Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 1
Вариант 7
Простые классы

Студент:	Чурсина Н. А.
Группа:	М8О-208Б-18
Преподаватель:	Журавлев А.А.
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++

```
// //
// //
// //
// //
// //  Created by kosha on 28/09/2019.
// //  Copyright © 2019 kosha. All rights reserved.
// //

#include <string>
#include <iostream>
#include "BitString.h"


std::ostream& operator<<(std::ostream& os, const BitString128& obj){
    for (int i = 63; i >= 0; --i) os << ((obj.high >>i) & 1);
    for (int i = 63; i >= 0; --i) os << ((obj.low >>i) & 1);
    os << "\n";
    return os;
}

std::istream& operator>>(std::istream& is, BitString128& obj){
    std::string input_number;
    is >> input_number;
    obj = BitString128(input_number);
    return is;
}



int main(){
    BitString128 num1;
    BitString128 num2;
    BitString128 n_and;
    BitString128 n_or;
    BitString128 n_xor;

    std::cin >> num1;
    std::cin >> num2;

    std::cout<< "num1: " << num1;
    std::cout <<"num2: " << num2;

    std::cout<<"not1: "<< NOT(num1);
    std::cout<<"not2: "<< NOT(num2);

    n_and = AND(num1, num2);
    std::cout <<"AND:  "<< n_and;

    n_or = OR(num1, num2);
    std::cout << "OR:   " << n_or;

    n_xor = XOR(num1, num2);
```

```
    std::cout << "XOR:  " << n_xor<<"\nNUM1:   \n\nrShift:  \n";
    num1.rShift(2);

    printf("lShift: \n");
    num1.lShift(2);
    std::cout<< "\nNUM2:  \n\nrShift: \n";
    num2.rShift(2);
    std::cout<<"lShift:\n";
    num2.lShift(2);


    std::cout<< "\nQuantity of 1-bits: \nnum1:  \n" <<num1.get_bits()<<"\n";
    std::cout<< "num2:  \n"<<num2.get_bits()<<"\nDifference:\n";
    std::cout<< num1.compare_by_bits(num2)<<"\n";
    std::cout <<"Comparing by bits:\nnum1 < num2 ? :" << is_less(num1,num2) << "\n";
    std::cout <<"num 1 = num2 ? :"<< is_equal(num1, num2)<< "\n";
    std::cout <<"num1 > num2 ? :"<< is_greater(num1,num2) << "\n";

    return 0;
}
```

Header:

```
//
//  BitString.h

//
//  Created by kosha on 14/10/2019.
//  Copyright © 2019 kosha. All rights reserved.
//

#ifndef BitString_h
#define BitString_h
#include <string>
#include <iostream>
#include <string>


class BitString128;
std::ostream& operator <<(std::ostream&, const BitString128&);
class BitString128 {
public:
    uint64_t high=0, low=0;

    BitString128() = default;

    BitString128(std::string num){              //конструктор класса
        int j =0;
        for (auto i = num.rbegin(); i!=num.rend(); ++i, ++j){
            j<64? low += uint64_t(*i - '0')<<j : high += uint64_t(*i - '0')<<(j-64);
        }
    }
```

```
    BitString128& operator>>=(int k){
        for (int i = 0; i< k; ++i, r_bit_shift());
        return *this;
    }

    friend const BitString128 operator>>(const BitString128& a, const int& k){
        BitString128 res = a;
        return res>>=k;
    }

    BitString128& operator<<=(int k){
        for (int i = 0; i< k; ++i, l_bit_shift());
        return *this;
    }

    friend const BitString128 operator<<(const BitString128& a, const int& k){
        BitString128 res = a;
        return res<<=k;
    }



    //возвращает количество единичных битов в числе
    int get_bits() const{
        int count = 0;
        for (int i = 63; i >= 0; --i){
            count += (high >>i) & 1;
        }
        for (int i = 63; i >= 0; --i){
            count += (low >>i) & 1;
        }
        return count;
    }
    //возвращает разницу в количестве единичных битов
    int compare_by_bits(const BitString128& rhs)const{
        return abs(this->get_bits() - rhs.get_bits());
    }
    void lShift(int k){
        std::cout << ((*this)<<k);
    }

    void rShift(int k){
        std::cout << ((*this)>>k);
    }

private:
    void l_bit_shift(){
        std::cout << (*this);
        high <<= 1;
        high |= (low >>63) & 1;
        low <<= 1;
    }
```

```
    void r_bit_shift(){
        std::cout << (*this);
        low >>= 1;
        low |= (high & 1)<<63;
        high >>= 1;
    }
};

bool is_equal (const BitString128 a, const BitString128 b){
    return a.high == b.high and a.low == b.low;
}
bool is_less (const BitString128 a, const BitString128 b){
     return (a.high < b.high or (a.high == b.high and a.low < b.low));
}
bool is_greater (const BitString128 a, const BitString128 b){
    return (a.high > b.high or (a.high == b.high and a.low > b.low));
}

const BitString128 XOR(const BitString128 a, const BitString128 b ){
    BitString128 c;
    c.low = a.low ^ b.low;
    c.high = a.high ^ b.high;
    return c;
}
const BitString128 AND(const BitString128 a, const BitString128 b ){
    BitString128 c;
    c.low = a.low & b.low;
    c.high = a.high & b.high;
    return c;
}

const BitString128 OR(const BitString128 a, const BitString128 b){
    BitString128 c;
    c.low = a.low | b.low;
    c.high = a.high | b.high;
    return c;
}

BitString128 NOT(BitString128 res) {
    res.high = ~res.high;
    res.low = ~res.low;
    return res;
}

#endif /* BitString_h */
```

