**Homework 4 – CS60 Linnell**

**Arjun Kohli**

**W1579330**

**Problem 1:**

#include <iostream>

using namespace std;

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

int p = \*a;

\*a = \*b;

\*b = p;

}

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;

}

**Problems 2-4:**

**Header File (dbiguint.h):**

#ifndef DBIGUINT\_H

#define DBIGUINT\_H

#include <cstdlib>

#include <string>

#include <iostream>

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);

// pre: none //NEW

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

// a copy of given dynamic bigint

dbiguint(const dbiguint &);

// 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

void operator =(const dbiguint &);

// 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 &) 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 &);

void operator -=(const dbiguint &);

void operator \*=(const dbiguint &);

// 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 &, const dbiguint &);

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

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

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

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

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

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

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

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

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

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

#endif // DBIGUINT\_H

**Class File (dbiguint.cpp):**

#include "dbiguint.h"

#include <math.h>

//1

dbiguint::dbiguint() {

data\_ = new unsigned short [1];

data\_[0] = 0;

capacity\_ = 1;

}

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

capacity\_ = s.size();

data\_ = new unsigned short [capacity\_];

int j = capacity\_-1;

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

data\_[i] = s[j] - '0';

j--;

}

}

//2

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

return capacity\_;

}

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

unsigned short i = 0;

if (pos >= 0 && pos < capacity\_) {

i = data\_[pos];

}

return i;

}

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

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

out << b[i-1];

}

return out;

}

//3

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

if (newcapacity\_ > capacity\_) {

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

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

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

}

for (std::size\_t i = capacity\_; i < newcapacity\_; ++i) {

temp\_[i] = 0;

}

delete [] data\_;

data\_ = temp\_;

temp\_ = nullptr;

capacity\_ = newcapacity\_;

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

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

for (std::size\_t i = 0; i < newcapacity\_; ++i) {

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

}

// for (std::size\_t i = capacity\_; i < newcapacity\_; ++i) {

// temp\_[i] = 0;

// }

delete [] data\_;

data\_ = temp\_;

temp\_ = nullptr;

capacity\_ = newcapacity\_;

}

}

//4

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

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

reserve(b.size());

}

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

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

if (i == capacity\_-1) {

reserve(capacity\_+1);

}

data\_[i] -= 10;

data\_[i+1]++;

} else {

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

}

}

}

//5

dbiguint::~dbiguint() {

delete [] data\_;

data\_ = nullptr;

capacity\_ = 0;

}

//HOMEWORK 4

//Problem 2

dbiguint::dbiguint(const dbiguint & b) {

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

reserve(b.size());

} else {

capacity\_ = b.size();

}

data\_ = new unsigned short [capacity\_];

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

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

}

}

//6

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

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

return 1;

} else if (capacity\_ < b.size()) {

return -1;

} else if (capacity\_ == b.size()) {

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

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

return 1;

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

return -1;

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

}

}

}

return 0;

}

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

dbiguint bsum;

bsum += b1;

bsum += b2;

return bsum;

}

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

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;

}

//7

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

if (this == &b) {

return;

} else {

capacity\_ = b.size();

reserve(capacity\_);

delete [] data\_;

data\_ = new unsigned short [capacity\_];

for (unsigned short i = 0; i < capacity\_; ++i) {

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

}

}

}

// //10

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

std::cout << "Enter number: " << std::endl;

std::string input;

in >> input;

dbiguint b1(input);

b = b1;

return in;

}

// //Problem 3

// //8

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

int temp = 0;

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

temp = data\_[i] - b.data\_[i];

if(temp < 0) {

if(i == (capacity\_ - 1)) {

reserve((capacity\_ -1));

int j = i+1;

while(data\_[j] == 0) {

if (data\_[j] == 0) {

reserve((capacity\_ -1));

}

--j;

}

}

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

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

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

} else {

data\_[i]=temp;

}

if ((i == capacity\_-1) && (data\_[i] == 0)) {

reserve(capacity\_-1);

}

}

int j = capacity\_;

int count = 0;

while (data\_[j] == 0) {

count++;

j--;

}

reserve(capacity\_-count);

}

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

dbiguint bsub;

bsub = b1;

bsub -= b2;

return bsub;

}

//9

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

int mult = 0;

int multb = 0;

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

mult += data\_[i] \* pow(10, i);

}

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

multb += b.data\_[i] \* pow(10, i);

}

int result = mult \* multb;

std::string s = "";

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

char c = result % 10 + '0';

s += c;

}

dbiguint ans(s);

int j = ans.size();

int count = 0;

while (ans[j] == 0) {

count++;

j--;

}

reserve(ans.size()-count);

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

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

}

}

**Main File (main2.cpp):**

#include "dbiguint.h"

#include <iostream>

using namespace std;

int main() {

dbiguint b1("5");

dbiguint b2("100");

cout << (b1.compare(b2)) << endl;

dbiguint b3 = (b1+b2);

cout << b3 << endl;

cout << (b1<b2) << endl; //1

cout << (b1<=b2) << endl; //1

cout << (b1!=b2) << endl; //1

cout << (b1==b2) << endl; //0

cout << (b1>=b2) << endl;//0

cout << (b1>b2) << endl; //0

(b1 = b2);

cout << b1 << endl;

cin >> b1;

cout << b1 << endl;

b2-=b1;

cout << b2 << endl;

dbiguint b4 = (b2-b1);

cout << b4 << endl;

b1\*=b2;

cout << b1 << endl;

return 0;

}

**Output:**

-1

105

1

1

1

0

0

0

100

Enter a number:

40 🡨 This is the input

40

60

20

500