Core Library Reference Manual

1.7

Generated by Doxygen 1.3

Fri Aug 19 21:10:33 2005

Contents

Chapter 1

Core Library Compound Index

1.1 Core Library Compound List

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

Chapter 2

Core Library Class Documentation

2.1 BigFloat Class Reference

BigFloat is a class of Float-Point number with error bounds.

`#include <BigFloat.h>`

Helper Functions

- int **sign** () const
  sign function
- bool **isZeroIn** () const
  check whether contains zero
- BigFloat **abs** () const
  absolute value function
- int **cmp** (const BigFloat &x) const
  comparison function
- const BigInt & **m** () const
  get mantissa
- unsigned long **err** () const
  get error bits
- long **exp** () const
  get exponent
- bool **isExact** () const
  check whether err == 0

- BigFloat & **makeExact** ()

  set err to 0

- BigFloat & **makeCeilExact** ()

  set err to 0, but first add err to the mantissa (m)

- BigFloat & **makeFloorExact** ()

  set err to 0, but subtract err from the mantissa (m)

- BigFloat & **makeInexact** ()

  set err to 1

- extLong **lMSB** () const

  return lower bound of Most Significant Bit

- extLong **uMSB** () const

  return upper bound of Most Significant Bit

- extLong **MSB** () const

  return Most Significant Bit

- extLong **flrLgErr** () const

  floor of Lg(err)

- extLong **clLgErr** () const

  ceil of Lg(err)

- BigFloat **div** (const BigFloat &x, const extLong &r) const

  division with relative precsion `r`

- BigFloat **div2** () const

  exact division by 2

- BigFloat **sqrt** (const extLong &a) const

  squareroot

- BigFloat **sqrt** (const extLong &a, const BigFloat &init) const

  squareroot with initial approximation `init`

- bool **hasExactDivision** ()

  Has Exact Division.

- const BigFloat & **getZero** ()

  return BigFloat(0)

- const BigFloat & **getOne** ()

  return BigFloat(1)

Public Member Functions

Constructors and Destructor

- **BigFloat** ()
  default constructor
- **BigFloat** (int i)
  constructor for `int`
- **BigFloat** (long l)
  constructor for `long`
- **BigFloat** (double d)
  constructor for `double`
- **BigFloat** (const char ∗s)
  constructor for `const char*` *(default base = 10)*
- **BigFloat** (const std::string &s)
  constructor for `std::string`*(default base = 10)*
- **BigFloat** (int &i, const extLong &r, const extLong &a)
  constructor for `int` *and* `long`
- **BigFloat** (long &x, const extLong &r, const extLong &a)
- **BigFloat** (const BigInt &I, unsigned long er=0, long ex=0)
  constructor from `BigInt`*, error and exponent values*
- **BigFloat** (const BigRat &R, const extLong &r=defRelPrec, const extLong &a=defAbsPrec)
  constructor for `BigRat`
- **BigFloat** (const **Expr** &E, const extLong &r=defRelPrec, const extLong &a=defAbsPrec)
  constructor for **Expr**
- **BigFloat** (BigFloatRep ∗r)
  constructor for `BigFloatRep`

Copy-Assignment-Destructor

- **BigFloat** (const BigFloat &rhs)
  copy constructor
- BigFloat & **operator=** (const BigFloat &rhs)
  assignment operator
- ∼**BigFloat** ()
  destructor

Compound Assignment Operators

- BigFloat & **operator+=** (const BigFloat &x)
  operator+=

- BigFloat & **operator-=** (const BigFloat &x)
  operator-=

- BigFloat & **operator *=** (const BigFloat &x)
  operator=*

- BigFloat & **operator/=** (const BigFloat &x)
  operator/=

Unary Minus Operator

- BigFloat **operator+** () const
  unary plus

- BigFloat **operator-** () const
  unary minus

String Conversion Functions

- void **fromString** (const char *s, const extLong &p=defBigFloatInputDigits)
  set value from `const char*` *(base = 10)*

- std::string **toString** (long prec=defBigFloatOutputDigits, bool sci=false) const
  convert to `std::string` *(base = 10)*

