МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №1

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Тихонов Фёдор Андреевич, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Задание:

Разработать программу на языке C++ согласно варианту задания. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Вариант №6:

Создать класс BitString для работы с 128-битовыми строками. Битовая строка должна быть представлена двумя полями типа unsigned long long. Должны быть реализованы все традиционные операции для работы с битами: and, or, xor, not. Реализовать сдвиг влево shiftLeft и сдвиг вправо shiftRight на заданное количество битов. Реализовать операцию вычисления количества единичных битов, операции сравнения по количеству единичных битов. Реализовать операцию проверки включения.

Описание программы:

Исходный код разделён на 2 файла:

- BitString.h – описание и реализация класса BitString
- main.cpp – основная программа

Дневник отладки:

Была синтаксическая ошибка при рефакторинге кода, в остальном сложностей не возникло.

Вывод:

В процессе выполнения работы я на практике познакомился и поработал с классами. Благодаря им упрощается написание кода для различных объемных программ, которые используют различные типы данных, содержащие сразу несколько различных полей.

Исходный код:

BitString.h:

```
#ifndef LAB0_1_BITSTRING_H
#define LAB0_1_BITSTRING_H

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

class BitString {
public:
    uint64_t high = 0, low = 0;

    BitString() = default;
    BitString(uint64_t lo) : high(0), low(lo) {};
    BitString(uint64_t hi, uint64_t lo) : high(hi), low(lo) {};
    BitString(std::string num) {
        int j = 0;
        for (auto i = num.rbegin(); i != num.rend(); i++, j++) {
            if (j < 64)
```

```
            low += uint64_t(*i - '0') << j;
        else
            high += uint64_t(*i - '0') << (j-64);
    }
}

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

friend std::ostream& operator<<(std::ostream& os, const BitString& 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;
}

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

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

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

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

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

BitString OR(const BitString b) {
    BitString c;
    c.low = low | b.low;
    c.high = high | b.high;
    return c;
}
```

```
    BitString NOT() {
        high = ~high;
        low = ~low;
        return *this;
    }

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

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

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

    int get_bits_1() const {
        int c = 0;
        for (int i = 63; i >= 0; i--)
            c += (int)(high >>i) & 1;
        for (int i = 63; i >= 0; i--)
            c += (int)(low >> i) & 1;
        return c;
    }

    int compare_by_bits(const BitString& rhs) const {
        return abs(this->get_bits_1() - rhs.get_bits_1());
    }

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 //LAB0_1_BITSTRING_H
```

main.cpp:

```
#include <string>
#include <iostream>
```

```
#include "BitString.h"

int main() {

    BitString n1;
    BitString n2;
    BitString n_not1;
    BitString n_not2;
    BitString n_and;
    BitString n_or;
    BitString n_xor;

    std::cout << "Enter the first number:";
    std::cin >> n1;
    std::cout << "Enter the second number:";
    std::cin >> n2;

    std::cout << "n1: " << n1;
    std::cout << "n2: " << n2;

    n_not1 = n1;
    n_not2 = n2;
    std::cout<<"not1: "<< n_not1.NOT();
    std::cout<<"not2: "<< n_not2.NOT();

    n_and = n1;
    n_and.AND(n2);
    std::cout <<"AND: "<< n_and;

    n_or = n1;
    n_or.OR(n2);
    std::cout << "OR: " << n_or;

    n_xor = n1;
    n_xor.XOR(n2);
    std::cout << "XOR: " << n_xor;

    std::cout <<"\nn1: rShift: ";
    n1.rShift(2);
    printf("lShift:");
    n1.lShift(2);
    std::cout << "\nn2: rShift: ";
    n2.rShift(2);
    std::cout << "lShift:\n";
    n2.lShift(2);

    std::cout << "\nQuantity of 1-bits: \nn1: " << n1.get_bits() << "\n";
    std::cout << "n2: " << n2.get_bits();
    std::cout << "\nDifference: " << n1.compare_by_bits(n2) << "\n\n";
    std::cout << "Comparing by bits:\n"
              << "n1 < n2 ? " << (n1.is_less(n2) ? "true" : "false") << "\n";
    std::cout << "n1 = n2 ? " << (n1.is_equal(n2) ? "true" : "false") << "\n";
    std::cout << "n1 > n2 ? " << (n1.is_greater(n2) ? "true" : "false") << "\n";

    return 0;
}
```

Пример работы:

```
Enter the first number: 1010010
Enter the second number: 1000101001
n1: 00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000001010010
n2:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000
001000101001
not1:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111
11111110101101
not2:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111
11110111010110
AND:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000
0000001010010
OR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000
000001010010
XOR:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000
0000001010010

n1: rShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000
0000000000001010010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
00101001
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
00010100
lShift:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
000000001010010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
10100100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000001
01001000

n2: rShift:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000
0000000001000101001
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000001
00010100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000
10001010
lShift:
```

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010
00101001
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000100
01010010
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000001000
10100100

Quantity of 1-bits:
n1: 3
n2: 4
Difference: 1

Comparing by bits:
n1 < n2 ? true
n1 = n2 ? false
n1 > n2 ? false

Process finished with exit code 0
```