**Lab 05**

**Object Oriented Programming Lab**

**Common Solution**

**16 Marks**

**Challenge-1:** *Big Number*

**BigNumber.h**

#ifndef BIG\_NUMBER\_H

#define BIG\_NUMBER\_H

enum Comparsion { EQUAL, SMALL,LARGE };

class BigNumber

{

char\* number;

int numberLength;

int getLength(const char \* num);

void copyCharArray(const char\* source);

bool isSame(BigNumber str) const ;

bool isEmpty() const ;

bool isCallingGreater(BigNumber s2) const ;

public:

BigNumber(const char\*);

BigNumber(const BigNumber&);

~BigNumber();

void setNumber(const char\* num);

void print() const ;

BigNumber subtract(const BigNumber& other) const;

BigNumber multiply(const BigNumber& other) const;

BigNumber add(const BigNumber & ) const ;

Comparsion compare(const BigNumber &) const ;

};

#endif // ! BIG\_NUMBER\_H

**BigNumber.cpp**

#include <iostream>

#include "BigNumber.h"

using namespace std;

//Private Functions:

int BigNumber::getLength(const char \* num)

{

int i = 0;

while (num[i] != '\0')

{

i = i + 1;

}

return i;

}

bool BigNumber::isEmpty() const

{

if (numberLength == 0)

return true;

return false;

}

bool BigNumber::isSame( BigNumber str) const

{

if (isEmpty() && str.isEmpty())

return 1;

if (numberLength != str.numberLength)

{

return false;

}

else

{

int i = 0;

while (i < numberLength)

{

if (number[i] != str.number[i])

return false;

i++;

}

return true;

}

}

void BigNumber::copyCharArray(const char\* source)

{

int i = 0;

while (source[i] != '\0')

{

number[i] = source[i];

i++;

}

number[i] = '\0';

}

bool BigNumber::isCallingGreater ( BigNumber s2) const

{

if (numberLength > s2.numberLength)

return true;

else if (numberLength < s2.numberLength)

return false;

int i = 0;

while (i < numberLength)

{

if (number[i] > s2.number[i])

return true;

i++;

}

return false;

}

//Public Functions:

BigNumber::BigNumber(const char\* num) ---- (-0.5) if not created.

{

if (num==nullptr||num=="")

{

number = new char;

\*number = '\0';

numberLength = 0;

}

else

{

int lengthOfNumber = getLength(num);

number = new char[lengthOfNumber + 1];

numberLength = lengthOfNumber;

copyCharArray(num);

}

}

BigNumber::BigNumber(const BigNumber& rfs) ---- (-0.5) if not created.

{

if (rfs.number == nullptr || rfs.number=="")

{

number = new char;

\*number = '\0';

numberLength = 0;

return;

}

number = new char[rfs.numberLength + 1];

numberLength = rfs.numberLength;

copyCharArray(rfs.number);

}

BigNumber::~BigNumber() ---- (-0.5) if not created.

{

if (number != nullptr)

{

delete[] number;

number = nullptr;

numberLength = 0;

}

}

void BigNumber::print() const ---- (-0.5) if not created.

{

if (number != nullptr )

{

if (numberLength != 0)

{

cout << number;

}

else

{

cout << "0";

}

}

}

Comparsion BigNumber::compare(const BigNumber& rfs) const ---- (-0.5) if not created.

{

if (isSame(rfs))

return EQUAL;

else if (isCallingGreater(rfs))

return LARGE;

else

return SMALL;

}

BigNumber BigNumber::add(const BigNumber& rfs) const ---- (-0.5) if not created.

{

int maxLength = (numberLength > rfs.numberLength) ? numberLength : rfs.numberLength;

char\* result = new char[maxLength + 1];

result[maxLength] = '\0';

int carry = 0;

int callingObjectIndex = numberLength - 1;

int parameterIndex = rfs.numberLength - 1;

int resultIndex = maxLength - 1;

while (callingObjectIndex >= 0 || parameterIndex >= 0)

{

int digit1 = (callingObjectIndex >= 0) ? (number[callingObjectIndex] - '0') : 0;

int digit2 = (parameterIndex >= 0) ? (rfs.number[parameterIndex] - '0') : 0;

int sum = digit1 + digit2 + carry;

carry = sum / 10;

result[resultIndex] = (sum % 10) + '0';

resultIndex--;

callingObjectIndex--;

parameterIndex--;

}

if (carry > 0)

result[resultIndex] = carry + '0';

BigNumber sumNumber(result);

delete[] result;

return sumNumber;

}

BigNumber BigNumber::multiply(const BigNumber& other) const ---- (8)

{

int resultLength = numberLength + other.numberLength;

char\* result = new char[resultLength + 1];

result[resultLength] = '\0';

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

{

result[i] = '0';

}

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

{

int carry = 0;

for (int j = other.numberLength - 1; j >= 0; j--)

