

Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа
по курсу «ООП»**

**Тема:
Простые классы.**

Студент:	Козлов А.Д.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	7
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

bigString.hpp:

```
#ifndef BIGSTRING_HPP
#define BIGSTRING_HPP

struct BigString{
    BigString();
    BigString(unsigned long long n1, unsigned long long n2);
    ~BigString();

    void set_lString(unsigned long long n);
    void set_rString(unsigned long long n);
    unsigned long long get_lString() const;
    unsigned long long get_rString() const;

    void print_bits();

    void shiftLeft(int shift);
    void shiftRight(int shift);

    BigString AND(const BigString& s);
    BigString OR(const BigString& s);
    BigString XOR(const BigString& s);
    BigString NOT();

    int num_of_units() const;
    BigString* comparison(BigString& s);
    int is_include(const BigString& s);

private:
    unsigned long long lString;
    unsigned long long rString;
};

#endif
```

bigString.cpp:

```
#include <iostream>
#include "bigString.hpp"

BigString::BigString():lString(0), rString(0){}

BigString::BigString(unsigned long long n1, unsigned long long n2):lString(n1),
rString(n2){}

BigString::~~BigString(){}

```

```

void BigString::set_lString(unsigned long long n) {
    this->lString = n;
}

void BigString::set_rString(unsigned long long n) {
    this->rString = n;
}

unsigned long long BigString::get_lString() const {
    return this->lString;
}

unsigned long long BigString::get_rString() const {
    return this->rString;
}

void BigString::print_bits() {
    unsigned long long Mask_firstBit = 0x8000000000000000; // первый бит 1
    остальные 0
    unsigned long long num = this->lString;
    for(int i = 0; i < 64; ++i) {
        if (num&Mask_firstBit) {
            std::cout << "1";
        }
        else {
            std::cout << "0";
        }
        num = num << 1;
    }
    num = this->rString;
    for(int i = 0; i < 64; ++i) {
        if (num&Mask_firstBit) {
            std::cout << "1";
        }
        else {
            std::cout << "0";
        }
        num = num << 1;
    }
    std::cout << "\n";
}

void BigString::shiftLeft(int shift) {
    unsigned long long Mask_firstBit = 0x8000000000000000; // первый бит 1
    остальные 0

```

```

int bit;
for(int i = 0; i < shift; ++i) {
    if(this->rString & Mask_firstBit) {
        bit = 1;
    }
    else {
        bit = 0;
    }
    this->rString = this->rString << 1;
    this->lString = this->lString << 1;
    this->lString = this->lString | bit;
}
}

void BigString::shiftRight(int shift) {
    unsigned long long Mask_lastBit = 1; // последний бит 1 остальные 0
    int bit;
    for(int i = 0; i < shift; ++i) {
        if (this->lString & Mask_lastBit) {
            bit = 1;
        }
        else {
            bit = 0;
        }
        this->lString = this->lString >> 1;
        this->rString = this->rString >> 1;
        if (bit) {
            this->rString = this->rString | 0x8000000000000000;
        }
    }
}

BigString BigString::AND(const BigString& s) {
    BigString res;
    unsigned long long mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if ((this->lString & s.get_lString()) & mask) {
            res.lString = res.lString | mask;
        }
        mask = mask >> 1;
    }
    mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if ((this->rString & s.get_rString()) & mask) {
            res.rString = res.rString | mask;
        }
    }
}

```

```

    }
    mask = mask >> 1;
}
return res;
}

```

```

BigString BigString::OR(const BigString& s) {
    BigString res;
    unsigned long long mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if ((this->lString | s.get_lString()) & mask) {
            res.lString = res.lString | mask;
        }
        mask = mask >> 1;
    }
    mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if ((this->rString | s.get_rString()) & mask) {
            res.rString = res.rString | mask;
        }
        mask = mask >> 1;
    }
    return res;
}

