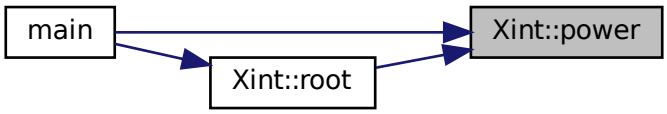
Returns

A reference to itself

5.1.3.24 power() [1/2] Xint Xint::power (long *exponent*)

Here is the call graph for this function:

Here is the caller graph for this function:



5.1.3.25 power() [2/2] Xint Xint::power (Xint *exponent*)

Here is the call graph for this function:

5.1.3.26 root() Xint Xint::root (long *deg*)

Calculates the root of a given degree.

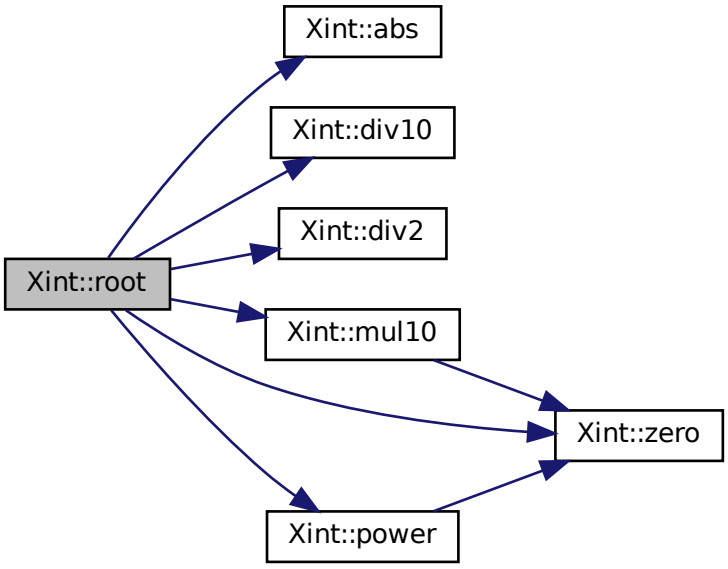
Parameters

in	*deg*	The degree of the root

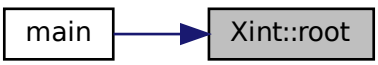
Returns

Root value

Here is the call graph for this function:



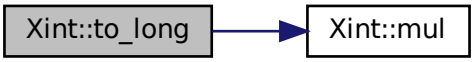
Here is the caller graph for this function:



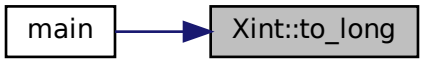
5.1.3.27 sub() `Xint Xint::sub ( Xint & b )`

5.1.3.28 to_long() `long long Xint::to_long ( )`

Here is the call graph for this function:

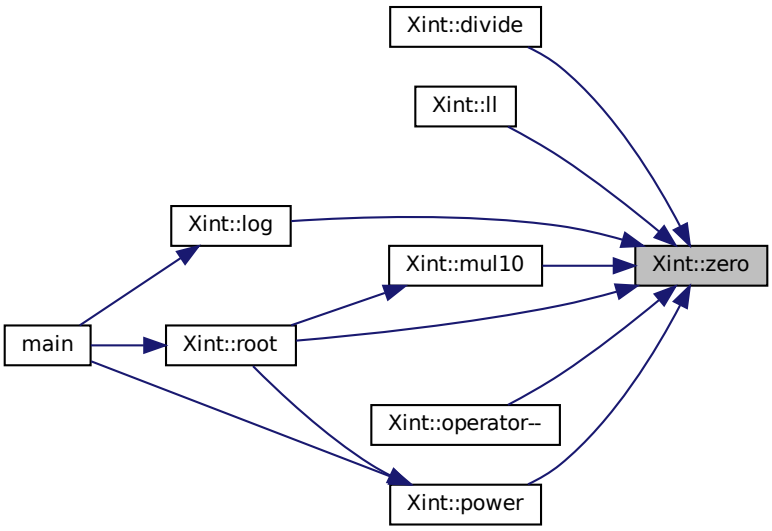


Here is the caller graph for this function:



5.1.3.29 zero() `bool Xint::zero ( ) [inline]`

Here is the caller graph for this function:



5.1.4 Friends And Related Function Documentation

5.1.4.1 operator"!= `bool operator!= (`

```
const Xint & x1,
const Xint & x2 )  [friend]
```

Determine if the Xint object is not equivalent to the other.

