Правительство Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Кафедра «Компьютерной безопасности»

Отчет к лабораторной работе №3

по дисциплине

«Языки программирования»

Работу выполнил студент группы СКБ242 _____ П.В. Жучков

подпись, дата

Работу проверил _____ С.А. Булгаков

подпись, дата

Москва, 2024

Содержание

1. Постановка задачи

Разработать класс «целое произвольной длины» (во внешней форме представления в виде строки символов-цифр, базовый тип данных: std::string). Для класса разработать необходимые конструкторы, а также методы, обеспечивающие изменение отдельных составных частей объекта. Используя перегрузку операторов (operator) разработать арифметику объектов, включающую действия сложения и вычитания над объектами и стандартными целыми типами, присваивание, ввод и вывод в стандартные потоки (используя операторы «<<» и «>>»), приведение к/от базового типа данных. Организовать операции в виде конвейера значений, с результатом (новым объектом) и сохранением значений входных операндов.

2. Алгоритм решения

Для выполнения задачи нужно создать класс для работы с целыми числами произвольной длинны. Мы будем представлять это число в виде строки. Класс будет состоять из: абсолютного значения (std::string) и знака числа (bool), конструкторов для инициализации и преобразования. Также для реализации сложения и вычитания нужно перегрузить оператор плюс и оператор минус. Для сложения и вычитания нужно дописать дополнительные функции, которые выполняют сложение и вычитание числового значения строк. Еще нужно переписать остальные операторы для ввода/вывода, присваивания и приведения у другим типам данных.

3. Реализация задачи

Программа разбита на три основных файла: заголовочный файл class.h, файл с реализацией методами класса class.cpp и основной запускаемый файл main.cpp. Рассмотрим каждый из них подробнее

3.1 Заголовочный файл class.h

Заголовочный файл class.h содержит определение класса class, предназначенного для работы с целыми числами произвольной длины, представленных в виде строк. Также файл содержит определение нескольких дополнительных функций. В файле определены операторы, конструкторы и сама структура класса.

Определение класса:

```
class BigInt{
    private:
        std::string value;
        bool sign;

    public:
    //constructors
        BigInt();
        BigInt(bool sign, std::string str);
        BigInt(std::string& str);
        BigInt(const char* str);
        BigInt(int num);
        BigInt(short num);
        BigInt(long num);
        BigInt(char num);

    //operators
        friend BigInt operator+(const BigInt& left, const
BigInt& right);
        friend BigInt operator-(const BigInt& left, const
BigInt& right);
        BigInt& operator=(const BigInt& other);
        friend std::ostream& operator<<(std::ostream& out,
const BigInt& num);
        friend std::istream& operator>>(std::istream& in,
BigInt& num);

        operator std::string() const;
        operator int() const;
        operator short() const;
        operator long() const;
```

```
        operator char() const;
};
```

Определение дополнительной перегрузки операторов для различных типов данных:

```
//++++++++++++++++++++

BigInt operator+(int left, const BigInt& right);
BigInt operator+(std::string left, const BigInt& right);
BigInt operator+(short left, const BigInt& right);
BigInt operator+(long left, const BigInt& right);
BigInt operator+(char left, const BigInt& right);
BigInt operator+(const BigInt& left, int right);
BigInt operator+(const BigInt& left, std::string right);
BigInt operator+(const BigInt& left, long right);
BigInt operator+(const BigInt& left, short right);
BigInt operator+(const BigInt& left, char right);

//---------------------
BigInt operator-(int left, const BigInt& right);
BigInt operator-(std::string left, const BigInt& right);
BigInt operator-(short left, const BigInt& right);
BigInt operator-(long left, const BigInt& right);
BigInt operator-(char left, const BigInt& right);
BigInt operator-(const BigInt& left, int right);
BigInt operator-(const BigInt& left, std::string right);
BigInt operator-(const BigInt& left, long right);
BigInt operator-(const BigInt& left, short right);
BigInt operator-(const BigInt& left, char right);
```

Определение дополнительных функций вне класса для реализации сложения и вычитания:

```
std::string stringDif(const std::string& big, const
std::string& small);
std::string stringSum(const std::string& adin, const
std::string& dva);
bool firstBigger(const std::string& adin, const std::string&
dva);
```

3.2 Файл class.cpp

В файле class.cpp реализованы функции, определенные в заголовочном файле.

