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

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Белоусов Егор Владимирович, группа М8О-207Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

Цель:

- Изучение основ работы с классами в C++;
- Перегрузка операций и создание литералов

Задание:

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 лежит в 3 файлах:

1. main.cpp: основная программа

2. bitstring.h : описание класса bitstring

3. bitstring.cpp: реализация класса bitstring

Дневник отладки

В main.cpp выполняются самые различные операции с двумя битовыми строчками.

D:\oop\lab02\cmake-build-debug\lab02.exe

00000000000000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000

00001010

00000000000000000000000000000000000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000

00000000

00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011001011010101111111 0

10010101

00000000000000000000000000000000000000000000000000000000000010000
00000000000000000000000000000001100101101010111111110

10011111

0

Process finished with exit code 0

Недочёты

По моему мнению недочетов нет.

Выводы

В данной лабораторной работе я закрепил знания, полученные в прошлой ЛР.

Теперь вместо функции для операций используется более естественный и простой способ: перегрузка операторов. Также для удобства был реализован пользовательский литерал. После выполнения работы я стал лучше понимать, как писать универсальные классы, с которыми удобно работать.

Исходный код

main.cpp

```cpp
#include <iostream>
#include "bitstring.cpp"

int main() {
    BitString a(5, 10);
    BitString b(1, 0);
    std::cout << (a ^ b) << "\n";
    std::cout << (a & b) << "\n";
    BitString lit = 1100101101010111111010010101_bit;
    std::cout << lit << "\n";
    std::cout << (b ^ a ^ lit) << "\n";
    std::cout << (lit < a) << "\n";
    system("pause");// if you want to run exe
    return 0;
}
```

bitstring.h

```cpp
#ifndef LAB02_BITSTRING_H
#define LAB02_BITSTRING_H
```

```
#include "iostream"


class BitString {
private:
    unsigned long long first;
    unsigned long long second;
public:
    BitString();

    BitString(unsigned long long first, unsigned long long second);

    int count() const;

    friend BitString operator&(const BitString &a, const BitString &b);

    friend BitString operator|(const BitString &a, const BitString &b);

    friend BitString operator^(const BitString &a, const BitString &b);

    friend BitString operator~(const BitString &a);

    friend BitString operator<<(const BitString &a, unsigned int cnt);

    friend BitString operator>>(const BitString &a, unsigned int cnt);

    friend bool operator==(const BitString &a, const BitString &b);

    friend bool subMask(const BitString &mask, const BitString &submask);

    friend bool operator<(const BitString &a, const BitString &b);

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

    friend std::istream &operator>>(std::istream &is, BitString &a);

    ~BitString() = default;
};


#endif //LAB02_BITSTRING_H
```

bitstring.cpp

```
#include "bitstring.h"
#include "cstring"

BitString::BitString() : first(0), second(0) {}

BitString::BitString(unsigned long long first, unsigned long long second) :
first(first), second(second) {}

int BitString::count() const {
    return __builtin_popcountll(first) + __builtin_popcountll(second);
}

BitString operator&(const BitString &a, const BitString &b) {
    return BitString(a.first & b.first, a.second & b.second);
}
```

```cpp
BitString operator|(const BitString &a, const BitString &b) {
    return BitString(a.first | b.first, a.second | b.second);
}

BitString operator^(const BitString &a, const BitString &b) {
    return BitString(a.first ^ b.first, a.second ^ b.second);
}

BitString operator~(const BitString &a) {
    return BitString(~a.first, ~a.second);
}

BitString operator<<(const BitString &a, unsigned int cnt) {
    return BitString((a.first << cnt) + (a.second >> (64 - cnt)), a.second << cnt);
}

BitString operator>>(const BitString &a, unsigned int cnt) {
    return BitString(a.first >> cnt, (a.second >> cnt) + (a.first << (64 - cnt)));
}

bool operator==(const BitString &a, const BitString &b) {
    return a.first == b.first && a.second == b.second;
}

bool subMask(const BitString &a, const BitString &b) {
    return (a | b) == a;
}

bool operator<(const BitString &a, const BitString &b) {
    return a.count() < b.count();
}

std::istream &operator>>(std::istream &is, BitString &a) {
    is >> a.first >> a.second;
    return is;
}

std::ostream &operator<<(std::ostream &os, const BitString &a) {
    for (int i = 63; i >= 0; --i) {
        if ((a.first >> i) & 1) {
            os << "1";
        } else os << "0";
    }
    for (int i = 63; i >= 0; --i) {
        if ((a.second >> i) & 1) {
            os << "1";
        } else os << "0";
    }
    return os;
}

BitString operator "" _bit(const char *str) {
    unsigned long long first = 0, second = 0;
    int len = strlen(str);
    if (len <= 64) {
        for (int i = 0; i < len; ++i) {
            second *= 2;
            if (str[i] == '1')second++;
        }
    } else {
        for (int i = 0; i < len - 64; ++i) {
```

```
            first *= 2;
            if (str[i] == '1')first++;
        }
        for (int i = len - 64; i < len; ++i) {
            second *= 2;
            if (str[i] == '1')second++;
        }
    }
    return BitString(first, second);
}
```