bigInt.cpp

#include <iostream>

#include <string>

#include <cctype>

#include <cstdlib>

#include "bigInt.h"

using namespace std;

bigInt::bigInt()

{

}

bigInt::bigInt(long int i)

/\*this function should determine whether i is positive or

negative and set the sign accordingly. It then extracts

the digits from i and stores them in the vector digit.

This can be done by nextdigit = i % 10; i = i / 10; For

example if i is 456 then nextdigit = 456 % 10; sets

nextdigit to 6 and i = i / 10; sets i to 45 so i is ready

to have the next digit extracted.

\*/

{

long int nextdigit;

if(i < 0)

{

sign = '-';

i=abs(i);

}

else sign='+';

while(i != 0)

{

nextdigit=i % 10;

digit.push\_back(nextdigit);

i=i/10;

}

}

bigInt::bigInt(const string& s)

{ convertString(s);

}

void bigInt::print(ostream& os) const

{ int k;

os<<sign;

for(k = digit.size() - 1; k >= 0; k--) //print right to left

os<<digit[k]; //high order digits first

}

void bigInt::read(istream& is)

{ string s;

is>>s;

convertString(s);

}

bigInt& bigInt::operator++() //++ only works correctly for positive bigInts

{ int i, sum, newdigit, carry;

sum = digit[0] + 1;

newdigit = sum % 10;

carry = sum / 10;

digit[0] = newdigit;

i = 1;

while(i < digit.size() && carry != 0)

{ sum = digit[i] + carry;

newdigit = sum % 10;

carry = sum / 10;

digit[i] = newdigit;

i++;

}

if (carry != 0) digit.push\_back(carry);

return \*this;

}

void bigInt::convertString(const string& s)

{ int j, k, nextdigit;

if (s[0] == '+' || s[0] == '-')

{ sign = s[0];

k = 1; //process from subscript 1 so sign not considered again

}

else

{ sign = '+'; //no sign in string then positive number

k = 0; //no sign so process from subscript 0

}

digit.resize(0); //resize the vector digit to 0

for(j = s.size() - 1; j >= k; j--) //process digits in string from right

if (isdigit(s[j])) //to left - low order digits first

{ nextdigit = s[j] - '0'; //convert character digit to int

digit.push\_back(nextdigit);

}

else

{ cerr<<"Bad string argument for convertString function"<<endl;

cin.get(); cin.get(); //to pause console i/o screen

exit(1);

}

}

ostream& operator <<(ostream& os, const bigInt& bi)

{

bi.print(os);

return os;

}

istream& operator >>(istream& is, bigInt& bi)

{

bi.read(is);

return is;

}

bigInt bigInt::operator \*(const bigInt& bi2)const

{

const int prodsize=bi2.digit.size()+digit.size()+1;

bigInt temp;

temp.digit.resize(prodsize);

unsigned long int x;

int iplusj;

for(int i=0; i < digit.size(); i++)

{

for(int j=0; j < bi2.digit.size(); j++)

{

x=bi2.digit[j]\*digit[i];

iplusj=i+j;

temp.digit[iplusj]+=x%10;

temp.digit[iplusj+1]=temp.digit[iplusj]/10 +x/10+ temp.digit[iplusj+1];

temp.digit[iplusj]%=10;

}

}

for( int g= temp.digit.size()-1; g>=0; g--)

{

if(temp.digit[g]==0)temp.digit.pop\_back();

else if(temp.digit[g]>0)break;

}

temp.sign='+';

return temp;

}

bigInt bigInt::operator +(const bigInt& bi2)const//works only for both numbers positive.

{

int sum;

int carry=0;

int k;

int len=digit.size();

bigInt temp;

if(digit.size() == bi2.digit.size())//this addition to take place if both numbers are of equal length.

{

for(k=0;k < len; k++)

{

sum=digit[k]+bi2.digit[k]+carry;

carry=sum/10;

sum=sum%10;

temp.digit.push\_back(sum);

}

if(carry != 0)temp.digit.push\_back(carry);

}//end of code for addition if both numbers are equal.

if(digit.size() < bi2.digit.size())//if this has fewer numbers

{

for(k=0;k < digit.size();k++)//this loop will go on till the one with fewer numbers will be exhausted

{

sum=digit[k]+bi2.digit[k]+carry;

carry=sum/10;

sum=sum%10;

temp.digit.push\_back(sum);

}

for(k=digit.size();k<bi2.digit.size();k++)//this will work on the left over digits of the big digit.

{

sum=bi2.digit[k]+carry;

carry=sum/10;

sum=sum%10;

temp.digit.push\_back(sum);

}

if(carry != 0)temp.digit.push\_back(carry);

}

if(bi2.digit.size() < digit.size())//if bi2 is the smaller number, then it will be on the bottom.

