Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 2

Тема: Перегрузка операторов в C++

Студент: Бирюков В. В.
Группа: 80-207
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи. Вариант 6.

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.

Операции and, or, xor, not, сравнения (на равенство, больше и меньше) должны быть выполнены в виде перегрузки операторов.

Необходимо реализовать пользовательский литерал для работы с константами типа BitString.

2. Описание программы

В программе реализован класс BitString, представляющий 96-битную строку в виде старшей части (unsigned long long) и младшей (unsigned int).

При помощи перегрузки операторов реализованы следующие операции:

- побитовые AND (&), OR (|), XOR (^) - производит соответствующую операцию над двумя строками, применяется по отдельности к каждой части строки.
- побитовое NOT (~) - выполняет инверсию каждой части одной строки.
- битовый сдвиг влево (<<) - младшая и старшая части сдвигаются на нужное число разрядов, к старшей добавляется часть младшей при помощи побитового OR.
- битовый сдвиг вправо (<<) - младшая и старшая части сдвигаются на нужное число разрядов, к младшей добавляется часть старшей при помощи побитового OR.
- сравнения (>, <, ==, !=) - сравнение происходит по количеству единичных бит в строке.

Метод count вычисляет количество единичных бит в строке.

Пользовательский литерал для работы с константами типа BitString представлен в виде строки, содержащей шестнадцатеричные цифры и

суффикса _bs.

Также, для удобства перегружен ввод и вывод в стандартные потоки, строки вводятся и выводятся в шестнадцатеричном виде.

Программа получает на вход две строки (a и b) и два целых числа (n1 и n2) и выводит результаты следующих операций: побитовый OR, AND и XOR a и b, побитовое NOT a, сдвиг влево a на n1, сдвиг вправо b на n2, строку состоящую из второй половины a и первой половины b, количество единичных битов a и b, сравнение a и b, сравнение первых четырех цифр a и вторых четырех цифр b, сравнение a и 111111111111, сравнение b и ffff0000

3. Набор тестов

Тест 1:
```
a = 1
b = 0
n1 = 3
n2 = 1
```

Тест 2:
```
a = 1234abcd
b = 74abcd0100
n1 = 16
n2 = 20
```

Тест 3:
```
a = ddc5d3b93d3d3b82866abdf8
b = 0fc3f8657ae0a7171f008189
n1 = 56
n2 = 48
```

4. Результаты выполнения тестов

Тест 1:
```
a = 1
b = 0
n1 = 3
n2 = 1
a | b = 1
a & b = 0
a ^ b = 1
~a = fffffffffffffffffffffffe
a << n1 = 8
```

```
b >> n2 = 0
(a << 48) | (b >> 48) = 1000000000000
count(a) = 1
count(b) = 0
a == b = 0
a & ffff != b & ffff0000 = 1
a > 111111111111 = 0
b < ffff0000 = 1
```

Тест 2:

```
a = 1234abcd
b = 74abcd0100
n1 = 16
n2 = 20
a | b = 74bbfdabcd
a & b = 2040100
a ^ b = 74b9f9aacd
~a = ffffffffffffffffedcb5432
a << n1 = 1234abcd0000
b >> n2 = 74abc
(a << 48) | (b >> 48) = 1234abcd000000000000
count(a) = 15
count(b) = 15
a == b = 1
a & ffff != b & ffff0000 = 1
a > 111111111111 = 0
b < ffff0000 = 1
```

Тест 3:

```
a = ddc5d3b93d3d3b82866abdf8
b = 0fc3f8657ae0a7171f008189
n1 = 56
n2 = 48
a | b = dfc7fbfd7ffdbf979f6abdf9
a & b = dc1d0213820230206008188
a ^ b = d2062bdc47dd9c95996a3c71
~a = 223a2c46c2c2c47d79954207
a << n1 = 82866abdf800000000000000
b >> n2 = fc3f8657ae0
(a << 48) | (b >> 48) = 3b82866abdf80fc3f8657ae0
count(a) = 55
count(b) = 44
a == b = 0
a & ffff != b & ffff0000 = 1
```

```
a > 111111111111 = 1
b < ffff0000 = 0
```

