Fraser Burch

CS 60 Winter

Homework 4

**Problem 1:**

**Code:**

#include <iostream>

#include <string>

using namespace std;

void swap(int \*a, int \*b){

int \* tmpHolder;

\*tmpHolder = \*a;

\*a = \*b;

\*b = \*tmpHolder;

}

int main(){

int a, b, \* x, \* y;

a = 10;

b = 5;

x = &a;

y = &b;

cout<<"a is: "<<a<<", b is: "<<b<<endl;

cout<<"\*x is: "<<\*x<<", \*y is: "<<\*y<<endl;

swap(x, y);

cout<<"a is: "<<a<<", b is: "<<b<<endl;

cout<<"\*x is: "<<\*x<<", \*y is: "<<\*y<<endl;

return 0;

}

**Output:**

FHosts-MacBook-Pro:Homework 4 fhost$ ./main

a is: 10, b is: 5

\*x is: 10, \*y is: 5

a is: 5, b is: 10

\*x is: 5, \*y is: 10

**Problem 2:**

**Main.cpp Code:**

#include "dbiguint.h"

#include <iostream>

using namespace std;

int main(){

string str = "10";

string str2 = "100";

dbiguint defaultConst = dbiguint();

dbiguint small = dbiguint(str);

dbiguint big = dbiguint(str2);

//Testing the Problem 2 - step 6 from lab

cout <<"Comparing " << small << " to " << big <<endl;

cout<<endl;

int resultOfCompare = small.compare(big);

cout << "Should be -1. " << resultOfCompare<<endl;

bool test1 = (small < big); //Should be true

cout<< "Should be true " << test1 << endl;

bool test3 = (small <= big); //Should be true

cout<< "Should be true " << test3 << endl;

bool test4 = (small != big); //Should be true

cout<< "Should be true " << test4 << endl;

bool test5 = (small == big); //Should be false

cout<< "Should be false " << test5 << endl;

bool test6 = (small > big);//Should be false

cout<< "Should be false " << test6 << endl;

bool test7 = (small >= big); //Should be false

cout<< "Should be false " << test7 << endl;

//Testing the overloaded operator (+)

dbiguint added = small+big;

cout<< added << endl;

//Testing copy constructor

dbiguint copyOfBig = dbiguint(big);

cout<<"This should be a copy of Big("<<big<<")"<< copyOfBig<<endl;

//Testing the overloaded = operator

dbiguint copyOfSmall = small;

cout<<"This should be a copy of Small("<<small<<")"<<copyOfSmall<<endl;

return 10000;

}

**Output:**

FHosts-MacBook-Pro:Lab 4 fhost$ ./main

Comparing 1 0 to 1 0 0

Should be -1. -1

Should be true 1

Should be true 1

Should be true 1

Should be false 0

Should be false 0

Should be false 0

1 1 0

This should be a copy of Big(1 0 0 )1 0 0

This should be a copy of Small(1 0 )1 0

**Problem 3:**

**Main.cpp Code just for testing:**

#include "dbiguint.h"

#include <iostream>

using namespace std;

int main(){

string str = "10";

string str2 = "100";

dbiguint defaultConst = dbiguint();

dbiguint small = dbiguint(str);

dbiguint big = dbiguint(str2);

/\*

----------------------START OF PROBLEM 3------------------------------------------

\*/

//Testing modified reserve

string reserveTest = "178";

string reserve2Test = "9999999";

dbiguint small22 = dbiguint(reserveTest);

dbiguint big22 = dbiguint(reserve2Test);

small22.reserve(big22.size());

for(size\_t i = 0; i < small22.size(); i++){

cout<<small22[i];

}

cout<<""<<endl;

small22.reserve(1);

for(size\_t i = 0; i < small22.size(); i++){

cout<<small22[i];

}

cout<<""<<endl;

//Testing -= operator

string top = "150";

string top1 = "250";

string bottom = "50";

dbiguint topBig = dbiguint(top);

dbiguint bottomBig = dbiguint(bottom);

topBig-=bottomBig;

cout<< "(" << top << "- "<<bottomBig << ") = " << topBig<< endl;

//Testing - operator

dbiguint topBig2 = dbiguint(top1);

dbiguint bottomBig2 = dbiguint(bottom);

dbiguint subtracted = topBig2-bottomBig2;

cout<< "(" << topBig2 << "- " <<bottomBig2 << ") = " << subtracted << endl;

