LILIB(Long Interval LIBrary)

松田 望

平成 25 年 9 月 23 日

1 概要

- C++ のライブラリ。
- LINUX 上の GCC でコンパイル可能。
- 多倍長実数クラス `LongFloat` を提供。
- 多倍長行列クラス `LongMatrix` を提供。
- 精度保証付き多倍長区間クラス `LongInterval` を提供。
- 精度保証付き多倍長区間行列クラス `LongIntervalMatrix` を提供。
- 区間値を整数の中心値・半径形式で保持。
- 多倍長演算の精度を任意に設定可能。
- 多倍長クラスはプログラム上、通常の「数」と同じように扱える。
- `LongFloat` および `LongInterval` の中心値に対する、四則演算および平方根の計算が、 correct rounding である。

2 使用方法

1. LILIB を使用するユーザプログラム `src.cpp` をコンパイルするには、
   - `src.cpp`
   - `lilib.h`
   - `libli.a`

   の三つのファイルが必要です。
2. ライブラリファイル `libli.a` は、`lilib.zip` を展開したディレクトリで、 make コマンドを実行すると生成されます。

3. src.cpp の冒頭に、

   ```
   #include "lilib.h"
   ```

   と書いてください。

4. クラス LongFloat, LongMatrix, LongInterval, LongIntervalMatrix を使用する前 (変数の宣言より前) に、関数

   ```
   void lilib::setPrecision(int precision)
   ```

   を一度だけ呼んでください。precision は、使用したい多倍長演算の精度 (10 進数の桁数) です。

5. 関数 setPrecision によって、指定された桁数以上の精度の多倍長演算が可能になります。実際に扱える桁数は、関数

   ```
   int lilib::getPrecision()
   ```

   によって取得できます。

3 サンプルプログラム

3.1 ソースコード

```
#include <iostream>
#include "lilib.h"

using namespace std;

int main(){
  lilib::setPrecision(100);
  cout << "Long precision is " << lilib::getPrecision() << "." << endl;
  cout << endl;

  LongInterval x;

  x = 1;
  cout << "x = " << x << endl;

  x /= 3;
  cout << "x = " << x << endl;
```

```
  x *= 3;
  cout << "x = " << x << endl;
  cout << endl;


  int m = 3, n = 2, i, j;
  LongIntervalMatrix a(m, n), b, c;

  for(i = 0; i < m; i++){
    for(j = 0; j < n; j++){
      a[i][j] = n * i + j;
    }
  }

  b = trans(a);
  c = a * b;

  cout << "a = " << endl << a << endl;
  cout << "b = " << endl << b << endl;
  cout << "c = " << endl << c << endl;

  return 0;
}
```

3.2 実行結果

```
Long precision is 105.

x = < 1.000000, 0.000000>
x = < 3.333333e-1, 2.537942e-116>
x = < 1.000000, 7.613826e-116>

a =
< 0.000000, 0.000000>   < 1.000000, 0.000000>
< 2.000000, 0.000000>   < 3.000000, 0.000000>
< 4.000000, 0.000000>   < 5.000000, 0.000000>

b =
```

```
< 0.000000, 0.000000>   < 2.000000, 0.000000>   < 4.000000, 0.000000>
< 1.000000, 0.000000>   < 3.000000, 0.000000>   < 5.000000, 0.000000>

c =
< 1.000000, 0.000000>   < 3.000000, 0.000000>   < 5.000000, 0.000000>
< 3.000000, 0.000000>   < 1.300000e1, 0.000000> < 2.300000e1, 0.000000>
< 5.000000, 0.000000>   < 2.300000e1, 0.000000> < 4.100000e1, 0.000000>
```

4 クラス

4.1 LongFloat クラス

4.1.1 コンストラクタ

`LongFloat()`	値は不定。

`LongFloat(int x)`
`LongFloat(double x)`
`LongFloat(LongFloat x)`
	`x` で初期化する。

4.1.2 メンバ関数

`void setDouble(double x)`	値を `x` にする。
`double getDouble()`	`double` 値を取得する。
`double getDouble(int round)`	丸めの方向を指定して `double` 値を取得する。 `round < 0` なら、下への丸め。 `round == 0` なら、最近点への丸め。 `round > 0` なら、上への丸め。
`std::string getString()`	値を表す文字列を取得する。
`std::string getInternalData()`	内部データを表す文字列を取得する。

4.1.3 非メンバ関数

`LongFloat abs(LongFloat x)`	$\|x\|$
`LongFloat pow(LongFloat x, int n)`	x^n
`LongFloat sqrt(LongFloat x)`	$\sqrt{x}$

4.2 LongMatrix クラス

4.2.1 コンストラクタ

`LongMatrix()`	1 行 1 列の行列。値は不定。
`LongMatrix(int rows, int columns)`	rows 行 columns 列の行列。値は不定。
`LongMatrix(LongMatrix x)`	`x` で初期化する。

4.2.2 メンバ関数

`void resize(int rows, int columns)`	サイズを rows 行 columns 列 にする。 値は不定になる。
`int rows()`	行数を取得する。
`int columns()`	列数を取得する。
`std::string getString()`	値を表す文字列を取得する。

