

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_STANDARD 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;
}

```

```

}

```

```

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 ~ () {
    BitString bs;

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

    return bs;
}

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

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

    return bs1;
}

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

    return bs1;
}

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

    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++) {
        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(){

    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;

}

```

//110100111 << 3 == 100111000

```

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

    return thisNumber == bsNumber;
}

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

    return thisNumber > bsNumber;
}

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

    return thisNumber < bsNumber;
}

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

    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) {

    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);
}

```

BitString.h

```

#ifndef __BitString_h__
#define __BitString_h__

#include <iostream>
#include <string>

class BitString
{
public:

    BitString();
    BitString(const char *);

    void input(std::istream &is);

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

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

```

```

    bool isArgInThis(const BitString &bs);

    void print(std::ostream &os);

private:
    unsigned long long firstHalf;
    unsigned long long secondHalf;

};

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

#endif

```

Source.cpp

```

#include "BitString.h"

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

    BitString bs;

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

    bs.input(std::cin);

    BitString bs1;

    std::cout << "Enter string\n";
    bs1.input(std::cin);

    bs.print(std::cout);
    bs1.print(std::cout);

    BitString bsTest = ~bs;

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

    bsTest.print(std::cout);

    bsTest = bs & bs1;

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

    bsTest.print(std::cout);

    bsTest = bs | bs1;

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

```



```

bsTest.print(std::cout);

bsTest = bs ^ bs1;

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

bsTest.print(std::cout);

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

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

std::cout << "Is second in first? : ";
bs.isArgInThis(bs1);

std::cout << '\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";
bs.print(std::cout);

bs1 >> shift;

std::cout << "Shifted second right : \n";
bs1.print(std::cout);

std::cout << "Literal using 123_bs : \n";
(123_bs).print(std::cout);

return 0;
}

```

file01.test
5000
458
3
file02.test

[illegible]

3

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