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 *Ядров Артем Леонидович, группа М8О-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 7: BitString Разработать программу на языке C++ согласно варианту задания.

Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод. Реализовать над объектами реализовать в виде перегрузки операторов. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

Описание программы

Исходный код лежит в файле main.cpp и содержит класс BitString, его реализацию и пользовательский литерал.

Дневник отладки

Недочёты

Выводы

Я закрепил основы работы с классами в C++, а также познакомился с пользовательскими литералами и перегрузками операторов.

Исходный код

main.cpp

```
#include <iostream>

typedef long long ll;

class BitString {
public:
    BitString() = default;

    BitString(BitString &cp) = default;

    BitString(ll fir, ll sec) : first(fir), second(sec) {};

    BitString operator&(const BitString &right) const {
        return {this->first & right.first, this->second & right.second};
    }

    BitString operator|(const BitString &right) const {
        return {this->first | right.first, this->second | right.second};
    }

    BitString operator^(const BitString &right) const {
        return {this->first ^ right.first, this->second ^ right.second};
    }

    BitString operator!() const {
        return {!this->first, !this->second};
    }

    BitString operator<<(int x) const {
        if (x >= 64) {
            return {second << (x - 64), 0};
        }
        return {(first << x) | (second >> (64 - x)), second << x};
    }

    BitString operator>>(int x) const {
        if (x >= 64) {
            return {0, first >> (x - 64)};
        }
```

```
        return {first >> x, (first << (64 - x)) | (second << x)};
    }

    int num_single_bits() {
        int cnt = 0;
        BitString cp(*this);
        for (; cp.first; cp.first << 1, cp.second << 1) {
            cnt += (int) cp.first & 1;
            cnt += (int) cp.second & 1;
        }
        return cnt;
    }
    std::ostream& operator<<(std::ostream& os){
        os << this->first << "." << this->second;
    }
    bool operator<(BitString& rhs){
        return this->num_single_bits() < rhs.num_single_bits();
    }
    bool operator>(BitString& rhs){
        return this->num_single_bits() > rhs.num_single_bits();
    }
    bool substring(BitString& substr) const{
        BitString cp = *this & substr;
        return (cp.first != 0) && (cp.second != 0);
    }
    ll first_getter(){
        return this->first;
    }
    ll second_getter(){
        return this->second;
    }
    void first_setter(ll x){
        first = x;
    }
    void second_setter(ll x){
        second = x;
    }
private:
    ll first, second;
};

unsigned long long operator""_division_by_2(unsigned long long x){
```

```
    return x / 2;
}
int main(){
    BitString a, b;
    std::cout << 4_division_by_2 << std::endl;
    std::cout << "Enter first 128-BitString:" << std::endl << "> ";
    ll first, second;
    std::cin >> first >> second;
    a.first_setter(first); a.second_setter(second);
    std::cout << "Enter second 128-BitString:" << std::endl << "> ";
    std::cin >> first >> second;
    b.first_setter(first); b.second_setter(second);
    BitString ans = a & b;
    std::cout << "A & B:" << ans.first_getter() << "." << ans.second_getter();
    return 0;
}
```