Yash Sadhwan

IIT HYDERABAD

cs13b1026

Big integer and big double library (Jan'15-Feb'15)

(B.Tech course assignment, Guide: Prof. Ramakrishna Upadrasta)

Report on Project of POPL on implementation of MyInteger and MyRational Classes

Summary

Fully Functional Big Integer and Big Double Library with Arbitary Precision.

Written in C++; Provides an API very similar to that of python terminal.

Provides almost all the operators which 'int' supports (overloaded).

Question

The basic question of the assignment was to implement MyInteger and Myrational function to ensure various mathematical operation that can't be handled by ordinary int value in C or C++.

Structure of the code

The code file contains 3 files , MyInteger.h and MyInteger.cpp for the Integer class. The other file is Myrational.cpp that contains all the class declaration and member functions of the class MyRational. The problem of circular dependencies can't allow me to have header file for MyRational .

The code in the header file includes all the declarations and the all the member function of that class.

The code in the .cpp file includes all the working code of the function. MyRational.cpp contains the main function of the whole program.

Many function the the MyInteger class has been used in the Myrational.cpp file.

Classes

The code has two classes named MyInteger and MyRational

MyInteger Class

Attributes

string a; string of any variable length

int length; //length of the string

char sign; //sign of the integer

Functions

string add(string s2); //take 2 string and add them .Using a character array and count

string sub(string s2); //Done same as addition

string mul(string s2); //takes 2 string and perform repeated addition (the basic multiplication)

string div(string s2); // takes 2 string and perform repeated subtraction (the basic division)

bool stringCompare(string s1,string s2);// Compare 2 string as they will be if they were integers

MyInteger operator+(const MyInteger &OtherObject); //Overloads the + operator by string addition and taking care of sign

MyInteger operator-(const MyInteger &OtherObject); //Overloads the - operator by string subtraction and taking care of sign

MyInteger operator*(const MyInteger &OtherObject); //Overloads the * operator by string multiplication and taking care of sign

MyInteger operator/(const MyInteger &OtherObject); //Overloads the / operator by string division and taking care of sign

MyInteger operator%(const MyInteger &OtherObject); //Overloads the % operator by string division and then subtraction

Bool Comparision Operators

This overloads the operator like >,<,>=,==,!= using the string Comparision

Overloading the cout operator

The cout Operator declaration is not in the class as it was not working. So, I have to declare it outside the class

My Rational Class

There are 2 private members of the class

MyInteger Num; //for numerator of the rational class

MyInteger Den; // for denominator of the rational class

Member Function in the MyRational class

MyRational parse(string s); //Parsing string values to make integers

string Remove_Front(string s); // Removes the front zeroes

MyRational pow(MyInteger n); //Calculates power of a number by repeated squaring

MyInteger intVal(); // Return the Mynteger value of the a float number

string floatVal(); // Return the string containing number after decimal point

All the basic mathematical operator and the comparisons operators are overloaded as same as the MyInteger class.

I have used all the operator of the MyInteger class by creating instances of them in my code.

Cout operator is overloaded to print maximum of 20 decimals places. But the answer will be stored with unlimited precision.

so, to avoid any round off errors

Test Case Data tested on the code

Input for the Test Case

7392822.7392822

*

326675651.326675651

+

24476398.24476398

pow

2

/

1408714459

/

1406360698

*

826330336.826330336

+

1552076277

+

867413454.867413454

*

868841443.868841443

/

1058756228

-

1575973466

pow

1

/

1995475535

-

193960750.193960750

-

1033392267

+

1847675179

/

18764947.18764947

pow

2

/

533046341.533046341

+

507211049.507211049

+

1215877488

pow

1

/

259107846.259107846

/

28000937.28000937

+

1130709199

*

1222379293

*

102072994.102072994

*

114075196.114075196

/

529678468.529678468

/

1055532856

*

493049562.493049562

/

1328844202

pow

3

pow

4

pow

1

-

594521642.594521642

+

729286909.729286909

pow

-1

/

417402252.417402252

*

1230623542

*

1234567887654321

*

123456789876543223456789876543 21.987654323456

*

123456789876543223456789876543 21.987654323456

*

123456789876543223456789876543 21.987654323456

*

123456789876543223456789876543 21.987654323456

*

123456789876543223456789876543 21.987654323456

*

123456789876543223456789876543212415055183497671.1387948159577122

Output for the Test Case-----

2415055207974069.3835587959577122

583249165756267552343471190063 3.41359166628641350069

414029374107721536734 4.15806875267799864776

2943977136850.57442308030003605283

2432697619102750761487.00362858929457662640

2432697619104302837764.00362858929457662640

2432697619105170251218.87104204329457662640

211362851187962899959567691 4829.48615342996937801784

199633159738034523240 1.95467811641165942683

199633159737876925893 5.95467811641165942683

199633159737876925893 5.95467811641165942683

1000429001690.95245707232119892485

1000235040940.75849632232119892485

999201648673.75849632232119892485

1001049323852.75849632232119892485

53346.77011569845403238643

2845877881.77717764662871500198

5.33889393855026906131

507211054.84610498755026906131

1723088542.84610498755026906131

1723088542.84610498755026906131

6.65008245687402475040

0.00000023749499491296

1130709199.000000023749499491296

1382155511262216597.30896397274645691674

14108075134921591837195346 2.90018355612059419514

160938143780828846965896625178177 59.78365095004810822199

303841204320751628790026484.20318961503328057594

287855752280582376.10182594278542250800

14192715272304879422304619 0.13251417604369689283

106804960663890373.97707379253216185237

12183561874532637598942235648413304642539379582272 70.27096509187244282736

22034190144396288981150089060779816556653716110756775153609477112327693592914379222459737227214179936678022754321025951555445827139646792041908703303836116801551383473733695865800488769744612039689544129 79.17036973078587515798

22034190144396288981150089060779816556653716110756775153609477112327693592914379222459737227214179936678022754321025951555445827139646792041908703303836116801551383473733695865800488769744612039689544129 79.17036973078587515798

22034190144396288981150089060779816556653716110756775153609477112327693592914379222459737227214179936678022754321025951555445827139646792041908703303836116801551383473733695865800488769744612039683598913 36.57584808878587515798

22034190144396288981150089060779816556653716110756775153609477112327693592914379222459737227214179936678022754321025951555445827139646792041908703303836116801551383473733695865800488769744612039690891782 46.30513499778587515798

0.00000000000000000000

0.00000000000000000000

0.00000000000000000000

0.00000000000000000000

0.00000000000000000000

0.00000000000000000000

0.00000000000000000000

0.00000000000000000000

0.00000000000000000000

0.00584895386308081666