 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-71

Орлівський С. П.

**Перевірив:**

ас. каф. БМК. Рисін С.В.

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

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

(підпис викладача)

Київ-2019

**🞏 Практичне заняття без зауважень**

**🞏 Практичне заняття має 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:**

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

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

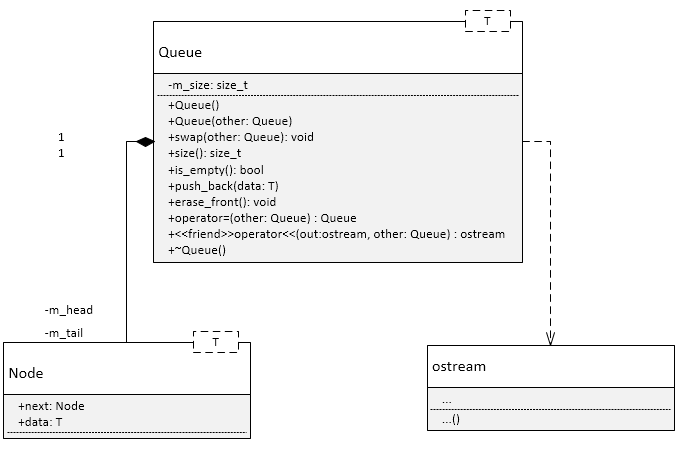
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.
3. Скласти і захистити звіт по роботі.

|  |  |  |
| --- | --- | --- |
| **Номер варіанту** | **Шаблон класів** | **Поля** |
| 15 | Шаблон класів для збереження елементів довільного типу за принципом черги. Клас має зберігати покажчик на початок та покажчик на кінець черги. Розмір черги необмежений. Для збереження елементів черги розробити шаблон структур. | Функція перевірки на порожність черги.  Функція, що повертає кількість елементів черги.  Функція повернення елементу з початку черги без видалення.  Функція додавання елементу в кінець черги.  Функція видалення (без повернення значення) елементу з початку черги.  Функція очищення черги.  Операція виведення черги в потік. |

**UML-діаграма**



**Лістинг**

**node.hpp:**

#pragma once

template <typename T>

struct **Node**

{

Node<T> \*next;

T data;

};

**queue.hpp:**

#pragma once

#include <iostream>

#include "node.hpp"

template <typename T>

class **Queue**

{

private:

Node<T> \*m\_head;

Node<T> \*m\_tail;

size\_t m\_size;

public:

**Queue**();

**Queue**(const Queue<T> &other);

void **swap**(Queue<T> &other);

size\_t **size**() const;

bool **is\_empty**() const;

void **push\_back**(T data);

void **erase\_front**();

Queue<T> &operator=(Queue<T> other);

template<typename Tt>

friend std::ostream &operator<<(std::ostream &out, const Queue<Tt> &queue);

void **clear**();

~**Queue**();

};

**queue\_methods.hpp:**

#include "queue.hpp"

template<typename T>

Queue<T>::**Queue**() : m\_head(nullptr), m\_tail(nullptr), m\_size(0)

{

}

template<typename T>

Queue<T>::**Queue**(const Queue<T> &other)

{

Node<T> \*current\_other\_node = other.m\_head;

Node<T> \*current\_current\_node = nullptr;

m\_head = nullptr;

if (current\_other\_node)

{

current\_current\_node = new Node<T>;

current\_current\_node->data = current\_other\_node->data;

m\_head = current\_current\_node;

}

while (current\_other\_node && current\_other\_node->next)

{

current\_current\_node->next = new Node<T>;

current\_current\_node->next->data = current\_other\_node->next->data;

current\_current\_node = current\_current\_node->next;

current\_other\_node = current\_other\_node->next;

}

if (current\_current\_node)

{

current\_current\_node->next = nullptr;

}

m\_size = other.m\_size;

}

template<typename T>

size\_t Queue<T>::**size**() const

{

return m\_size;

}

template<typename T>

bool Queue<T>::**is\_empty**() const

{

return (!m\_size) ? true : false;

}

template<typename T>

void Queue<T>::**push\_back**(T data)

{

Node<T> \*temp = new Node<T>;

temp->data = data;

temp->next = nullptr;

if (m\_tail)

{

m\_tail->next = temp;

m\_tail = temp;

}

else

{

m\_head =temp;

m\_tail = temp;

}

++m\_size;

}

template<typename T>

void Queue<T>::**erase\_front**()

{

Node<T> \*temp = m\_head;

if (temp)

{

m\_head = m\_head->next;

delete temp;

--m\_size;

}

else

{

std::cout << "dkw but noting to remove\n";

}

}

template<typename T>

void Queue<T>::**swap**(Queue<T> &other)

{

m\_size = other.m\_size;

m\_head = other.m\_head;

m\_tail = other.m\_tail;

}

template<typename T>

Queue<T> &Queue<T>::operator=(Queue<T> other)

{

this->swap(other);

return \*this;

}

template<typename T>

void Queue<T>::**clear**()

{

Node<T> \*temp;

while (m\_head)

{

erase\_front();

// temp = m\_head;

// m\_head = m\_head->next;

