Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 2: «Операторы, литералы»

Группа:	М8О-206Б-18, №27
Студент:	Шорохов Алексей Павлович
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	

Москва, 2019

Задание

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, заданное количество битов. Реализовать операцию вычисления количества единичных битов, операции сравнения по количеству единичных битов. Реализовать операцию проверки включения. Операции and, or, xor, not, >>,

виде перегрузки операторов. Необходимо реализовать пользовательский литерал для работы с константами типа BitString.

Адрес репозитория на GitHub

Код программы на C++

```
cmake_minimum_required(VERSION 3.2)

project(BitString)

add_executable(BitString
        Source.cpp
        BitString.cpp)

set_property(TARGET BitString PROPERTY CXX_STANDART 11)
```

BitString.cpp

```
#include "BitString.h"
#include <stdlib.h>
#include <iostream>
#include <string>
#include <vector>

BitString::BitString() {

    firstHalf = 0;
    secondHalf = 0;

}

BitString::BitString(const char * in) : BitString() {
    std::string str = std::string(in);
    std::string sec(str.size(), '0');
    std::vector<int> v;
```

```
    while (str != sec) {
        int a = 0;
        for (int i = 0; i < str.size(); i++) {
            a *= 10;
            a += str[i] - '0';
            str[i] = char('0' + a / 2);
            a %= 2;
        }
        v.push_back(a);
    }



    unsigned long long shs = 1;

    for (int i = 0; i < 64 && i < v.size(); i++) {
        secondHalf += v[i] * shs;
        shs *= 2;
    }

    unsigned long long fhs = 1;

    for (int i = 64; i < v.size(); i++) {
        firstHalf += v[i] * fhs;
        fhs *= 2;
    }

}

BitString::BitString(std::string str) : BitString() {
    std::string sec(str.size(), '0');
    std::vector<int> v;

    while (str != sec) {
        int a = 0;
        for (int i = 0; i < str.size(); i++) {
            a *= 10;
            a += str[i] - '0';
            str[i] = char('0' + a / 2);
            a %= 2;
        }
        v.push_back(a);
    }



    unsigned long long shs = 1;

    for (int i = 0; i < 64 && i < v.size(); i++) {
        secondHalf += v[i] * shs;
        shs *= 2;
```

```
    }

    unsigned long long fhs = 1;

    for (int i = 64; i < v.size(); i++) {
        firstHalf += v[i] * fhs;
        fhs *= 2;
    }
}

void BitString::input(std::istream &is) {
    std::string str;


    is >> str;

    std::string sec(str.size(), '0');
    std::vector<int> v;

    while (str != sec) {
        int a = 0;
        for (int i = 0; i < str.size(); i++) {
            a *= 10;
            a += str[i] - '0';
            str[i] = char('0' + a / 2);
            a %= 2;
        }
        v.push_back(a);
    }



    unsigned long long shs = 1;

    for (int i = 0; i < 64 && i < v.size(); i++) {
        secondHalf += v[i] * shs;
        shs *= 2;
    }

    unsigned long long fhs = 1;

    for (int i = 64; i < v.size(); i++) {
        firstHalf += v[i] * fhs;
        fhs *= 2;
    }
}

BitString BitString::operator ~ () const{
    BitString bs;

    bs.firstHalf = ~(firstHalf);
    bs.secondHalf = ~(secondHalf);
```

```
    return bs;
}


BitString BitString::operator & (const BitString &bs) const{

    BitString bs1;
    bs1.firstHalf = firstHalf & bs.firstHalf;
    bs1.secondHalf = secondHalf & bs.secondHalf;

    return bs1;
}

BitString BitString::operator | (const BitString &bs) const{
    BitString bs1;
    bs1.firstHalf = firstHalf | bs.firstHalf;
    bs1.secondHalf = secondHalf | bs.secondHalf;

    return bs1;
}

BitString BitString::operator ^ (const BitString &bs) const{

    BitString bs1;
    bs1.firstHalf = firstHalf ^ bs.firstHalf;
    bs1.secondHalf = secondHalf ^ bs.secondHalf;

    return bs1;
}

void BitString::operator << (unsigned long long size) {
    unsigned long long pow63 = 1;
    for (int i = 0; i < 63; i++) {
        pow63 *= 2;
    }

    for (int i = 0; i < size; i++) {                    //110100111 << 3 == 100111000
