 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №5**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Шаблони»

Варіант №15

**Виконав:**

студент гр. БС-81

Сєров О. В.

**Перевірив:**

ас. каф. БМК

Рисін С.В

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

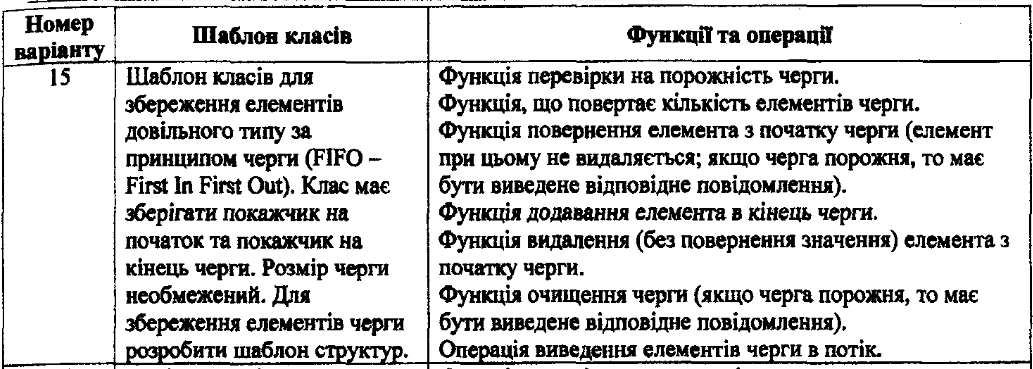
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2020

**Завдання:**

1. Ознайомитися з теоретичними відомостями роботи із шаблонами функцій та шаблонами класів.
2. Відповідно до свого варіанту розробити шаблон класів абстрактної структури даних із визначенням заданих функцій та операцій (в кожному шаблонному класі також має бути визначений конструктор копіювання та операція привласнення) і відповідну діаграму класів в нотації UML, написати програму тестування, в якій перевіряється використання шаблона для стандартних типів даних:

******

1. Скласти і захистити звіт.

**🞏 Виконано без зауважень**

**🞏 Маються зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до UML діаграми:**

**🞏 діаграма класу не відповідає коду**

**🞏 виконані не за стандартом:**

**🞏 атрибути**

**🞏 відношення**

**🞏 потужність**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 хибно задані специфікатори доступу**

**🞏 помилки у визначенні конструкторів / деструкторів**

**🞏 відсутні списки ініціалізації в конструкторах**

**🞏 константні методи**

**🞏 використано глобальні змінні**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Діаграма класів:**



**Код:**

**FIFO\_template.h**

#pragma once

#include <iostream>

#include <string>

#include <cstdlib>

using namespace std;

template <class X>

class Queue

{

X\* \_arr; // array of elements

unsigned \_capacity; // maximum capacity of the queue

unsigned \_first; // number of front element in the queue

unsigned \_last; // number of rear element in the queue

unsigned \_length; // current size of the queue

public:

Queue(unsigned size);

~Queue();

bool isEmpty() const;

bool isFull() const;

unsigned size() const;

X peek() const;

void enqueue(X new\_element); // insert element

void dequeue(); // remove element

void delete\_q();

void print(ostream& os = cout) const;

};

template <class X>

Queue<X>::Queue(unsigned size) :

\_capacity(size), \_first(0), \_last(-1), \_length(0)// constructor to initialize queue

{

\_arr = new X[size];

}

template <class X>

Queue<X>::~Queue() // destructor to delete queue

{

delete\_q();

}

template <class X>

bool Queue<X>::isEmpty() const

{

return (size() == 0);

}

template <class X>

bool Queue<X>::isFull() const

{

return (size() == \_capacity);

}

template <class X>

unsigned Queue<X>::size() const // current size of the queue

{

return \_length;

}

template <class X>

X Queue<X>::peek() const // return front element in the queue

{

if (isEmpty())

{

cout << "\nUnderFlow\nProgram Terminated\n";

exit(EXIT\_FAILURE);

}

return \_arr[\_first];

