**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 1**

Тема: Простые классы на языке С++

Студент: Тумаков Данила Владимирович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи
2. Описание программы
3. Листинг программы
4. Набор тестов
5. Вывод
6. Список литературы

## Постановка задачи

## 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

I. Класс BitString

a) Конструкторы

б) Геттеры и сеттеры

в) Функция печати битовой строки

г) Свойства класса

II. а) Перегрузка побитовых операторов конъюнкции, дизъюнкции, отрицания, сдвигов влево и вправо.

б) Перегрузка операторов сравнения

в) Функция подсчета количества единичных битов в строке

г) Функция проверки включения

III. Главная функция

а) Ввод данных

б) Проверка операторов

в) Проверка функций

д) Вывод результата

# Листинг программы.

/\*\*

  Тумаков М8О-206Б-19

  Создать класс BitString для работы с 96битовыми 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.

 \*/

**Ссылка на репозиторий:** https://github.com/Danila565/oop\_exercise\_01.

#include <iostream>

#include <string>

#include <bitset>

using namespace std;

typedef unsigned long long ull;

typedef unsigned int ui;

class bit\_str

{

    public:

    //конструктор класса по умолчанию;

    bit\_str(){

        up=0;

        down=0;

    }

    //конструктор класса;

    bit\_str(ull a, ui b){

        up=a;

        down=b;

    }

    //получить значение старшей строки(64бит)

    ull GetUp(){

        return up;

    }

    //получить значение младшей строки(32бит)

    ui GetDown(){

        return down;

    }

    //установить значение строки

    void Set(ull up, ui down) {

        this->up = up;

        this->down = down;

    }

    //перегрузка оператора & (И)

    bit\_str operator&(bit\_str b){

        bit\_str c;

        c.up = up & b.up;

        c.down = down & b.down;

        return c;

    }

    //перегрузка оператора &= (И)

    bit\_str operator&=(bit\_str b){

        up = up & b.up;

        down = down & b.down;

        return \*this;

    }

    //перегрузка оператора | (ИЛИ)

    bit\_str operator|(bit\_str b){

        bit\_str c;

        c.up = up | b.up;

        c.down = down | b.down;

        return c;

    }

    //перегрузка оператора |= (ИЛИ)

    bit\_str operator|=(bit\_str b){

        up = up | b.up;

        down = down | b.down;

        return \*this;

    }

    //перегрузка оператора ^

    bit\_str operator^(bit\_str b){

        bit\_str c;

        c.up = up ^ b.up;

        c.down = down ^ b.down;

        return c;

    }

    //перегрузка оператора ^=

    bit\_str operator^=(bit\_str b){

        up = up ^ b.up;

        down = down ^ b.down;

        return \*this;

    }

    //перегрузка оператора ~ (НЕ)

    bit\_str operator~(){

        up = ~up;

        down = ~down;

        return \*this;

    }

    //перегрузка оператора сдвига <<

    bit\_str operator<<(const int& num) {

        ui downL;

        for (int i = 0; i < num; i++) {

            downL= (down & (1<<31)) >> 31;

            up = (up<<1) | downL;

            down = ((down)<<1);

        }

        return \*this;

    }

    //перегрузка оператора сдвига <<=

    bit\_str operator<<=(const int& num) {

        return (\*this=\*this<<num);

    }

    //перегрузка оператора сдвига >>

    bit\_str operator>>(const int& num){

        ui up1;

        for (int i = 0; i < num; i++) {

            up1= up & 1;

            up = ((up)>>1);

            down = ((down)>>1) | (up1<<31);

        }

        return \*this;

    }

    //перегрузка оператора сдвига >>=

    bit\_str operator>>=(const int& num){

        return (\*this=\*this>>num);

    }

//перегрузка оператора сравнения >

    bool operator>(const bit\_str& b){

        if(up > b.up){

            return true;

        }

        else if(up < b.up){

            return false;

        }

        else if(down > b.down){

            return true;

        }

        else {

            return false;

        }

    }

    //перегрузка оператора сравнения <

    bool operator<(const bit\_str& b){

        if(up < b.up){

            return true;

        }

        else if(up > b.up){

            return false;

        }

        else if(down < b.down){

            return true;

