**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 1**

Тема: Простые классы на языке С++

Студент: Шубин Григорий Сергеевич

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата: 27.09.2020

Оценка:

Москва, 2020

Репозиторий Github: <https://github.com/Garigoriy/oop_exercise_01>

1. Постановка задачи

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.

1. Описание программы

Программа представлена в виде текстового интерфейса, с помощью которого можно ввести данные и проверить каждую из операций данных в условии.

1. Набор тестов
2. 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000110011
3. 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000101001
4. 000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001111000
5. 110010000011111110101001001001101010111011011011101001111110010000000001010001101100000010010110
6. 001111100010101100011110001011101000100011111111111010000010010101010111001000010100101100001101
7. Результаты выполнения тестов

xor 1) и 2):

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011010

and 1) и 2):

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100001

or 1) и 2):

000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000111011

not 1):

111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111001100

Сдвиг 4) влево на 20:

100100100110101011101101101110100111111001000000000101000110110000001001011000000000000000000000

Сдвиг 5) вправо на 20:

000000000000000000000011111000101011000111100010111010001000111111111110100000100101010101110010

Подсчитать количество единиц в 4):

Result:46

Сравнить количество единиц. 4) > 5):

False

Содержится ли 3) в 5):

True

1. Листинг программы

#include<iostream>

#include<bitset>

#include<fstream>

#include<string>

using namespace std;

class BitString {

private:

unsigned long long old;

unsigned int jun;

public:

BitString(){};

BitString(unsigned long long lhs, unsigned int rhs) : old(lhs), jun(rhs) {}

unsigned long long Getold(){return old;}

unsigned int Getjun() {return jun;}

friend int Countbin(const BitString);

friend bool Inclusion(const BitString lhs, const BitString rhs);

friend BitString Compare(const string data);

friend BitString operator^(const BitString lhs, const BitString rhs);

friend BitString operator&(const BitString lhs, const BitString rhs);

friend BitString operator|(const BitString lhs, const BitString rhs);

friend BitString operator~(const BitString tmp);

friend BitString operator<<(const BitString tmp, int shift);

friend BitString operator>>(const BitString tmp, int shift);

friend bool operator>(const BitString lhs, const BitString rhs);

friend bool operator<(const BitString lhs, const BitString rhs);

};

BitString operator ^(const BitString lhs, const BitString rhs) {

return BitString(lhs.old^rhs.old, lhs.jun^rhs.jun);

}

BitString operator &(const BitString lhs, const BitString rhs) {

return BitString(lhs.old&rhs.old, lhs.jun&rhs.jun);

}

BitString operator |(const BitString lhs, const BitString rhs) {

return BitString(lhs.old|rhs.old, lhs.jun|rhs.jun);

}

BitString operator ~(const BitString tmp) {

return BitString(~tmp.old, ~tmp.jun);

}

BitString operator<<(const BitString tmp, int shift){

BitString a = tmp;

for (int i = 0; i < shift; ++i) {

a.old = a.old<<1;

if ((a.jun >> 31)&1) a.old=a.old|1;

a.jun = a.jun<<1;

}

return BitString(a.old, a.jun);

}

BitString operator>>(const BitString tmp, int shift){

BitString a = tmp;

for (int i = 0; i < shift; ++i) {

a.jun = a.jun>>1;

if (a.old&1) a.jun = a.jun|((a.jun|1) << 31);

a.old = a.old>>1;

}

return BitString(a.old, a.jun);

}

int Countbin(const BitString tmp) {

BitString a = tmp;

int res = 0;

while (a.old || a.jun) {

if (a.old&1) ++res;

if (a.jun&1) ++res;

a.old = a.old >> 1;

a.jun = a.jun >> 1;

}

return res;

}

bool operator>(const BitString lhs, const BitString rhs){

return Countbin(lhs)>Countbin(rhs);

}

bool operator<(const BitString lhs, const BitString rhs){

return Countbin(lhs)<Countbin(rhs);

}

bool Inclusion(const BitString lhs, const BitString rhs){

BitString a = rhs;

for (int i = 0; i < 95; ++i) {

if (((lhs.old&a.old)||(a.old == lhs.old))&&((lhs.jun&a.jun)||(a.jun == lhs.jun))) return true;

a = a >> 1;

}

return false;

}

BitString Compare(const string data){

unsigned long long lhs = 0, l = 0;

unsigned int rhs = 0, r = 0;

int j = 0;

for (int i = 63; i >=0; --i) {

if (data[j] == '1') {

lhs = lhs | ((l|1)<<i);

}

++j;

}

for (int i = 31; i >=0; --i) {

if (data[j] == '1') {

rhs = rhs | ((r|1)<<i);

}

++j;

}

return BitString(lhs, rhs);

}

void Interface() {

cout << "Select an option:" << '\n';

cout << "1.Add string" << '\n';

cout << "2.Show all strings" << '\n';

cout << "3.Operation XOR" << '\n';

cout << "4.Operation AND" << '\n';

cout << "5.Operation OR" << '\n';

cout << "6.Operation NOT" << '\n';

cout << "7.Bit Shift Left" << '\n';

cout << "8.Bit Shift Right" << '\n';

cout << "9.Count 1 in string" << '\n';

cout << "10.Compare strings" << '\n';

cout << "11.Inclusion of string " << '\n';

cout << "12.Exit" << '\n' << '\n';

