МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №2**

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. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

**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:**

Проблем gh

**Вывод:**  
 В рамках данной лабораторной работы я поработал с важной вещью - перегрузкой операторов. Также я познакомился с пользовательскими литералами . Это очень удобная и практическая вещь, о которой я не знал до изучения курса ООП. Использование этого средства позволяет получать из заданных типов данных какие-либо другие данные.

**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;

}

const BitString operator~() const{

BitString res = \*this;

res.high = ~res.high;

res.low = ~res.low;

return res;

}

BitString& operator^=(const BitString& rhs) {

high ^= rhs.high;

low ^= rhs.low;

return \*this;

}

friend BitString operator^(const BitString& rhs, const BitString& lhs) {

BitString res = lhs;

return res ^= rhs;

}

BitString& operator|=(const BitString& rhs) {

high |= rhs.high;

low |= rhs.low;

return \*this;

}

friend BitString operator|(const BitString& rhs, const BitString& lhs) {

BitString res = lhs;

return res |= rhs;

}

BitString& operator&=(const BitString& rhs) {

high &= rhs.high;

low &= rhs.low;

return \*this;

}

friend BitString operator&(const BitString& rhs, const BitString& lhs) {

BitString res = lhs;

return res &= rhs;

}

BitString& operator>>=(int rhs) {

for (int i = 0; i < rhs; i++, r\_bit\_shift());

return \*this;

}

friend BitString operator>>(const BitString& lhs, const int& rhs) {

BitString res = lhs;

return res >>= rhs;

}

BitString& operator<<=(int rhs) {

for (int i = 0; i < rhs; i++, l\_bit\_shift());

return \*this;

}

friend BitString operator<<(const BitString& lhs, const int& rhs) {

BitString res = lhs;

return res <<= rhs;

}

bool operator<(const BitString& rhs) const {

return (high < rhs.high or (high == rhs.high and low < rhs.low));

}

bool operator>(const BitString& rhs) const {

return rhs < \*this;

}

bool operator>=(const BitString& rhs) const {

return !(\*this < rhs);

}

bool operator<=(const BitString& rhs) const {

return !(\*this > rhs);

}

bool operator==(const BitString& rhs) const {

return high == rhs.high and low == rhs.low;

}

bool operator!=(const BitString& rhs) const {

return !(\*this == rhs);

}

int get\_bits() 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() - 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;

}

};

BitString operator "" \_BitString(const char \*str, size\_t n) {

return {std::string(str, n)};

}

#endif //LAB0\_1\_BITSTRING\_H

**main.cpp:**

#include <string>

#include <iostream>

#include "BitString.h"

int main() {

BitString n1;

BitString n2;

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;

std::cout<<"~n1 = "<< ~n1;

std::cout<<"~n2 = "<< ~n2;

std::cout <<"n1 & n2 = "<< (n1 & n2);

std::cout << "n1 | n2 = " << (n1 | n2);

std::cout << "n1 ^ n2 = " << (n1 ^ n2);

std::cout <<"\nn1: rShift: " << (n1 >> 2);

std::cout << "lShift:" << (n1 << 2);

std::cout << "\nn2: rShift: " << (n2 >> 2);

std::cout << "lShift: " << (n2 << 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 < n2 ? "true" : "false") << "\n";

std::cout << "n1 = n2 ? " << (n1 == n2 ? "true" : "false") << "\n";

std::cout << "n1 > n2 ? " << (n1 > n2 ? "true" : "false") << "\n\n";

BitString n\_str = "0101010101001"\_BitString;

std::cout << "Literal operator: \"0101010101001\"\_BitString;\noutput:\n"

<< n\_str << "\n";

return 0;

}

**Пример работы:**

Enter the first number:001010100101001

Enter the second number:01010100100101010010

n1 = 00000000000000000000000000000000000000000000000000000000000000000000

000000000000000000000000000000000000000000000001010100101001

n2 = 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101010

0100101010010

~n1 = 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111

10101011010110

~n2 = 111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111101010

11011010101101

n1 & n2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

000000000100000000

n1 | n2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

010101110101111011

n1 ^ n2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

010101110001111011

n1: rShift: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

00000001010100101001

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010

10010100

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101

01001010

lShift:00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

001010100101001

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101010

01010010

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010100

10100100

n2: rShift: 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

01010100100101010010

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010100100

10101001

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101010010

01010100

lShift: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101

0100100101010010

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101010010010

10100100

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010100100101

01001000

Quantity of 1-bits:

n1: 6

n2: 8

Difference: 2

Comparing by bits:

n1 < n2 ? true

n1 = n2 ? false

n1 > n2 ? false

Literal operator: "0101010101001"\_BitString;

output:

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001010

10101001

Process finished with exit code 0