Parameters

x1	Xint object.
x2	another Xint object.

Returns

Whether the two Triangle objects are not the same.

5.1.4.2 operator% `Xint operator% (`

```
Xint & a,
Xint & b )  [friend]
```

5.1.4.3 operator%= `Xint & operator%= (`

```
Xint & a,
Xint & b )  [friend]
```

5.1.4.4 operator* `Xint operator* (`

```
Xint a,
Xint b )  [friend]
```

5.1.4.5 operator*= `Xint & operator*= (`

```
Xint & a,
Xint & b )  [friend]
```

5.1.4.6 operator+ `Xint operator+ (`

```
Xint a,
Xint b )  [friend]
```

5.1.4.7 operator+=

```
Xint & operator+= (
        Xint & a,
        Xint & b )  [friend]
```

5.1.4.8 operator-

```
Xint operator- (
        Xint a,
        Xint b )  [friend]
```

5.1.4.9 operator-=

```
Xint & operator-= (
        Xint & a,
        Xint & b )  [friend]
```

5.1.4.10 operator/

```
Xint operator/ (
        Xint a,
        Xint b )  [friend]
```

5.1.4.11 operator/=

```
Xint & operator/= (
        Xint & a,
        Xint & b )  [friend]
```

5.1.4.12 operator< [1/4]

```
bool operator< (
        long long a,
        Xint b )  [friend]
```

5.1.4.13 operator< [2/4]

```
bool operator< (
        vector< int8_t > number,
        Xint & b )  [friend]
```

5.1.4.14 operator< [3/4]

```
bool operator< (
        Xint & a,
        long long b )  [friend]
```

5.1.4.15 operator< [4/4]

```
bool operator< (
        Xint & a,
        Xint & b )  [friend]
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a < b, false otherwise.

5.1.4.16 operator<< `ostream & operator<< (`

```
ostream & out,
const Xint & a )  [friend]
```

5.1.4.17 operator<= [1/3] `bool operator<= (`

```
long long a,
Xint b )  [friend]
```

5.1.4.18 operator<= [2/3] `bool operator<= (`

```
Xint & a,
long long b )  [friend]
```

5.1.4.19 operator<= [3/3] `bool operator<= (`

```
Xint & a,
Xint & b )  [friend]
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a <= b, false otherwise.

5.1.4.20 operator== `bool operator== (`

```
const Xint & x1,
const Xint & x2 )  [friend]
```

Determine if the Xint object is equivalent to the other.

Parameters

x1	Xint object.
x2	another Xint object.

Returns

Whether the two Triangle objects are the same.

5.1.4.21 operator> [1/3] bool operator> (

```
long long a,
Xint b )  [friend]
```

5.1.4.22 operator> [2/3] bool operator> (

```
Xint & a,
long long b )  [friend]
```

5.1.4.23 operator> [3/3] bool operator> (

```
Xint & a,
Xint & b )  [friend]
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a > b, false otherwise.

5.1.4.24 operator>= [1/3] bool operator>= (

```
long long a,
Xint b )  [friend]
```

5.1.4.25 operator>= [2/3] bool operator>= (

```
Xint & a,
long long b )  [friend]
```

5.1.4.26 operator>= [3/3] bool operator>= (

```
Xint & a,
Xint & b )  [friend]
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a >= b, false otherwise.

5.1.4.27 operator>> istream & operator>> (

```
istream & in,
Xint & a )  [friend]
```

5.1.5 Member Data Documentation

5.1.5.1 len int Xint::len [private]

5.1.5.2 number vector<int8_t> Xint::number [private]

The number representation is stored in the number vector. Least significant digit in number[0]. Most significant digit in number[len-1].

