Luke Doukakis

CS256

Professor Nima Davarpanah

Final Part 1- Big Numbers Library

https://github.com/lukedoukakis/cs256-Final-Part-1

```
using namespace System;
using namespace std;
#pragma once
#include <vector>
#include <math.h>

namespace LibraryBigNumbers {

	class BigNumbers {

	public:

		vector<int> digits;

		////--------------CONSTRUCTORS-----------------------

		BigNumbers(vector<int> _digits) {
			for (int i = 0; i < _digits.size(); i++) {
				digits.at(i) = _digits.at(i);
			}
		}

		BigNumbers() {}

		//--------------FUNCTION ADD-----------------------

		static BigNumbers add(BigNumbers a, BigNumbers b) {

			BigNumbers output;

			int carry;

			//if BigNumbers a is larger
			if (a.digits.size() > b.digits.size()) {
				for (int i = a.digits.size() - 1; i >= b.digits.size(); i--) {
					output.digits.insert(output.digits.begin(),
a.digits.at(i));
				}

				for (int i = b.digits.size() - 1; i >= 0; i--) {

					int tempSum = a.digits.at(i) + b.digits.at(i) + carry;
					output.digits.insert(output.digits.begin(), tempSum % 10);
					if (tempSum > 9) {
```

```
                                        carry = 1;
                                }
                                else { carry = 0; }
                        }
                }

                //if BigNumbers b is larger
                if (b.digits.size() > a.digits.size()) {
                        for (int i = b.digits.size() - 1; i >= a.digits.size(); i--) {
                                output.digits.insert(output.digits.begin(),
b.digits.at(i));
                        }

                        for (int i = a.digits.size() - 1; i >= 0; i--) {

                                int tempSum = b.digits.at(i) + a.digits.at(i) + carry;
                                output.digits.insert(output.digits.begin(), tempSum % 10);
                                if (tempSum > 9) {
                                        carry = 1;
                                }
                                else { carry = 0; }
                        }
                }

                return output;
        }


        //--------------FUNCTION SUBTRACT-----------------------

        static BigNumbers subtract(BigNumbers a, BigNumbers b) {

                BigNumbers output;

                //IF BigNumbers a IS LARGER
                if (a.digits.size() >= b.digits.size()) {
                        for (int i = a.digits.size() - 1; i >= b.digits.size(); i--) {
                                output.digits.insert(output.digits.begin(), a.digits.at(i
- 1));
                        }

                        for (int i = b.digits.size(); i > 0; i--) {
                                if (a.digits.at(i - 1) - b.digits.at(i - 1) < 0) {
                                        a.digits.at(i - 1) += 10;
                                        a.digits.at(i - 2)--;
                                }
                                int diff = a.digits.at(i - 1) - b.digits.at(i - 1);
                                output.digits.insert(output.digits.begin(), diff);
                        }
                }

                //IF BigNumbers b IS LARGER
                if (b.digits.size() >= a.digits.size()) {
                        for (int i = b.digits.size() - 1; i > a.digits.size(); i--) {
                                output.digits.insert(output.digits.begin(), b.digits.at(i
- 1));
                        }

                        for (int i = a.digits.size(); i > 0; i--) {
                                if (b.digits.at(i - 1) - a.digits.at(i - 1) < 0) {
                                        b.digits.at(i - 1) += 10;
```

```
                                b.digits.at(i - 2)--;
                        }
                        int diff = b.digits.at(i - 1) - a.digits.at(i - 1);
                        output.digits.insert(output.digits.begin(), diff);
                        output.digits.at(0) *= -1;
                    }
                }

                return output;
        }


        //--------------FUNCTION MULTIPLY-----------------------

        static BigNumbers multiply(BigNumbers a, BigNumbers b) {

                BigNumbers output;

                vector<BigNumbers> subProds;
                vector<int> intVec;

                int carry = 0;

                for (int i = b.digits.size() - 1; i >= 0; i--) {
                        for (int j = a.digits.size() - 1; j >= 0; j--) {

                                int temp = b.digits.at(i) * a.digits.at(j) + carry;
                                intVec.insert(intVec.begin(), temp % 10);
                                carry = temp / 10;
                        }

                        subProds.insert(subProds.begin(), *new BigNumbers);

                        for (int i = 0; i < intVec.size() - 1; i++) {

subProds.at(0).digits.insert(subProds.at(0).digits.begin(), intVec.at(i));
                        }
                        intVec.clear();

                }
                for (int i = 0; i < subProds.size(); i++) {
                        output = BigNumbers::add(output, subProds.at(i));
                }

                return output;
        }


        //--------------FUNCTION DIVIDE-----------------------

        static BigNumbers divide(BigNumbers a, BigNumbers b) {

                BigNumbers output;

                //if b is larger than a, return a BigNumbers of 0
                if (b.digits.size() > a.digits.size()) {
                        output.digits.insert(output.digits.begin(), 0);
                }

                else {
```

```
                        int dividend;
                        for (int i = 0; i < a.digits.size(); i++) {
                                dividend += a.digits.at(i) * pow(10, a.digits.size() - i);
                        }

                        int divisor;
                        for (int i = 0; i < a.digits.size(); i++) {
                                divisor += b.digits.at(i) * pow(10, b.digits.size() - i);
                        }

                        int quotient = dividend / divisor;

                        for (int i = 1; i <= quotient; i = i * 10) {
                                output.digits.insert(output.digits.begin(), (quotient / i)
% 10);
                        }
                }

                return output;
        }


        //--------------FUNCTION MODULO-----------------------

        static BigNumbers mod(BigNumbers a, BigNumbers b) {

                BigNumbers output;

                int num1;
                for (int i = 0; i < a.digits.size(); i++) {
                        num1 += a.digits.at(i) * pow(10, a.digits.size() - i);
                }

                int num2;
                for (int i = 0; i < b.digits.size(); i++) {
                        num2 += b.digits.at(i) * pow(10, b.digits.size() - i);
                }

                int mod = num1 % num2;

                for (int i = 1; i <= mod; i = i * 10) {
                        output.digits.insert(output.digits.begin(), (mod / i) % 10);
                }

                return output;
        }


        //--------------OPERATOR OVERLOADING----------------------

        BigNumbers BigNumbers ::operator + (BigNumbers a) {
                BigNumbers tmp = BigNumbers::add(tmp, a);
                return (tmp);
        }

        BigNumbers BigNumbers ::operator - (BigNumbers a) {
                BigNumbers tmp = BigNumbers::subtract(tmp, a);
                return (tmp);
        }

        BigNumbers BigNumbers ::operator * (BigNumbers a) {
```

```
                BigNumbers tmp = BigNumbers::multiply(tmp, a);
                return (tmp);
        }

        BigNumbers BigNumbers ::operator / (BigNumbers a) {
                BigNumbers tmp = BigNumbers::divide(tmp, a);
                return (tmp);
        }

        BigNumbers BigNumbers ::operator % (BigNumbers a) {
                BigNumbers tmp = BigNumbers::mod(tmp, a);
                return (tmp);
        }

};
}
```