2. Ссылка на репозиторий на Github

https://github.com/kwk18/oop_exercise_01

3.Набор testcases

1. Входные данные:

1111111111111000000000000000000000000000000000000000000000000000101010001
1101011101010111111

2. Входные данные:

11111111111110000000000000000000000000000000000000000000000000001010100010000111
11110010101011010

00000000000000000000000000000111111001010100110010101010101001010100101000101
10100000000000000000101010010

3. Входные данные:

0111010
101

4. Входные данные:

0
10

5.

1001
1001

4. Результат выполнения тестов

1.

num1:
00000000000000000000000000000000000000000000000111111111111100000000000000000000000000000000000
0000000000000101010001

num2:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0001101011101010111111

not1:
11111111111111111111111111111111111111111111111000000000000011111111111111111111111111111111111
1111111111111010101110

not2:
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1110010100010101000000

AND:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000010001

OR:
00000000000000000000000000000000000000000000000111111111111100000000000000000000000000000000000
0001101011101111111111

XOR:
00000000000000000000000000000000000000000000000111111111111100000000000000000000000000000000000
0001101011101111101110

NUM1:

rShift:

000000000000000000000000000000000000000000000001111111111111000000000000000000000000000000000
0000000000000000101010001

00000000000000000000000000000000000000000000000011111111111110000000000000000000000000000000000000000000000000000010101000

00000000000000000000000000000000000000000000000011111111111110000000000000000000000000000000000000000000000000000001010100

lShift:

00000000000000000000000000000000000000000000000011111111111110000000000000000000000000000000000000000000000000000101010001

00000000000000000000000000000000000000000000000011111111111110000000000000000000000000000000000000000000000000001010100010

00000000000000000000000000000000000000000000000011111111111110000000000000000000000000000000000000000000000000010101000100

NUM2:

rShift:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000011010111010101011111

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000001101011101010111111

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000000110101110101011111

lShift:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000011010111010101011111

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000110101110101011111110

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00001101011101010111111100

Quantity of 1-bits:

num1:

17

num2:

15

Difference:

2

Comparing by bits:

num1 < num2 ? :0

num 1 = num2 ? :0

num1 > num2 ? :1

2.

num1:

11111111111111111111111111111111111111110000000000000000001111000000000000000000000000000000000000010100010000011111110010101011011

num2:

00000000000000000000000000000000000000000000000001111111001010111111111111111111111111111111111111111001101010101001011001111010

not1:

00000000000000000000000000000000000000001111111111111111110000011111111111111111111111111111111101011101111100000001101010100100

not2:

11111111111111111111111111111111111111111111111110000000110101000000000000000000000000000000000000110010101010101101001100000101

AND:

00000000000000000000000000000000000000000000000000000000000000001010100000000000000000000000000000000000010000000000101010000100010111010

OR:

11111111111111111111111111111111111111110000000000111111100111111111111111111111111111111111111111101111010111111111011101111011

XOR:
1111111111111111111111111111111111111111100000000001111110001010111111111111111111111111111111110110111
10100101011100110010000l