        firstHalf = firstHalf << 1;
        if (secondHalf >= pow63) {
            firstHalf += 1;
        }
        secondHalf = secondHalf << 1;
    }
}

void BitString::operator >> (unsigned long long size) {

    unsigned long long pow63 = 1;
    for (int i = 0; i < 63; i++) {
        pow63 *= 2;
    }
```

```
    for (int i = 0; i < size; i++) {
        secondHalf = secondHalf >> 1;
        if (firstHalf % 2 == 1) {
            secondHalf += pow63;
        }
        firstHalf = firstHalf >> 1;
    }


}

unsigned long long BitString::posBitNumber() const{

    BitString bs1;
    bs1.firstHalf = firstHalf;
    bs1.secondHalf = secondHalf;

    unsigned long long number = 0;

    while (bs1.firstHalf != 0) {
        if (bs1.firstHalf % 2 == 1) number++;
        bs1.firstHalf /= 2;
    }

    while (bs1.secondHalf != 0) {
        if (bs1.secondHalf % 2 == 1) number++;
        bs1.secondHalf /= 2;
    }

    return number;

}

bool BitString::operator == (BitString &bs) const{
    unsigned long long thisNumber = posBitNumber();
    unsigned long long bsNumber = bs.posBitNumber();

    return thisNumber == bsNumber;
}

bool BitString::operator > (BitString &bs) const{
    unsigned long long thisNumber = posBitNumber();
    unsigned long long bsNumber = bs.posBitNumber();

    return thisNumber > bsNumber;
}

bool BitString::operator < (BitString &bs) const{
    unsigned long long thisNumber = posBitNumber();
    unsigned long long bsNumber = bs.posBitNumber();

    return thisNumber < bsNumber;
```

```
}

bool BitString::isArgInThis(const BitString &bs) const{

    BitString ans;
    ans.firstHalf = firstHalf & bs.firstHalf;
    ans.secondHalf = secondHalf & bs.secondHalf;
    if (ans.firstHalf == bs.firstHalf && ans.secondHalf == bs.secondHalf) return true;
    else return false;

}

void BitString::print(std::ostream &os) const{

    BitString bs1;
    bs1.firstHalf = firstHalf;
    bs1.secondHalf = secondHalf;

    std::vector<int> v;

    while (bs1.firstHalf != 0) {
        v.push_back(bs1.firstHalf % 2);
        bs1.firstHalf /= 2;
    }

    for (int i = 0; i < 64 - v.size(); i++) {
        std::cout << 0;
    }

    for (int i = v.size() - 1; i >= 0; i--) {
        std::cout << v[i];
    }
    v.clear();

    std::cout << " ";

    while (bs1.secondHalf != 0) {
        v.push_back(bs1.secondHalf % 2);
        bs1.secondHalf /= 2;
    }
    for (int i = 0; i < 64 - v.size(); i++) {
        os << 0;
    }

    for (int i = v.size() - 1; i >= 0; i--) {
        os << v[i];
    }
    std::cout << '\n';

}

BitString operator ""_bs(const char * in) {
```

```
    return BitString(in);
}


std::istream & operator>> (std::istream& is, BitString& bs) {
    std::string a;
    is >> a;

    bs = BitString(a);
}

std::ostream & operator<< (std::ostream& os, const BitString& bs) {
    bs.print(os);
}
```

BitString.h

```
#ifndef __BitString_h__
#define __BitString_h__


#include <iostream>
#include <string>

class BitString
{
public:

    BitString();
    BitString(const char *);
    BitString(std::string);

    void input(std::istream &is);

    BitString operator ~ () const;
    BitString operator & (const BitString &bs) const;
    BitString operator | (const BitString &bs) const;
    BitString operator ^ (const BitString &bs) const;
    void operator >> (unsigned long long size);
    void operator << (unsigned long long size);
    bool operator == (BitString &bs) const;
    bool operator > (BitString &bs) const;
    bool operator < (BitString &bs) const;

    unsigned long long posBitNumber() const;
    int compPosBitNumber(const BitString &bs) const;

    bool isArgInThis(const BitString &bs) const;

    void print(std::ostream &os) const;

private:
    unsigned long long firstHalf;
```

```
    unsigned long long secondHalf;

};

BitString operator ""_bs(const char * in);

std::istream& operator>> (std::istream& is, BitString& bs);

std::ostream& operator<< (std::ostream& os, const BitString& bs);


#endif
```

