Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра ІУС

Дисципліна: «Технології об’єктно орієнтованого програмування»

**ПРАКТИЧНА РОБОТА № 1**

**«КОНСТРУЮВАННЯ ПРОСТИХ КЛАСІВ ТА ОБ’ЄКТІВ»**

|  |  |
| --- | --- |
|  | Прийняла:  Білова Т. Г.  з оцінкою «\_\_»  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2020р. |

|  |  |
| --- | --- |
|  | Виконав:  Ст. гр. ІТУ-19-2  Куренков Б.М. |

**МЕТА РОБОТИ**

Засвоєння поняття класів та об’єктів; набуття практичних навичок їх оголошення та використання.

**ПОСТАНОВКА ЗАДАЧІ**

**Загальне завдання для всіх варіантів**:

1. Здійснити проектування відповідного класу. Включити в інтерфейс

класу, крім функцій, зазначених у завданні, такі функції:

• всі типи конструкторів (якщо в додаткових вказівках до групи

завдань не вказано інше). Конструктори обов’язково повинні перевіряти

коректність ініціалізації даних класу.

• функції доступу і ініціалізації;

• функцію порівняння об’єктів на рівність (якщо в завданні не вказано

інше);

• функції введення c консолі і виведення на консоль.

2. Завдання розбиті на розділи за рівнем складності. Деякі розділи

можуть супроводжуватися специфічними додатковими вказівками до

виконання завдань саме цієї групи.

4. Реалізувати отриманий проект засобами мови С ++. При реалізації

конструкторів і функцій класу, що змінюють стан об’єкта (значення його

даних), стежити за тим, щоб об’єкт завжди знаходився в несуперечливому

стані.

5. У функції main створити об’єкти відповідного типу і здійснити

демонстрацію функціональності розробленого класу.

6. Операції додавання, множення та ін. не перевантажувати,

реалізовувати їх звичайними методами класу (крім завдання на сутності,

що передбачають динамічне розміщення, там потрібно перевантажити

операцію присвоювання operator= ).

**ХІД РОБОТИ**

#pragma once

#include <iostream>

#include <iomanip>

#include <stdio.h>

class Queue

{

int max\_size;

int\* root;

int i\_bot = -1;

public:

Queue();//ok

Queue(const Queue& tmp);//ok

Queue(int max\_size);//ok

Queue(int max\_size, int elem);//ok

Queue& operator=(const Queue& m2);//ok

//Queue& operator+(const Queue& m2);//ok

//Queue& operator+=(const Queue& m2);//ok

//bool operator==(const Queue& m2) const;//ok

bool Empty() { return i\_bot == -1 ? 1 : 0; };//ok

bool Push(int elem);//ok

bool Push(int\* elem, int size);//ok

bool Pop();//ok

bool Pop(int num);//ok

bool Clear();//ok

bool Show() const;//ok

void Show\_size() { std::cout << "Max size is: " << this->max\_size << "\n"; }//ok

void Show\_i() { std::cout << "i\_bot is: " << this->i\_bot << "\n"; }//ok

Queue& Unite(Queue& m2);//ok ~ +

bool Compare(Queue& m2) const;//ok ~ ==

~Queue();//ok

};

Queue::Queue(const Queue &tmp) : max\_size(tmp.max\_size), i\_bot(tmp.i\_bot) {

root = new int[tmp.i\_bot + 1];

for (int i = -1; i != tmp.i\_bot; ++i) {

root[i + 1] = tmp.root[i + 1];

}

std::cout << "Queue copyed with " << i\_bot + 1 << " elements!\n";

}

Queue::Queue() :max\_size(0) {

root = NULL;

std::cout << "Queue created with 0 elements!\n";

}

Queue::Queue(int max\_size) {

max\_size < 0 ? this->max\_size = (max\_size \* -1) : this->max\_size = (max\_size);

root = new int[this->max\_size];

std::cout << "Queue created with 0 elements!\n";

}

Queue::Queue(int max\_size, int elem) {

max\_size < 0 ? this->max\_size = (max\_size \* -1) : this->max\_size = (max\_size);

root = new int[this->max\_size];

root[0] = elem;

i\_bot = 0;

std::cout << "Queue created with 1 elements!\n";

}

Queue& Queue::operator=(const Queue& m2) {

if (this == &m2) { std::cout << "You cannot assign an element to itself!\n"; return \*this; }

delete[] root;

root = new int[m2.i\_bot + 1];

for (int i = -1; i != m2.i\_bot; ++i) {

root[i + 1] = m2.root[i + 1];

}

i\_bot = m2.i\_bot;

max\_size = m2.max\_size;

return \*this;

