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: 000000000000000000000000000000000000000000000000000000000000000000000

00000000000000000000000000000000000000000000000000001010010

n2: 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

001000101001

not1: 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

11111110101101

not2: 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

11110111010110

AND: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

0000001010010

OR: 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

000001010010

XOR: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

0000001010010

n1: rShift: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000000000001010010

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00101001

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00010100

lShift:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

000000001010010

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

10100100

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

01001000

n2: rShift: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000000001000101001

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

00010100

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

10001010

lShift:

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010

00101001

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100

01010010

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001000

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