#include <iostream>

#include <string>

using namespace std;

struct bignum {

protected:

int length;

int capacity;

int \* numbers;

bool neg;

public:

bignum(string n) {

if (n[0]=='-') neg = true;

else neg = false;

if (neg==true) { //if reading in a negative number don't

length = n.length()-1; //include the negative '-' in the numbers array

capacity = n.length()-1; //the operator functions will read this number

numbers = new int[length]; //as if it was positive

for (int i=length; i>0; i--)

numbers[length-i] = n[i]-'0';

}

else if (neg==false) { //the normal case where a positive number is given

length = n.length();

capacity = n.length();

numbers = new int[length];

for (int i=length-1; i>=0; i--)

numbers[length-i-1] = n[i]-'0'; //copying the list into "numbers" backwards

} //so that we can work with it

}

bignum(int size) { //this constructor creates a bignum with a given size

capacity = size; //setting all the values to initially NULL

length = 0;

neg = false;

numbers = new int[capacity];

for (int i=capacity-1; i>=0; i--)

numbers[i] = NULL;

}

~bignum() {

delete [] numbers;

}

void print() {

if (length==1 && numbers[0]==0) { //in the case that the final answer is 0

cout << numbers[0] << "\n"; //prevents the next while loop from removing it

return;

}

while (numbers[length-1]==0) { //this loop removes 0's from the beginning of a number

length--;

int \* a = new int[length];

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

a[i] = numbers[i];

delete numbers;

numbers = a;

}

if (neg) cout << "-";

for (int i=length-1; i>=0; i--)

cout << numbers[i];

cout << "\n";

}

bool getneg()

{ return neg; }

void setpos()

{ neg = false; }

void setneg()

{ neg = true; }

void resize() {

capacity \*= 2;

int \* a = new int[capacity];

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

a[i] = numbers[i];

delete [] numbers;

numbers = a;

}

int get\_length() {

return length;

}

int get(int pos) {

if (pos>=length) {

// cerr << "Error in get - position is out of scope\n";

return -1;

}

if (pos<0) {

// cerr << "Error in get - position is negative\n";

return -1;

}

return numbers[pos];

}

void pushback(int value) {

if (length>=capacity) resize();

if (value>=0 && value<10) {

numbers[length] = value;

length++;

// cout << "Pushing back " << value << "\n";

}

else if (value<0) cerr << "Error in pushback - value is negative\n";

else if (value>9) cerr << "Error in pushback - value is greater than 9\n";

}

};

bool absfirstbigger(bignum\* a, bignum\* b) {

if (a->get\_length()>b->get\_length())

return true;

else if (b->get\_length()>a->get\_length())

return false;

if (a->get\_length()==b->get\_length()) {

for (int i=(a->get\_length()-1); i>=0; i--) {

if (a->get(i)>b->get(i))

return true;

else if (b->get(i)>a->get(i))

return false;

}

return true; //if the two numbers are the same, it outputs true

}

return true;

}

bool firstbigger(bignum\* a, bignum\* b) { //accounts for negatives when comparing bignums

if (a->getneg() && b->getneg())

return absfirstbigger(b,a);

else if (a->getneg()==false && b->getneg())

return true;

else if (a->getneg() && b->getneg()==false)

return false;

else if (a->getneg()==false && b->getneg()==false)

return absfirstbigger(a,b);

}

bignum\* absadd(bignum\* x, bignum\* y) {

int cap;

if (x->get\_length()>=y->get\_length())

cap = x->get\_length()+1;

else cap = y->get\_length()+1;

bignum \* result = new bignum(cap);

int sum = 0;

bool carry = false; //carry is initially not needed

for (int i=0; i<cap; i++) {

if (x->get(i)==-1 && y->get(i)==-1) { //if the last place has been reached

if (carry) result->pushback(1); //if the last place has only a carry

return result;

}

else if (x->get(i)==-1) //if one bignum has reached its end

sum = y->get(i);

else if (y->get(i)==-1) //if the other has reached its end

sum = x->get(i);

else sum = x->get(i) + y->get(i); //the normal addition of both bignums

if (carry==true) { //accounting for the carry

sum++;

carry = false;

}

if (sum>9) { //checking if a carry needs to be used

sum = sum-10;

carry = true;

}

else if (sum<10) { //the normal condition. setting carry to not be used

carry = false;

}

result->pushback(sum);

}

return result;

