Правительство Российской Федерации

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

Кафедра «Компьютерной безопасности»

Отчет к лабораторной работе №5

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

Доработать класс «целое произвольной длины» из Лабораторной работы №4. Для класса разработать деструктор. Используя перегрузку операторов (operator) разработать арифметику объектов, включающую действия деления с остатком / и взятия остатка от деления % для объектов класса и стандартных целочисленных типов. Организовать операции в виде конвейера значений, с результатом (новым объектом) и сохранением значений входных операндов

2. Алгоритм решения

Для реализации умножения созданного класса, нужно перегрузить операцию целочисленного деления и взятие деления от остатка. Для этого напишем вспомогательную функцию, делящую абсолютные значения объектов в виде строки. Будем пользоваться ранее написанным сложением, вычитанием и функцией, сравнивающей значения объектов. Алгоритм деления будет реализован аналогично делению столбиком. Для более удобного тестирования реализуем перегрузку оператора унарного минуса. Также для класса напишем деструктор.

3. Реализация задачи

Деление реализовано аналогично делению стандартных типов в языке C++. Частное положительно при одинаковых знаках делимого и делителя, остаток сохраняет знак делимого.

```
BigInt operator/(const BigInt& left, const BigInt& right) {
        if (left.sign == right.sign)
        return BigInt(true, stringDivision(left.value,
right.value).first);
    else
        return BigInt(false, stringDivision(left.value,
right.value).first);
}
BigInt operator%(const BigInt& left, const BigInt& right) {
        if (left.sign)
        return BigInt(true, stringDivision(left.value,
right.value).second);
    else
        return BigInt(false, stringDivision(left.value,
right.value).second);
}
```

При перегрузке операторов деления используется вспомогательная функция `string Division`, которая выполняет деление абсолютных значений.

```
std::pair<std::string, std::string> stringDivision(const
std::string& dividend, const std::string& divisor){

    std::pair<std::string, std::string> ans;
    const size_t divsize = divisor.length();
    std::string dim = dividend.substr(0, divsize);
    std::string quotient = "", rem;
    int p = divsize;
    std::string sub[11];
    sub[1] = divisor;

    if(divisor == "0"){
        std::cerr << "ERROR: Division by zero\n";
        exit(EXIT_FAILURE);
    }
    if (divisor == dividend) return std::make_pair("1",
"0");
    if (firstBigger(dividend, divisor) == false) return
std::make_pair("0", dividend);
```

```
    while(p != dividend.length() || divsize ==
dividend.length()){

        bool addZero = false;
        while(firstBigger(sub[1], dim) && sub[1] != dim){
            if (addZero) quotient += '0';
            if (p != dividend.length()){
                addZero = true;
                if(dim != "0") dim += dividend[p];
                else dim = (dividend[p] == '0' ? "" : dim =
dividend[p]);
                p++;
            } else {
                return std::make_pair(quotient, (dim == "" ?
"0" : dim));
            }
        }

        for (int i = 2; i <= 10; i++){
            if (sub[i] == "") sub[i] = stringSum(sub[i-1],
divisor);
            if (firstBigger(sub[i], dim)){
                rem = stringDif(dim, (sub[i] == dim ? sub[i]
: sub[--i]));
                rem = (rem == "0" ? "" : rem);
                quotient += (i + '0');
                break;
            }
        }
        dim = rem;
    }
    return std::make_pair(quotient, (rem == "" ? "0" :
rem));
}
```

Функция одновременно находит частное и остаток от деления. При делении на ноль выводится ошибка и выполнение программы прекращается. Общий цикл работает до тех пор, пока деление не «дойдет» до последней цифры делимого. Первый внутренний цикл `while` сносит цифру делимого для вычитания и нахождения цифры частного. При сноски нескольких цифр подряд к частному добавляется 0. Второй внутренний цикл `for` реализует это вычитание. Для этого есть массив, в котором содержатся возможные вычитаемые. Если нужного вычитаемого в этом массиву не находится, то к максимальному (последнему определенному) элементу массива добавляется делитель. Это значение записывается в следующую ячейку. При

получении вычитаемого большего, чем уменьшаемое, берется предыдущее вычитаемое. Индекс предыдущего вычитаемого добавляется к частному. Функция не рассчитана на числа с ведущими нулями, дробными числами, строки с иными символами кроме цифр. Деление определено для строк и всех стандартных целочисленных типах данных.

```
//////////////////
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);}
```

Перегрузка оператора унарного минуса

```
BigInt BigInt::operator-(){              //unary minus
    BigInt a;
    a.value = value;
    a.sign = !sign;
    return a;
}
```

Деструктор класса

```
BigInt::~BigInt(){}
```

4. Тестирование

Для тестирования будем сравнивать деление стандартных типов и реализованного деления. При одинаковом результате последние две колонки будут состоять только из 1. Будем вводить только положительные числа, проверка с другими знаками будет проводится автоматически. При вводе огромных чисел последние две колонки не будут состоять только из 1. Такие тесты будем проверять вручную.