5.1.5.3 sign short Xint::sign [private]

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

- xint.h
- xint.cpp

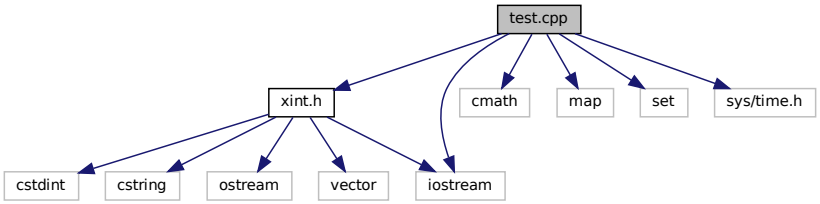
6 File Documentation

6.1 README.md File Reference

6.2 test.cpp File Reference

```
#include "xint.h"
#include <cmath>
#include <iostream>
#include <map>
#include <set>
#include <sys/time.h>
```

Include dependency graph for test.cpp:



Macros

- #define TEST
- #define pb push_back
- #define mp make_pair
- #define fi(a, b) for (int i = a; i <= b; i++)
- #define fj(a, b) for (int j = a; j <= b; j++)
- #define fo(a, b) for (int o = a; o <= b; o++)
- #define fdi(a, b) for (int i = a; i >= b; i--)
- #define fdj(a, b) for (int j = a; j >= b; j--)
- #define fdo(a, b) for (int o = a; o >= b; o--)
- #define sz(x) (int)x.size()
- #define init_v(tab, n) fo(1, n) tab.pb(0)
- #define dbg(x) { cerr << __LINE__ << "\t" << #x << ": " << x << endl; }
- #define dbg0(x, n)

Typedefs

- typedef long long ll
- typedef long double ld
- typedef vector< int > vi
- typedef pair< int, int > pii
- typedef vector< pii > vpii
- typedef pair< ll, ll > pll
- typedef vector< pll > vpll
- typedef vector< ll > vll

Functions

- template<typename T >
  ostream & operator<< (ostream &os, vector< T > v)
- template<typename A , typename B >
  ostream & operator<< (ostream &os, pair< A, B > p)
- template<typename T >
  ostream & operator<< (ostream &os, set< T > t)
- template<typename T1 , typename T2 >
  ostream & operator<< (ostream &os, map< T1, T2 > t)
- void time_mark ()
- long losuj (long pocz, long kon)
- int main ()

Variables

- vector< ll > stamps

6.2.1 Detailed Description

tests for class xint.

Author

Tadeusz Kielak tadeusz@kielak.com

Date

2023

6.2.2 Macro Definition Documentation

6.2.2.1 dbg

```
#define dbg(
        x )  { cerr << __LINE__ << "\t" << #x << ":  " << x << endl; }
```

6.2.2.2 dbg0

```
#define dbg0(
        x,
        n )
```

Value:

```
{
                \
  cerr « __LINE__ « "\t" « #x « ": ";
          \
  for (int ABC = 0; ABC < n; ABC++)
                \
    cerr « x[ABC] « ' ';
              \
  cerr « endl;
              \
}
```

