**Lab 04**

**Object Oriented Programming Lab**

**Individual Solution**

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

**BigNumber.h**

#ifndef BIG\_NUMBER\_H

#define BIG\_NUMBER\_H

#include<iostream>

using namespace std;

enum Comparison

{

EQUAL,SMALL,LARGE

};

class BigNumber

{

char\* num;

int numberLength;

int getStrLength(const char\*);

void copyStr(const char\* , char\* );

public:

BigNumber(const char \*);

BigNumber(const BigNumber &);

~BigNumber();

BigNumber add(BigNumber);

void print();

Comparison compare(BigNumber);

};

#endif // !BIG\_NUMBER\_H

**BigNumber.cpp**

#include "BigNumber.h"

//Private Functions:

int BigNumber::getStrLength(const char\* str)

{

int length = 0;

while (str[length]!='\0')

{

length++;

}

return length;

}

void BigNumber::copyStr(const char\* source, char\* destination)

{

if (source == nullptr)

{

return;

}

int i = 0;

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

{

destination[i] = source[i];

i++;

}

destination[i] = '\0';

}

//Public Functions:

BigNumber::BigNumber(const char\* input)

{

numberLength = getStrLength(input);

num = new char[numberLength+1];

copyStr(input,num);

}

BigNumber::BigNumber(const BigNumber& ref)

{

numberLength = ref.numberLength;

num = new char[numberLength + 1];

copyStr(ref.num, num);

}

BigNumber::~BigNumber()

{

delete[]num;

num = nullptr;

numberLength = 0;

}

void BigNumber::print()

{

cout << num;

}

BigNumber BigNumber::add(BigNumber other)

{

// Find the maximum length among the two numbers

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

// Allocate memory for the result, add 1 for potential carry

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

result[maxLength] = '\0'; // Null-terminate the result

// Initialize carry

int carry = 0;

// Traverse both numbers from right to left and add digits

int i = numberLength - 1;

int j = other.numberLength - 1;

int k = maxLength - 1;

while (i >= 0 || j >= 0)

{

// Get digits from both numbers or use 0 if index is out of range

int digit1 = (i >= 0) ? (num[i] - '0') : 0;

int digit2 = (j >= 0) ? (other.num[j] - '0') : 0;

// Add digits and carry

int sum = digit1 + digit2 + carry;

// Update carry for the next iteration

carry = sum / 10;

// Store the result digit

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

// Move to the next digit in both numbers

i--;

j--;

}

// If there is a remaining carry, prepend it to the result

if (carry > 0)

result[k] = carry + '0';

// Create a new BigNumber object from the result

BigNumber sumNumber(result);

// Delete the dynamically allocated memory

delete[] result;

return sumNumber;

}

Comparison BigNumber::compare(BigNumber ref)

{

if (numberLength < ref.numberLength)

return SMALL;

else if (numberLength > ref.numberLength)

return LARGE;

else

{

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

{

if (num[i] < ref.num[i])

return SMALL;

else if (num[i] > ref.num[i])

return LARGE;

}

return EQUAL;

}

}