4.2.3 非メンバ関数

`LongMatrix abs(LongMatrix a)`	$y_{ij} = \|a_{ij}\|$ となる行列 Y を取得する。
`LongMatrix sqrt(LongMatrix a)`	$y_{ij} = \sqrt{a_{ij}}$ となる行列 Y を取得する。
`LongMatrix trans(LongMatrix a)`	転置行列 A^T を取得する。
`LongMatrix zeros(int rows, int columns)`	rows 行 columns 列 の 全要素が 0 の行列を取得する。
`LongMatrix ones(int rows, int columns)`	rows 行 columns 列 の 全要素が 1 の行列を取得する。
`LongMatrix eye(int size)`	size 行 size 列 の単位行列を取得する。
`void qr(LongMatrix &q, LongMatrix &r,` `  LongMatrix &a)`	QR 分解を行う。

4.3 LongInterval クラス

4.3.1 コンストラクタ

`LongInterval()`	値は不定。

`LongInterval(int x)`
`LongInterval(double x)`
`LongInterval(LongFloat x)`
`LongInterval(LongInterval x)`
	`x` で初期化する。

`LongInterval(int mid, int rad)`
`LongInterval(LongFloat mid, int rad)`
`LongInterval(int mid, LongFloat rad)`
`LongInterval(LongFloat mid, LongFloat rad)`
	中心値を `mid` 、
	半径を `rad` で初期化する。

4.3.2 メンバ関数

`void setDouble(double x)`	中心値を `x` 、半径を `0` にする。

`void setMidRad(int mid, int rad)`
`void setMidRad(LongFloat mid, int rad)`
`void setMidRad(int mid, LongFloat rad)`
`void setMidRad(LongFloat mid, LongFloat rad)`
	中心値を `mid` 、半径を `rad` にする。

`double getDouble()`	`double` 値を取得する。
`std::string getMidRad()`	中心値と半径を表す文字列を取得する。
`std::string getInfSup()`	下端と上端を表す文字列を取得する。
`std::string getInternalData()`	内部データを表す文字列を取得する。
`LongFloat mid()`	中心値を取得する。
`LongFloat rad()`	半径を取得する。
`LongFloat diam()`	直径を取得する。
`LongFloat inf()`	下端を取得する。
`LongFloat sup()`	上端を取得する。
`LongFloat mag()`	最大絶対値を取得する。
`LongFloat mig()`	最小絶対値を取得する。

`int contains(int x)`
`int contains(LongFloat x)`
`int contains(LongInterval x)`
	区間が `x` を含むか判定する。
	含むなら 1 、含まないなら 0 、
	不明なら -1 を返す。
	区間の境界に重なっている場合、
	含まないとする。

`int containsEqual(int x)`
`int containsEqual(LongFloat x)`
`int containsEqual(LongInterval x)`
	区間が `x` を含むか判定する。
	含むなら 1 、含まないなら 0 、
	不明なら -1 を返す。
	区間の境界に重なっている場合、
	含むとする。

4.3.3 非メンバ関数

`LongInterval pow(LongInterval x, int n)`	x^n
`LongInterval sqrt(LongInterval x)`	$\sqrt{x}$

4.4 LongIntervalMatrix クラス

4.4.1 コンストラクタ

`LongIntervalMatrix()`	1 行 1 列の行列。値は不定。
`LongIntervalMatrix(int rows, int columns)`	rows 行 columns 列の行列。値は不定。
`LongIntervalMatrix(LongMatrix x)` `LongIntervalMatrix(LongIntervalMatrix x)`	x で初期化する。

4.4.2 メンバ関数

`void resize(int rows, int columns)`	サイズを rows 行 columns 列 にする。 値は不定になる。
`int rows()`	行数を取得する。
`int columns()`	列数を取得する。
`std::string getMidRad()`	中心値と半径を表す文字列を取得する。
`std::string getInfSup()`	下端と上端を表す文字列を取得する。
`LongMatrix mid()`	中心値を取得する。
`LongMatrix rad()`	半径を取得する。
`LongMatrix diam()`	直径を取得する。
`LongMatrix inf()`	下端を取得する。
`LongMatrix sup()`	上端を取得する。
`LongMatrix mag()`	最大絶対値を取得する。
`LongMatrix mig()`	最小絶対値を取得する。

4.4.3 非メンバ関数

`LongIntervalMatrix sqrt(LongIntervalMatrix a)`	$y_{ij} = \sqrt{a_{ij}}$ となる行列 Y を取得する。
`LongIntervalMatrix trans(LongIntervalMatrix a)`	転置行列 A^T を取得する。

5 区間の比較演算について

`LongInterval` クラスの変数は、比較演算子を用いて `int`, `LongFloat`, `LongInterval` クラスと比較できます。ただし、戻り値の型が通常の比較演算と異なるので、
注意が必要です。通常、比較演算は `bool` 型の値を返します。一方、

- `LongInterval < int`
- `LongFloat > LongInterval`
- `LongInterval >= LongInterval`

などの比較演算は、 `int` 型の値を返します。この `int` 値の内容は以下のようになります。

- 真のとき、 1
- 偽のとき、 0
- 不明のとき、 -1

LILIB では区間値を、下端・上端ではなく中心値・半径の形で保持しています。この方法には、計算の高速化や記憶容量の節約などのメリットがありますが、下端・上端が厳密に求められないことがあるというデメリットも存在します。このため、区間の比較結果として「不明」の場合が存在します。

なお、「二つの区間が等しいか」は厳密に調べることができるので、

- `LongInterval == LongInterval`
- `LongInterval != LongInterval`

の二つの演算は、従来通り `bool` 型の値を返します。