```

```

BigString BigString::XOR(const BigString& s) {
    BigString res;
    unsigned long long mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if ((!(this->lString & mask) && (s.get_lString() & mask)) ||
            ((this->lString & mask) && !(s.get_lString() & mask))) {
            res.lString = res.lString | mask;
        }
        mask = mask >> 1;
    }
    mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if ((!(this->rString & mask) && (s.get_rString() & mask)) ||
            ((this->rString & mask) && !(s.get_rString() & mask))) {
            res.rString = res.rString | mask;
        }
        mask = mask >> 1;
    }
    return res;
}

```

```

BigString BigString::NOT() {
    BigString res;
    unsigned long long mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if (!(this->lString & mask)) {
            res.lString = res.lString | mask;
        }
        mask = mask >> 1;
    }
    mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if (!(this->rString & mask)) {
            res.rString = res.rString | mask;
        }
        mask = mask >> 1;
    }
    return res;
}

```

```

int BigString::num_of_units() const {
    unsigned long long mask = 0x8000000000000000;
    int counter = 0;
    for(int i = 0; i < 64; ++i) {
        if (this->lString & mask) {
            counter++;
        }
        mask = mask >> 1;
    }
    mask = 0x8000000000000000;
    for(int i = 0; i < 64; ++i) {
        if (this->rString & mask) {
            counter++;
        }
        mask = mask >> 1;
    }
    return counter;
}

```

```

BigString* BigString::comparison(BigString& s) {
    int value1 = this->num_of_units();
    int value2 = s.num_of_units();
    if (value1 >= value2) {
        return this;
    }
}

```

```
    else {  
        return &s;  
    }  
}
```

```
int BigString::is_include(const BigString& s) {  
    if(((this->lString & s.get_lString()) == this->get_lString()) || ((this->rString &  
s.get_rString()) == this->get_rString())) {  
        return 1;  
    }  
    else {  
        return 0;  
    }  
}
```

main.cpp:

```
#include <iostream>
#include <fstream>
#include "bigString.hpp"

void separator();
void meny();
unsigned long long get_num_in_file(std::ifstream& fin);

int main(int argc, char* argv[]) {
    int key;
    int shift;
    unsigned long long n1;
    unsigned long long n2;
    BigString str1;
    BigString str2;

    if (argc > 1) {
        std::ifstream fin(argv[1], std::ios_base::in);
        if (!fin.is_open()) {
            std::cout << "Нельзя отрыть файл!\n";
            return 2;
        }

        n1 = get_num_in_file(fin);
        n2 = get_num_in_file(fin);
        str1.set_lString(n1);
        str1.set_rString(n2);

        n1 = get_num_in_file(fin);
        n2 = get_num_in_file(fin);
        str2.set_lString(n1);
        str2.set_rString(n2);

        fin.close();
    }
    else {
        for(int i = 0; i < 2; ++i) {
            std::cin >> n1 >> n2;
            str1.set_lString(n1);
            str1.set_rString(n2);

            std::cin >> n1 >> n2;
            str2.set_lString(n1);
            str2.set_rString(n2);
        }
    }
}
```



```
}
```

```
meny();
```

```
std::cin >> key;
```

```
switch(key) {
```

```
    case 1:
```

```
        std::cin >> shift;
```

```
        str1.print_bits();
```

```
        str1.shiftLeft(shift);
```

```
        str1.print_bits();
```

```
        break;
```

```
    case 2:
```

```
        std::cin >> shift;
```

```
        str1.print_bits();
```

```
        str1.shiftRight(shift);
```

```
        str1.print_bits();
```

```
        break;
```

```
    case 3:
```

```
        str1.print_bits();
```

```
        str2.print_bits();
```

```
        str1.AND(str2).print_bits();
```

```
        break;
```

```
    case 4:
```

```
        str1.print_bits();
```

```
        str2.print_bits();
```

```
        str1.OR(str2).print_bits();
```

```
        break;
```

```
    case 5:
```

```
        str1.print_bits();
```

```
        str2.print_bits();
```

```
        str1.XOR(str2).print_bits();
```

```
        break;
```

```
    case 6:
```

```
        str1.print_bits();
```

```
        str2.print_bits();
```

```
        str1.NOT().print_bits();
```

```
        break;
```

```
    case 7:
```

```
        str1.print_bits();
```

```
        str2.print_bits();
```

```
        std::cout << "num_of_units :" << str1.num_of_units() <<std::endl;
```

```
        std::cout << "num_of_units :" << str1.num_of_units() <<std::endl;
```

```
        break;
```

```
    case 8:
```

```

        str1.print_bits();
        str2.print_bits();
        std::cout << "comparison \n";
        str1.comparison(str2)->print_bits();
    case 9:
        str1.print_bits();
        str2.print_bits();
        if (str1.is_include(str2)) {
            std::cout << "Yes\n";
        }
        else {
            std::cout << "No\n";
        }
    default:
        std::cout << "error" << std::endl;
        break;
    }
    return 0;
}

void separator() {
    std::cout << "===== ";
    for(int i = 0; i < 128; ++i) {
        std::cout << "=";
    }
    std::cout << std::endl;
}

void meny() {
    std::cout << "1)shiftLeft" << std::endl;
    std::cout << "2)shiftLeft" << std::endl;
    std::cout << "3)OR" << std::endl;
    std::cout << "4)AND" << std::endl;
    std::cout << "5)XOR" << std::endl;
    std::cout << "6)NOT" << std::endl;
    std::cout << "7)num_of_units" << std::endl;
    std::cout << "8)comparison" << std::endl;
    std::cout << "9)is_include" << std::endl;
}

unsigned long long get_num_in_file(std::ifstream& fin) {
    unsigned long long num = 0;
    char ch;
    while(fin.get(ch)) {