- std::string **str** () const

Conversion Functions

- int **intValue** () const
  return int value

- long **longValue** () const
  return long value

- float **floatValue** () const
  return float value

- double **doubleValue** () const
  return double value

- BigInt **BigIntValue** () const
  return BigInt value

- BigRat **BigRatValue** () const
  return BigRat value

Utility Functions

- void **approx** (const BigInt &I, const extLong &r, const extLong &a)
  approximate BigInt number
- void **approx** (const BigFloat &B, const extLong &r, const extLong &a)
  approximate BigFloat number
- void **approx** (const BigRat &R, const extLong &r, const extLong &a)
  approximate BigRat number
- void **dump** () const
  dump internal data

Static Public Member Functions

- BigFloat **exp2** (int e)
  returns a BigFloat of value 2^e

2.1.1 Detailed Description

BigFloat is a class of Float-Point number with error bounds.

2.1.2 Member Function Documentation

2.1.2.1 bool BigFloat::isExact () const `[inline]`

check whether err == 0

Returns:
true if err == 0, otherwise false

2.1.2.2 bool BigFloat::isZeroIn () const `[inline]`

check whether contains zero

Returns:
true if contains zero, otherwise false

2.1.2.3 BigFloat& BigFloat::makeCeilExact () `[inline]`

set err to 0, but first add err to the mantissa (m)

Returns:
the ceiling exact BigFloat, variant of makeExact

2.1.2.4 BigFloat& BigFloat::makeExact () `[inline]`

set err to 0

Returns:

an exact BigFloat, see Tutorial for why this is useful!

2.1.2.5 BigFloat& BigFloat::makeFloorExact () `[inline]`

set err to 0, but subtract err from the mantissa (m)

Returns:

the floor exact BigFloat, variant of makeExact

2.1.2.6 BigFloat& BigFloat::makeInexact () `[inline]`

set err to 1

Returns:

an inexact BigFloat, see Tutorial for why this is useful!

2.1.2.7 int BigFloat::sign () const `[inline]`

sign function

Note:

This is only the sign of the mantissa, it can be taken to be the sign of the BigFloat only if !(**isZeroIn()**).

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

- BigFloat.h

2.2 Expr Class Reference

Expr is a class of Expression in Level 3.

`#include <Expr.h>`

Helper Functions

- int **sign** () const
  get the sign
- bool **isZero** () const
  is zero?
- Expr **abs** () const
  absolute value
- int **cmp** (const Expr &e) const
  compare function
- ExprRep * **Rep** () const
  return the internal representation
- long **getExponent** () const
  get exponent of current approximate value
- BigInt **getMantissa** () const
  get mantissa of current approximate value
- const Expr & **getZero** ()
  return Expr(0)
- const Expr & **getOne** ()
  return Expr(1)
- bool **hasExactDivision** ()
  Has Exact Division.

Public Types

- enum { **LIST_MODE**, **TREE_MODE**, **SIMPLE_LEVEL**, **DETAIL_LEVEL** }
  debug information levels

Public Member Functions

Constructors and Destructor

- **Expr** ()

default constructor

- **Expr** (int i)
  constructor for `int`
- **Expr** (unsigned int ui)
  constructor for `unsigned int`
- **Expr** (long l)
  constructor for `long`
- **Expr** (unsigned long ul)
  constructor for `unsigned long`
- **Expr** (float f)
  constructor for `float`
- **Expr** (double d)
  constructor for `double`
- **Expr** (const BigInt &I)
  constructor for `BigInt`
- **Expr** (const BigRat &R)
  constructor for `BigRat`
- **Expr** (const **BigFloat** &F)
  constructor for **BigFloat**
- **Expr** (const char ∗s, const extLong &p=defInputDigits)
  constructor for `const char*`
- **Expr** (const std::string &s, const extLong &p=defInputDigits)
  constructor for `std::string`
- **Expr** (const Real &r)
  constructor for `Real`
- template<class NT> **Expr** (const Polynomial< NT > &p, int n=0)
  constructor for Polynomial node (n-th root)
- template<class NT> **Expr** (const Polynomial< NT > &p, const BFInterval &I)
  constructor for Polynomial node (root in Interval `I`*)*
- **Expr** (ExprRep ∗p)
  constructor for ExprRep

Copy-Assignment-Destructors

- **Expr** (const Expr &rhs)
  copy constructor
- Expr & **operator=** (const Expr &rhs)