}

//ok

/\*Queue& Queue::operator+(const Queue& m2) {

Queue\* tmp = new Queue(this->max\_size + m2.max\_size);

tmp->Push(root, i\_bot + 1);

tmp->Push(m2.root, m2.i\_bot + 1);

return \*tmp;

}\*/

//ok

/\*Queue& Queue::operator+=(const Queue& m2) {

this->Push(m2.root, m2.i\_bot + 1);

return \*this;

}\*/

//ok

/\*bool Queue::operator==(const Queue& m2) const {

if (i\_bot == m2.i\_bot) {

bool cond\_comp = 1;

for (int i = -1; i + 1 != i\_bot && i + 1 != m2.i\_bot && cond\_comp == 1; ++i)

if (root[i + 1] != m2.root[i + 1])

cond\_comp = 0;

if (cond\_comp) return 1;

else return 0;

}

return 0;

}\*/

bool Queue::Push(int elem) {

if (i\_bot + 1 == max\_size) { std::cout << "Queue is full!\n"; return 0; }

root[i\_bot + 1] = elem;

++i\_bot;

std::cout << "Element succesfully pushed!\n";

return 1;

}

bool Queue::Push(int\* elem, int size) {

if (i\_bot + 1 == max\_size) { std::cout << "Queue is full!\n"; return 0; }

int i = 1;

size < 0 ? size \*= -1 : size = size;

while ( size + 1 > i && ((i\_bot + i ) != max\_size)) {

root[i\_bot + i ] = elem[i-1];

++i;

}

std::cout << i - 1 << " elements succesfully pushed!!!!\n";

i\_bot += i - 1;

return 1;

}

bool Queue::Show() const {

if (i\_bot == -1) { std::cout << "Queue is empty!\n"; return 0; }

int i = i\_bot + 1;

std::cout << "Queue has " << i << " elements:\n";

while (i) {

std::cout << std::setw(4) << root[i\_bot - i + 1];

--i;

} std::cout << std::endl;

return 1;

}

bool Queue::Pop() {

if (i\_bot == -1) { std::cout << "You cannot delete an element because of queue emptyness!\n"; return 0; }

if (i\_bot + 2 == max\_size)

root = NULL;

else

++root;

if (i\_bot != -1)

--i\_bot;

std::cout << "Element succesfully deleted!\n";

return 1;

}

bool Queue::Pop(int num) {

if (i\_bot == -1) { std::cout << "You cannot delete an element because of queue emptyness!\n"; return 0; }

num < 0 ? num \*= -1 : num = num;

if (i\_bot + 1 + num == max\_size)

root = NULL;

else

root += num;

int i = 0;

while (num > i) {

++i;

if (i\_bot != -1)

--i\_bot;

else {

std::cout << "All elements succesfully deleted!\n"; return 1;

}

}

std::cout << "Deleted " << i << " elements!\n";

return 1;

}

bool Queue::Clear() {

if (i\_bot != -1) {

root = NULL;

std::cout << "Queue cleared!\n";

i\_bot = -1;

return 1;

}

std::cout << "Queue is alredy empty!\n";

return 0;

}

Queue& Queue::Unite(Queue& m2) {

Queue\* tmp = new Queue(this->max\_size + m2.max\_size);

tmp->Push(root, i\_bot + 1);

tmp->Push(m2.root, m2.i\_bot + 1);

return \*tmp;

}

bool Queue::Compare(Queue& m2) const {

if (i\_bot == m2.i\_bot) {

bool cond\_comp = 1;

for (int i = -1; i + 1 != i\_bot && i + 1 != m2.i\_bot && cond\_comp == 1; ++i)

if (root[i + 1] != m2.root[i + 1])

cond\_comp = 0;

if (cond\_comp) return 1;

else return 0;

}

return 0;

}

Queue::~Queue() {

delete[] root;

}

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

#include "Queue\_.h"

int main() {

setlocale(LC\_ALL, "");

srand(time(0));

int tmp\_in;

std::cout << "Введите максимальный размер первой очереди: ";

std::cin >> tmp\_in;

Queue\* g1 = new Queue(tmp\_in);

std::cout << "Заполните её: \n";

tmp\_in < 0 ? tmp\_in \*= -1 : tmp\_in = tmp\_in;

for (int i = 0, tmp; i < tmp\_in; ++i) {

std::cout << " ";

std::cin >> tmp;

g1->Push(tmp);

}

std::cout << "Первая очередь заполнена!\n\n\n";

std::cout << "\nВведите максимальный размер второй очереди: ";

std::cin >> tmp\_in;