}

bignum\* abssubtract(bignum\* first, bignum\* second) {

if (first->get\_length()<second->get\_length()) { //checking if the result will produce a negative # (not allowed)

cerr << "Error in subtract - first number is smaller than second number\n";

return NULL;

}

if (first->get\_length()==second->get\_length()) { //checking when the two are the same length

for (int i=(first->get\_length())-1; i>=0; i--) {

if (i==(first->get\_length()-1) && first->get(i)<second->get(i)) { //only compares the most significant digit to check if the first is smaller

cerr << "Error in subtract - first number is smaller than second number 2\n";

return NULL;

}

}

}

int cap = first->get\_length();

bignum \* result = new bignum(cap); //defining and initializing result bignum here

int diff = 0; //used in place of "sum"

bool carry = false; //used for "carrying over" when subtracting a bigger from a smaller

for (int i=0; i<cap; i++) {

if (first->get(i)==-1 && second->get(i)==-1) { //if the last place has been reached

return result;

}

else if (second->get(i)==-1) { //if the the smaller num has reached its end

if (carry) { //if there was a carry from the last position

if (first->get(i)==0) { //if a FURTHER carry must be applied

diff = first->get(i)+10-1;

carry = true;

}

else {

diff = first->get(i)-1;

carry = false;

}

}

else diff = first->get(i);

}

else if (carry) { //checking if a carry was previously used

if ((first->get(i)-1)<second->get(i)) { //checking if a carry needs to be used (again)

diff = (first->get(i)+10-1)-second->get(i); //+10 accounts for the current carry

carry = true; //-1 accounts for the last carry that had to be executed

}

else { //else not. difference here is carry = false and 10

diff = (first->get(i)-1)-second->get(i); //does not need to be added

carry = false;

}

}

else if (first->get(i)<second->get(i)) { //checking if a carry needs to be used

diff = (first->get(i)+10)-second->get(i);

carry = true;

}

else if (first->get(i)>=second->get(i)) { //the normal case. no carry is needed

diff = first->get(i)-second->get(i);

carry = false;

}

if (i==cap-1 && diff==0) return result;

result->pushback(diff);

}

return result;

}

bignum\* absmultiply(bignum\* a, bignum\* b) {

if (a->get\_length()==1 && a->get(0)==0) { //if either bignum is 0 result is 0

bignum \* result = new bignum(1);

result->pushback(0);

return result;

}

if (b->get\_length()==1 && b->get(0)==0) {

bignum\* result = new bignum(1);

result->pushback(0);

return result;

}

int cap = a->get\_length()+b->get\_length();

bignum \* result = new bignum(cap);

int sum = 0, carrycounter = 0;

bool carry = false;

for (int i=0; i<cap; i++) {

sum = 0;

for (int j=0; j<cap; j++) {

if (i-j<0)

break;

if (a->get(i-j)==-1 || b->get(j)==-1) //if the position accessed for either number does not exist

sum = sum;

else //the normal case

sum = sum + (a->get(i-j) \* b->get(j));

}

if (carry) { //if the previous digit's sum was over 9

sum = sum+carrycounter;

carrycounter = 0;

}

if (sum>9) { //if the current digit's sum is over 9 (needs carry)

while (sum>9) {

sum = sum-10;

carrycounter++;

carry = true;

}

}

else carry = false;

result->pushback(sum);

}

return result;

}

bignum\* absfactorial(bignum\* a) {

bignum \* result = new bignum("1");

bignum \* one = new bignum("1");

bignum \* decrement = a;

bignum \* temp;

while (firstbigger(decrement,one)) {

temp = result;

result = absmultiply(result,decrement); //decrement is always 1 less. in the first case this will

delete temp; //multiply (for example, 65!) 65 (decrement) by 1 (result)

temp = decrement; //in the second run through it will multiply 64 (decrement)

decrement = abssubtract(decrement,one); //by 65 (result) etc

delete temp; //temp is used to prevent memory leaks

}

return result;

}

bignum\* add(bignum\* x, bignum\* y) {

bignum\* result;

if (x->getneg()==true && y->getneg()==false) {

if (firstbigger(x,y)) {

result = abssubtract(x,y);

result->setneg();

return result;

}

else if (firstbigger(x,y)==false) {

result = abssubtract(y,x);

result->setpos();

return result;

}

}

else if (x->getneg()==false && y->getneg()==true) {

if (firstbigger(x,y)) {

result = abssubtract(x,y);

result->setpos();

return result;