6.2.2.3 fdi

```
#define fdi(
    a,
    b ) for (int i = a; i >= b; i--)
```

6.2.2.4 fdj

```
#define fdj(
    a,
    b ) for (int j = a; j >= b; j--)
```

6.2.2.5 fdo

```
#define fdo(
    a,
    b ) for (int o = a; o >= b; o--)
```

6.2.2.6 fi

```
#define fi(
    a,
    b ) for (int i = a; i <= b; i++)
```

6.2.2.7 fj

```
#define fj(
    a,
    b ) for (int j = a; j <= b; j++)
```

6.2.2.8 fo

```
#define fo(
    a,
    b ) for (int o = a; o <= b; o++)
```

6.2.2.9 init_v

```
#define init_v(
    tab,
    n ) fo(1, n) tab.pb(0)
```

6.2.2.10 mp

```
#define mp make_pair
```

6.2.2.11 pb

```
#define pb push_back
```

6.2.2.12 sz

```
#define sz(
    x ) (int)x.size()
```

6.2.2.13 TEST

```
#define TEST
```

6.2.3 Typedef Documentation

6.2.3.1 ld

```
typedef long double ld
```

6.2.3.2 ll

```
typedef long long ll
```

6.2.3.3 pii

```
typedef pair<int, int> pii
```

6.2.3.4 pll

```
typedef pair<ll, ll> pll
```

6.2.3.5 vi

```
typedef vector<int> vi
```

6.2.3.6 vll

```
typedef vector<ll> vll
```

6.2.3.7 vpii

```
typedef vector<pii> vpii
```

6.2.3.8 vpll

```
typedef vector<pll> vpll
```

6.2.4 Function Documentation

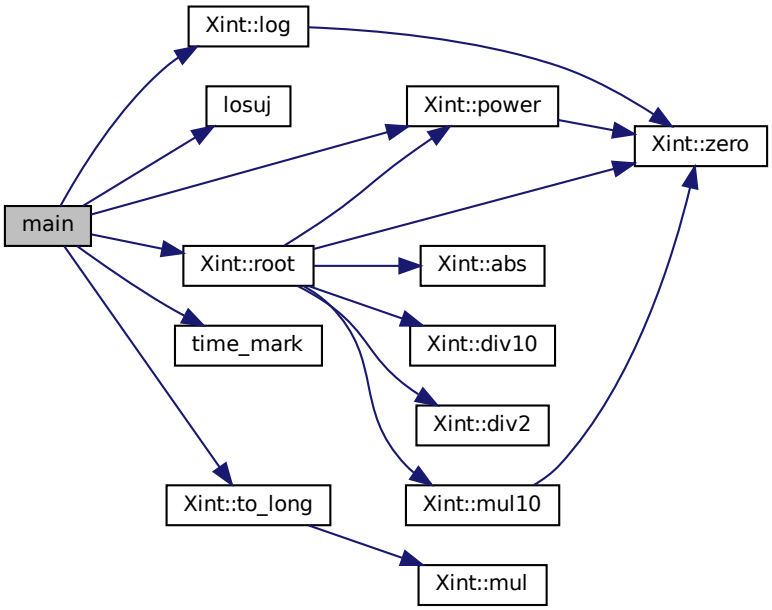
6.2.4.1 losuj() `long losuj (`

```
            long pocz,
            long kon )
```

Here is the caller graph for this function:

6.2.4.2 main() `int main ( )`

Here is the call graph for this function:



6.2.4.3 operator<<() [1/4] `template<typename T1 , typename T2 >`

```
ostream & operator<< (
            ostream & os,
            map< T1, T2 > t )
```

6.2.4.4 operator<<() [2/4] template<typename A , typename B >

```
ostream & operator<< (
            ostream & os,
            pair< A, B > p )
```

6.2.4.5 operator<<() [3/4] template<typename T >

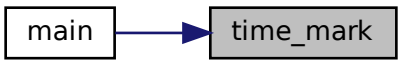
```
ostream & operator<< (
            ostream & os,
            set< T > t )
```

6.2.4.6 operator<<() [4/4] template<typename T >

```
ostream & operator<< (
            ostream & os,
            vector< T > v )
```

6.2.4.7 time_mark() void time_mark ( )

Here is the caller graph for this function:



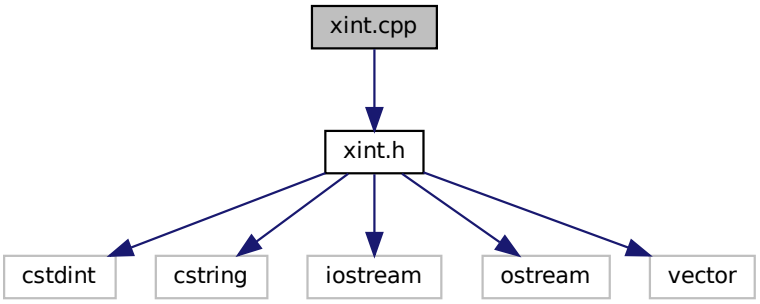
6.2.5 Variable Documentation

6.2.5.1 stamps vector<ll> stamps

6.3 xint.cpp File Reference

```
#include "xint.h"
```

Include dependency graph for xint.cpp:



Macros

- #define dbgx(x) ;
- #define dbg(x) ;

Functions

- bool operator< (Xint &a, Xint &b)

  Comparing the values of two Xint objects.
- bool operator< (Xint &a, long long b)
- bool operator< (long long a, Xint b)
- bool operator> (Xint &a, long long b)
- bool operator> (long long a, Xint b)
- bool operator<= (Xint &a, long long b)
- bool operator<= (long long a, Xint b)
- bool operator>= (Xint &a, long long b)
- bool operator>= (long long a, Xint b)
- bool operator> (Xint &a, Xint &b)

  Comparing the values of two Xint objects.
- bool operator>= (Xint &a, Xint &b)

  Comparing the values of two Xint objects.
- bool operator<= (Xint &a, Xint &b)

  Comparing the values of two Xint objects.
- bool operator< (vector< int8_t > number, Xint &b)
- istream & operator>> (istream &in, Xint &a)
- ostream & operator<< (ostream &out, const Xint &a)
- Xint & operator+= (Xint &a, Xint &b)
- Xint & operator-= (Xint &a, Xint &b)
- Xint operator+ (Xint a, Xint b)
- Xint operator- (Xint a, Xint b)
- Xint & operator*= (Xint &a, Xint &b)

- Xint operator* (Xint a, Xint b)
- Xint operator* (Xint &a, int b)
- Xint operator+ (Xint &a, int b)
- Xint & operator/= (Xint &a, Xint &b)
- Xint & operator%= (Xint &a, Xint &b)
- Xint operator/ (Xint a, Xint b)
- Xint operator% (Xint &a, Xint &b)
- bool operator== (const Xint &x1, const Xint &x2)

  Determine if the Xint object is equivalent to the other.
- bool operator!= (const Xint &x1, const Xint &x2)

  Determine if the Xint object is not equivalent to the other.

6.3.1 Detailed Description

xint implementation

Author

Tadeusz Kielak tadeusz@kielak.com

Copyright

Tadeusz Kielak

Date

2023

6.3.2 Macro Definition Documentation

6.3.2.1 dbg

```
#define dbg(
    x ) ;
```

6.3.2.2 dbgx

```
#define dbgx(
    x ) ;
```

6.3.3 Function Documentation

6.3.3.1 operator"!=()

```
bool operator!= (
        const Xint & x1,
        const Xint & x2 )
```

Determine if the Xint object is not equivalent to the other.

Parameters

x1	Xint object.
x2	another Xint object.

Returns

Whether the two Triangle objects are not the same.

6.3.3.2 operator%()

```
Xint operator% (
        Xint & a,
        Xint & b )
```

6.3.3.3 operator%=()

```
Xint & operator%= (
        Xint & a,
        Xint & b )
```

6.3.3.4 operator∗() [1/2]

```
Xint operator* (
        Xint & a,
        int b )
```

6.3.3.5 operator∗() [2/2]

```
Xint operator* (
        Xint a,
        Xint b )
```

6.3.3.6 operator∗=()

```
Xint & operator*= (
        Xint & a,
        Xint & b )
```

6.3.3.7 operator+() [1/2]

```
Xint operator+ (
        Xint & a,
        int b )
```

