**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 С++

Студент: Попов Илья Павлович

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

# Постановка задачи

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 |, 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.

# Описание программы

## Методы класса BitsString:

* BitString() : lo(0), hi(0), ko\_vo\_ed(0) {} - создание BitString без аргументов
* BitString(ull h, ull l) : lo(l), hi(h) {}- создание BitString с заданными аргументами
* ull hight() const { return hi; } - геттер старших байтов
* ull low() const { return lo; } - геттер младших байтов
* void cmp(const BitString& b1) - сравнение BitString по кол-ву единичных битов
* void shiftRight(int a) - сдвиг BitString на а вправо
* void shiftLeft(int a)- сдвиг BitString на а влево
* void scan(const string& str) - считывание BitString из строки, которая подается на вход
* bool is\_include(const BitString& b1) - проверка содержания в BitString числа b1

## Перегружены операторы

1. >>, << - ввод и вывод BitString
2. & - операция and для работы с битами
3. | - операция or для работы с битами
4. ^ - операция xor для работы с битами
5. ~ - операция not для работы с битами

В программе присутствует проверка корректности введенных значений, как при создании класса, так и при обращениям к его методам.

# Набор тестов

## Тест № 1:*(заданы 2 числа, оба используют только младшие байты BitString)*

Число №1: 1010111101

Число №2: 10101

Числа для сдвига влево и вправо: 13 8

## Тест № 2:*(заданы 2 числа, одно использует старшие байты BitString, а другое - нет)*

Число№1: 1001010010101001010010101011111111111111111111111111111111111111

Число №2: 10101

Числа для сдвига влево и вправо: 80 3

## Тест № 3:*(заданы 2 числа, оба используют старшие байты BitString)*

Число№1: 100101001010100101001010101111111111111111111111111111111111111111111111111

Число№2: 10000000000000000000000000000000000000000000000000000000000000001

Числа для сдвига влево и вправо: 3 2

## Тесты № 4, 5:(демонстрируют устойчивость программы к неверно введенным параметрам)

Число №1: 10101abcd10101

Число №2: 10101 //для теста №4

Числа для сдвига влево и вправо: -3 -4 //для теста №5

# Результаты выполнения тестов

## Тест № 1

Введите свою Bitstring:

1010111101

Введите свою Bitstring:

10101

----------------------------------------------------------------------------

Ваши BitStrings:

1010111101

10101

----------------------------------------------------------------------------

Традиционные операции для работы с битами: and &, or |, xor ^, not ~:

a & b = 10101

a | b = 1010111101

a ^ b = 1010101000

~a = 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110101000010

~b = 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111101010

----------------------------------------------------------------------------

Введите числа для демонстрации сдвигов чисел(первое - для сдвига влево, второе - для сдвига вправо):

13 8

Сдвиги числа №1:

Влево: 10101111010000000000000

Вправо: 10

Сдвиги числа №2:

Влево: 101010000000000000

Вправо:

----------------------------------------------------------------------------

Сравним количество единичных битов в числах:

7 > 3

----------------------------------------------------------------------------

Проверим включение числа №2 в число №1:

1010111101

10101

INCLUDE!

Проверим включение числа №1 в число №2:

10101

1010111101

NOT INCLUDE!

## Тест № 2

Введите свою Bitstring:

1001010010101001010010101011111111111111111111111111111111111111

Введите свою Bitstring:

10101

----------------------------------------------------------------------------

Ваши BitStrings:

1001010010101001010010101011111111111111111111111111111111111111

10101

----------------------------------------------------------------------------

Традиционные операции для работы с битами: and &, or |, xor ^, not ~:

a & b = 10101

a | b = 1001010010101001010010101011111111111111111111111111111111111111

a ^ b = 1001010010101001010010101011111111111111111111111111111111101010

~a = 11111111111111111111111111111111111111111111111111111111111111110110101101010110101101010100000000000000000000000000000000000000

~b = 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111101010

----------------------------------------------------------------------------

Введите числа для демонстрации сдвигов чисел(первое - для сдвига влево, второе - для сдвига вправо):

80 3

Сдвиги числа №1:

Влево: 1001010101111111111111111111111111111111111111100000000000000000000000000000000000000000000000000000000000000000000000000000000

Вправо: 1001010010101001010010101011111111111111111111111111111111111

Сдвиги числа №2:

Влево: 1010100000000000000000000000000000000000000000000000000000000000000000000000000000000

Вправо: 10

----------------------------------------------------------------------------

Сравним количество единичных битов в числах:

49 > 3

----------------------------------------------------------------------------

Проверим включение числа №2 в число №1:

1001010010101001010010101011111111111111111111111111111111111111

10101

INCLUDE!

