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 2
Вариант 7
Операторы, литералы

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

Москва 2019

1. Код программы на языке C++

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


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;
    std::cin >> num1;
    std::cin >> num2;
    printf("num1: ");
    std::cout<< num1;
    printf("num2: ");
    std::cout << num2;
    printf("not1: ");
    std::cout<< (~num1);
    printf("not2: ");
    std::cout<< (~num2);

    printf("|:    ");
    std::cout<<(num1 | num2);
    printf("^:    ");

    std::cout<< (num1 ^ num2);
    printf("&:    ");

    std::cout<< (num1 & num2);
    printf("NUM 1:  \n");
    printf("rShift: \n");
    std::cout<< (num1>>2) <<"\n";

    printf("lShift \n");
    std::cout<< (num1<<2);
    std::cout<< "\n";
    printf("NUM2:  \n");
    printf("rShift \n");
    std::cout<< (num2>>2) <<"\n";
```

```
    printf("lShift \n");
    std::cout<< (num2<<2);
    std::cout<< "\n";
    printf("num1:  \n");
    std::cout<< num1.get_bits()<<"\n";
    printf("num2:  \n");
    std::cout<< num2.get_bits()<<"\n";
    std::cout<< num1.compare_by_bits(num2)<<"\n";
    std::cout <<"num1 < num2 ? :" << (num1 < num2) << "\n";
    std::cout <<"num 1 = num2 ? :"<< (num1 == num2)<< "\n";
    std::cout <<"num1 > num2 ? :"<< (num1 > num2) << "\n";
    return 0;
}
```

Header:

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

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

    BitString128() = default;

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

    const BitString128 operator~() const{
        BitString128 res = *this;
        res.high = ~res.high;
        res.low = ~res.low;
        return res;
    }

    BitString128& operator^=(const BitString128& rhs){
        high ^= rhs.high;
        low ^= rhs.low;
        return *this;
    }

    friend const BitString128 operator^(const BitString128& rhs, const BitString128& lhs){
        BitString128 res = lhs;
```

```
    return res^=rhs;
}

BitString128& operator|=(const BitString128& rhs){
    high |= rhs.high;
    low |= rhs.low;
    return *this;
}

friend const BitString128 operator|(const BitString128& rhs, const BitString128& lhs){
    BitString128 res = lhs;
    return res|=rhs;
}

BitString128& operator&=(const BitString128& rhs){
    high &= rhs.high;
    low &= rhs.low;
    return *this;
}

friend const BitString128 operator&(const BitString128& rhs, const BitString128& lhs){
    BitString128 res = lhs;
    return res &= rhs;;
}

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

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

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

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

bool operator<(const BitString128& rhs)const{
    return (high < rhs.high or (high == rhs.high and low < rhs.low));
}

bool operator>(const BitString128& rhs)const{
    return rhs < *this;
}
```

```
    bool operator>=(const BitString128& rhs)const{
        return !(*this < rhs);
    }

    bool operator<=(const BitString128& rhs)const{
        return !(*this > rhs);
    }

    bool operator==(const BitString128& rhs)const{
        return high == rhs.high and low == rhs.low;
    }

    bool operator!=(const BitString128& rhs)const{
        return !(*this == rhs);
    }

    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;
    }

    int compare_by_bits(const BitString128& rhs)const{
        return abs(this->get_bits() - rhs.get_bits());
    }

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;
    }
};

#endif /* BitString_h */
```

2. Ссылка на репозиторий на Github

https://github.com/kwk18/oop_exercise_02

3.Набор testcases

1. Входные данные:

1111111111111000000000000000000000000000000000000000000000000000000101010001

110101110101010111111

2. Входные данные:

11111111111110000000000000000000000000000000000000000000000000000000101010001000011111110010101011010

000000000000000000000000000000000001111110010101001100101010101010010101001010001011010000000000000000101010010

3. Входные данные:

0111010

101

4. Входные данные:

0

10

4. Результат выполнения тестов

1.

num1:

000000000000000000000000000000000000000000000000000111111111111100000000000000000000000000000000000000000000000000000000101010001

num2:

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000110101110101010111111

not1:

111111111111111111111111111111111111111111111111111000000000000011111111111111111111111111111111111111111111111111111111010101110

not2:

11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111001010001010101000000

|:

000000000000000000000000000000000000000000000000000111111111111100000000000000000000000000000000000000000000110101110101111111111

^:

000000000000000000000000000000000000000000000000000111111111111100000000000000000000000000000000000000000000110101110101111101110

&:

0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010001

NUM 1:

rShift:

000000000000000000000000000000000000000000000000000111111111111100000000000000000000000000000000000000000000000000000000101010001