6.3.3.8 operator+() [2/2]

```
Xint operator+ (
    Xint a,
    Xint b )
```

6.3.3.9 operator+=()

```
Xint & operator+= (
    Xint & a,
    Xint & b )
```

6.3.3.10 operator-()

```
Xint operator- (
    Xint a,
    Xint b )
```

6.3.3.11 operator-=()

```
Xint & operator-= (
    Xint & a,
    Xint & b )
```

6.3.3.12 operator/()

```
Xint operator/ (
    Xint a,
    Xint b )
```

6.3.3.13 operator/=()

```
Xint & operator/= (
    Xint & a,
    Xint & b )
```

6.3.3.14 operator<() [1/4]

```
bool operator< (
    long long a,
    Xint b )
```

6.3.3.15 operator<() [2/4]

```
bool operator< (
    vector< int8_t > number,
    Xint & b )
```

6.3.3.16 operator<() [3/4]

```
bool operator< (
    Xint & a,
    long long b )
```

6.3.3.17 operator<() [4/4]

```
bool operator< (
    Xint & a,
    Xint & b )
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a < b, false otherwise.

6.3.3.18 operator<<() `ostream & operator<< (`

```
ostream & out,
const Xint & a )
```

6.3.3.19 operator<=() [1/3] `bool operator<= (`

```
long long a,
Xint b )
```

6.3.3.20 operator<=() [2/3] `bool operator<= (`

```
Xint & a,
long long b )
```

6.3.3.21 operator<=() [3/3] `bool operator<= (`

```
Xint & a,
Xint & b )
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a <= b, false otherwise.

6.3.3.22 operator==() `bool operator== (`

```
const Xint & x1,
const Xint & x2 )
```

Determine if the Xint object is equivalent to the other.

Parameters

x1	Xint object.
x2	another Xint object.

Returns

Whether the two Triangle objects are the same.

6.3.3.23 operator>() [1/3]

```
bool operator> (
    long long a,
    Xint b )
```

6.3.3.24 operator>() [2/3]

```
bool operator> (
    Xint & a,
    long long b )
```

6.3.3.25 operator>() [3/3]

```
bool operator> (
    Xint & a,
    Xint & b )
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a > b, false otherwise.

6.3.3.26 operator>=() [1/3]

```
bool operator>= (
    long long a,
    Xint b )
```

6.3.3.27 operator>=() [2/3]

```
bool operator>= (
    Xint & a,
    long long b )
```

6.3.3.28 operator>=() [3/3] bool operator>= (

```
Xint & a,
Xint & b )
```

Comparing the values of two Xint objects.

Parameters

a	Xint &.
b	Xint &.

Returns

A true if a >= b, false otherwise.

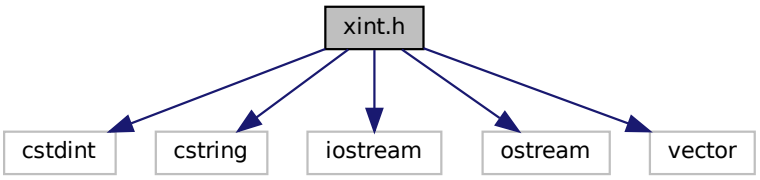
6.3.3.29 operator>>() istream & operator>> (

```
istream & in,
Xint & a )
```

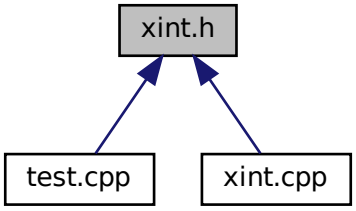
6.4 xint.h File Reference

```
#include <cstdint>
#include <cstring>
#include <iostream>
#include <ostream>
#include <vector>
```

Include dependency graph for xint.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Xint

  A Xint class.

6.4.1 Detailed Description

xint header

Author

Tadeusz Kielak tadeusz@kielak.com

Copyright

Tadeusz Kielak

Date

2023

6.5 xint.h

Go to the documentation of this file.