Проверим включение числа №1 в число №2:

10101

1001010010101001010010101011111111111111111111111111111111111111

NOT INCLUDE!

## Тест № 3

Введите свою Bitstring:

100101001010100101001010101111111111111111111111111111111111111111111111111

Введите свою Bitstring:

10000000000000000000000000000000000000000000000000000000000000001

----------------------------------------------------------------------------

Ваши BitStrings:

100101001010100101001010101111111111111111111111111111111111111111111111111

10000000000000000000000000000000000000000000000000000000000000001

----------------------------------------------------------------------------

Традиционные операции для работы с битами: and &, or |, xor ^, not ~:

a & b = 10000000000000000000000000000000000000000000000000000000000000001

a | b = 100101001010100101001010101111111111111111111111111111111111111111111111111

a ^ b = 100101001000100101001010101111111111111111111111111111111111111111111111110

~a = 11111111111111111111111111111111111111111111111111111011010110101011010110101010000000000000000000000000000000000000000000000000

~b = 11111111111111111111111111111111111111111111111111111111111111101111111111111111111111111111111111111111111111111111111111111110

----------------------------------------------------------------------------

Введите числа для демонстрации сдвигов чисел(первое - для сдвига влево, второе - для сдвига вправо):

3 2

Сдвиги числа №1:

Влево: 100101001010100101001010101111111111111111111111111111111111111111111111111000

Вправо: 1001010010101001010010101011111111111111111111111111111111111111111111111

Сдвиги числа №2:

Влево: 10000000000000000000000000000000000000000000000000000000000000001000

Вправо: 100000000000000000000000000000000000000000000000000000000000000

----------------------------------------------------------------------------

Сравним количество единичных битов в числах:

60 > 2

----------------------------------------------------------------------------

Проверим включение числа №2 в число №1:

10010100101100101001010101111111111111111111111111111111111111111111111111

1000000000000000000000000000000000000000000000000000000000000001

NOT INCLUDE!

Проверим включение числа №1 в число №2:

1000000000000000000000000000000000000000000000000000000000000001

10010100101100101001010101111111111111111111111111111111111111111111111111

NOT INCLUDE!

## Тест № 4

Введите свою Bitstring:

10101abcd10101

Ошибка! Некорректный ввод.

## Тест № 5

Введите числа для демонстрации сдвигов чисел(первое - для сдвига влево, второе - для сдвига вправо):

-3 -4

Сдвиги числа №1:

Ошибка! Сдвиг не может производиться на отрицательное число!

# Листинг программы

/\*

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 |, 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.

\*/

#include <iostream>

#include <cmath>

#include <string>

using namespace std;

using ull = unsigned long long;

string dec\_to\_two(const ull a);

class BitString {

private:

ull lo;//младшие байты числа

ull hi;//старшие байты числа

int ko\_vo\_ed;//кол-во единичных битов в числе

public:

BitString() : lo(0), hi(0), ko\_vo\_ed(0) {}

BitString(ull h, ull l) : lo(l), hi(h) {}

ull hight() const { return hi; }

ull low() const { return lo; }

void cmp(const BitString& b1) {//сравнение по кол-ву единичных битов

cout << ko\_vo\_ed;

if (ko\_vo\_ed > b1.ko\_vo\_ed) {

cout << " > ";

}

else if (ko\_vo\_ed = b1.ko\_vo\_ed) {

cout << " = ";

}

else {

cout << " < ";

}

cout << b1.ko\_vo\_ed;

}

/////////////////////////////////////////////////////////////////////////////////

void shiftRight(int a) {//Сдавиг вправо на заданное количество битов

if (a < 0) {

cout << "Ошибка! Сдвиг не может производиться на отрицательное число!" << endl;

exit(3);

}

if (a < 64) {

lo = lo >> a;

/\*\*/

ull f1 = (hi << (64 - a));

lo = lo | f1;

hi = hi >> a;

}

else if (a >= 64 && a < 128) {

lo = 0;

ull f1 = hi >> (a - 64);

lo = lo | f1;

hi = hi >> (128 - a);

}

else {//случай, когда сдвиг > 128, тогда все число сдвигается и остается 0

lo = 0;

hi = 0;