// delete temp;

}

m\_size = 0;

}

template<typename T>

Queue<T>::~**Queue**()

{

clear();

}

template<typename Tt>

std::ostream &operator<<(std::ostream &out, const Queue<Tt> &queue)

{

if (!queue.m\_head)

{

out << "queue is empty";

return out;

}

Node<Tt> \*temp = queue.m\_head;

while (temp)

{

out << temp->data << ' ';

temp = temp->next;

}

return out;

}

**main.cpp:**

#include <iostream>

#include "queue.hpp"

#include "queue\_methods.hpp"

#define LOOP\_NUMBER 10

int **main**()

{

Queue<int> q;

Queue<int> q2(q);

std::cout << q2 << std::endl;

for (size\_t i = 0; i < LOOP\_NUMBER; ++i)

{

q.push\_back(static\_cast<int>(i));

}

std::cout << q << std::endl;

std::cout << "q is empty: " << q.is\_empty() << std::endl;

for (size\_t i = 0; i <= LOOP\_NUMBER; ++i)

{

q.erase\_front();

}

std::cout << q << std::endl;

q.clear();

std::cout << q << std::endl;

std::cout << "q is empty: " << q.is\_empty() << std::endl;

Queue<short> qs;

Queue<long long> qll;

Queue<float> qf;

Queue<double> qd;

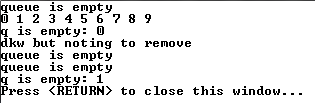
Queue<bool> qb;

Queue<char> qc;

return 0;

}

**Результати роботи програми**



**Контрольні запитання**

1. ***Для чого використовують шаблони?***

Шаблони у мовах програмування використовують з метою зменшити об’єми написаного коду та в той же час отримати його(коду) універсальну версію для роботи з типами даних не залежно від їхнього типу.

1. ***Що таке шаблон класів, як його визначити у С++?***

Шаблон класу у С++ це особлива конструкція декларації класу, що дозволяє останньому отримувати різні типи даних у якості полів класу. У С++ визначається за допомогою констукції template<…>:

template<typename або class ім’я(псевдонім) універсального типу, подальший список типів…>

class ім’я класу

{

…(замість типу універсальних полів використовувати ім’я типу, визначене в <>)

};

1. ***Що таке шабон функції та як її визначити у С++:***

Шаблон ф-кції у С++ це конструкція, що дозволяє створити функцію, котра отримує параметри не залежно від їх типу:

template<typename або class ім’я(псевдонім) універсального типу, подальший список типів…>

тип ім’я\_ф-ції (список параметрів)

{

тіло

}

***4.Що таке конкретизація шаблону та в яких випадках необхідно виконувати явну конкретизацію?***

Конкретизація шаблону – це використання ф-кції або класу з конкретним існуючим типом замість шаблонного. Реалізується завдяки явному вказанню типу шаблону у “<>” при інстанціюванні класу або у випадку функцій компілятор визначає необхідну конкретизацію в залежності від параметрів.

void f

{

TemplatedClass<int> a; //інстанціювання шаблонного класу TemplatedClass з конкретним типом – int

foo(10); // якщо функція foo() – шаблонна, у данному випадку вона конкретизується типом int

}

***5. Що таке спеціалізація шаблона у С++, коли вона використовується.***

Спеціалізація шаблона у С++ - це визначення шаблонної ф-кції або класу при роботі з конкретним отриманим параметром. Потрібно, коли шаблонний клас або ф-кція при отриманні параметрів усіх можливих типів буде мати одну незмінну імплементацію для роботи з усіма цими типами і лише для роботи з якимось одним або декількома типами її імплементація буде відрізнятися. Наприклад, якщо для роботи з числами будь якого типу функція буде мати одну й ту саму імплементацію. А при роботі з рядками ця імплементація зміниться.

Наприклад, ф-кція додавання:

template <typename T>

T addition(T first, T second)

{

return first + second;

}

template <>

char \*addition<char \*>(char \*first, char \*second)

{

char \*new\_str = (strlen(first) + strlen(second) > 0) ? new char[strlen(first) + strlen(second) – 1]: nullptr;

strcpy(new\_str, first);

strcat(new\_str, second);

return new\_str;

}

***6. Для чого використовують параметри за замовчуванням для шаблонів класів?***

Параметри за замовчуванням для шаблонів класів використовують для того щоб можна було створити клас не выказавши тип, з яким він буде працювати – в такому разі буде використаний тип за замовчуванням у шаблоні класу.

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

Так, якщо ф-ція має усі параметри за замовчуванням .

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

Так, якщо ми визначаємо спеціалізацію для шаблона, тобто реалізацію шаблонної ф-ції або класу для конкретного типу. Як показано у відповіді на 4 запитання.

***9. Чи можна за допомогою шаблонів створити функцію з таким самим ім’ям як і у явно визначенної функції. Відповідь пояснити.***

Так можна, бо у С++ є механізм перевантаження ф-цій. у такому випадку компілятор буде спочатку звертатися до нешаблонної фції і якщо тип її параметрів не підходить, то тоді буде викликана шаблонна ф-ція

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

Так, можуть