000000000000000000000000000000000000000000000000000011111111111110000000000000000000000000000000000000000000000000000000010101000

00000000000000000000000000000000000000000000000000000111111111111100000000000000000000
00000000000000000000000000000000000001010100

lShift
00000000000000000000000000000000000000000000000000011111111111110000000000000000000000
00000000000000000000000000000000000101010001
00000000000000000000000000000000000000000000000000111111111111100000000000000000000000
00000000000000000000000000000000001010100010
00000000000000000000000000000000000000000000000001111111111111000000000000000000000000
00000000000000000000000000000000010101000100

NUM2:
rShift
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000110101110101010111111
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000011010111010101011111
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000001101011101010101111

lShift
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000110101110101010111111
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000001101011101010101111110
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000011010111010101011111100

num1:
17
num2:
15
2
num1 < num2 ? :0
num 1 = num2 ? :0
num1 > num2 ? :1

2.
num1:
111111111111111111111111111111111111111100000000000000000001111100000000000000000000000
00000000001010001000011111110010101011011
num2:
000000000000000000000000000000000000000000000000001111111001010111111111111111111111111
11111111111100110101010101001011001111010
not1:
000000000000000000000000000000000000000011111111111111111110000011111111111111111111111
11111111110101110111100000001101010100100
not2:
111111111111111111111111111111111111111111111111110000000110101000000000000000000000000
00000000000011001010101010110100110000101
|:
111111111111111111111111111111111111111100000000001111111001111111111111111111111111111
11111111111110111101011111111011101111011

^:
1111111111111111111111111111111111111111110000000000111111000101011111111111111111111111
11111111110110111101001010111001100100001

&:
0000000000000000000000000000000000000000000000000000000000010101000000000000000000000000
00000000010000000000101010000100010111010

NUM 1:

rShift:

111111111111111111111111111111111111111111100000000000000000000111110000000000000000000000
0000000000000101000100001111111001010101011011

0111111111111111111111111111111111111111111100000000000000000000111110000000000000000000000
00000000000000101000100001111111001010101101

00111111111111111111111111111111111111111111100000000000000000000111110000000000000000000000
000000000000000101000100001111111001010101 10

lShift

111111111111111111111111111111111111111111100000000000000000000111110000000000000000000000
0000000000000101000100001111111001010101011011

1111111111111111111111111111111111111111110000000000000000000011111000000000000000000000000
000000000000101000100001111111001010101 10110

11111111111111111111111111111111111111111000000000000000000000111110000000000000000000000000
00000000000101000100001111111001010101101100

NUM2:

rShift

0000000000000000000000000000000000000000000000000000001111111001010111111111111111111111
1111111111111110011010101010100101100111 1010

00000000000000000000000000000000000000000000000000000000111111100101011111111111111111111
11111111111111110011010101010100101100111101

000000000000000000000000000000000000000000000000000000000111111100101011111111111111111111
111111111111111110011010101010100101100111 10

lShift

0000000000000000000000000000000000000000000000000000001111111001010111111111111111111111
1111111111111110011010101010100101100111 1010

000000000000000000000000000000000000000000000000000001111111001010111111111111111111111111
11111111111111001101010101010010110011110100

00000000000000000000000000000000000000000000000000001111111001010111111111111111111111111
1111111111110011010101010100101100111101000

num1:

62

num2:

60

2

num1 < num2 ? :0

num 1 = num2 ? :0

num1 > num2 ? :1

3.

num1:

0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000111010

num2:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101
not1:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000101
not2:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111010
|:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000111111
^:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000111111
&:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
NUM 1:
rShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000111010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011101
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001110

lShift
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000111010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001110100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011101000

NUM2:
rShift
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

lShift
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010100

num1:

4
num2:
2
2
num1 < num2 ? :0
num 1 = num2 ? :0
num1 > num2 ? :1

4.
num1:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000
num2:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000010
not1:
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111
not2:
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111101
|:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000010
^:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000010
&:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000
NUM 1:
rShift:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000

lShift
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000

NUM2:
rShift
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000010
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000001

000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000

lShift
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000010
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000100
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000001000

num1:
0
num2:
1
-1
num1 < num2 ? :1
num 1 = num2 ? :0
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 или оператора:

1. Оператор |
2. Оператор ~
3. Оператор &
4. Оператор >>
5. Оператор <<
6. Оператор ^
7. Оператор <
8. Оператор >
9. Оператор ==
10. Метод get_bits
11. Метод compare_by_bits

При использовании операторов 1-6 программа выполняет стандартные операции для чисел, представленных в двоичной системе счисления.
Операторы 7-9 это операторы 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 научилась их применять. Так же освоила перегрузку операторов.