}

else if (firstbigger(x,y)==false) {

result = abssubtract(y,x);

result->setneg();

return result;

}

}

else if (x->getneg()==false && y->getneg()==false) {

result = absadd(x,y);

result->setpos();

return result;

}

else if (x->getneg()==true && y->getneg()==true) {

result = absadd(x,y);

result->setneg();

return result;

}

}

bignum\* subtract(bignum\* x, bignum\* y) {

bignum\* result;

if (x->getneg()==true && y->getneg()==false) {

result = absadd(x,y);

result->setneg();

return result;

}

else if (x->getneg()==false && y->getneg()==true) {

result = absadd(x,y);

result->setpos();

return result;

}

else if (x->getneg()==false && y->getneg()==false) {

if (firstbigger(x,y)) {

result = abssubtract(x,y);

result->setpos();

return result;

}

else if (firstbigger(x,y)==false) {

result = abssubtract(y,x);

result->setneg();

return result;

}

}

else if (x->getneg()==true && y->getneg()==true) {

if (firstbigger(x,y)) {

result = abssubtract(x,y);

result->setneg();

return result;

}

else if (firstbigger(x,y)==false) {

result = abssubtract(y,x);

result->setpos();

return result;

}

}

}

bignum\* multiply(bignum\* x, bignum\* y) {

bignum\* result;

result = absmultiply(x,y);

if ((x->getneg()==true && y->getneg()==false) || (x->getneg()==false && y->getneg()==true))

result->setneg();

else if ((x->getneg()==true && y->getneg()==true) || (x->getneg()==false && y->getneg()==false))

result->setpos();

return result;

}

bignum\* factorial(bignum\* x) {

bignum\* result;

if (x->getneg()==false) {

result = absfactorial(x);

result->setpos();

return result;

}

else if (x->getneg()==true)

return NULL;

}

bignum\* absdivide(bignum\* x, bignum\* y) {

if (y->get\_length()==1 && y->get(0)==0) //when trying to divide by 0

return NULL;

else if (y->get\_length()>x->get\_length()) { //if the divisor is bigger than the dividend the quotient will

bignum \* result = new bignum(1); //be rounded to 1

result->pushback(0);

return result;

}

else if (y->get\_length()==x->get\_length()) {

for (int i=(y->get\_length()-1); i>=0; i--) {

if (y->get(i)>x->get(i)) { //same as before

bignum \* result = new bignum(1);

result->pushback(1);

return result;

}

if (i==0 && y->get(i)==x->get(i)) { //if the two numbers are exactly the same the quotient will be 1

bignum \* result = new bignum(1);

result->pushback(1);

return result;

}

}

}

bignum\* counter = new bignum("0");

bignum\* temp = x;

bignum\* zero = new bignum("0");

bignum\* one = new bignum("1");

bignum\* cleanup;

while(true) {

cleanup = temp;

temp = subtract(temp,y);

delete cleanup;

if (firstbigger(temp,zero)==false)

break;

counter = add(counter,one);

// cout << "test\n";

// temp->print();

// y->print();

// counter->print();

}

bignum \* result = counter;

return result;

}

bignum\* divide(bignum\* x, bignum\* y) {

bignum\* result;

if ((x->getneg()==true && y->getneg()==false) || (x->getneg()==false && y->getneg()==true)) {

x->setpos(); y->setpos();

result = absdivide(x,y);

result->setneg();

return result;

}

else if ((x->getneg()==true && y->getneg()==true) || (x->getneg()==false && y->getneg()==false)) {

x->setpos(); y->setpos();

result = absdivide(x,y);

result->setpos();

return result;

}

}

void main() {

bignum \* a = new bignum("-100");

bignum \* b = new bignum("25");

bignum \* c = new bignum("49");

bignum \* d = new bignum("12");

bignum \* e = new bignum("67");

bignum \* f = new bignum("66");

bignum \* g = divide(a,b);

if (g!=NULL) {

cout << "-100/25 = ";

g->print();

}

else cout << "Error divide by 0\n";

bignum \* h = divide(c,d);

if (h!=NULL) {

cout << "49/12 = ";

h->print();

}

else cout << "Error divide by 0\n";

bignum \* i = divide(e,f);

if (i!=NULL) {

cout << "67/66 = ";

i->print();

}

else cout << "Error divide by 0\n";

}

**Output**

Macintosh HD:Users:nthn888:Desktop:Screen shot 2013-04-30 at 1.20.09 AM.png