}

}

void shiftLeft(int a) {//Сдавиг влево на заданное количество битов

if (a < 0) {

cout << "Ошибка! Сдвиг не может производиться на отрицательное число!" << endl;

exit(3);

}

if (a < 64) {

hi = hi << a;

ull f2 = lo >> (64 - a);

lo = lo << a;

hi = hi | f2;

}

else if (a >= 64 && a < 128) {

hi = hi << a;

ull f2 = lo;

f2 = f2 << (a - 64);

lo = 0;

hi = hi | f2;

}

else if (a >= 128) {//случай, когда сдвиг > 128, тогда все число сдвигается и остается 0

hi = 0;

lo = 0;

}

}

/////////////////////////////////////////////////////////////////////////////////

void scan(const string& str) {//считываем из строки два unsigned long long числа

ull cur\_hi = 0;

ull cur\_lo = 0;

int count\_step = 0;

if(str.size() > 64) {//случай, когда надо задействовать старшие байты числа

for (int i = 0; i < str.size(); i++) {

if ((str[i] - '0') == 1) {//операция вычисления количества единичных битов

ko\_vo\_ed++;

}

if (i < str.size() - 64) {//условие, позволяющее отличить старшие биты от младших

cur\_hi = (cur\_hi << 1) + (str[i] - '0');

if (cur\_hi == 0) {//условие, позволяющее не потерять нулевые биты(для чисел, старшие байты которых начинаются с 0)

count\_step++;

}

}

else {

cur\_lo = (cur\_lo << 1) + (str[i] - '0');

}

}

}

else {//случай, когда не надо задействовать старшие байты числа

for (int i = 0; i < str.size(); i++) {

if ((str[i] - '0') == 1) {

ko\_vo\_ed++;

}

cur\_lo = (cur\_lo << 1) + (str[i] - '0');

}

}

hi = cur\_hi;

lo = cur\_lo;

if (count\_step) {

hi \*= pow(2, count\_step);

}

}

bool is\_include(const BitString& b1) {//операция проверки включения

string s, p;

if (hi){

s = dec\_to\_two(hi);

if (dec\_to\_two(lo).length() < 64) {

string str;

for (int i = dec\_to\_two(lo).length(); i < 64; i++) {

str.push\_back('0');

}

s = s + str;

}

s = s + dec\_to\_two(lo);

}

else { s = dec\_to\_two(lo); }

if (b1.hight()) {

p = dec\_to\_two(b1.hight());

if (dec\_to\_two(b1.low()).length() < 64) {

string str;

for (int i = dec\_to\_two(b1.low()).length(); i < 64; i++) {

str.push\_back('0');

}

p = p + str;

}

p = p + dec\_to\_two(b1.low());

}

else { p = dec\_to\_two(b1.low()); }

cout << s << endl << p << endl;

int i, j;

for (i = 0; i < s.length(); i++) {

for (j = 1; j < p.length(); j++) {

if (s[i + j] != p[j]) {

break;

}

}

if (j == p.length()) {

cout << "INCLUDE!\n";

return true;

}

}

cout << "NOT INCLUDE!\n";

}

friend istream& operator>> (istream& in, BitString& b);

};

/////////////////////////////////////////////////////////////////////////////////

bool is\_bin\_number(const string& s) {//проверка входных данных

bool otr = false;

for (int i = 0; i < s.length(); ++i) {

if (s[i] < '0' || s[i] > '1') {

return false;

}

}

return true;

}

istream& operator>> (istream& in, BitString& b) {//перегруженный оператор для ввода

cout << "\nВведите свою Bitstring:\n";

string str;

cin >> str;

if (str.length() > 128) {

cout << "Ошибка! Число слишком большое!" << endl;

exit(1);

}

if (!is\_bin\_number(str)) {

cout << "Ошибка! Некорректный ввод." << endl;

exit(2);

}

b.scan(str);

return in;

}

string dec\_to\_two(const ull a) {//числа в полях ull представлены в 10-ичном виде, чтобы сделать их читаемыми юзаем эту функцию

string str;

ull cur\_a = a;

while (cur\_a > 0) {

str.push\_back((cur\_a % 2) + '0');

cur\_a /= 2;

}

string res;

for (int i = str.length(); i >= 0; i--) {

res.push\_back(str[i]);

}

return res;

}

ostream& operator<< (ostream& out, const BitString& b) {//перегруженный оператор для выввода