Конструкторы класса:

```
BigInt::BigInt() {value = "0"; sign =
true;}           //defolt constructor

BigInt::BigInt(bool insign, std::string str) {
    sign = insign;
    value = str;
    while (value.size() > 1 && value[0] == '0') {
            value.erase(0, 1);
    }
    if (value == "0") sign = true;
}

BigInt::BigInt(std::string& str) {
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);
}

BigInt::BigInt(const char* str)
{                   //cstring constructor
    sign = str[0] == '-' ? false : true;
    value = sign ? std::string(str) : std::string(str + 1);
}

BigInt::BigInt(int num){
    std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);

}

BigInt::BigInt(short num){
     std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);
}

BigInt::BigInt(long num){
     std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
```

```
    value = sign ? str : str.substr(1);
}
BigInt::BigInt(char num){
     std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);
}
```

Вспомогательные функции. Эти функции реализуют сложение и вычитание строк «столбиком». Также одна функция сравнивает числа записанные в строках, чтобы при вычитание всегда из большего вычиталось меньшее.

```
std::string stringDif(const std::string& big, const
std::string& small){     //substract abs values
    std::string result;
    int b = big.size() - 1; int s = small.size() - 1;
    int dop = 0;
    while (b >= 0 || s >= 0){
        int dif, right = (s >= 0 ? small[s--] - '0' : 0),
left = big[b--] - '0';
        dif = left - right - dop;
        if (dif < 0){
            dif += 10;
            dop = 1;
        } else {
            dop = 0;
        }
        result.push_back(dif + '0');
    }
    while(result.size() > 1 && result[result.size() - 1] ==
'0') {
            result.erase(result.size() - 1, 1);
    }

    std::reverse(result.begin(), result.end());
    return result.empty() ? "0" : result;
}


std::string stringSum(const std::string& adin, const
std::string& dva){     //sumarize abs values
    std::string result;
    int i = adin.size() - 1, j = dva.size() - 1;
    int dop = 0;
```

```
    while (i >= 0 || j >= 0 || dop == 1){
        int sum = 0;
        sum = (i >= 0 ? adin[i--] - '0' : 0) + (j >= 0 ?
dva[j--] - '0' : 0) + dop;
        dop = sum / 10;
        result.push_back((sum % 10) + '0');
    }
    std::reverse(result.begin(), result.end());
    return result;
}

bool firstBigger(const std::string& adin, const std::string&
dva){     //compare abs values
    if(adin.length() > dva.length()) return true;
    if(adin.length() < dva.length()) return false;
    for (int i = 0; i < int(adin.length()); ++i){
        if (adin[i] > dva[i]) return true;
        if (adin[i] < dva[i]) return false;
    }
    return true;
}
```

Перегрузка операторов. Перегружаются операторы ввода\вывода, сложения, вычитания и присваивания. Оператор вычитания реализуется через сложение числа с противоположным знаком.

```
BigInt& BigInt::operator=(const BigInt& other) {
    if (this != &other) {
        value = other.value;
        sign = other.sign;
    }
    return *this;
}

std::ostream& operator<<(std::ostream& out, const BigInt&
num) {
    if (num.sign == false) out << '-';
    out << num.value;
    return out;
}

std::istream& operator>>(std::istream& in, BigInt& num){
    std::string input;
    in >> input;
    if (input[0] == '-') {num.sign = false; num.value =
input.substr(1);}
```

```
    else {num.sign = true; num.value = input;}
    return in;
}

BigInt operator+(const BigInt& left, const BigInt& right) {
    if (left.sign == right.sign) {
        return BigInt(left.sign, stringSum(left.value,
right.value));
    }
    if (left.sign){
        if(firstBigger(left.value, right.value)){
            return BigInt(true, stringDif(left.value,
right.value));
        }
        return BigInt(false, stringDif(right.value,
left.value));
    } else {
        if(firstBigger(right.value, left.value)){
            return BigInt(true, stringDif(right.value,
left.value));
        }
        return BigInt(false, stringDif(left.value,
right.value));
    }
}

BigInt operator-(const BigInt& left, const BigInt& right){
    return left + BigInt(!right.sign, right.value);
}
```

