ММИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №2**

по курсу “Объектно-ориентированное программирование”

III семестр, 2021/22 учебный год

Выполнила студентка группы *М8О-208Б-20*

*Шатунова Юлия Викторовна*

Преподаватель: *Дорохов Евгений Павлович*

Москва, 2021

**Цель работы**

* Изучение основ работы с классами в C++.
* Перегрузка операций и создание литералов.

**Задание**Разработать программу на языке C++ согласно варианту задания. Действия над объектами реализовать в виде перегрузки операторов. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

**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 включения.

**Описание программы**

Исходный код разделён находится в 1 файле – lab02.cpp.

**Дневник отладки:**

Программа в отладке не нуждалась.

**Недочеты**

Недочеты не были обнаружены.

**Вывод**

В результате выполнения лабораторной работы №2 были изучены основы работы с классами в C++, перегрузка операций и создание пользовательских литералов. Данная лабораторная работа является модификацией лабораторной работы №1, но отличается от нее не сильно.

**Исходный код**

lab02.cpp

#include <iostream>

#include <string>

class BitString {

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) {}

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&);

private:

unsigned long long high;

unsigned int low;

};

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();

}

// user literal

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;

}