```
1
19 #include <cstdint>
20 #include <cstring>
21 #include <iostream>
22 #include <ostream>
23 #include <vector>
24
25 using namespace std;
26
28
31 class Xint
32 {
33 private:
39   vector<int8_t> number;
40   short sign;
41   int len;
42
```

```
43 public:
44   // constructors
45   Xint();
46   Xint(unsigned long long n);
47   Xint(unsigned long n);
48   Xint(long long n);
49   Xint(long n);
50   Xint(unsigned int n);
51   Xint(int n);
52   Xint(string &);
53   Xint(const char *);
54   Xint(const Xint &);
55   Xint(vector<int8_t> &);
56   // operator int() const {
57   //   long long outcome = 0;
58   //   long long mul = 1;
59   //   for (int i = 0; i < number.size(); i++)
60   //   {
61   //     outcome += number[i] * mul;
62   //     mul *= 10;
63   //   }
64   //   return outcome * sign;
65   // }
66   // operator long long() const {
67   //   long long outcome = 0;
68   //   long long mul = 1;
69   //   for (int i = 0; i < number.size(); i++)
70   //   {
71   //     outcome += number[i] * mul;
72   //     mul *= 10;
73   //   }
74   //   return outcome * sign;
75   // }
76   // operator unsigned long() const {
77   //   long long outcome = 0;
78   //   long long mul = 1;
79   //   for (int i = 0; i < number.size(); i++)
80   //   {
81   //     outcome += number[i] * mul;
82   //     mul *= 10;
83   //   }
84   //   return outcome * sign;
85   // }
86
87 private:
88   // auxiliary methods
89   Xint &inc_pos();
90   Xint &dec_pos();
91   Xint &plus_assign(Xint &b);
92   Xint &minus_assign(Xint &b);
93   bool less(Xint &b);
94   bool greater(Xint &b);
95   pair<vector<int8_t>, vector<int8_t» divide(Xint &b);
96
97 public:
98   void mul10();
99   void div10();
100   void mul2();
101   void div2();
102   bool zero() { return (number.size() == 1 && number[0] == 0) ? true : false; }
103   long long ll();
104   // incrementation and decrementation
105   Xint &operator++();
106   Xint operator++(int temp);
107   Xint &operator--();
108   Xint operator--(int temp);
109   Xint &operator=(long long &b);
110   Xint &operator=(unsigned long &b);
111   Xint add(Xint &b);
112   Xint sub(Xint &b);
113   Xint mul(Xint &b);
114   Xint div(Xint &b);
115   friend Xint &operator+=(Xint &, Xint &);
116   friend Xint &operator-=(Xint &, Xint &);
117   friend bool operator==(const Xint &x1, const Xint &x2);
118   friend bool operator!=(const Xint &x1, const Xint &x2);
119
120   friend Xint &operator/=(Xint &, Xint &);
121   friend Xint &operator%=(Xint &, Xint &);
122   friend bool operator<(Xint &a, Xint &b);
123   friend bool operator<(Xint &a, long long b);
124   friend bool operator<(long long a, Xint b);
125   friend bool operator>(Xint &a, long long b);
126   friend bool operator>(long long a, Xint b);
127   friend bool operator<=(Xint &a, long long b);
128   friend bool operator<=(long long a, Xint b);
129   friend bool operator>=(Xint &a, long long b);
```

```
130   friend bool operator>=(long long a, Xint b);
131
132   friend bool operator<(vector<int8_t> number, Xint &b);
133   friend Xint &operator*=(Xint &a, Xint &b);
134
135   friend Xint operator+(Xint a, Xint b);
136   friend Xint operator-(Xint a, Xint b);
137   friend Xint operator*(Xint a, Xint b);
138   friend bool operator>(Xint &a, Xint &b);
139   friend bool operator>=(Xint &a, Xint &b);
140   friend bool operator<=(Xint &a, Xint &b);
141   friend Xint operator%(Xint &a, Xint &b);
142   friend Xint operator/(Xint a, Xint b);
143   friend ostream &operator«(ostream &, const Xint &);
144   friend istream &operator»(istream &, Xint &);
145
146   long long to_long();
147
148   // maths functions
149   Xint power(long exponent);
150   Xint power(Xint exponent);
151   long log(long base);
152   Xint root(long base);
153   Xint abs();
154 };
```

Index