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 *Каширин Кирилл Дмитриевич, группа М8О-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

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. Исходный код лежит в трёх файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню
2. BitString.h: описание класса адресов
3. BitString.cpp: реализация класса адреса

Протокол работы

12345 6789
BitString a:
00000000000000000000000000000000000000000000000000101101110000
00100000000000000000000000000000000000000000000000111111010011101
BitString b:
00000000000000000000000000000000000000000000000011000000111
00100000000000000000000000000000000000000000000001101010000101
AND
00000000000000000000000000000000000000000000000001000000000
00100000000000000000000000000000000000000000000001101010000101
OR
00000000000000000000000000000000000000000000000111101110111001000
00000000000000000000000000000000000000000000000111111010011101
XOR
00000000000000000000000000000000000000000000000110101110111000000
00000000000000000000000000000000000000000000000110010000011000
NOT
11111111111111111111111111111111111111111111111101001000111111101111
11111111111111111111111111111111111111111111111000000101100010
1 in a
18
0
0

Дневник отладки

Проблем и ошибок при написании данной работы не возникло.

Недочёты

Выводы

В процессе выполнения работы я на практике познакомился с классами. Эта лабораторная является вводной по изучению объекто-ориентированной парадигмы. Классы удобны для упрощения написание кода для различных объемных программ, ипользующих различные типы данных, содержащие сразу несколько различных полей. Например, при необходимости можно реализовать новый тип данных - битовая строка, которая работает с двумя полями unsigned long long.

Исходный код:

BitString.h

```
#ifndef BITSTRING_H
#define BITSTRING_H
#include <iostream>

class BitString {
    public:
    BitString();
    BitString(std::istream &is);
    BitString(unsigned long long a, unsigned long long b);
    BitString(const BitString &other);
    unsigned long long get_one();
    unsigned long long get_two();
    friend std::ostream& operator<<(std::ostream& os,const BitString& a);
    int count();
    BitString shiftLeft(int n);
    BitString shiftRight(int n);
    bool include(BitString& other);
    bool operator <(BitString& other);
    bool operator >(BitString& other);
    bool operator ==(BitString& other);
    BitString operator &( BitString &other);
    BitString operator ||(BitString &other);
    BitString operator ^(BitString &other);
    BitString operator ~();
    BitString operator >>(int n) const;
    BitString operator<<(int n) const;
    ~BitString();
    private:
    unsigned long long one_, two_;
};

#endif // BITSTRING_H
```

BitString.cpp

```
#include <iostream>
#include "BitString.h"
#include <cmath>
using namespace std;
BitString::BitString(){
```

```
    one_ = 0;
    two_ = 0;
};
BitString::BitString(std::istream &is) {
    is >> one_;
    is >> two_;
};
BitString:: BitString(unsigned long long a, unsigned long long b) {
    one_ = a;
    two_ = b;
};
BitString::BitString(const BitString &other) {
    one_ = other.one_;
    two_ = other.two_;
};
unsigned long long BitString::get_one() {
    return one_;
}
unsigned long long BitString::get_two() {
    return two_;
}
BitString BitString:: operator &(BitString &other)  {
    BitString a(one_ & other.one_, two_ & other.two_);
    return a;
}
BitString BitString:: operator ||(BitString &other) {
    BitString a(one_ | other.one_, two_ | other.two_);
    return a;
}
BitString BitString::operator ^(BitString &other) {
    BitString a(one_ ^ other.one_, two_ ^ other.two_);
    return a;
}
BitString BitString::operator ~() {
    BitString a(~one_,~two_);
    return a;
}
BitString BitString::operator >>(int n) const {
    if(n > 128) {
        return BitString(0,0);
    }
    BitString bs(*this);
```

```
    unsigned long long b = pow(2, 63);
    unsigned long long a = 1;
    for(int i = 0; i < n; ++i){
        bs.two_ >>= 1;
        if(bs.one_ & 1){
            bs.two_ |= b;
        }
        bs.one_ >>= 1;
    }
    return bs;
}
BitString BitString::operator<<(int n) const{
    if(n > 128) {
        return BitString(0,0);
    }
    BitString bs(*this);
    unsigned long long a = pow(2, 63);
    for(int i = 0; i < n; ++i){
        bs.one_ <<= 1;
        if(bs.two_ & a) {
            bs.one_ |= 1;
            }
        bs.two_ <<= 1;
        }
        return bs;
    }
std::ostream& operator<<(std::ostream& os,const BitString& bs){
    int i = 63;
    while (i >= 0) {
        os << ((bs.one_ >> i) & 1);
        i--;
    }
    i = 63;
    while (i >= 0) {
        os << ((bs.two_ >> i) & 1);
        i--;
    }
    return os;
}
BitString BitString::shiftLeft(int n) {
    if(n > 128) {
        return BitString(0,0);
```

```
    }
    for(int i = 0; i < n; ++i){
        this->one_ <<= 1;
        if(this->two_ & (unsigned long long)pow(2, 63)) {
            this->one_ |= 1;
            }
        this->two_ <<= 1;
    }
    return *this;
}
BitString BitString::shiftRight(int n) {
     if(n > 128) {
        return BitString(0,0);
    }
    for(int i = 0; i < n; ++i){
        this->two_ >>= 1;
        if(this->one_ & 1){
            this->two_ |= (unsigned long long)pow(2, 63);
        }
        this->one_ >>= 1;
    }
    return *this;
}
int BitString::count() {
    int i = 63;
    int cnt = 0;
    while (i >= 0) {
        if ((one_ >> i) & 1) {
            cnt++;
        }
        i--;
    }
    i = 63;
    while (i >= 0) {
        if ((two_ >> i) & 1){
            cnt++;
        }
        i--;
    }
    return cnt;
}
bool BitString::operator <(BitString& a) {
```

```
    return this->count() < a.count();
}
bool BitString::operator >(BitString& a) {
    return this->count() > a.count();
}
bool BitString::operator ==(BitString& a) {
    return this->count() == a.count();
}
bool BitString::include(BitString& other) {
    BitString s = other&(*this);
    return s==other;
}
BitString::~BitString(){
}
```

main.cpp

```
#include <iostream>
#include "BitString.h"
#include <cmath>
using namespace std;
int main() {
    BitString a(23425, 32413);
    BitString b(cin);
    BitString c(678686, 345346);
    cout << "BitString a:" << endl << a << endl;
    cout << "BitString b:" << endl << b << endl;
    cout << "AND" << endl << (a&b) << endl;
    cout << "OR" << endl << (a||b) << endl;
    cout << "XOR" << endl << (a^b) << endl;
    cout << "NOT" << endl << ~a << endl;
    cout << "1 in a" << endl << a.count() << endl;
    cout << (b > a) << endl;
    cout << b.include(c) << endl;
}
```