Queue g2(tmp\_in);

std::cout <<"Введите размер массива элементов, который будет помещен в очередь: ";

std::cin >> tmp\_in;

tmp\_in < 0 ? tmp\_in \*= -1 : tmp\_in = tmp\_in;

int\* arr = new int[tmp\_in];

for (int i = 0; i < tmp\_in; ++i) {

arr[i] = rand() % (2 \* 30 + 1) - 30;

}

std::cout << "Массив был заполнен случайными числами в диапазоне [-30;30].\n";

g2.Push(arr, tmp\_in);

std::cout << "Массив был помещен во вторую очередь.\n\n\n\n";

Queue\* g3 = new Queue();

std::cout << "Третья очередь была созданна пустой.\n\n\n";

std::cout << "\nПроверим пусты ли очереди:\n1я - " << g1->Empty() << "\n2я - " << g2.Empty() << "\n3я - " << g3->Empty() << "\n\n\n\n";

std::cout << "Выведем очереди на экран:\n\nПервая:";

g1->Show();

std::cout << "\nВторая:\n";

g2.Show();

std::cout << "\nТретья:\n";

g3->Show();

std::cout << "\n\n\nСоединим первую и вторую очереди, заполнив их объединением третью.\n";

\*g3 = g1->Unite(g2);

std::cout << "Третья очередь выглядит вот так:\n";

g3->Show();

std::cout << "\n\n\nСколько элементов желаете удалить из третьей очереди, перед сравнением её со второй? ";

std::cin >> tmp\_in;

tmp\_in < 0 ? tmp\_in \*= -1 : tmp\_in = tmp\_in;

g3->Pop(tmp\_in);

std::cout << "Теперь третья очередь выглядит вот так:\n";

g3->Show();

std::cout << "\n\n\nДля наглядности выведем на экран вторую очередь.\n";

g2.Show();

std::cout << "Равны ли вторая и третья очереди? ";

std::cout << g2.Compare(\*g3);

std::cout << "\n\n\n\nВыведем на экран содержимое всех очередей в последний раз, затем очистим их.\n";

std::cout << "\nПервая:\n";

g1->Show(); // 1

g1->Show\_i();

g1->Show\_size();

std::cout << "\nВторая:\n";

g2.Show(); // 2

g2.Show\_i();

g2.Show\_size();

std::cout << "\nТретья:\n";

g3->Show(); // 3

g3->Show\_i();

g3->Show\_size();

std::cout << "\n";

g1->Clear(); // очистка

g2.Clear();

g3->Clear();

std::cout << "\nПервая:\n";

g1->Show(); // 1

g1->Show\_i();

g1->Show\_size();

std::cout << "\nВторая:\n";

g2.Show(); // 2

g2.Show\_i();

g2.Show\_size();

std::cout << "\nТретья:\n";

g3->Show(); // 3

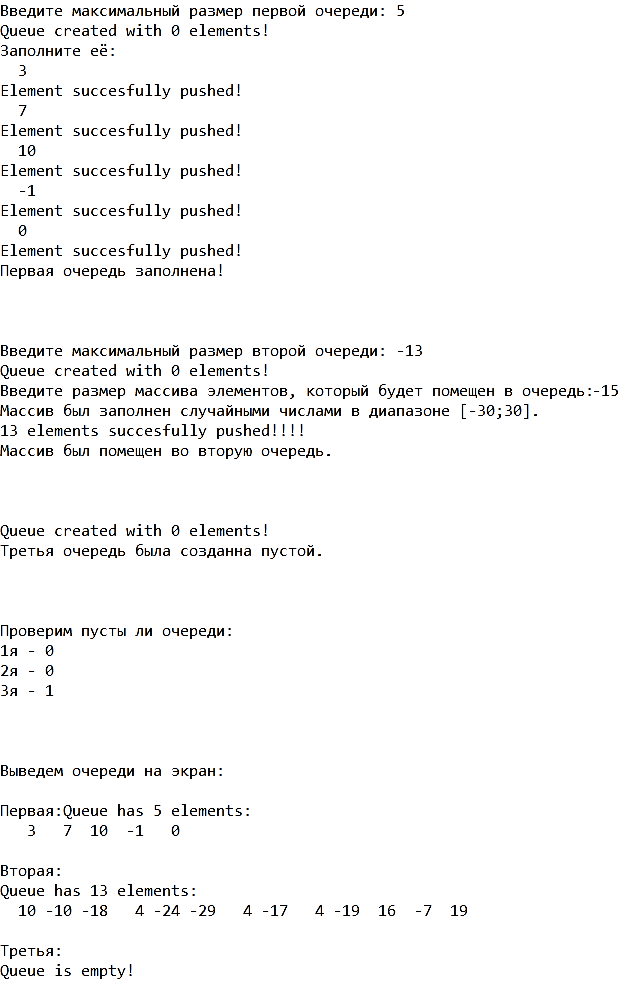
g3->Show\_i();