Приведение класса к стандартным типам данных:

```
BigInt::operator std::string() const {              //ne
rabotaet epxlicit c takimi kluchami kompilyatsii
    return (sign ? "" : "-") + value;
}
BigInt::operator int() const {
    return (sign ? 1 : -1) * std::atoi(value.c_str());
}
BigInt::operator short() const {
    return (sign ? 1 : -1) * std::atoi(value.c_str());
}
BigInt::operator char() const {
    return (sign ? 1 : -1) * std::atoi(value.c_str());
}
BigInt::operator long() const {
```

```
    return (sign ? 1 : -1) * std::atol(value.c_str());
}
```

Дополнительная реализация арифметических операторов:

```
//++++++++++++++

BigInt operator+(int left, const BigInt& right)  {return
BigInt(left) + right;}
BigInt operator+(std::string left, const BigInt& right)
{return BigInt(left) + right;}
BigInt operator+(short left, const BigInt& right) {return
BigInt(left) + right;}
BigInt operator+(long left, const BigInt& right) {return
BigInt(left) + right;}
BigInt operator+(char left, const BigInt& right) {return
BigInt(left) + right;}
BigInt operator+(const BigInt& left, int right) {return left
+ BigInt(right);}
BigInt operator+(const BigInt& left, std::string
right){return left + BigInt(right);}
BigInt operator+(const BigInt& left, long right){return left
+ BigInt(right);}
BigInt operator+(const BigInt& left, short right){return
left + BigInt(right);}
BigInt operator+(const BigInt& left, char right){return left
+ BigInt(right);}
//-------------

BigInt operator-(int left, const BigInt& right)  {return
BigInt(left) - right;}
BigInt operator-(std::string left, const BigInt& right)
{return BigInt(left) - right;}
BigInt operator-(short left, const BigInt& right) {return
BigInt(left) - right;}
BigInt operator-(long left, const BigInt& right) {return
BigInt(left) - right;}
BigInt operator-(char left, const BigInt& right) {return
BigInt(left) - right;}
BigInt operator-(const BigInt& left, int right) {return left
- BigInt(right);}
BigInt operator-(const BigInt& left, std::string
right){return left - BigInt(right);}
BigInt operator-(const BigInt& left, long right){return left
- BigInt(right);}
```

```
BigInt operator-(const BigInt& left, short right){return
left - BigInt(right);}
BigInt operator-(const BigInt& left, char right){return left
- BigInt(right);}
```

3.3 Файл main.cpp

В этом файле реализуется базовое тестирование. Использование ввода\вывода, конструкторов, сложение и вычитание класса с различными типами данных слева и справа.

```
    std::string strp = "1000", strm = "-1000";
    int ip = 1000, im = -1000;
    BigInt a = 100, b = 1000;

    std::cout << a + strp << ' ' << strp + a << ' ' << a +
ip << ' ' << ip + a << ' ' << a + b << ' ' << b + a << "\n";
    std::cout << a + strm << ' ' << strm + a << ' ' << a +
im << ' ' << im + a << ' ' << a + b << ' ' << b + a << "\n";
    std::cout << a - strp << ' ' << strp - a << ' ' << a -
ip << ' ' << ip - a << ' ' << a - b << ' ' << b - a << "\n";
    std::cout << a - strm << ' ' << strm - a << ' ' << a -
im << ' ' << im - a << ' ' << a - b << ' ' << b - a << "\n";
```