**Output:**

FHosts-MacBook-Pro:Lab 4 fhost$ ./main

8710000

871

(150- 5 0) = 1 0 0

(2 5 0 - 5 0) = 2 0 0

**Problem 4:**

**Main.cpp Code just for testing:**

#include "dbiguint.h"

#include <iostream>

using namespace std;

int main(){

string str = "10";

string str2 = "100";

dbiguint defaultConst = dbiguint();

dbiguint small = dbiguint(str);

dbiguint big = dbiguint(str2);

\*/

cout<<"--------Start of Problem 4--------"<<endl;

dbiguint mult1 = dbiguint(top);

dbiguint mult2 = dbiguint(bottom);

mult1\*=mult2;

cout<< mult1<<endl;

//Testing just the \* operator

dbiguint multTest = mult1\*mult2;

cout<< mult1 << " times " << mult2 << " equals " << multTest<<endl;

return 10000;

}

**Output:**

--------Start of Problem 4--------

7 5 0 0

7 5 0 0 times 5 0 equals 3 7 5 0 0 0

**-------------------------DBIGUINT.CPP:----------------------------------**

#include "dbiguint.h"

#include <iostream>

dbiguint::dbiguint(){

data\_ = new unsigned short[1];

data\_[0] = 0;

capacity\_ = 1;

}

dbiguint::dbiguint(const std::string &a){

data\_ = new unsigned short[a.size()];

capacity\_ = a.size();

int j = 0;

char zero = '0';

int codeZero = zero;

for(std::size\_t i = 1; i < a.size()+1; i++){

int tempNum = a[a.length() - i];

int numCode = tempNum - codeZero;

data\_[j]=numCode;

j++;

}

}

std::size\_t dbiguint::size() const{

return capacity\_;

}

unsigned short dbiguint::operator [](std::size\_t pos) const{

if (pos < capacity\_){return dbiguint::data\_[pos];}

else{return 0;}

}

std::ostream& operator <<(std::ostream &out, const dbiguint &b){

for(size\_t i = b.size()-1; i >0; i--){

//std::cout<<i<<std::endl;

out<<b[i]<<" ";

}

out<<b[0]<<" ";

//out << std::endl;

return out;

}

void dbiguint::reserve(std::size\_t newcapacity\_){

//For increasing size

if(newcapacity\_ > capacity\_){

unsigned short \* tempData\_ = new unsigned short[newcapacity\_];

for (size\_t i = 0; i < capacity\_; i++){

tempData\_[i] = data\_[i];

}

for(size\_t j = capacity\_; j < newcapacity\_; j++){

tempData\_[j] = 0;

}

capacity\_ = newcapacity\_;

delete [] data\_;

data\_ = tempData\_;

tempData\_ = nullptr;

}

//For decreasing size

else if(newcapacity\_ < capacity\_){

size\_t initialBootOut = 0;

size\_t leadingZeros = 0;

for(size\_t i = (capacity\_-1); i >0; i--){

if(data\_[i] == 0 && initialBootOut == 0){

leadingZeros++;

}

else{

initialBootOut = 1;

}

}

size\_t newCap = capacity\_-leadingZeros;

unsigned short \* tempData2\_ = new unsigned short[newCap];

for(size\_t j = 0; j < newCap; j++){

tempData2\_[j] = data\_[j];

}

capacity\_ = newCap;

delete [] data\_;

data\_ = tempData2\_;

tempData2\_ = nullptr;

}

}

void dbiguint::operator +=(const dbiguint & b){

if(b.size() > capacity\_){

reserve(b.size());

}

else{

reserve(capacity\_);

}

for(size\_t i = 0; i < capacity\_; i++){

if (data\_[i] + b.data\_[i] > 9){

data\_[i+1] = data\_[i+1] + 1;

int tempHolder = data\_[i] + b.data\_[i];

tempHolder = tempHolder%10;

data\_[i] = tempHolder;

}

else{

data\_[i] = (data\_[i] + b.data\_[i]);

}

}

}

/\*

--------------------------------------------------------------------------------

PROBLEM 2 - Steps 5 - 7 from Lab 4

\*/

dbiguint:: ~dbiguint(){

delete [] data\_;//This is the only piece we NEED

data\_ = nullptr;

capacity\_ = 0;