Тест 1:

12094 135

---

89 79

-89 79

89 -79

-89 -79

---

89 79

-89 79

89 -79

-89 -79

---

1 1

1 1

1 1

1 1

Тест 2:

10011205 13

770092 9

-770092 9

770092 -9

-770092 -9

770092 9

-770092 9

770092 -9

-770092 -9

---

1 1

1 1

1 1

1 1

Тест 3:

10 0

ERROR: Division by zero

Тест 4:

10 100

---

0 10

0 10

0 -10

0 -10

---

0 10

0 10

0 -10

0 -10

---

1 1

1 1

1 1

1 1

Тест 5:

1500 1

1500 0

-1500 0

1500 0

-1500 0

1500 0

-1500 0

1500 0

-1500 0

---

1 1

1 1

1 1

1 1

Тест 6:

100000000000000000000 3

33333333333333333333 1

-33333333333333333333 1

33333333333333333333 -1

-33333333333333333333 -1

715827882 1

-715827882 1

715827882 -1

-715827882 -1

---

0 1

0 1

0 1

0 1

Тест 7:

0 12384

---

0 0

0 0

0 0

0 0

---

0 0

0 0

0 0

0 0

---

1 1

1 1

1 1

1 1

Приложение А (main.cpp)

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

int main(){

    BigInt a;
    BigInt b;

    std::cin >> a >> b;
    int ai = int(a), bi = int(b);

    std::cout << "------------------\n";
    std::cout << a / b << " " << a % b << "\n";
    std::cout << a / -b << " " << a % -b << "\n";
    std::cout << -a / -b << " " << -a % -b << "\n";
    std::cout << -a / b << " " << -a % b << "\n";
    std::cout << "------------------\n";
    std::cout << ai / bi << " " << ai % bi << "\n";
    std::cout << ai / -bi << " " << ai % -bi << "\n";
    std::cout << -ai / -bi << " " << -ai % -bi << "\n";
    std::cout << -ai / bi << " " << -ai % bi << "\n";
    std::cout << "------------------\n";
    std::cout << (ai / bi == int(a / b)) << " " << (ai % bi
== int(a % b)) << "\n";
    std::cout << (ai / -bi == int(a / -b)) << " " << (ai % -
bi == int(a % -b)) << "\n";
    std::cout << (-ai / -bi == int(-a / -b)) << " " << (-ai
% -bi == int(-a % -b)) << "\n";
    std::cout << (-ai / bi == int(-a / b)) << " " << (-ai %
bi == int(-a % b)) << "\n";
    return 0;
}
```

Приложение Б (class.cpp)

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

//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);
}
BigInt::BigInt(BigInt const &obj){                    //copy
constructor
    value = obj.value;
    sign = obj.sign;
    std::cout << "copy" << "\n";
}
BigInt::~BigInt(){}                                   //destru
ctour

//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;
}
std::string stringMultiply(const std::string& adin, const
std::string& dva){    //multiply
    int a, d = dva.size() - 1;
    int dop = 0;
    std::string result = "0", tmp;
    while (d >= 0){
        tmp = "";
        a = adin.size() - 1;
        for (int i = 0; i < (int)dva.size() - d - 1; ++i){
            tmp.push_back('0');
        }
        while(a >= 0 || dop){
            int prod;
            prod = (a >= 0 ? adin[a--] - '0' : 1) * (dva[d]
- '0') + dop;
            dop = prod / 10;
            tmp.push_back((prod % 10) + '0');
        }
```

```
        d--;
        std::reverse(tmp.begin(), tmp.end());
        result = stringSum(result, tmp);
    }
    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;
}
std::pair<std::string, std::string> stringDivision(const
std::string& dividend, const std::string& divisor){
    std::pair<std::string, std::string> ans;
    const size_t divsize = divisor.length();
    std::string dim = dividend.substr(0, divsize);
    std::string quotient = "", rem;
    size_t p = divsize;
    std::string sub[11];
    sub[1] = divisor;

    if(divisor == "0"){
        std::cerr << "ERROR: Division by zero\n";
        exit(EXIT_FAILURE);
    }
    if (divisor == dividend) return std::make_pair("1",
"0");
    if (firstBigger(dividend, divisor) == false) return
std::make_pair("0", dividend);

    while(p != dividend.length() || divsize ==
dividend.length()){

        bool addZero = false;
        while(firstBigger(sub[1], dim) && sub[1] != dim){
            if (addZero) quotient += '0';
            if (p != dividend.length()){
                addZero = true;
                if(dim != "0") dim += dividend[p];
```

```
                else dim = (dividend[p] == '0' ? "" : dim =
dividend[p]);
                p++;
            } else {
                return std::make_pair(quotient, (dim == "" ?
"0" : dim));
            }
        }

        for (int i = 2; i <= 10; i++){
            if (sub[i] == "") sub[i] = stringSum(sub[i-1],
divisor);
            if (firstBigger(sub[i], dim)){
                rem = stringDif(dim, (sub[i] == dim ? sub[i]
: sub[--i]));
                rem = (rem == "0" ? "" : rem);
                quotient += (i + '0');
                break;
            }
        }
        dim = rem;
    }
    return std::make_pair(quotient, (rem == "" ? "0" :
rem));
}

//get set function
const std::string& BigInt::get_value() const{
    return value;
}
void BigInt::set_value(const std::string& str) {
    value = str;
}
const bool BigInt::get_sign() const{
    return sign;
}
void BigInt::set_sign(const bool s) {
    sign = s;
}

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