Также проверяется перевод к другим типам класса:

```
std::cout << typeid(a).name() << " : " << a << '\n';
std::cout << typeid(int(a)).name() << " : " << int(a)  <<
'\n';
std::cout << typeid(short(a)).name() << " : " <<
short(a)  << '\n';
std::cout << typeid(long(a)).name() << " : " << long(a)  <<
'\n';
std::cout << typeid(std::string(a)).name() << " : " <<
std::string(a)  << '\n';
```

4. Тестирование

Используя ввод из файла main.cpp программа выводит получим:

```
1100 1100 1100 1100 1100 1100
-900 -900 -900 -900 1100 1100
-900 900 -900 900 -900 900
1100 -1100 1100 -1100 -900 900
--------------------------------
6BigInt : 100
i : 100
```

```
s : 100
l : 100
```

Заметим, что при изменении мест слагаемых сумма не меняется, а при изменении в вычитании меняется знак. При изменении типа данных, значение не меняется. Все верно.

Приложение A(class.h)

```
#ifndef CLASS_H
#define CLASS_H

#include <string>
#include <iostream>
#include <algorithm>

class BigInt{
    private:
        std::string value;
        bool sign;

    public:
    //constructors
        BigInt();
        BigInt(bool sign, std::string str);
        BigInt(std::string& str);
        BigInt(const char* str);
        BigInt(int num);
        BigInt(short num);
        BigInt(long num);
        BigInt(char num);

    //operators
        friend BigInt operator+(const BigInt& left, const
BigInt& right);
        friend BigInt operator-(const BigInt& left, const
BigInt& right);
        BigInt& operator=(const BigInt& other);
        friend std::ostream& operator<<(std::ostream& out,
const BigInt& num);
        friend std::istream& operator>>(std::istream& in,
BigInt& num);

        operator std::string() const;
        operator int() const;
        operator short() const;
```

```
        operator long() const;
        operator char() const;
};
//++++++++++++++++++++
BigInt operator+(int left, const BigInt& right);
BigInt operator+(std::string left, const BigInt& right);
BigInt operator+(short left, const BigInt& right);
BigInt operator+(long left, const BigInt& right);
BigInt operator+(char left, const BigInt& right);
BigInt operator+(const BigInt& left, int right);
BigInt operator+(const BigInt& left, std::string right);
BigInt operator+(const BigInt& left, long right);
BigInt operator+(const BigInt& left, short right);
BigInt operator+(const BigInt& left, char right);

//----------------------
BigInt operator-(int left, const BigInt& right);
BigInt operator-(std::string left, const BigInt& right);
BigInt operator-(short left, const BigInt& right);
BigInt operator-(long left, const BigInt& right);
BigInt operator-(char left, const BigInt& right);
BigInt operator-(const BigInt& left, int right);
BigInt operator-(const BigInt& left, std::string right);
BigInt operator-(const BigInt& left, long right);
BigInt operator-(const BigInt& left, short right);
BigInt operator-(const BigInt& left, char right);

std::string stringDif(const std::string& big, const
std::string& small);
std::string stringSum(const std::string& adin, const
std::string& dva);
bool firstBigger(const std::string& adin, const std::string&
dva);


#endif
```