//cout << b.hight() << " " << b.low() << endl;

if (b.hight()){//если у числа есть старшие байты, а младшие байты не заполнены до конца(их не 64), то дозаполняем вывод нулями

cout << dec\_to\_two(b.hight());// << " ";

int length = dec\_to\_two(b.low()).length();

while (length <= 64) {

cout << "0";

length++;

}

//cout << " " << dec\_to\_two(b.low()) << endl << endl;

cout << dec\_to\_two(b.low()) << endl << endl;

}

else {

cout << dec\_to\_two(b.low()) << endl << endl;

}

return out;

}

/////////////////////////////////////////////////////////////////////////////////

//традиционные операции для работы с битами: and &, or |, xor ^, not ~

BitString operator & (const BitString& b1, const BitString& b2) {

ull res\_hi = b1.hight() & b2.hight();

ull res\_lo = b1.low() & b2.low();

BitString res(res\_hi, res\_lo);

return res;

}

BitString operator | (const BitString& b1, const BitString& b2) {

ull res\_hi = b1.hight() | b2.hight();

ull res\_lo = b1.low() | b2.low();

BitString res(res\_hi, res\_lo);

return res;

}

BitString operator ^ (const BitString& b1, const BitString& b2) {

ull res\_hi = b1.hight() ^ b2.hight();

ull res\_lo = b1.low() ^ b2.low();

BitString res(res\_hi, res\_lo);

return res;

}

BitString operator ~ (const BitString& b1) {

BitString res(~b1.hight(), ~b1.low());

return res;

}

/////////////////////////////////////////////////////////////////////////////////

int main() {

setlocale(LC\_ALL, "rus");

BitString a, b;

cin >> a >> b;

cout << "----------------------------------------------------------------------------\n";

cout << "Ваши BitStrings:\n";

cout << a << b;

/\*\*/

cout << "----------------------------------------------------------------------------\n";

cout << "Традиционные операции для работы с битами: and &, or |, xor ^, not ~:\n";

cout << "a & b = " << (a & b);

cout << "a | b = " << (a | b);

cout << "a ^ b = " << (a ^ b);

cout << "~a = " << ~a;

cout << "~b = " << ~b;

cout << "----------------------------------------------------------------------------\n";

cout << "Введите числа для демонстрации сдвигов чисел(первое - для сдвига влево, второе - для сдвига вправо):\n";

int num1, num2;

cin >> num1 >> num2;

cout << "\nСдвиги числа №1:\n";

BitString cur\_la = a;

BitString cur\_ra = a;

cur\_la.shiftLeft(num1);

cur\_ra.shiftRight(num2);

cout << "Влево: " << cur\_la;

cout << "Вправо: " << cur\_ra;

cout << "Сдвиги числа №2:\n";

BitString cur\_lb = b;

BitString cur\_rb = b;

cur\_lb.shiftLeft(num1);

cur\_rb.shiftRight(num2);

cout << "Влево: " << cur\_lb;

cout << "Вправо: " << cur\_rb;

cout << "----------------------------------------------------------------------------\n";

cout << "Сравним количество единичных битов в числах:\n\n";

a.cmp(b);

cout << "\n\n----------------------------------------------------------------------------\n";

cout << "\nПроверим включение числа №2 в число №1:\n";

a.is\_include(b);

cout << "\nПроверим включение числа №1 в число №2:\n";

b.is\_include(a);

}

//Набор пробных чисел:

//100101001010100101001010101111111111111111111111111111111111111111111111111 --- длина 75

//1001010010101001010010101011111111111111111111111111111111111111 --- длина 64

//1111111111111111111111111111111111111111111111111111111111111111 --- старшие байты числа не используются

//10000000000000000000000000000000000000000000000000000000000000001 --- старшие байты числа используются

//11111100000100000000111111111111111111111100000000000000011111111111111111

# Выводы:

В ходе этой лабораторной работы я познакомился с таким понятием, как перегрузка операторов. Это очень удобный способ вызова функций который сильно упрощает написание кода, ведь повышает его читаемость, и ошибки быстрее попадаются на глаза. Ведь, как известно, код читается намного чаще, чем пишется.

Также стоит принимать во внимание, что возможность перегрузки есть только для пользовательских типов/классов.

# ЛИТЕРАТУРА

<https://ravesli.com/urok-130-vvedenie-v-peregruzku-operatorov/>

Введение в перегрузку операторов в с++ / Ravesli