{

for(k=0;k<bi2.digit.size();k++)

{

sum=digit[k]+bi2.digit[k]+carry;

carry=sum/10;

sum=sum%10;

temp.digit.push\_back(sum);

}

for(k=bi2.digit.size();k<digit.size();k++)//this will work on the left over digits of big int.

{

sum=digit[k]+carry;

carry=sum/10;

sum=sum%10;

temp.digit.push\_back(sum);

}

if(carry != 0)temp.digit.push\_back(carry);

}

temp.sign='+';

return temp;

}

const bigInt& bigInt::operator =(long int i)

{

int k=0;

int nextdigit;

digit.erase(this->digit.begin(), this->digit.end());

while(i != 0)

{

nextdigit=i % 10;

digit.push\_back(nextdigit);

i=i/10;

}

return \*this;

}

const bigInt& bigInt::operator =(const bigInt& bi2)

{

int size = bi2.digit.size(); // get vector size

digit.erase(this->digit.begin(), this->digit.end()); // erase current vector

this->digit.resize(size); // set new vector size

for(int i= 0; i < size; i++)

{

this->digit[i]=bi2.digit[i];

}

this->sign='+';

return \*this;

}

bool bigInt::operator ==(const bigInt& bi2)const

{

bool isIT=false;

if( bi2.digit.size() == this->digit.size() )

{

for(int i=0; i < this->digit.size();i++)

{

if(bi2.digit[i] != this->digit[i])return isIT;

}

isIT=true;

}

return isIT;;

}

bool bigInt::operator < (const bigInt& bi2)const

{

bool isIt=false;

if(this->digit.size() != bi2.digit.size())return( this->digit.size() < bi2.digit.size());

for(int i=0;i < this->digit.size();i++)

{

if(this->digit[i] < bi2.digit[i])return isIt=true;

}

return isIt;

}

Step-by-step explanation

**bigInt.h**

#ifndef \_bigInt\_h

#define \_bigInt\_h

#include <iostream>

#include <vector>

#include <string>

using namespace std;

class bigInt

{

public:

bigInt(); //default constructor, defined

bigInt(long int k); //construct bigInt with value like k, defined

bigInt(const string& s); //construct bigInt with value like string s ,defined

void print(ostream& os) const; //function to output a bigInt, defined

void read(istream& is); //function to input a bigInt, defined

bigInt operator+(const bigInt& bi2) const; //overloads + operator

bigInt operator\*(const bigInt& bi2) const; //overloads \* operator

bigInt& operator++(); //overloads pre-increment op, defined

const bigInt& operator=(const bigInt& bi2); //copy assignment operator, defined

const bigInt& operator=(long int i); //converts i to bigInt and assigns

//overload comparison operators

bool operator==(const bigInt& bi2) const; //defined

bool operator<(const bigInt& bi2) const;//defined

private:

char sign; //stores the sign '+' or '-'. Not really

//needed in this program because you only have

//to handle positive numbers (numbers >= 0), but

//would be needed in a complete program.

vector<int> digit; //vector to store digits of a bigInt with

//one digit in each element of the vector.

//A more efficient version of the program

//would store several digits in each element.

//digit[0] is the 1's digit, digit[1] is the 10's

//digit, digit[2] is the 100's digit, etc.

void convertString(const string& s); //converts string into bigInt

//representaion., defined

};

//overloads << and >> operators as non-member functions using

//public member functions print and read.

ostream& operator<<(ostream& os, const bigInt& bi); //defined

istream& operator>>(istream& is, bigInt& bi);//defined

#endif

Test program

#include <iostream>

#include "bigInt.h"

int main()

{ bigInt bi1, bi2("99999999999999999999999999999999"), bi3(7777777);

cout<<"bi2 = "<<bi2<<endl;

cout<<"bi3 = "<<bi3<<endl;

bi1 = bi2 + bi3;

cout<<"sum = "<<bi1<<endl;

bi1 = bi2 \* bi3;

cout<<"product = "<<bi1<<endl;

cout<<"Enter a bigInt: ";

cin>>bi1; //enter as input for this 9876543210

cout<<"input value = "<<bi1<<endl;

if(!(bi1 == bi2)) cout<<"bi1 != bi2"<<endl;

bi2 = bi1;

if(bi1 == bi2) cout<<"bi1 == bi2"<<endl;

bi1 = 1;

bi3 = 31;

for(bi2 = 2; bi2 < bi3; ++bi2)

bi1 = bi1 \* bi2;

cout<<"factorial of 30 = "<<bi1<<endl;

cin.get(); cin.get(); //to pause console i/o screen

//press enter key 1 or 2 times to end

return 0;

}