Приложение Б(class.cpp)

```
#include "class.h"
#include <string>
#include <iostream>
#include <algorithm>
#include <stdexcept>
```

```
#include <cstdlib>
#include <sstream>

//constructors
BigInt::BigInt() {value = "0"; sign =
true;}           //defolt constructor

BigInt::BigInt(bool insign, std::string str) {     //bool +
string
    sign = insign;
    value = str;
    while (value.size() > 1 && value[0] == '0') {
            value.erase(0, 1);
    }
    if (value == "0") sign = true;
}

BigInt::BigInt(std::string& str) {                  //string
constructor
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);
}

BigInt::BigInt(const char* str)
{                   //cstring constructor
    sign = str[0] == '-' ? false : true;
    value = sign ? std::string(str) : std::string(str + 1);
}

BigInt::BigInt(int num){                            //int
constructor
    std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);

}

BigInt::BigInt(short num){                          //short
constructor
     std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
```

```
    value = sign ? str : str.substr(1);
}

BigInt::BigInt(long
num){                              //long  constructor
     std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);
}
BigInt::BigInt(char
num){                              //char  constructor
     std::stringstream ss;
    ss << num;
    std::string str = ss.str();
    sign = str[0] == '-' ? false : true;
    value = sign ? str : str.substr(1);
}

//dop functions
std::string stringDif(const std::string& big, const
std::string& small){     //substract abs values
    std::string result;
    int b = big.size() - 1; int s = small.size() - 1;
    int dop = 0;
    while (b >= 0 || s >= 0){
        int dif, right = (s >= 0 ? small[s--] - '0' : 0),
left = big[b--] - '0';
        dif = left - right - dop;
        if (dif < 0){
            dif += 10;
            dop = 1;
        } else {
            dop = 0;
        }
        result.push_back(dif + '0');
    }
    while(result.size() > 1 && result[result.size() - 1] ==
'0') {
            result.erase(result.size() - 1, 1);
    }

    std::reverse(result.begin(), result.end());
    return result.empty() ? "0" : result;
```

```
}

std::string stringSum(const std::string& adin, const
std::string& dva){     //sumarize abs values
    std::string result;
    int i = adin.size() - 1, j = dva.size() - 1;
    int dop = 0;
    while (i >= 0 || j >= 0 || dop == 1){
        int sum = 0;
        sum = (i >= 0 ? adin[i--] - '0' : 0) + (j >= 0 ?
dva[j--] - '0' : 0) + dop;
        dop = sum / 10;
        result.push_back((sum % 10) + '0');
    }
    std::reverse(result.begin(), result.end());
    return result;
}

bool firstBigger(const std::string& adin, const std::string&
dva){     //compare abs values
    if(adin.length() > dva.length()) return true;
    if(adin.length() < dva.length()) return false;
    for (int i = 0; i < int(adin.length()); ++i){
        if (adin[i] > dva[i]) return true;
        if (adin[i] < dva[i]) return false;
    }
    return true;
}


//operators
BigInt& BigInt::operator=(const BigInt& other) {
    if (this != &other) {
        value = other.value;
        sign = other.sign;
    }
    return *this;
}

std::ostream& operator<<(std::ostream& out, const BigInt&
num) {
    if (num.sign == false) out << '-';
    out << num.value;
    return out;
}
```

```
std::istream& operator>>(std::istream& in, BigInt& num){
    std::string input;
    in >> input;
    if (input[0] == '-') {num.sign = false; num.value =
input.substr(1);}
    else {num.sign = true; num.value = input;}
    return in;
}

BigInt operator+(const BigInt& left, const BigInt& right) {
    if (left.sign == right.sign) {
        return BigInt(left.sign, stringSum(left.value,
right.value));
    }
    if (left.sign){
        if(firstBigger(left.value, right.value)){
            return BigInt(true, stringDif(left.value,
right.value));
        }
        return BigInt(false, stringDif(right.value,
left.value));
    } else {
        if(firstBigger(right.value, left.value)){
            return BigInt(true, stringDif(right.value,
left.value));
        }
        return BigInt(false, stringDif(left.value,
right.value));
    }
}

BigInt operator-(const BigInt& left, const BigInt& right){
    return left + BigInt(!right.sign, right.value);
}

//+++++++++++++
BigInt operator+(int left, const BigInt& right)  {return
BigInt(left) + right;}
BigInt operator+(std::string left, const BigInt& right)
{return BigInt(left) + right;}
BigInt operator+(short left, const BigInt& right) {return
BigInt(left) + right;}
BigInt operator+(long left, const BigInt& right) {return
BigInt(left) + right;}
```

```
BigInt operator+(char left, const BigInt& right) {return
BigInt(left) + right;}
BigInt operator+(const BigInt& left, int right) {return left
+ BigInt(right);}
BigInt operator+(const BigInt& left, std::string
right){return left + BigInt(right);}
BigInt operator+(const BigInt& left, long right){return left
+ BigInt(right);}
BigInt operator+(const BigInt& left, short right){return
left + BigInt(right);}
BigInt operator+(const BigInt& left, char right){return left
+ BigInt(right);}
//-------------

BigInt operator-(int left, const BigInt& right)  {return
BigInt(left) - right;}
BigInt operator-(std::string left, const BigInt& right)
{return BigInt(left) - right;}
BigInt operator-(short left, const BigInt& right) {return
BigInt(left) - right;}
BigInt operator-(long left, const BigInt& right) {return
BigInt(left) - right;}
BigInt operator-(char left, const BigInt& right) {return
BigInt(left) - right;}
BigInt operator-(const BigInt& left, int right) {return left
- BigInt(right);}
BigInt operator-(const BigInt& left, std::string
right){return left - BigInt(right);}
BigInt operator-(const BigInt& left, long right){return left
- BigInt(right);}
BigInt operator-(const BigInt& left, short right){return
left - BigInt(right);}
BigInt operator-(const BigInt& left, char right){return left
- BigInt(right);}



BigInt::operator std::string() const {              //ne
rabotaet epxlicit c takimi kluchami kompilyatsii
    return (sign ? "" : "-") + value;
}
BigInt::operator int() const {
    return (sign ? 1 : -1) * std::atoi(value.c_str());
}
BigInt::operator short() const {
```

```
    return (sign ? 1 : -1) * std::atoi(value.c_str());
}
BigInt::operator char() const {
    return (sign ? 1 : -1) * std::atoi(value.c_str());
}
BigInt::operator long() const {
    return (sign ? 1 : -1) * std::atol(value.c_str());
}
```

