Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа
по курсу «ООП»

Тема:
Операторы, литералы.

Студент:	Касимов М.М.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	6
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

bits.h:

```
//
// Created by magom on 17.09.2019.
//

#ifndef OOP_EXERCISE_02_BITS_H
#define OOP_EXERCISE_02_BITS_H

#include <iostream>
#include <string>
#include <vector>


class bit {
private:
    unsigned long long a;
    unsigned int b;
public:
    bit();

    bit(unsigned long long a1, unsigned int b1) {
        a = a1;
        b = b1;
    };

    void ShiftLeft(int i);

    void ShiftRight(int i);

    int Inclusion(const bit &other) const;

    void Get(unsigned long long i, unsigned int j);

    unsigned long long Hight() const;

    unsigned int Low() const;

    bit operator&(const bit &other) const;

    bit operator|(const bit &other) const;

    bit operator^(const bit &other) const;

    bit operator~() const;

    int operator<=(const bit &other) const;

    int operator>=(const bit &other) const;
```

```
    int operator==(const bit &other) const;

    int operator<(const bit &other) const;

    int operator>(const bit &other) const;

    friend std::ostream &operator<<(std::ostream &os, const bit &other);

    friend std::istream &operator>>(std::istream &is, bit &other);
};


#endif //OOP_EXERCISE_02_BITS_H
```

bits.cpp:

```
//
// Created by magom on 17.09.2019.
//

#include "bits.h"

int number(unsigned long long m) {
    int i = 0;
    while (m > 0) {
        i += m % 2;
        m = m / 2;
    }
    return i;
}

void to2(unsigned long long a, std::ostream &os, int *Count, bool flag) {
    if (a == 0)
        return;
    if (a % 2 == 1 && flag) {
        *Count += 1;
        flag = false;
    }
    to2(a / 2, os, Count, flag);
    std::cout << a % 2;
}

void Count0(unsigned long long a, std::ostream &os, int *Count, bool flag) {
    if (a == 0)
        return;
    Count0(a / 2, os, Count, flag);
    *Count+=1;
}

bit bit::operator&(const bit &other) const {
    unsigned long long temp_a = a & other.Hight();
    unsigned int temp_b = b & other.Low();
    bit temp(temp_a, temp_b);
    return temp;
```

```
}

bit bit::operator|(const bit &other) const {
    unsigned long long temp_a = a | other.Hight();
    unsigned int temp_b = b | other.Low();
    bit temp(temp_a, temp_b);
    return temp;
}

bit bit::operator^(const bit &other) const {
    unsigned long long temp_a = a ^other.Hight();
    unsigned int temp_b = b ^other.Low();
    bit temp(temp_a, temp_b);
    return temp;
}

bit bit::operator~() const {
    bit other(~a,~b);
    return other;
}

void bit::ShiftLeft(int i) {
    int k = 0;
    while (k < i) {
        if (b >= (1u << 31u)) {
            a = a << 1;
            ++a;
            b = b << 1;
            ++k;
        } else {
            a = a << 1;
            b = b << 1;
            ++k;
        }
    }
}

void bit::ShiftRight(int i) {
    unsigned int k = 0, f = (1u << 31u);
    while (k < i) {
        if (a % 2 == 1) {
            a = a >> 1;
            b = b >> 1;
            b = b | f;
            ++k;
        } else {
            a = a >> 1;
            b = b >> 1;
            ++k;
        }
    }
}
```

```
void bit::Get(unsigned long long i, unsigned int j) {
    a = i;
    b = j;
}

unsigned long long bit::Hight() const {
    return a;
}

unsigned int bit::Low() const {
    return b;
}

int bit::operator==(const bit &other) const {
    int i1 = number(a) + number(b);
    int i2 = number(other.a) + number(other.b);
    return i1 == i2;
}

int bit::operator<=(const bit &other) const {
    int i1 = number(a) + number(b);
    int i2 = number(other.a) + number(other.b);
    return i1 <= i2;

}

int bit::operator>=(const bit &other) const {
    int i1 = number(a) + number(b);
    int i2 = number(other.a) + number(other.b);
    return i1 >= i2;

}

int bit::operator<(const bit &other) const {
    int i1 = number(a) + number(b);
    int i2 = number(other.a) + number(other.b);
    return i1 < i2;

}

int bit::operator>(const bit &other) const {
    int i1 = number(a) + number(b);
    int i2 = number(other.a) + number(other.b);
    return i1 > i2;
}

int bit::Inclusion(const bit &other) const {
    if ((a & other.a) == a && (b & other.b) == b)
        return 1;
    else
        return 0;
```

```
}

bit::bit() {
    a = 0;
    b = 0;
}

std::ostream &operator<<(std::ostream &os, const bit &other) {
    bool flag = true;
    int aCount = 0, bCount = 0;
    os << other.a << " " << other.b << std::endl;
    Count0(other.a, os, &aCount, flag);
    for (int i = 0; i < 64 - aCount; ++i) {
        os << 0;
    }
    to2(other.a, os, &aCount, flag);
    os << " ";
    flag = true;
    Count0(other.b, os, &bCount, flag);
    for (int i = 0; i < 32 - bCount; ++i) {
        os << 0;
    }
    to2(other.b, os, &bCount, flag);
    os << std::endl;
    return os;
}

std::istream &operator>>(std::istream &is, bit &other) {
    std::string s;
    is >> s;
    int n = s.size();
    std::string t(n, '0');
    std::vector<int> v;
    while (s != t) {
        int d = 0;
        for (int i = 0; i < s.size(); i++) {
            d *= 10;
            d += (s[i] - 48);
            s[i] = char(48 + d / 2);
            d %= 2;
        }
        v.push_back(d);
    }

    unsigned int b_step = 1;
    for (int i = 0; i < 32 && i < v.size(); i++) {
        other.b += v[i] * b_step;
        b_step *= 2;
    }
    unsigned long long a_step = 1;
    for (int i = 32; i < v.size(); i++) {
        other.a += v[i] * a_step;
```

```
        a_step *= 2;
    }
    return is;
}
```
main.cpp:
```
#include <iostream>
#include <sstream>
#include "bits.h"

bit operator "" _sr(const char *st, size_t siz) {
    std::istringstream is(st);
    bit val;
    is >> val;
    return val;
}

int main() {
    bit a;
    std::cin>>a;
    std::cout<<a;
    bit b;
    std::cin>>b;
    std::cout<<b;
    a.ShiftLeft(1);
    bit f = a & b;
    std::cout<<f;
    f = a|b;
    std::cout<<f;
    f = ~a;
    std::cout<<f;
    return 0;
}
```
CmakeLists.txt:
```
cmake_minimum_required(VERSION 3.10.2)
project(oop_exercise_02)

set(CMAKE_CXX_STANDARD 14)

add_executable(oop_exercise_01 main.cpp bits.h bits.cpp)
```
test.sh:
```
executable=$1

for file in test_??.test
do
  $executable < $file > tmp
  if cmp tmp ${file%%.test}.result
  then
    echo Test "$file": SUCCESS
  else
```

```
    echo Test "$file": FAIL
  fi
  rm tmp
done
```