{

int digit1 = number[i] - '0';

int digit2 = other.number[j] - '0';

int product = digit1 \* digit2 + carry + (result[i + j + 1] - '0');

result[i + j + 1] = (product % 10) + '0';

carry = product / 10;

}

result[i] += carry;

}

int startPos = 0;

while (result[startPos] == '0' && startPos < resultLength - 1)

{

startPos++;

}

BigNumber resultBigNum(result + startPos);

delete[] result;

return resultBigNum;

}

Sample Runs;

1. 123 \* 45 = 5535 ---- (2)
2. 987654 \* 123456 = 121,931,812,224 ---- (2)
3. 999 \* 999 = 998001---- (1)
4. 12345 \* 0 = 0 ---- (1)
5. 1 \* 9999 = 9999 ---- (2)

Automicity ---- (-0.5)

BigNumber BigNumber::subtract(const BigNumber& other) const ---- (6)

{

if (compare(other) == SMALL)

{

return BigNumber("0");

}

int len1 = numberLength;

int len2 = other.numberLength;

int carry = 0;

char\* result = new char[len1 + 1];

for (int i = len1 - 1, j = len2 - 1, k = len1 - 1; i >= 0; --i, --j, --k)

{

int num1 = number[i] - '0';

int num2 = j >= 0 ? other.number[j] - '0' : 0;

if (num1 < num2 + carry)

{

num1 += 10;

result[k] = (num1 - num2 - carry) + '0';

carry = 1;

}

else

{

result[k] = (num1 - num2 - carry) + '0';

carry = 0;

}

}

result[len1] = '\0';

int start = 0;

while (result[start] == '0')

{

++start;

}

BigNumber res(result + start);

delete[] result;

return res;

}

Sample Runs;

1. 100 - 99 = 1 ---- (2)
2. 12345 - 6789 = 5556 ---- (2)
3. 500 - 500 = 0 ---- (1)
4. 1234567890 - 987654321 = 246913569 ---- (1)

Automicity ---- (-0.5)

void BigNumber::setNumber(const char\* num) ---- (2)

{

this->~BigNumber(); ---- (0.5)

if (num == nullptr || num == "") ---- (0.5)

{

number = new char;

\*number = '\0';

numberLength = 0;

}

else

{

int lengthOfNumber = getLength(num);

number = new char[lengthOfNumber + 1];

numberLength = lengthOfNumber;

copyCharArray(num);

}

}

nullptr and empty check ---- (0.5)

Destructor or delete current num as it is on heap ---- (0.5)

Rest of the logic (Create array and deep copy) ---- (1)

Automicity ---- (-0.5)

**Quick Revision:**

void setNumber(const char\* num); ---- (2)

nullptr and empty check ---- (0.5)

Destructor or delete current num as it is on heap ---- (0.5)

Rest of the logic (Create array and deep copy) ---- (1)

Automicity ---- (-0.5)

BigNumber subtract(BigNumber&)const; ---- (6)

Sample Runs;

1. 100 - 99 = 1 ---- (2)
2. 12345 - 6789 = 5556 ---- (2)
3. 500 - 500 = 0 ---- (1)
4. 1234567890 - 987654321 = 246913569 ---- (1)

Automicity ---- (-0.5)

BigNumber multiply(BigNumber&)const; ---- (8)

Sample Runs;

1. 123 \* 45 = 5535 ---- (2)
2. 987654 \* 123456 = 121,931,812,224 ---- (2)
3. 999 \* 999 = 998001---- (1)
4. 12345 \* 0 = 0 ---- (1)
5. 1 \* 9999 = 9999 ---- (2)

Automicity ---- (-0.5)

Penalties: If functions of previous lab are not made then

---- (-0.5) for each function.

**Penalty Matrix:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Penalty List | Labs | | | | | | | | | | | | | | | |
| 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Indentation putting { Infront of loop header, in do while, putting while with closing } | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Meaningful Variable Names |  | -2 | -2 | -2 | -2 |  |  |  |  |  |  |  |  |  |  |  |
| Camel Case Notation | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Atomicity |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Syntax error | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Linker error | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Wrong function prototypes | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Class interface or additional members |  |  | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Use of library function/class without permission | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Continue statement | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| cin/cout where it isn’t needed | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Multi-filing |  |  | 0 | 0 | 0 |  |  |  |  |  |  |  |  |  |  |  |
| Wrong #ifndef or name of header file |  |  | -2 | -2 | -2 |  |  |  |  |  |  |  |  |  |  |  |
| Global functions |  |  | -3 | -3 | -3 |  |  |  |  |  |  |  |  |  |  |  |
| Multiple classes in one header file |  |  | -3 | -3 | -3 |  |  |  |  |  |  |  |  |  |  |  |
| Function of preceding lab |  |  |  |  | -0.5 |  |  |  |  |  |  |  |  |  |  |  |