}

int dbiguint::compare(const dbiguint & b) const{

int initialBoot = 0;

int initialBoot2 = 0;

size\_t numDigits = 0;

size\_t bNumDigits = 0;

int toReturn = 0;

//For class within dbiguint

for(size\_t i = capacity\_-1; i > 0; i--){

if(data\_[i] == 0 && initialBoot == 0){numDigits++;}

else{

initialBoot = 1;

}

}

//For parameter dbiguint

for(size\_t i = b.size()-1; i > 0; i--){

if(b[i] == 0 && initialBoot2 == 0){bNumDigits++;}

else{

initialBoot2 = 1;

}

}

size\_t numTrueDigits = capacity\_ - numDigits;

size\_t bnumTrueDigits = b.capacity\_ - bNumDigits;

if (numTrueDigits < bnumTrueDigits){ //If parameter has more true digits

toReturn = -1;

}

else if (numTrueDigits > bnumTrueDigits){ //If parameter has more true digits

toReturn = 1;

}

else{ //if number of true digits match

for(size\_t i = b.size()-1; i >= 1; i--){

if(data\_[i] > b[i]){

toReturn = 1;

break;

}

else if(data\_[i] < b[i]){

toReturn = -1;

break;

}

else{

toReturn = 0;

}

}

}

return (toReturn);

}

dbiguint operator +(const dbiguint &b1, const dbiguint &b2){

dbiguint tempBigUInt;

tempBigUInt += b1;

tempBigUInt += b2;

return tempBigUInt;

}

bool operator < (const dbiguint &b1, const dbiguint &b2){

if(b1.compare(b2) == -1){

return true;

}

return false;

}

bool operator <= (const dbiguint &b1, const dbiguint &b2){

if(b1.compare(b2) == -1 || b1.compare(b2) == 0){

return true;

}

return false;

}

bool operator == (const dbiguint &b1, const dbiguint &b2){

if(b1.compare(b2) == 0){

return true;

}

return false;

}

bool operator != (const dbiguint &b1, const dbiguint &b2){

if(b1.compare(b2) != 0){

return true;

}

return false;

}

bool operator >= (const dbiguint &b1, const dbiguint &b2){

if(b1.compare(b2) == 1 || b1.compare(b2) == 0){

return true;

}

return false;

}

bool operator > (const dbiguint &b1, const dbiguint &b2){

if(b1.compare(b2) == 1){

return true;

}

return false;

}

//Copy constructor

dbiguint::dbiguint(const dbiguint &b){

data\_ = new unsigned short[b.size()];

capacity\_ = b.size();

for(std::size\_t i = 0; i < b.size(); i++){

data\_[i]=b[i];

}

}

//Overloaded = operator to return a DBIGUINT

dbiguint dbiguint::operator =(const dbiguint &b){

dbiguint toReturn = dbiguint(b);

return(toReturn);

}

/\*

--------------------------------------------------------------------------------

END OF PROBLEM 2

\*/

/\*

--------------------------------------------------------------------------------

START OF PROBLEM 3

\*/

dbiguint dbiguint::operator -= (const dbiguint & b){

reserve(1);

for(size\_t i = 0; i < capacity\_; i++){

if(data\_[i] > b[i]){

data\_[i] = (data\_[i] - b[i]);

}

else if(data\_[i] < b[i]){

int tempHolder = data\_[i] + 10;

tempHolder = tempHolder - b[i];

data\_[i+1] = data\_[i+1]-1;

data\_[i] = tempHolder;

}

else{

data\_[i] = 0;

}

}

reserve(1);

return(\*this);

}

dbiguint operator -(const dbiguint &b1, const dbiguint &b2){

dbiguint newTemp = dbiguint(b1);

newTemp-= b2;

return(newTemp);

}

/\*

--------------------------------------------------------------------------------

END OF PROBLEM 3

\*/

/\*

--------------------------------------------------------------------------------

START OF PROBLEM 4

\*/

dbiguint dbiguint::operator \*=(const dbiguint & b){

//b.reserve(1);

size\_t firstBigInt = 0;

size\_t secondBigInt = 0;

size\_t tensSpot = 1;

dbiguint newTemp;

for(size\_t i = 0; i < capacity\_; i++){

firstBigInt += data\_[i]\*tensSpot;

tensSpot = tensSpot\*10;