```

```
    if (ch != ' ' && ch != '\n' && ch != EOF) {  
        num = num * 10 + (ch - '0');  
    }  
    else {  
        break;  
    }  
}  
return num;  
}
```

CmakeLists.txt:

```
cmake_minimum_required(VERSION 3.1)
```

```
project(oop_exercise_01)
```

```
add_executable(oop_exercise_01 main.cpp bigString.cpp)
```

2. Ссылка на репозиторий на GitHub.

https://github.com/ArtemKD/oop_exercise_01

3. Набор testcases.

test_01.txt:

```
12868426913406380000 5193880475836733000  
12918000598641058000 11398999006967146000
```

test_02.txt:

```
11093947366930840000 12809315413885479000  
8591448501686567000 7091160512692316000
```

test_03.txt:

```
5647118480700488000 15870949720163300000  
13670168416534604000 8221050084544006000
```

test_04.txt:

```
11023188960974280000 10712243251891943000  
15260885590755232000 11658364395506348000
```

test_05.txt:

```
17490701296889190000 1907006900119359500  
696737765169393700 17953908319343747000
```

4. Результаты выполнения тестов.

test_01.txt:

=====

str1

1011001010010101110101100001100100001111101010011110011111100000010010
0000010100010111101100011001000000000101011101011001001000

str2

101100110100010111110101000110100001101001001000101000001101000010011
11000110001011000010011001111001101010011101010011000010000

str1 and str2

101100100000010111010100000110000000101000001000101000001100000000001
000000100000100000000000010010000000000010010000110000000000

=====

str1

1011001010010101110101100001100100001111101010011110011111100000010010
0000010100010111101100011001000000000101011101011001001000

str2

101100110100010111110101000110100001101001001000101000001101000010011
11000110001011000010011001111001101010011101010011000010000

str1 or str2

10110011110101011111011100011011000111111110100111100111111100001101111
000110101011111111111011111001101010111111111011001011000