Source.cpp

```
#include "BitString.h"

int main(int argc, char** argv) {

    BitString bs;

    std::cout << "Enter string\n";

    std::cin >> bs;

    BitString bs1;


    std::cout << "Enter string\n";
    std::cin >> bs1;

    std::cout << bs;
    std::cout << bs1;

    BitString bsTest = ~bs;

    std::cout << "not first number:\n";

    std::cout << bsTest;

    bsTest = bs & bs1;

    std::cout << "first and second:\n";

    std::cout << bsTest;

    bsTest = bs | bs1;

    std::cout << "first or second:\n";

    std::cout << bsTest;
```

```
    bsTest = bs ^ bs1;

    std::cout << "first xor second:\n";

    std::cout << bsTest;

    std::cout << "Positive Bit Number of First is " << bs.posBitNumber() << '\n';

    if (bs > bs1) {
        std::cout << "Bit Comparence of first and second shows that first is larger\n";
    } else if (bs < bs1) {
        std::cout << "Bit Comparence of first and second shows that second is larger\n";
    } else {
        std::cout << "Bit Comparence of first and second shows that they are equal\n";
    }

    std::cout << "Is second in first? : ";
    if (bs.isArgInThis(bs1)) std::cout << "YES\n";
    else std::cout << "NO\n";

    int shift;

    std::cout << "Enter number of bits to shift first number left and second right : ";
    std::cin >> shift;

    bs << shift;

    std::cout << "Shifted first left : \n";
    std::cout << bs;

    bs1 >> shift;

    std::cout << "Shifted second right : \n";
    std::cout << bs1;

    std::cout << "Literal using 123_bs : \n";
    std::cout << 123_bs;

    BitString bs2;

    std::cin >> bs2;
    std::cout << bs2;

    return 0;
}
```

file01.test

```
5000
458
3
```

file02.test

```
15
```

615
8
file03.test
15000000
16000000
60
Результаты тестов
1
Enter string
5000

Enter string
458
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000001001110001000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000111001010
not first number:
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111110110001110111
first and second:
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000110001000
first or second:
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000001001111001010
first xor second:
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000001001001000010
Positive Bit Number of First is 5
Bit Comparence of first and second shows that they are equal
Is second in first? : NO
Enter number of bits to shift first number left and second right : 3
Shifted first left :
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000001001110001000000
Shifted second right :
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000111001
Literal using 123_bs :
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000001111011
2
Enter string
15

Enter string
615
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000001111

00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000001001100111
not first number:
11111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111110000
first and second:
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000111
first or second:
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000001001101111
first xor second:
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000001001101000
Positive Bit Number of First is 4
Bit Comparence of first and second shows that second is larger
Is second in first? : NO
Enter number of bits to shift first number left and second right : 8
Shifted first left :
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000111100000000000
Shifted second right :
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000010
Literal using 123_bs :
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000001111011
3
Enter string
15000000

Enter string
16000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000111001001110000111000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000111101000010010000000000
not first number:
11111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111000110110001111000111111
first and second:
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000111001000010000000000000
first or second:
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000111101001110010111000000
first xor second:
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000100001100010111000000
Positive Bit Number of First is 10
Bit Comparence of first and second shows that first is larger
Is second in first? : NO

Enter number of bits to shift first number left and second right : 60
Shifted first left :
00000000000000000000000000000000000000000000000011100100111000011100
00000000000000000000000000000000000000000000000000000000000000000000
Shifted second right :

Объяснение результатов

Программа получает на вход две строки, содержащие числа, которые далее преобразует в 128-битовые строки и выполняет преобразования, требуемые задание лабораторной работы.

Вывод

Были изучены операторы и литералы, применены в лабораторной работе. Применение перегрузки операторов и создание пользовательских литералов упрощают понимание кода и делают его более лаконичным.