}

int main() {

BitString arr[255];

int arg, j = 0;

bool flag = false;

while (true) {

Interface();

unsigned long long left;

unsigned int right;

int lhs, rhs, cache, cnt;

BitString tmp;

string line;

char simv;

ifstream in("input.txt");

cin >> arg;

cout << '\n';

switch (arg) {

case 1:

cout << "Add string with 1)keyboard or 2)file?" << '\n';

cin >> cache;

if (cache == 2) {

cout << "Enter file name" << '\n';

cin >> line;

ifstream in(line);

while (getline(in, line)) {

arr[j] = Compare(line);

++j;

}

} else {

cin >> line;

arr[j] = Compare(line);

++j;

cout << "Added successful!" << '\n';

}

break;

case 2:

cout << "Available strings:" << '\n';

for (int i = 0; i < j; ++i) {

cout << i+1 << ')' << bitset<sizeof(arr[i].Getold()) \* CHAR\_BIT>(arr[i].Getold()) << bitset<sizeof(arr[i].Getjun()) \* CHAR\_BIT>(arr[i].Getjun()) << '\n';

}

cout << '\n';

break;

case 3:

cout << "Enter 2 strings numbers to XOR" << '\n';

cin >> lhs >> rhs;

--lhs; --rhs;

tmp = arr[lhs]^arr[rhs];

cout << "Result:" << '\n';

cout << bitset<sizeof(tmp.Getold()) \* CHAR\_BIT>(tmp.Getold()) << bitset<sizeof(tmp.Getjun()) \* CHAR\_BIT>(tmp.Getjun()) << '\n';

break;

case 4:

cout << "Enter 2 strings numbers to AND" << '\n';

cin >> lhs >> rhs;

--lhs; --rhs;

tmp = arr[lhs]&arr[rhs];

cout << "Result:" << '\n';

cout << bitset<sizeof(tmp.Getold()) \* CHAR\_BIT>(tmp.Getold()) << bitset<sizeof(tmp.Getjun()) \* CHAR\_BIT>(tmp.Getjun()) << '\n';

break;

case 5:

cout << "Enter 2 strings numbers to OR" << '\n';

cin >> lhs >> rhs;

--lhs; --rhs;

tmp = arr[lhs]|arr[rhs];

cout << "Result:" << '\n';

cout << bitset<sizeof(tmp.Getold()) \* CHAR\_BIT>(tmp.Getold()) << bitset<sizeof(tmp.Getjun()) \* CHAR\_BIT>(tmp.Getjun()) << '\n';

break;

case 6:

cout << "Enter string number to NOT" << '\n';

cin >> cache;

--cache;

tmp = ~arr[cache];

cout << "Result:" << '\n';

cout << bitset<sizeof(tmp.Getold()) \* CHAR\_BIT>(tmp.Getold()) << bitset<sizeof(tmp.Getjun()) \* CHAR\_BIT>(tmp.Getjun()) << '\n';

break;

case 7:

cout << "Enter string number to leftShift and number of shifts" << '\n';

cin >> cache >> cnt;

--cache;

tmp = arr[cache] << cnt;

cout << bitset<sizeof(tmp.Getold()) \* CHAR\_BIT>(tmp.Getold()) << bitset<sizeof(tmp.Getjun()) \* CHAR\_BIT>(tmp.Getjun()) << '\n';

break;

case 8:

cout << "Enter string number to rightShift and number of shifts" << '\n';

cin >> cache >> cnt;

--cache;

tmp = arr[cache] >> cnt;

cout << bitset<sizeof(tmp.Getold()) \* CHAR\_BIT>(tmp.Getold()) << bitset<sizeof(tmp.Getjun()) \* CHAR\_BIT>(tmp.Getjun()) << '\n';

break;

case 9:

cout << "Enter string number to count 1 in it" << '\n';

cin >> cache;

--cache;

cout << "Result:" << Countbin(arr[cache]) << '\n';

break;

case 10:

cout << "Enter 1st string number, operation, 2nd string number" << '\n';

cin >> lhs >> simv >> rhs;

--lhs; --rhs;

if (simv == '>') {

if (arr[lhs]>arr[rhs]) cout << "True" << '\n'; else cout << "False" << '\n';

} else

if (simv == '<') {

if (arr[lhs]<arr[rhs]) cout << "True" << '\n'; else cout << "False" << '\n';

} else {

cout << "Error!" << '\n';

break;

}

break;

case 11:

cout << "Enter 1st string number that includes in 2nd string number" << '\n';

cin >> lhs >> rhs;

if (Inclusion(arr[lhs], arr[rhs])) cout << "True" << '\n'; else cout << "False" << '\n';

break;

case 12:

flag = true;

break;

}

if (flag) break;

}

return 0;

}