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

ЛАБОРАТОРНАЯ РАБОТА №1 по курсу

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

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

Цель:

- Изучение системы сборки на языке C++, изучение систем контроля версии.
- Изучение основ работы с классами в C++;

Порядок выполнения работы

1. Ознакомиться с теоретическим материалом.
2. Получить у преподавателя вариант задания.
3. Реализовать задание своего варианта в соответствии с поставленными требованиями.
4. Подготовить тестовые наборы данных.
5. Создать репозиторий на GitHub.
6. Отправить файлы лабораторной работы в репозиторий.
7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

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.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться **oop_exercise_01** (в случае использования Windows **oop_exercise_01.exe)**

Необходимо зарегистрироваться на GitHub (если студент уже имеет регистрацию на GitHubто можно использовтаь ее) и создать репозитарий для задания лабораторной работы.

Преподавателю необходимо предъявить ссылку на публичный репозиторий на Github. Имя репозитория должно быть https://github.com//*login*/oop_exercise_01

Где login – логин, выбранный студентом для своего репозитория на Github.

Репозиторий должен содержать файлы:

- main.cpp //файл с заданием работы
- CMakeLists.txt // файл с конфигураций CMake
- test_xx.txt // файл с тестовыми данными. Где xx – номер тестового набора 01, 02 , … Тестовых наборов должно быть больше 1.
- report.doc // отчет о лабораторной работе

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 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.

Недочёты

Недочётов не было обнаружено.

Выводы

В процессе выполнения работы я на практике познакомился с классами. Благодаря им, упрощается написание кода для различных объемных программ, ипользующих различные типы данных, содержащие сразу несколько различных полей. Например, при необходимости использовать тип данных, соответствующий адресу дома, вместо хранения трех различных полей в программе, можно создать структуру типа адреса и использовать ее.

Исходный код

main.cpp

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

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);

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;
}

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();
}

int main() {
    BitString bs1(123455, 84);
    BitString bs2(8885, 7774);
    std::cout << "Test1:\n" << bs1 << std::endl << bs2 << std::endl
        << "and:\n" << (bs1 & bs2) << std::endl << "or:\n" << (bs1 | bs2) << std::endl
        << "xor:\n" << (bs1 ^ bs2) << std::endl << "not for bs1:\n" << ~bs1 <<
std::endl
        << "shiftleft2 for bs1:\n" << (bs1 << 2) << std::endl << "shiftright2 for bs1:\n"
<< (bs1 >> 2) << std::endl;
    std::cout << "----------------------------------------------------------" << std::endl;
    BitString bs3(3, 5);
    BitString bs4(1, 1);
    std::cout << "Test2:\n" << bs3 << std::endl << bs4 << std::endl
        << "CountOne for bs3: " << bs3.CountOne() << "\nCountOne for bs4: " <<
bs4.CountOne()
        << "\nbs3 > bs4? " << (bs3 > bs4) << "\nbs3 == bs4? " << (bs3 == bs4) <<
"\nbs3 < bs4? " << (bs3 < bs4)
        << std::endl << "bs3 include bs3? " << bs3.includeBS(bs4) << std::endl;
}
```