Приложение В

```
#include <iostream>
#include <string>
#include <typeinfo>
#include "class.h"

int main(){
    std::string strp = "1000", strm = "-1000";
    int ip = 1000, im = -1000;
    BigInt a = 100, b = 1000;

    std::cout << a + strp << ' ' << strp + a << ' ' << a +
ip << ' ' << ip + a << ' ' << a + b << ' ' << b + a << "\n";
    std::cout << a + strm << ' ' << strm + a << ' ' << a +
im << ' ' << im + a << ' ' << a + b << ' ' << b + a << "\n";
    std::cout << a - strp << ' ' << strp - a << ' ' << a -
ip << ' ' << ip - a << ' ' << a - b << ' ' << b - a << "\n";
    std::cout << a - strm << ' ' << strm - a << ' ' << a -
im << ' ' << im - a << ' ' << a - b << ' ' << b - a << "\n";
    std::cout << "------------------------------\n";
    std::cout << typeid(a).name() << " : " << a << '\n';
    std::cout << typeid(int(a)).name() << " : " <<
int(a)  << '\n';
    std::cout << typeid(short(a)).name() << " : " <<
short(a)  << '\n';
    std::cout << typeid(long(a)).name() << " : " <<
long(a)  << '\n';
    std::cout << typeid(std::string(a)).name() << " : " <<
std::string(a)  << '\n';
}
```

Приложение Г

class
-value: std::string -sign: bool
+BigInt(); +BigInt(bool sign, std::string str) +BigInt(std::string& str) +BigInt(const char* str) +BigInt(int num) +BigInt(short num) +BigInt(long num) +BigInt(char num) +operator std::string() const +operator int() const +operator short() const +operator long() const +operator char() const +friend BigInt operator+(const BigInt& left, const BigInt& right) +friend BigInt operator-(const BigInt& left, const BigInt& right) +BigInt& operator=(const BigInt& other) +friend std::ostream& operator<<(std::ostream& out, const BigInt& +friend std::istream& operator>>(std::istream& in, BigInt& num)