}

template <class X>

void Queue<X>::enqueue(X new\_element) // add an item to the queue

{

// check for queue overflow

if (isFull())

{

cout << "\nOverFlow of queue\nProgram stopped\n";

exit(EXIT\_FAILURE);

}

\_last = (\_last + 1) % \_capacity;

\_arr[\_last] = new\_element;

\_length++;

cout << "\nItem '" << new\_element << "' added in queue\n";

}

template <class X>

void Queue<X>::dequeue() // remove front element from the queue

{

// check for queue underflow

if (isEmpty())

{

cout << "\nUnderFlow of queue\nProgram stopped\n";

exit(EXIT\_FAILURE);

}

cout << "\nRemoving item '" << \_arr[\_first] << "' from queue\n";

\_first = (\_first + 1) % \_capacity;

\_length--;

}

template <class X>

void Queue<X>::delete\_q() // delete all elements

{

if (isEmpty())

cout << "\nQueue is already empty!\n";

else

{

while (\_length != 0)

Queue<X>::dequeue();

cout << "\nQueue is empty now!\n";

}

}

template <class X>

void Queue<X>::print(ostream& os) const // show all elements

{

cout << "\nThe QUEUE now:\n[ ";

for (unsigned i = 0; i < \_length; ++i)

os << \_arr[(\_first + i) % \_capacity] << ' ';

os << "]" << endl;

}

**Main.cpp**

#include "FIFO\_template.h"

void main()

{

Queue<string> q\_1(4);

q\_1.enqueue("first");

q\_1.enqueue("second");

q\_1.enqueue("third");

q\_1.print();

cout << "\nFront element is: " << q\_1.peek() << endl;

q\_1.dequeue();

q\_1.enqueue("fourth");

q\_1.print();

cout << "\nQueue size is " << q\_1.size() << endl;

q\_1.dequeue();

q\_1.dequeue();

q\_1.print();

q\_1.delete\_q();

system("pause");

Queue<int> q\_2(2);

q\_2.enqueue(10);

q\_2.enqueue(100);

q\_2.print();

cout << "\nFront element is: " << q\_2.peek() << endl;

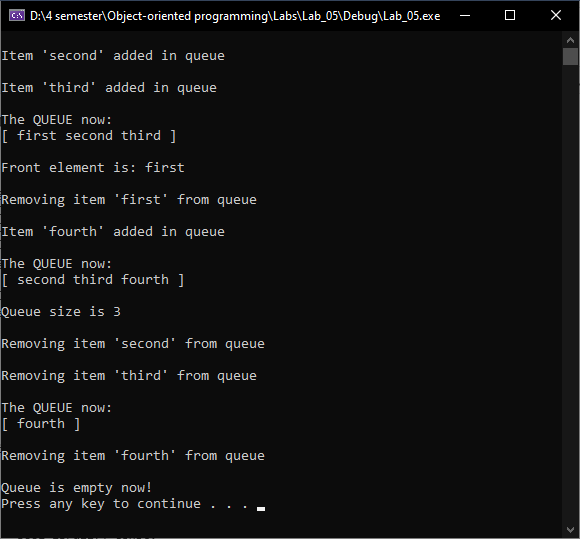
q\_2.dequeue();

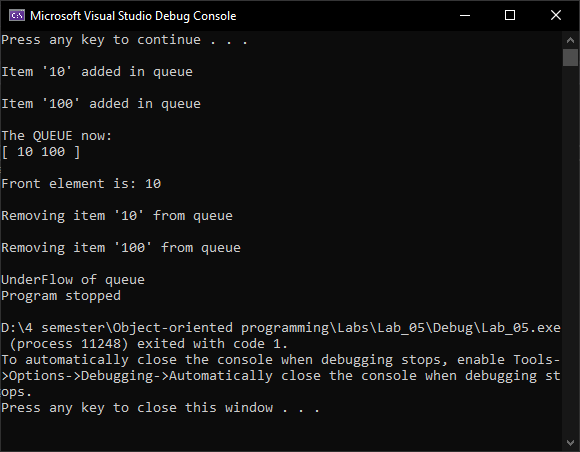
q\_2.dequeue();

q\_2.dequeue();

}

**Результати:**

****

****

**Контрольні питання:**

1. **Для чого використовують шаблони?**

Для відокремлення загального алгоритму від його реалізації стосовно до конкретних типів даних.