=====

str1

1011001010010101110101100001100100001111101010011110011111100000010010
0000010100010111101100011001000000000101011101011001001000

str2

101100110100010111110101000110100001101001001000101000001101000010011
11000110001011000010011001111001101010011101010011000010000

str1 xor str2

000000011101000000100011000000110001010111100001010001110011000011010
11000100101001111111111010110001101010110110111000001011000

=====

str1

1011001010010101110101100001100100001111101010011110011111100000010010
0000010100010111101100011001000000000101011101011001001000

not str1

0100110101101010001010011110011011110000010101100001100000011111101101
1111101011101000010011100110111111111010100010100110110111

=====

str2

101100110100010111110101000110100001101001001000101000001101000010011
11000110001011000010011001111001101010011101010011000010000

not str2

0100110010111010000010101110010111100101101101110101111100101111011000
0111001110100111101100110000110010101100010101100111101111

```
=====
Num of units in str1: 57
Num of units in str2: 56
=====
Max in str1 and str2:
1011001010010101110101100001100100001111101010011110011111100000010010
0000010100010111101100011001000000000101011101011001001000
=====
str1 >> 3
0001011001010010101110101100001100100001111101010011110011111100000010
0100000010100010111101100011001000000000101011101011001001
=====
str1 << 5
1100101001010111010110000110010000111110101001111001111110000001001000
0001010001011110110001100100000000010101110101100100100000
=====
```

test_02.txt:

=====

str1

1001100111110101100111100101101000011010110010101111100111000000101100
0111000011110101001000000010010100100101000110100001011000

str2

011101110011101011110100111001001001000100010111000000000101100001100
01001101000110111001110101001101100010011010110011101100000

str1 and str2

00010001001100001001010001000000000100000000001000000000010000000010
000001000000110101001000000000000100000001000110000001000000

=====

str1

1001100111110101100111100101101000011010110010101111100111000000101100
0111000011110101001000000010010100100101000110100001011000

str2

011101110011101011110100111001001001000100010111000000000101100001100
01001101000110111001110101001101100010011010110011101100000

str1 or str2

1111111111111111111110111111101001101111011111111110011101100011110011
11101011110111001110101011111100110111010110111101111000

=====

str1

1001100111110101100111100101101000011010110010101111100111000000101100
0111000011110101001000000010010100100101000110100001011000

str2

011101110011101011110100111001001001000100010111000000000101100001100
01001101000110111001110101001101100010011010110011101100000

str1 xor str2

1110111011001111011010101011111010001011110111011111100110011000110100
1110101011000010000110101011111000110110010000111100111000

=====

str1

1001100111110101100111100101101000011010110010101111100111000000101100
0111000011110101001000000010010100100101000110100001011000

not str1

0110011000001010011000011010010111100101001101010000011000111111010011
1000111100001010110111111101101011011010111001011110100111

=====

str2

011101110011101011110100111001001001000100010111000000000101100001100
01001101000110111001110101001101100010011010110011101100000

not str2

1000100011000101000010110001101101101110111010001111111110100111100111
0110010111001000110001010110010011101100101001100010011111

```
=====
Num of units in str1: 59
Num of units in str2: 60
=====
Max in str1 and str2:
011101110011101011110100111001001001000100010111000000000101100001100
01001101000110111001110101001101100010011010110011101100000
=====
str1 >> 3
0001001100111110101100111100101101000011010110010101111100111000000101
1000111000011110101001000000010010100100101000110100001011
=====
str1 << 5
0110011111010110011110010110100001101011001010111110011100000010110001
1100001111010100100000001001010010010100011010000101100000
=====
```

test_03.txt:

=====

str1

010011100101111010011000010101101010011010110010001110010100000011011
10001000000111100001010111010111110011010110011001010100000

str2

101111011011011000110001101111101000111010011110010100011100000011100
1000010111000010010111111010110001111100001100011101110000

str1 and str2

000011000001011000010000000101100000011000000010001010000100000001010
0000000000000000000010111010110000011000000000001000100000

