Лабораторная работа №3

**Хашимов Амир Азизович**

**М8О-207Б-20**

**Вариант - 27**

[OOPLabs/lab01 at main · isAmirKhashimov/OOPLabs (github.com)](https://github.com/isAmirKhashimov/OOPLabs/tree/main/lab01)

# Задание

* + Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод. Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop\_exercise\_01 (в случае использования Windows oop\_exercise\_01.exe)
  + 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.

# Структура проекта и алгоритм программы:

// CMakeLists

cmake\_minimum\_required(VERSION 3.5)

project (oop\_exercise\_01)

set(SOURCES

bitString.cpp

main.cpp

)

add\_executable(oop\_exercise\_01 ${SOURCES})

target\_include\_directories(oop\_exercise\_01

PRIVATE

${PROJECT\_SOURCE\_DIR}

)

// bitString.h

#pragma once

#define ull unsigned long long

class BitString

{

private:

ull left;

ull right;

public:

BitString(ull left, ull right);

void Print();

int CountOnes();

friend BitString operator ^ (const BitString left, const BitString right);

friend BitString operator & (const BitString left, const BitString right);

friend BitString operator | (const BitString left, const BitString right);

friend BitString operator ~ (const BitString bits);

friend BitString operator << (const BitString bits, int count);

friend BitString operator >> (const BitString left, int count);

friend bool operator < (BitString bits1, BitString bits2);

friend bool operator > (BitString bits1, BitString bits2);

friend bool operator == (BitString bits1, BitString bits2);

friend bool operator != (BitString bits1, BitString bits2);

};

// bitString.cpp

#include "bitString.h"

#include <iostream>

#include <bitset>

#include <climits>

BitString operator^(const BitString bits1, const BitString bits2)

{

return BitString(bits1.left ^ bits2.left, bits1.right ^ bits2.right);

}

BitString operator&(const BitString bits1, const BitString bits2)

{

return BitString(bits1.left & bits2.left, bits1.right & bits2.right);

}

BitString operator|(const BitString bits1, const BitString bits2)

{

return BitString(bits1.left | bits2.left, bits1.right | bits2.right);

}

BitString operator~(const BitString bits1)

{

return BitString(~bits1.left, ~bits1.right);

}

BitString operator<<(const BitString bits, int count)

{

return BitString((bits.left << count) | (bits.right >> (64 - count)), bits.right << count);

}

BitString operator>>(const BitString bits, int count)

{

return BitString(bits.left >> count, (bits.right >> count) | (bits.left << (64 - count)));

}

bool operator<(BitString bits1, BitString bits2)

{

return bits1.CountOnes() < bits2.CountOnes();

}

bool operator>(BitString bits1, BitString bits2)

{

return bits1.CountOnes() > bits2.CountOnes();

}

bool operator==(BitString bits1, BitString bits2)

{

return bits1.CountOnes() == bits2.CountOnes();

}

bool operator!= (BitString bits1, BitString bits2)

{

return bits1.CountOnes() != bits2.CountOnes();

}

BitString::BitString(ull left, ull right)

{

this->left = left;

this->right = right;

}

void BitString::Print()

{

std::cout << std::bitset<sizeof(left)\* CHAR\_BIT>(left) << std::bitset<sizeof(left)\* CHAR\_BIT>(right) << " " << "Count of ones: " << CountOnes() << std::endl;

}

int BitString::CountOnes()

{

ull it = 1;

int count = 0;

for (int i = 0; i < 64; i++)

{

count += (left & it) ? 1: 0;

count += (right & it) ? 1 : 0;

it <<= 1;

}

return count;

}

// main.cpp

#include <iostream>

#include "bitString.h"

using namespace std;

int main()

{

ull x, y;

cin >> x >> y;

BitString a = BitString(x, y);

cin >> x >> y;

BitString b = BitString(x, y);

BitString c = BitString(0, 0);

cout << "a = ";

a.Print();

cout << "b = ";

b.Print();

c = a & b;

cout << "a and b = ";

c.Print();

c = a | b;

cout << "a or b = ";

c.Print();

c = a ^ b;

cout << "a xor b = ";

c.Print();

c = ~a;

cout << "not a = ";

c.Print();

cout << "a << 1 = ";

c = a << 1;

c.Print();

cout << "b >> 2 = ";

c = a >> 2;

c.Print();

}

// test\_01.txt

123123123 1231232321 424324342 4234324324

// test\_02.txt

0 1 18446744073709551614 18446744073709551614

Сложных алгоритмов в данной лабораторной работе нет, просто создал класс, переопределил для него необходимые операторы. Результаты тестов показать увы не получится, потому что придется создавать проект заново (старый потерялся)