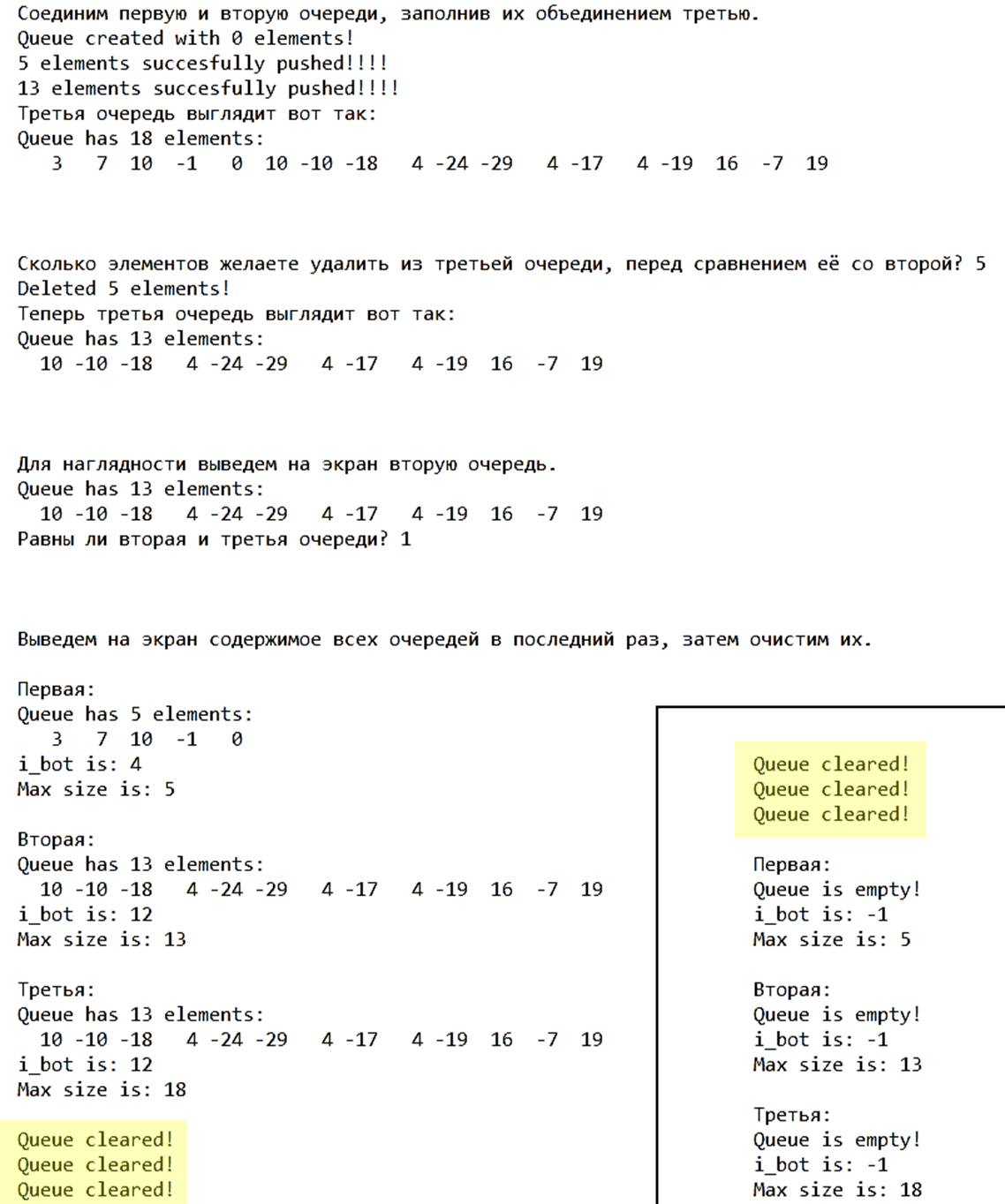
g3->Show\_size();

return 0;

}

**ВИКОНАННЯ ЕКСПЕРИМЕНТІВ ПО НАЛАГОДЖЕННЮ ПРОГРАМИ**

****



Висновки.

Протягом лабораторної роботи мною було засвоєно поняття класів та об’єктів; набуто практичних навичок їх оголошення та використання.