=====

str1

010011100101111010011000010101101010011010110010001110010100000011011
10001000000111100001010111010111110011010110011001010100000

str2

101111011011011000110001101111101000111010011110010100011100000011100
1000010111000010010111111010110001111100001100011101110000

str1 or str2

11111111111111010111001111111111100111111111001110011110000011111110
01010111111110011111111010111111111101111111011111110000

=====

str1

010011100101111010011000010101101010011010110010001110010100000011011
10001000000111100001010111010111110011010110011001010100000

str2

101111011011011000110001101111101000111010011110010100011100000011100
1000010111000010010111111010110001111100001100011101110000

str1 xor str2

111100111110100010101001111010011110000111111010001000110100000101011
10010101111111001110100000000111100110111111010111010000

=====

str1

010011100101111010011000010101101010011010110010001110010100000011011
10001000000111100001010111010111110011010110011001010100000

not str1

1011000110100001011001111010100101011001010011011100011010111111001000
1110111111000011110101000101000001100101001100110101011111

=====

str2

101111011011011000110001101111101000111010011110010100011100000011100
1000010111000010010111111010110001111100001100011101110000

not str2

0100001001001001110011100100000010111000101100001101011100011111100011
0111101000111101101000000101001110000011110011100010001111


```
=====
Num of units in str1: 60
Num of units in str2: 67
=====
Max in str1 and str2:
1011110110110110001100011011111101000111010011110010100011100000011100
1000010111000010010111111010110001111100001100011101110000
=====
str1 >> 3
000010011100101111010011000010101101010011010110010001110010100000011
01110001000000111100001010111010111110011010110011001010100
=====
str1 << 5
001110010111101001100001010110101001101011001000111001010000001101110
00100000011110000101011101011111001101011001100101010000000
=====
```

test_04.txt:

=====

str1

1001100011111010001110111111010011100110111000010001010101000000100101
0010101001100010000111101101010000100110000011011001011000

str2

110100111100100110001110100101101101001011010111110010010000000010100
0011100101011010100101001100010100000110111101011111100000

str1 and str2

100100001100100000001010100101001100001011000001000000010000000010000
00010001000100000000010001000000000000100000010011001000000

=====

str1

1001100011111010001110111111010011100110111000010001010101000000100101
0010101001100010000111101101010000100110000011011001011000

str2

110100111100100110001110100101101101001011010111110010010000000010100
00111001010110101001010011000101000001101111010111111100000

str1 or str2

1101101111110111011111111101101111011011110111110111101010000001011010
111101011110111001111111101111000101111111011111111111000

=====

str1

1001100011111010001110111111010011100110111000010001010101000000100101
0010101001100010000111101101010000100110000011011001011000

str2

110100111100100110001110100101101101001011010111110010010000000010100
00111001010110101001010011000101000001101111010111111100000

str1 xor str2

010010110011001110110101011000100011010000110110110111000100000000110
10101100011010111001101110101111000101011111001100110111000

=====

str1

1001100011111010001110111111010011100110111000010001010101000000100101
0010101001100010000111101101010000100110000011011001011000

not str1

0110011100000101110001000000101100011001000111101110101010111111011010
1101010110011101111000010010101111011001111100100110100111

=====

str2

110100111100100110001110100101101101001011010111110010010000000010100
00111001010110101001010011000101000001101111010111111100000

not str2

0010110000110110011100010110100100101101001010000011011011111111010111
1000110101001010110101100111010111110010000101000000011111

```
=====
Num of units in str1: 59
Num of units in str2: 62
=====
Max in str1 and str2:
110100111100100110001110100101101101001011010111110010010000000010100
0011100101011010100101001100010100000110111101011111100000
=====
str1 >> 3
0001001100011111010001110111111010011100110111000010001010101000000100
1010010101001100010000111101101010000100110000011011001011
=====
str1 << 5
0110001111101000111011111101001110011011100001000101010100000010010100
1010100110001000011110110101000010011000001101100101100000
=====
```

test_05.txt:

=====

str1

1111001010111011011101000010100011101001100010110101011001110000000110
1001110111000011001010110000000001110101010101000000001100

str2

000010011010101101001111010011100101001010000101011100000010010011111
00100101001000110000111111000011000011010100001111110111000

str1 and str2

00000000101010110100010000001000010000001000000101010000001000000001
100000100001000010000010110000000000010000000001000000001000

=====

str1

1111001010111011011101000010100011101001100010110101011001110000000110
1001110111000011001010110000000001110101010101000000001100

str2

000010011010101101001111010011100101001010000101011100000010010011111
00100101001000110000111111000011000011010100001111110111000

str1 or str2

1111101110111011011111110110111011110111000111101110110011101001111101
10111111100011100111111100001100111111111010111110111100

=====

str1

1111001010111011011101000010100011101001100010110101011001110000000110
1001110111000011001010110000000001110101010101000000001100

str2

000010011010101101001111010011100101001010000101011100000010010011111
00100101001000110000111111000011000011010100001111110111000

str1 xor str2

1111101100010000001110110110011010111011000011100010011001010100111000
110101111000010100110100100001100110111110100111110110100

=====

str1

1111001010111011011101000010100011101001100010110101011001110000000110
1001110111000011001010110000000001110101010101000000001100

not str1

0000110101000100100010111101011100010110011101001010100110001111111001
01100010001111001101010011111111000101010101011111110011

=====

str2

000010011010101101001111010011100101001010000101011100000010010011111
00100101001000110000111111000011000011010100001111110111000

not str2

1111011001010100101100001011000110101101011110101000111111011011000001
1011010110111001111000000111100111100101011110000001000111

```
=====
Num of units in str1: 58
Num of units in str2: 59
=====
Max in str1 and str2:
000010011010101101001111010011100101001010000101011100000010010011111
00100101001000110000111111000011000011010100001111110111000
=====
str1 >> 3
000111100101011101101110100001010001110100110001011010101100111000000
01101001110111000011001010110000000001110101010101000000001
=====
str1 << 5
110010101110110111010000101000111010011000101101010110011100000001101
00111011100001100101011000000000111010101010100000000100000
=====
```

5. Объяснение результатов работы программы.

- 1) При запуске программы oop_exercise_01 с аргументами test_?.txt объекты str1, str2 получают данные из файлов test_?.test.
- 2) Программа выполняет операции: and, or, xor, not, для строк полученных из тестов.
- 3) Побитовый сдвиг строк str1 и str2 вычисляется с помощью функций shiftLeft и shiftRight, затем выводится в стандартный поток вывода с помощью функции print_bits.
- 4) Для строк str1 и str2 вычисляется результат битовых операций (and, or, xor, not) с помощью функций AND, OR, XOR, NOT; результат выводится с помощью функции print_bits.
- 5) Вычисление количества единиц в строке осуществляется с помощью функции num_of_units; функция возвращает значение типа int и выводится в стандартный вывод с помощью std::cout.
- 6) Сравнение строк по количеству единиц осуществляется с помощью функции comparison, которая использует предыдущую функцию num_of_units; функция возвращает указатель на объект класса BigString, затем с помощью метода класса print_bits выводим слово с наибольшим количеством единиц в стандартный вывод.
- 7) Проверка включения осуществляется с помощью функции is_include.

6. Вывод.

Выполняя данную лабораторную я получил опыт работы с простыми классами, с системой сборки Cmake, с системой контроля версий GitHub, а также изучил основы работы с классами в C++. Создал класс, соответствующий варианту моего задания, реализовал для него битовые операции: «И»(and), «ИЛИ»(or), «ОТРИЦАНИЕ»(not), «Исключающее или»(xor), а также операции побитового сдвига(shiftLeft и shiftRight).