= operator

- **~Expr** ()
  destructor

Compound Assignment Operators

- Expr & **operator+=** (const Expr &e)
  += operator
- Expr & **operator-=** (const Expr &e)
  -= operator
- Expr & **operator *=** (const Expr &e)
  **= operator*
- Expr & **operator/=** (const Expr &e)
  /= operator

Unary Minus, Increment and Decrement Operators

- Expr **operator+** () const
  unary plus
- Expr **operator-** () const
  unary minus
- Expr & **operator++** ()
  left increment operator (++i)
- Expr **operator++** (int)
  right increment operator (i++)
- Expr & **operator–** ()
  left decrement operator (–i)
- Expr **operator–** (int)
  right deccrement operator (i–)

String Conversion Functions

- void **fromString** (const char *s, const extLong &prec=defInputDigits)
  set value from `const char*`
- std::string **toString** (long prec=defOutputDigits, bool sci=false) const
  convert to `std::string`

Conversion Functions

- int **intValue** () const
  convert to `int`
- long **longValue** () const
  convert to `long`
- float **floatValue** () const
  convert to `float`
- double **doubleValue** () const
  convert to `double`
- void **doubleInterval** (double &lb, double &ub) const
  convert to an interval defined by a pair of `double`
- BigInt **BigIntValue** () const
  convert to `BigInt` *(approximate it first!)*
- BigRat **BigRatValue** () const
  convert to `BigRat` *(approximate it first!)*
- **BigFloat BigFloatValue** () const
  convert to **BigFloat** *(approximate it first!)*

Approximation Function

- const Real & **approx** (const extLong &relPrec=defRelPrec, const extLong &absPrec=defAbsPrec) const
  *Compute approximation to combined precision [*r, a*].*

Debug Helper Function

- void **debug** (int mode=TREE_MODE, int level=DETAIL_LEVEL, int depthLimit=INT_MAX) const
  debug function

2.2.1 Detailed Description

Expr is a class of Expression in Level 3.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 Expr::Expr (float *f*) `[inline]`

constructor for `float`

Note:

the results of this constructor may appear unpredictable to the user. E.g., one may assume that new Expr(.1) is exactly equal to .1, but it will be print as .1000000000000000055511151231257827021181583404541015625. This is so because .1 cannot be represented exactly as a double (or, for that matter, as a binary fraction of any finite length). The value is the closest double value determined by the compiler.

2.2.2.2 Expr::Expr (const char ∗ *s*, const extLong & *p* = defInputDigits) `[inline]`

constructor for `const char*`

construct Expr from a string representation *s* with precision *prec*. It is perfectly predictable: new Expr(".1") is exactly equal to .1, as one would expect. Therefore, it is generally recommended that the (String) constructor be used in preference to the (double) constructor.

2.2.2.3 template<class NT> Expr::Expr (const Polynomial< NT > & *p*, int *n* = 0) `[inline]`

constructor for Polynomial node (n-th root)

default value n=0 means the first positive root

2.2.3 Member Function Documentation

2.2.3.1 const Real& Expr::approx (const extLong & *relPrec* = defRelPrec, const extLong & *absPrec* = defAbsPrec) const `[inline]`

Compute approximation to combined precision $[r, a]$.

Here is the definition of what this means: If e is the exact value and ee is the approximate value, then |e - ee| <= 2^{-a} or |e - ee| <= 2^{-r} |e|.

2.2.3.2 BigFloat Expr::BigFloatValue () const `[inline]`

convert to **BigFloat** (approximate it first!)

Ought to allow **BigFloatValue()** take an optional precision argument

2.2.3.3 void Expr::doubleInterval (double & *lb*, double & *ub*) const

convert to an interval defined by a pair of `double`

If value is exact, the two `double` will coincide

2.2.3.4 double Expr::doubleValue () const `[inline]`

convert to `double`

chen: - use equivalent precision (rel:53, abs: 1024) as in IEEE double. enforce an evaluation in case before this has been done before casting.

2.2.3.5 std::string Expr::toString (long *prec* = defOutputDigits, bool *sci* = false) const `[inline]`

convert to `std::string`

give decimal string representation

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

- Expr.h

Index