}

tensSpot = 1;

for(size\_t i = 0; i < b.size(); i++){

secondBigInt += b[i]\*tensSpot;

tensSpot = tensSpot\*10;

}

size\_t holder = secondBigInt\*firstBigInt;

size\_t digitCount = 10;

size\_t numDigitsHolder = 1;

while((holder % digitCount) != holder){

digitCount = digitCount\*10;

numDigitsHolder++;

}

size\_t modBy = 10;

size\_t divideBy = 1;

int hit = 0;

reserve(numDigitsHolder);

for(size\_t i = 0; i < numDigitsHolder; i++){

data\_[i] = (holder % modBy)/divideBy;

modBy = modBy \*10;

divideBy = divideBy\*10;

}

return (\*this);

}

dbiguint operator \*(const dbiguint &b1, const dbiguint &b2){

dbiguint toReturn = dbiguint(b1);

toReturn\*=b2;

return toReturn;

}

std::istream & operator >> (std::istream &in, dbiguint &b){

char charInt;

for(size\_t i = b.size()-1; i >0; i--){

charInt = (char)b[i];

in >> charInt;

}

return in;

}

/\*

--------------------------------------------------------------------------------

END OF PROBLEM 4

\*/

**-------------------------DBIGUINT.H:----------------------------------**

#ifndef DBIGUINT\_H

#define DBIGUINT\_H

#include <cstdlib>

#include <string>

class dbiguint

{

public:

// pre: none

// post: creates a dynamic bigint value 0

dbiguint();

// pre: s[0], ..., s[s.size()-1] are digits

// post: creates a dbiguint whose digits are given in s

dbiguint(const std::string & s);

//We aren't implementing this constructor yet

// pre: none //NEW

// post: copy constructor: creates a new dynamic bigint which is

// a copy of given dynamic bigint

dbiguint(const dbiguint &b);

// pre: none //NEW

// post: returns dynamically allocated memory to heap

~dbiguint();

// pre: none //NEW

// post: makes this dynamic bigint a copy of given dynamic bigint

dbiguint operator =(const dbiguint &b);

// pre: none //NEW

// post: returns the size of the memory block of this dbiguint

std::size\_t size() const;

// pre: none

// post: returns the digit at given pos (0 if does not exist)

// pos 0 is the least significant (units) digit

unsigned short operator [](std::size\_t pos) const;

// pre: none

// post: returns 0 if this dbiguint equals given dbiguint

// 1 if this dbiguint > given dbiguint

// -1 otherwise

int compare(const dbiguint & b) const;

// pre: none

// post: returns a string containing the digits and sign of this dbiguint

// std::string toStdString() const;

// pre: none

// post: adds/subtracts given dbiguint to this dbiguint

void operator +=(const dbiguint & b);

dbiguint operator -=(const dbiguint & b);

dbiguint operator \*=(const dbiguint & b);

// pre: none

// post: if newcapacity\_ <= capacity\_ then do nothing (cannot shrink)

// else allocate a new block with size newcapacity\_

// copy existing digits and fill the rest with 0

void reserve(std::size\_t newcapacity\_);

private:

unsigned short \*data\_;

std::size\_t capacity\_;

// INVARIANTS:

// data\_ points to (has the address of) a dynamic array

// of capacity\_ digits

// data\_[0] = least significant (units) digits

// data\_[k] = digit at position k (or 0 if not used)

};

// nonmember functions

dbiguint operator \*(const dbiguint &b1, const dbiguint &b2);

dbiguint operator +(const dbiguint &b1, const dbiguint &b2);

dbiguint operator -(const dbiguint &b1, const dbiguint &b2);

bool operator < (const dbiguint &b1, const dbiguint &b2);

bool operator <= (const dbiguint &b1, const dbiguint &b2);

bool operator == (const dbiguint &b1, const dbiguint &b2);

bool operator != (const dbiguint &b1, const dbiguint &b2);

bool operator >= (const dbiguint &b1, const dbiguint &b2);

bool operator > (const dbiguint &b1, const dbiguint &b2);

std::ostream & operator << (std::ostream &out, const dbiguint &b);

std::istream & operator >> (std::istream &in, dbiguint &b);

#endif // DBIGUINT\_H

**-------------------------Entire Homework 4 Main.CPP:-----------------------------**

#include "dbiguint.h"