1. **Що таке шаблон класів, як його визначити в C++?**

Шаблон класів – це конструкція, яка визначає узагальнений опис класу, тобто його опис для узагальненого типу даних. Під час створення шаблонного класу створюється ціле сімейство споріднених класів, які можна застосувати до будь-якого типу даних. Таким чином, тип даних, яким оперує клас, буде вказаний як параметр під час створення об’єкта шаблонного класу.

Формат оголошення шаблона класів наступний:

template <class ТІ [.class Т2,. . class TN]>

class class jiam e

{

// визначення класу

};

де Ті - довільний ідентифікатор узагальненого типу

1. **Що таке шаблон функцій, як його визначити в C++?**

Шаблон функцій – це конструкція, яка визначає узагальнений опис функцій, тобто їх опис для узагальненого типу даних. При виклику шаблону функцій вказується конкретний тип даних (стандартний тип даних int, double або будь-який користувацький тип даних), який підставляється на місце узагальненого типу.

Формат шаблона функцій:

template <class ТІ [,class Т2, ... ,class TN]>

return-type function\_name ( список\_параметрів )

{

// тіло функції

}

де Ті - довільний ідентифікатор узагальненого типу; return-type - тип значення, що повертає функція, в якості ‘якого може бути вказаний як конкретний тип даних, так і один зі списку узагальнених типів; список\_параметрів - список формальних параметрів шаблону функції, які можуть бути описані як із зазначенням конкретного типу, так і одного типу зі списку узагальнених типів.

1. **Що таке конкретизація шаблона? В яких випадках необхідне виконання явної конкретизації шаблона?**

Конкретизація шаблона – це створення компілятором на основі шаблона класів класу для певних параметрів (типів і констант). Таким чином на основі шаблона класів отримують клас, який потім в програмі може бути використаний через його уточнене ім’я:

ім’я шаблона\_класів

<параметри\_конкретизації>

Наприклад, використання уточненого імені шаблона класів під час оголошення параметрів функції і локальних змінних:

void function(stac<int> &s)

{

stack<int> \*ptr = &s;

}

1. **Що таке спеціалізація шаблона, коли її необхідно виконувати?**

Спеціалізація шаблона – це створення в коді реалізації шаблона класів для конкретних значень його параметрів. Є необхідною, якщо шаблон класів не придатний для конкретизації певниими типами даних або якщо конкретизація шаблона класів не є ефективною за реалізацією.

Формат спеціалізації шаблона класів:

Template<>

Class|struct|union ім’я\_класу

<ім’я\_типу\_спеціалізації>

{

// тіло

};

1. **Для чого використовують параметри за замовченням для шаблону класів?**

Використовують для того ж, що і параметри за замовчуванням у звичайних функцій – для гнучкості коду.

Приклад:

template <typename T, typename CONT = vector<T>>

class MyClass{

CONT data;

public:

void push(const T& value);

void show() const;

};

В даному прикладі можна навести різні типи вхідних даних та різні контейнери для роботи з даними.

1. **Чи можна викликати параметризовану функцію без параметрів?**

Так, якщо параметри встановлені за замовчуванням, інакше – ні.

1. **Чи може бути порожнім список параметрів шаблона? Відповідь пояснити.**

Список параметрів не може бути пустим, він має містити хоча б один параметр, інакше клас не буде мати типу даних з яким він оперує. Також буде наявна помилка компіляції.

1. **Чи можна за допомогою шаблона створити функцію з таким самим ім’ям, як і в явно визначеної функції? Відповідь пояснити.**

Так, але у випадку співпадіння параметрів – буде викликана явно визначена функція.

1. **Чи можуть шаблони класів містити віртуальні функції?**

Так, але якщо віртуальна функція не є шаблоном функцій.