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 *Зубко Дмитрий Валерьевич М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Цель:

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

Требования к программе

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.

6. Создать класс BitString для работы с 96-битовыми строками. Битовая строка должна быть представлена двумя полями: старшая часть unsigned long long, младшая часть unsigned int. 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 файле:

main.cpp - исполняемый код.

Дневник отладки

Во время выполнения лабораторной работы программа не нуждалась в отладке, все ошибки компиляции были исправлены с первой попытки. После их исправления программа работала так, как было задумано изначально.

Недочёты

Недочётов не было обнаружено.

Выводы

В процессе выполнения работы я на практике познакомился с пользовательскими литералами. Это очень удобная и практическая вещь, о которой я не знал до курса ООП. Использование этого средства позволяет получать из заданных типов данных какие то данные, вычислять что то, без использования функций, а с помощью переопределения специального оператора

Исходный код

main.cpp

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

class BitString;
bool operator==(const BitString& bs1, const BitString& bs2);

class BitString{
public:
    BitString():part1(0), part2(0){}

    BitString(const BitString& t):part1(t.part1), part2(t.part2){}

    BitString(unsigned long long a, unsigned int b):part1(a), part2(b){}

    BitString operator&(const BitString& t) const{
        BitString a(part1 & t.part1, part2 & t.part2);
        return a;
    }

    BitString operator|(const BitString& t) const{
        BitString a(part1 | t.part1, part2 | t.part2);
        return a;
    }

    BitString operator^(const BitString& t) const{
        BitString a(part1 ^ t.part1, part2 ^ t.part2);
        return a;
    }

    BitString operator~() const{
        BitString a(~part1, ~part2);
        return a;
    }

    BitString operator<<(int n) const{
```

```
    if(n > 96)
        return BitString();
    BitString bs(*this);
    unsigned int a = pow(2, 31);
    for(int i = 0; i < n; ++i){
        bs.part1 <<= 1;
        if(bs.part2 & a) {
            bs.part1 |= 1;
        }
        bs.part2 <<= 1;
    }
    return bs;
}

BitString operator >>(int n) const{
    if(n > 96)
        return BitString();
    BitString bs(*this);
    unsigned int b = pow(2, 31);
    unsigned long long a = 1;
    for(int i = 0; i < n; ++i){
        bs.part2 >>= 1;
        if(bs.part1 & 1){
            bs.part2 |= b;
        }
        bs.part1 >>= 1;
    }
    return bs;
}

friend std::ostream& operator<<(std::ostream& os, const BitString& bs);
friend std::istream& operator>>(std::istream& is, BitString& bs);

int CountOne() const{
    int n = 0;
    for(unsigned i = 0; i < sizeof(unsigned long long) * 8; ++i){
        if(1 & (part1 >> i)) ++n;
    }
    for(unsigned i = 0; i < sizeof(unsigned int) * 8; ++i){
        if(1 & (part2 >> i)) ++n;
    }
    return n;
}

bool includeBS(const BitString& bs) const{
```

```
        BitString s = bs & *this;
        return s == bs;

    }

private:
    unsigned long long part1;
    unsigned int part2;
};

std::ostream& operator<<(std::ostream& os, const BitString& bs){
    for(int i = sizeof(unsigned long long) * 8 - 1; i >= 0; --i){
        os << (1 & (bs.part1 >> i));
    }
    for(int i = sizeof(unsigned int) * 8 - 1; i >= 0; --i){
        os << (1 & (bs.part2 >> i));
    }
    return os;
}

std::istream& operator>>(std::istream& is, BitString& bs){
    is >> bs.part1;
    is >> bs.part2;
    return is;
}

bool operator<(const BitString& bs1, const BitString& bs2){
    return bs1.CountOne() < bs2.CountOne();
}

bool operator>(const BitString& bs1, const BitString& bs2){
    return bs1.CountOne() > bs2.CountOne();
}

bool operator==(const BitString& bs1, const BitString& bs2){
    return bs1.CountOne() == bs2.CountOne();
}

BitString operator "" _96bs(const char* str, size_t size) {
    int cnt = 0;
    std::string s = "";
    while (str[cnt] != ' ')
        s += str[cnt++];

    unsigned long long first = 0;
```

```
    unsigned int second = 0;
    for (int i = 0; i < s.size(); ++i) {
        first *= 10;
        first += str[i] - '0';
    }
    s = "";
    cnt++;
    while (str[cnt++] != '\0') {
        s += str[cnt];
    }
    for (int i = 0; i < s.size(); ++i) {
        second *= 10;
        second += str[i] - '0';
    }
    BitString res(first, second);
    return res;
}

int main() {
    BitString str(1234, 123);
    std::cout << str << std::endl << "1234 123"_96bs << std::endl;
    BitString bs1;
    BitString bs2;
    std::cout << "Enter bs1 and bs2: ";
    std::cin >> bs1 >> bs2;
    std::cout << "bs1: = " << bs1;
    std::cout << "\nbs2: = " << bs2;
    std::cout << "\nand: " << (bs1 & bs2);
    std::cout << "\nor: " << (bs1 | bs2);
    std::cout << "\nnot bs1: " << ~bs1;
    std::cout << "\nnot bs2: " << ~bs2;
    std::cout << "\nxor: " << (bs1 ^ bs2);
    std::cout << "\nshiftleft2 bs1: " << (bs1 << 2);
    std::cout << "\nshiftright2 bs1: " << (bs1 >> 2);
    std::cout << "\ncount one bs1: " << bs1.CountOne();
    std::cout << "\ncount one bs2: " << bs2.CountOne();
    std::cout << "\nbs1 > bs2: " << (bs1 > bs2);
    std::cout << "\nbs1 include bs2: " << bs1.includeBS(bs2);
}
```