2. Ссылка на репозиторий на GitHub.

https://github.com/magomed2000kasimov/oop_exercise_02

3. Набор тестов.

test_01.test:

```
2147483648
8589934591
```

test_02.test:

```
1
0
```

4. Результаты выполнения тестов.

test_01.result:

```
0 2147483648
00000000000000000000000000000000000000000000000000000000000000000000000000000000
10000000000000000000000000000000
1 4294967295
00000000000000000000000000000000000000000000000000000000000000000000000000000001
11111111111111111111111111111111
1 0
00000000000000000000000000000000000000000000000000000000000000000000000000000001
00000000000000000000000000000000
1 4294967295
00000000000000000000000000000000000000000000000000000000000000000000000000000001
11111111111111111111111111111111
18446744073709551614 4294967295
11111111111111111111111111111111111111111111111111111111111111111111111111111110
11111111111111111111111111111111
```

test_02.result:

```
0 1
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000001
0 0
```

00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000
0 0
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000
0 2
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000010
18446744073709551615 4294967293
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111101

5. Объяснение результатов работы программы.

1) Функция **friend std::istream &operator>>(std::istream &is, bit &other)** является главной функцией в моей программе, именно она правильно «расскладывает» число в 96 бит. Для этого она использует вспомогательный вектор и с помощью простых операций переводит его на 1 и 0, а дальше первые 32 бита уходят в переменную типа uns int, остальные в uns. long.

2) Методы **void shiftLeft(int k)**, **void shiftRight(int k)** производят левый и правый битовый сдвиг. При левом сдвиге учитывается 32 бит в переменной тип uns. int, а при правом сдвиге учитывается первый бит переменной типа uns. long.

3) **friend std::ostream &operator<<(std::ostream &os, const bit &other)** выводит младшую и старшую переменную и битовое представление строки соответственно. Для этого используется операция деления на 2.

4) функции включения и равенства помещают двоичную строку в вектор и цикл считает количество единиц и совпадения.

5) Все остальные функции тривиальные и используют готовые битовые операции языка C++.

6. Вывод.

Выполняя данную лабораторную я получил опыт работы с простыми классами, с системой сборки Cmake, с системой контроля версий git, а также изучил основы работы с классами в C++. Создал класс, соответствующий варианту моего задания, реализовал его методы.

7. Литература.

1)лекции по ООП МАИ.
2) Г.Шилдт «C++».