        }

        else {

            return false;

        }

    }

    //перегрузка оператора сравнения >=

    bool operator>=(const bit\_str& b){

        if(up > b.up){

            return true;

        }

        else if(up < b.up){

            return false;

        }

        else if(down > b.down){

            return true;

        }

        else if(down < b.down){

            return false;

        }

        else{

            return true;

        }

    }

    //перегрузка оператора сравнения <=

    bool operator<=(const bit\_str& b){

        if(up < b.up){

            return true;

        }

        else if(up > b.up){

            return false;

        }

        else if(down < b.down){

            return true;

        }

        else if(down > b.down){

            return false;

        }

        else{

            return true;

        }

    }

    //перегрузка оператора сравнения ==

    bool operator==(const bit\_str& b){

        if((up == b.up) && (down == b.down)){

            return true;

        }

        else{

            return false;

        }

    }

    //вывод строки как 2 числа(ull и ui)

    void Print(){

        cout << "Up:"<< up << "\tDown" << down << endl;

    }

    //побитовый вывод строки

    void Print\_bit() {

        bitset<64> upB(up);

        bitset<32> downB(down);

        cout<<upB<<" | "<<downB<<endl;

    }

    //подсчет количество единичных битов

     int count\_1(){

        bitset<64> upB(up);

        bitset<32> downB(down);

        int res = upB.count() + downB.count();

        return res;

    }

    //проверка включения через единичные биты

    bool check\_inclusion(ull a){

        ull temp=a;

        bit\_str str;

        str.up=up;

        str.down=down;

        if((str.down == 0 || str.up == 0) && temp == 0){

            return true;

        }

        while(str.down!=0) {

            if(temp==0) {

                return true;

            }

            if((temp & 1) == (str.down & 1)) {

                temp=temp>>1;

                //(str.down)<<1;

            } else {

                temp=a;

                //(str.down)<<1;

            }

            str.down=(str.down)>>1;

        }

        while(str.up!=0) {

            if(temp==0) {

                return true;

            } else if((temp & 1) == (str.up & 1)) {

                temp=temp>>1;

                //(str.up)<<1;

            } else {

                temp=a;

                //(str.up)<<1;

            }

            (str.up)=(str.up)>>1;

        }

        return false;

    }

    private:

    ull up;   //unsigned long long старшая часть строки

    ui down;  //unsigned int младшая часть строки

};

int main() {

    ull up, num;

    ui down;

    cout<< INT32\_MAX<<endl;

    cout<< LONG\_LONG\_MAX<<endl;

    cout<<"Enter strings a values"<<endl;

    cin >> up >> down;

    bit\_str a(up, down),b,c;

    //ввод строк

    cout<<"Enter values"<<endl;

    cin >> up >> down;

    b.Set(up, down);

    c.Set(1<<31,1<<31);

    //вывод строки а

    cout<<"String a:"<< endl;

    a.Print();

    a.Print\_bit();

    //вывод количества единичных битов в а

    cout<<"count\_a: "<< a.count\_1() <<endl;

    //вывод строки b

    cout<<"String b:"<< endl;

    b.Print();

    b.Print\_bit();

    //проверка включения числа num

    cout<<"Enter num"<<endl;

    cin >> num;

    if(a.check\_inclusion(num)) {

        cout<<"There is inclusion"<<endl;

    }

    else {

        cout<<"There is no inclusion"<<endl;

    }

    //проверка операции &

    a=a&b;

    cout<<"a & b = ";

    a.Print\_bit();

    //проверка операции |

    cout<<"a | b = ";

    b |= a;

    b.Print\_bit();

    //побитовый вывод a и b

    a.Print\_bit();

    b.Print\_bit();

    //проверка операции ~

    cout<<"~b = ";

    b=~b;

    b.Print\_bit();

    //проверка сдвигов

    cout<<"Shift left on ... ";

    cin >> down;

    a<<=down;

    b=b<<down;

    a.Print\_bit();

    b.Print\_bit();

    cout<<"Shift right on ... ";

    cin >> down;

    a>>=down;

    b=b>>down;

    a.Print\_bit();

    b.Print\_bit();

    return 0;

}

# Набор тестов.

Тест №1:

0 0

0 0

0

0

0

Результат:

Enter strings a values

0 0

Enter strings b values

0 0

String a:

Up: 0 Down: 0

0000000000000000000000000000000000000000000000000000000000000000 | 00000000000000000000000000000000

count\_a: 0

String b:

Up: 0 Down: 0

0000000000000000000000000000000000000000000000000000000000000000 | 00000000000000000000000000000000

a equal b

Enter num

0

There is inclusion

a & b = 0000000000000000000000000000000000000000000000000000000000000000 | 00000000000000000000000000000000

a | b = 0000000000000000000000000000000000000000000000000000000000000000 | 00000000000000000000000000000000

~b = 1111111111111111111111111111111111111111111111111111111111111111 | 11111111111111111111111111111111

Shift left on ... 0

0000000000000000000000000000000000000000000000000000000000000000 | 00000000000000000000000000000000

1111111111111111111111111111111111111111111111111111111111111111 | 11111111111111111111111111111111

Shift right on ... 0

0000000000000000000000000000000000000000000000000000000000000000 | 00000000000000000000000000000000

1111111111111111111111111111111111111111111111111111111111111111 | 11111111111111111111111111111111

Тест №2:

1234567 7654321

2049 1025

5

3

4

Результат:

Enter strings a values

1234567 7654321

Enter strings b values

2049 1025

String a:

Up: 1234567 Down: 7654321

0000000000000000000000000000000000000000000100101101011010000111 | 00000000011101001100101110110001

count\_a: 24

String b:

Up: 2049 Down: 1025

0000000000000000000000000000000000000000000000000000100000000001 | 00000000000000000000010000000001

a more then b

Enter num

5

There is inclusion

a & b = 0000000000000000000000000000000000000000000000000000000000000001 | 00000000000000000000000000000001

a | b = 0000000000000000000000000000000000000000000000000000100000000001 | 00000000000000000000010000000001

~b = 1111111111111111111111111111111111111111111111111111011111111110 | 11111111111111111111101111111110

Shift left on ... 3

0000000000000000000000000000000000000000000000000000000000001000 | 00000000000000000000000000001000

1111111111111111111111111111111111111111111111111011111111110111 | 11111111111111111101111111110000

Shift right on ... 4

0000000000000000000000000000000000000000000000000000000000000000 | 10000000000000000000000000000000

0000111111111111111111111111111111111111111111111111101111111111 | 01111111111111111111110111111111

Тест №3:

9223372036854775807 2147483647

1 2147483647

5

4

4

Результат:

Enter strings a values

9223372036854775807 2147483647

Enter strings b values

1 2147483647

String a:

Up: 9223372036854775807 Down: 2147483647

0111111111111111111111111111111111111111111111111111111111111111 | 01111111111111111111111111111111

count\_a: 94

String b:

Up: 1 Down: 2147483647

0000000000000000000000000000000000000000000000000000000000000001 | 01111111111111111111111111111111

a more then b

Enter num

5

There is inclusion

a & b = 0000000000000000000000000000000000000000000000000000000000000001 | 01111111111111111111111111111111

a | b = 0000000000000000000000000000000000000000000000000000000000000001 | 01111111111111111111111111111111

~b = 1111111111111111111111111111111111111111111111111111111111111110 | 10000000000000000000000000000000

Shift left on ... 4

0000000000000000000000000000000000000000000000000000000000010111 | 11111111111111111111111111110000

1111111111111111111111111111111111111111111111111111111111101000 | 00000000000000000000000000000000

Shift right on ... 4

0000000000000000000000000000000000000000000000000000000000000001 | 01111111111111111111111111111111

0000111111111111111111111111111111111111111111111111111111111110 | 10000000000000000000000000000000

# Вывод

Данная лабораторная работа помогает изучить классы и их методы в С++. Выполнив её, можно научиться создавать собственный класс, а также приобрести навыки работы с перегрузками операторов.

# Список литературы

1. <https://www.youtube.com/playlist?list=PLQOaTSbfxUtBm7DxblJZShqBQnBAVzlXX>.
2. <https://code-live.ru/post/cpp-classes/>.
3. https://codelessons.ru/cplusplus/klassy-v-c-rukovodstvo-dlya-nachinayushhix.html.