5. Листинг программы

```
#include <iostream>
#include <string>

class BitString {
private:
    unsigned long long high;
    unsigned int low;

public:
    BitString(): high(0), low(0) {}
    BitString(unsigned long long h, unsigned int l) : high(h), low(l) {}
    BitString(const BitString& bs) : high(bs.high), low(bs.low) {}
    // BitString(const BitString&& bs) : high(bs.high), low(bs.low) {}

    unsigned int count() const;

    BitString& operator= (const BitString&);
    BitString operator~ ();

    friend BitString operator& (const BitString&, const BitString&);
    friend BitString operator| (const BitString&, const BitString&);
    friend BitString operator^ (const BitString&, const BitString&);
    friend BitString operator<< (const BitString&, const unsigned int);
    friend BitString operator>> (const BitString&, const unsigned int);
    friend bool operator< (const BitString&, const BitString&);
    friend bool operator> (const BitString&, const BitString&);
    friend bool operator== (const BitString&, const BitString&);
    friend bool operator!= (const BitString&, const BitString&);

    friend std::ostream& operator<< (std::ostream&, const BitString&);
    friend std::istream& operator>> (std::istream&, BitString&);
};

unsigned int BitString::count() const {
    unsigned int count = 0;
    unsigned int l = low;
    while (l != 0) {
       count += l & 1;
       l >>= 1;
    }
    unsigned long long h = high;
    while (h != 0) {
       count += h & 1;
       h >>= 1;
    }
    return count;
}
```

```
BitString& BitString::operator= (const BitString& bs){
    if (this == &bs) {
            return *this;
    }
    high = bs.high;
    low = bs.low;
    return *this;
}

BitString BitString::operator~ () {
    return BitString(~high, ~low);
}

BitString operator& (const BitString& lhs, const BitString& rhs) {
    return BitString(lhs.high & rhs.high, lhs.low & rhs.low);
}

BitString operator| (const BitString& lhs, const BitString& rhs) {
    return BitString(lhs.high | rhs.high, lhs.low | rhs.low);
}

BitString operator^ (const BitString& lhs, const BitString& rhs) {
    return BitString(lhs.high ^ rhs.high, lhs.low ^ rhs.low);
}

BitString operator<< (const BitString& bs, unsigned int count) {
    unsigned int low = (unsigned long long)bs.low << count;
    unsigned long long high = bs.high << count;
    if (count < 32) {
       high |= bs.low >> (32 - count);
    } else {
       high |= (unsigned long long)bs.low << (count - 32);
    }
    return BitString(high, low);
}

BitString operator>> (const BitString& bs, unsigned int count) {
    unsigned int low = (unsigned long long)bs.low >> count;
    unsigned long long high = bs.high >> count;
    if (count < 32) {
       low |= bs.high << (32 - count);
    } else {
       low |= bs.high >> (count - 32);
    }
    return BitString(high, low);
}

bool operator< (const BitString& lhs, const BitString& rhs) {
    return lhs.count() < rhs.count();
}

bool operator> (const BitString& lhs, const BitString& rhs) {
    return lhs.count() > rhs.count();
```

```
}

bool operator== (const BitString& lhs, const BitString& rhs) {
    return lhs.count() == rhs.count();
}

bool operator!= (const BitString& lhs, const BitString& rhs) {
    return lhs.count() != rhs.count();
}

BitString operator"" _bs(const char* str, size_t len) {
    if (len <= 8) {
       return BitString(0, (unsigned int)strtoul(str, nullptr, 16));
    } else {
       return BitString(strtoul(str + 9, nullptr, 16),
                        (unsigned int)strtoul(str, nullptr, 16));
    }
}

std::ostream& operator<< (std::ostream& os, const BitString& bs) {
    char str[25];
    if (bs.high != 0) {
       sprintf(str, "%llx%08x", bs.high, bs.low);
    } else {
       sprintf(str, "%x", bs.low);
    }
    os << str;
    return os;
}

std::istream& operator>> (std::istream& is, BitString& bs) {
    std::string str;
    str.reserve(24);
    is >> str;
    if (str.size() <= 8) {
       bs.high = 0;
       bs.low = (unsigned int)stoul(str, nullptr, 16);
    } else {
       bs.high = stoull(str.substr(0, str.size()-8), nullptr, 16);
       bs.low = stoull(str.substr(str.size()-8), nullptr, 16);
    }
    return is;
}

int main() {
    BitString a, b;
    int n1, n2;
    std::cout << "a = "; std::cin >> a;
    std::cout << "b = "; std::cin >> b;
    std::cout << "n1 = "; std::cin >> n1;
    std::cout << "n2 = "; std::cin >> n2;

    std::cout << "a | b = " << (a | b) << std::endl;
```

```
    std::cout << "a & b = " << (a & b) << std::endl;
    std::cout << "a ^ b = " << (a ^ b) << std::endl;
    std::cout << "~a = " << ~a << std::endl;
    std::cout << "a << n1 = " << (a << n1) << std::endl;
    std::cout << "b >> n2 = " << (b >> n2) << std::endl;
    std::cout << "(a << 48) | (b >> 48) = " << ((a << 48) | (b >> 48))
              << std::endl;

    std::cout << "count(a) = " << a.count() << std::endl;
    std::cout << "count(b) = " << b.count() << std::endl;

    std::cout << "a == b = " << (a == b) << std::endl;
    std::cout << "a & ffff != b & ffff0000 = "
              << ((a & "ffff"_bs) != (b & "f0f0f0f0"_bs)) << std::endl;
    std::cout << "a > 111111111111 = " << (a > "111111111111"_bs)
              << std::endl;
    std::cout << "b < ffff0000 = " << (b < "ffff0000"_bs) << std::endl;
}
```

6. Выводы

В ходе лабораторной я ознакомился с механизмом перегрузки операторов в языке C++, а также с созданием пользовательских литералов.

7. Литература

1 Справочник по языку C++ [Электронный ресурс]. URL: https://ru.cppreference.com