NUM1:

rShift:
1111111111111111111111111111111111111111100000000000000000001111100000000000000000000000000000000001010
0010000111111100101010110111
0111111111111111111111111111111111111111110000000000000000000111110000000000000000000000000000000000101
000100001111111001010101101
0011111111111111111111111111111111111111111000000000000000000011111000000000000000000000000000000000010
1000100001111111001010101110
lShift:
1111111111111111111111111111111111111111100000000000000000001111100000000000000000000000000000000001010
0010000111111100101010110111
1111111111111111111111111111111111111111000000000000000000011111000000000000000000000000000000000010100
010000111111100101010110110
1111111111111111111111111111111111111110000000000000000000111110000000000000000000000000000000000101000
100001111111001010101101100

NUM2:

rShift:
0000000000000000000000000000000000000000000000000011111110010101111111111111111111111111111111111111100
1101010101010010110011111010
0000000000000000000000000000000000000000000000000001111111001010111111111111111111111111111111111111110
011010101010100101100111101
0000000000000000000000000000000000000000000000000000111111100101011111111111111111111111111111111111111
0011010101010100101100111110
lShift:
0000000000000000000000000000000000000000000000000011111110010101111111111111111111111111111111111111100
1101010101010010110011111010
0000000000000000000000000000000000000000000000000111111100101011111111111111111111111111111111111111001
1010101010100101100111101000
0000000000000000000000000000000000000000000000001111111001010111111111111111111111111111111111111110011
0101010101001011001111010000

Quantity of 1-bits:
num1:
62
num2:
60
Difference:
2
Comparing by bits:
num1 < num2 ? :0
num 1 = num2 ? :0
num1 > num2 ? :1

3.

num1:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000111010
num2:
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000101
not1:
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111000101
not2:
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111010

AND:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000
OR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000111111
XOR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000111111

NUM1:

rShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000111010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000011101
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000001110
lShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000111010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000001110100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000011101000

NUM2:

rShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000101
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000001
lShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000101
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000001010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000010100

Quantity of 1-bits:
num1:
4
num2:
2
Difference:
2
Comparing by bits:
num1 < num2 ? :0
num 1 = num2 ? :0
num1 > num2 ? :1

4.

num1:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000
num2:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000010

not1:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111
not2:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111101
AND:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
OR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010
XOR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010

NUM1:

rShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
lShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000

NUM2:

rShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000001
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
lShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000001000

Quantity of 1-bits:
num1:
0
num2:
1
Difference:
1
Comparing by bits:
num1 < num2 ? :1
num 1 = num2 ? :0
num1 > num2 ? :0

5.

num1:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000010001

num2:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000010001

not1:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111101110

not2:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111101110

AND:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000010001

OR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000010001

XOR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000

NUM1:

rShift:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000010001

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000001000

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000100

lShift:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000010001

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000100010

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000001000100

NUM2:

rShift:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000010001

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000001000

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000100

lShift:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000010001

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000100010

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000001000100

Quantity of 1-bits:

```
num1:
2
num2:
2
Difference:
0
Comparing by bits:
num1 < num2 ? :0
num 1 = num2 ? :1
num1 > num2 ? :0
```

5. Объяснение результатов программы

Данная программа создает класс BitString128 для работы с 128-битовыми числами. Числа представлены двумя полями: типа unsigned long long для старшей и младшей частей. Реализованы операции AND, OR, XOR, NOT, сдвиги на заданное количество битов. Так же операции сравнения (на равенство, больше и меньше), операция вычисления количества единичных битов, операция сравнения по количеству единичных битов.

Результат работы зависит от выбора нужного метода класса :

1. Метод OR
2. Метод NOT
3. Метод AND
4. Метод XOR
5. Метод lShift
6. Метод rShift
7. Метод is_less
8. Оператор is_greater
9. Оператор is_equal
10. Метод get_bits
11. Метод compare_by_bits

При использовании методов 1-6 программа выполняет стандартные операции для чисел, представленных в двоичной системе счисления.
Методы 7-9 это методы сравнения чисел.
Метод 10 возвращает количество единичных битов в числе.
Метод 11 сравнивает два числа по количеству единичных битов. Возвращает разницу в их количестве.

Вывод: Проделав данную работу я изучила классы и компоненты классов. Освоила работу с методами и объектами класса.