#include <iostream>

using namespace std;

int main(){

string str = "10";

string str2 = "100";

dbiguint defaultConst = dbiguint();

dbiguint small = dbiguint(str);

dbiguint big = dbiguint(str2);

//----------------------START OF PROBLEM 2------------------------------------------

cout<<"--------Start of Problem 2--------"<<endl;

cout <<"Comparing " << small << " to " << big <<endl;

cout<<endl;

int resultOfCompare = small.compare(big);

cout << "Should be -1. " << resultOfCompare<<endl;

bool test1 = (small < big); //Should be true

cout<< "Should be true " << test1 << endl;

bool test3 = (small <= big); //Should be true

cout<< "Should be true " << test3 << endl;

bool test4 = (small != big); //Should be true

cout<< "Should be true " << test4 << endl;

bool test5 = (small == big); //Should be false

cout<< "Should be false " << test5 << endl;

bool test6 = (small > big);//Should be false

cout<< "Should be false " << test6 << endl;

bool test7 = (small >= big); //Should be false

cout<< "Should be false " << test7 << endl;

//Testing the overloaded operator (+)

dbiguint added = small+big;

cout<< added << endl;

//Testing copy constructor

dbiguint copyOfBig = dbiguint(big);

cout<<"This should be a copy of Big("<<big<<")"<< copyOfBig<<endl;

//Testing the overloaded = operator

dbiguint copyOfSmall = small;

cout<<"This should be a copy of Small("<<small<<")"<<copyOfSmall<<endl;

/\*

----------------------END OF PROBLEM 2------------------------------------------

\*/

/\*

----------------------START OF PROBLEM 3------------------------------------------

\*/

cout<<"--------Start of Problem 3--------"<<endl;

//Testing modified reserve

string reserveTest = "178";

string reserve2Test = "9999999";

dbiguint small22 = dbiguint(reserveTest);

dbiguint big22 = dbiguint(reserve2Test);

small22.reserve(big22.size());

for(size\_t i = 0; i < small22.size(); i++){

cout<<small22[i];

}

cout<<""<<endl;

small22.reserve(1);

for(size\_t i = 0; i < small22.size(); i++){

cout<<small22[i];

}

cout<<""<<endl;

//Testing -= operator

string top = "150";

string top1 = "250";

string bottom = "50";

dbiguint topBig = dbiguint(top);

dbiguint bottomBig = dbiguint(bottom);

topBig-=bottomBig;

cout<< "(" << top << "- "<<bottomBig << ") = " << topBig<< endl;

//Testing - operator

dbiguint topBig2 = dbiguint(top1);

dbiguint bottomBig2 = dbiguint(bottom);

dbiguint subtracted = topBig2-bottomBig2;

cout<< "(" << topBig2 << "- " <<bottomBig2 << ") = " << subtracted << endl;

/\*

----------------------END OF PROBLEM 3------------------------------------------

\*/

/\*

----------------------START OF PROBLEM 4------------------------------------------

\*/

cout<<"--------Start of Problem 4--------"<<endl;

dbiguint mult1 = dbiguint(top);

dbiguint mult2 = dbiguint(bottom);

mult1\*=mult2;

cout<< mult1<<endl;

//Testing just the \* operator

dbiguint multTest = mult1\*mult2;

cout<< mult1 << " times " << mult2 << " equals " << multTest<<endl;

return 10000;

}

/\*

----------------------END OF PROBLEM 4------------------------------------------

\*/

**Entire Homework 4 Output:**

FHosts-MacBook-Pro:Lab 4 fhost$ g++ \*.cpp -o main

FHosts-MacBook-Pro:Lab 4 fhost$ ./main

--------Start of Problem 2--------

Comparing 1 0 to 1 0 0

Should be -1. -1

Should be true 1

Should be true 1

Should be true 1

Should be false 0

Should be false 0

Should be false 0

1 1 0

This should be a copy of Big(1 0 0 )1 0 0

This should be a copy of Small(1 0 )1 0

--------Start of Problem 3--------

8710000

871

(150- 5 0 ) = 1 0 0

(2 5 0 - 5 0 ) = 2 0 0

--------Start of Problem 4--------

7 5 0 0

7 5 0 0 times 5 0 equals 3 7 5 0 0 0

FHosts-MacBook-Pro:Lab 4 fhost$