```
    }
    return *this;
}
BigInt BigInt::operator=( short num){
    *this = BigInt(num);
    return *this;
}
BigInt BigInt::operator=( char num){
    *this = BigInt(num);
    return *this;
}
BigInt BigInt::operator=( int num){
    *this = BigInt(num);
    return *this;
}
BigInt BigInt::operator=( long num){
    *this = BigInt(num);
    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*(const BigInt& left, const BigInt& right) {
    if (left.sign == right.sign)
        return BigInt(true, stringMultiply(left.value,
right.value));
    else
        return BigInt(false, stringMultiply(left.value,
right.value));
}
BigInt operator/(const BigInt& left, const BigInt& right) {
        if (left.sign == right.sign)
        return BigInt(true, stringDivision(left.value,
right.value).first);
    else
        return BigInt(false, stringDivision(left.value,
right.value).first);
}
BigInt operator%(const BigInt& left, const BigInt& right) {
        if (left.sign)
        return BigInt(true, stringDivision(left.value,
right.value).second);
    else
        return BigInt(false, stringDivision(left.value,
right.value).second);
}
BigInt BigInt::operator-(){                              //unary
minus
    BigInt a;
    a.value = value;
    a.sign = !sign;
    return a;
}
```

```
//+++++++++++++
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*(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%(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);}

//to other types
BigInt::operator std::string() const {
    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 (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();
        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);
        BigInt(const BigInt& obj);

    //set and get methods

    const std::string& get_value() const;
    void set_value(const std::string& str);
    const bool get_sign() const;
    void set_sign(const bool s);
    //operators
        friend BigInt operator+(const BigInt& left, const
BigInt& right);
        friend BigInt operator-(const BigInt& left, const
BigInt& right);
        friend BigInt operator*(const BigInt& left, const
BigInt& right);
        friend BigInt operator/(const BigInt& left, const
BigInt& right);
        friend BigInt operator%(const BigInt& left, const
BigInt& right);
```

```
        friend std::ostream& operator<<(std::ostream& out,
const BigInt& num);
        friend std::istream& operator>>(std::istream& in,
BigInt& num);

        BigInt operator-(); //unary minus

        BigInt operator=(const BigInt& other);
        BigInt operator=( short num);
        BigInt operator=( char num);
        BigInt operator=( int num);
        BigInt operator=( long 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);

//************/
BigInt operator*(const BigInt& left, int right);
```

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

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

std::string stringDif(const std::string& big, const
std::string& small);
std::string stringSum(const std::string& adin, const
std::string& dva);
std::string stringMultiply(const std::string& adin, const
std::string& dva);
std::pair<std::string, std::string> stringDivision(const
std::string& dividend, const std::string& divisor);
bool firstBigger(const std::string& adin, const std::string&
dva);
#endif
```