 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №6**

з дисципліни «Об’єктно-орієнтоване програмування»

на тему: «Робота з бібліотекою STL»

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 для роботи зі стандартною бібліотекою STL за списком рекомендованої літератури.
2. Переробити розроблений в попередній лабораторній роботі шаблонний клас із використанням контейнерів STL.
3. Скласти і захистити звіт по роботі.

**🞏 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:**

**STL\_FIFO\_template.h**

#pragma once

#include <queue>

#include <iostream>

#include <string>

using namespace std;

template <class X>

class Queue

{

queue <X> \_queue;

public:

Queue();

~Queue();

bool isEmpty() 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>

inline Queue<X>::Queue()

{ }

template <class X>

Queue<X>::~Queue()

{

delete\_q();

}

template <class X>

bool Queue<X>::isEmpty() const

{

return \_queue.empty();

}

template <class X>

unsigned Queue<X>::size() const

{

return \_queue.size();

}

template <class X>

X Queue<X>::peek() const

{

if (isEmpty())

cout << "\nNo elements in queue to peek!\n";

else

return \_queue.front();

}

template <class X>

void Queue<X>::enqueue(X new\_element)

{

\_queue.push(new\_element);

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

}

template <class X>

void Queue<X>::dequeue()

{

if (isEmpty())

cout << "\nNo elements in queue to pop!\n";

else

{

cout << "\nRemoving item '" << \_queue.front() << "' from queue\n";

\_queue.pop();

}

}

template <class X>

void Queue<X>::delete\_q()

{

if (isEmpty())

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

else

while (!isEmpty())

dequeue();

}

template <class X>

void Queue<X>::print(ostream& os) const

{

queue <X> temp\_queue = \_queue;

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

while(!temp\_queue.empty())

{

os << temp\_queue.front() << ' ';

temp\_queue.pop();

}

os << "]" << endl;

}

**Main.cpp**

#include "STL\_FIFO\_template.h"

void main()

{

Queue<string> q\_1;

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;

q\_2.enqueue(10);

q\_2.enqueue(100);

q\_2.print();

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

q\_2.dequeue();

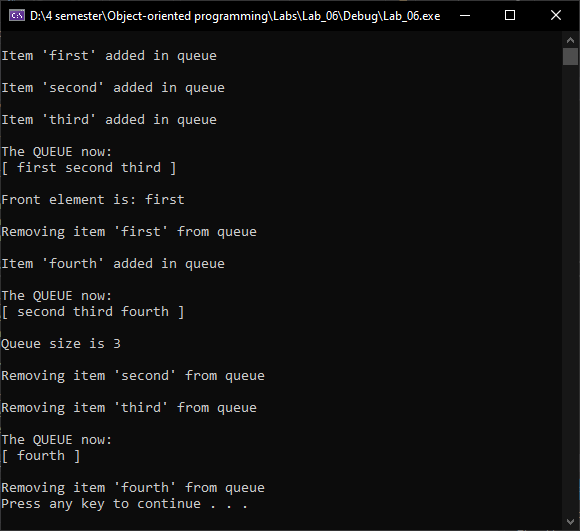
q\_2.dequeue();

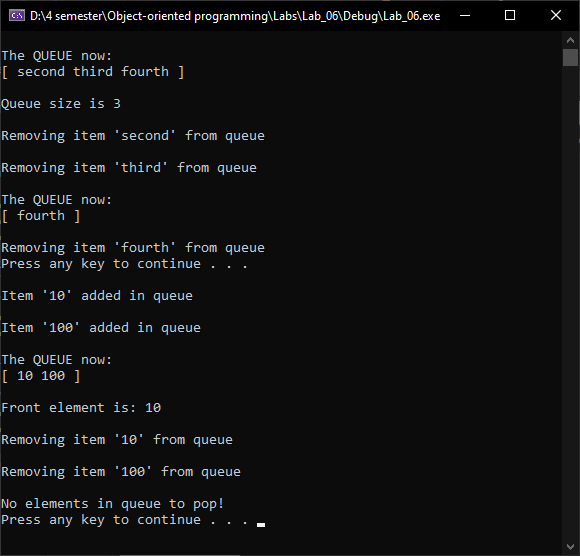
q\_2.dequeue();

system("pause");

}

**Результати:**

****

****

**Контрольні питання:**

1. **Опишіть організацію контейнера STL.**

Контейнери – це об’єкти, що зберігають інші елементи і реалізують механізми доступу до них. Прикладами контейнерів є вектори і списки. Кожний контейнер описується шаблонним класом, у якому реалізуються механізми доступу і функція для обробки елементів, що містяться у контейнері. Кожний контейнер має свої ітератори для перебору елементів і функції для їх обробки.

1. **Які класи належать до категорії контейнерів?**

Vector, array, deque, list, forward\_list, map, unordered\_set, multimap, unordered\_multimap, multiset, unordered\_ multiset, queue, priority\_ queue, stack.

1. **Які вимоги до типів, які можна використовувати з контейнерами STL?**

Контейнерами можуть використовуватися будь-які типи даних.

1. **Яка різниця між контейнерами та адаптерами контейнерів?**

Контейнери використовуються безпосередньо для зберігання даних, а адаптери використовують для того щоб реалізувати певний інтерфейс цього адаптера для нового класу.

1. **Що таке ітератори, яких типів вони бувають та в чому їх принципова різниця?**

Ітератори підкоряються принципу чистої абстракції, тобто будь-який об'єкт, який веде себе як ітератор, є ітератором.

**Категорії ітераторів**

* Ітератори введення і виведення
* Однонаправлені ітератори і двонаправлені ітератори
* Ітератори довільного доступу
* Допоміжні ітератори
* Реверсивні ітератори
* Ітератори потоків
* Ітератори вставки
* Константний ітератор

1. **Які операції можна виконувати над усіма типами ітераторів?**

Над усіма типами ітераторів можна виконувати такі операції: operator++, operator\*.

1. **Що таке ітератор довільного доступу? Які операції він дозволяє виконувати?**

Ітератори довільного доступу (random access) - через них можна мати доступ до будь-якого елементу. Такі ітератори реалізовані в деяких контейнерних типах stl (vector, deque, string, array).

1. **Які вимоги до ітераторів з боку алгоритмів?**

Різні алгоритми потребують різних вимог до літераторів. STL визначає 5 видів літераторів і описує алгоритми в термінах літераторів, які їм необхідні. Ці 5 видів літераторів наступні: вхідний, вихідний, однонаправлений, двохнаправлений та літератор довільного доступу. Кожен вид реалізує різні операції

1. **Що таке стандартні алгоритми? Наведіть приклад.**

Стандартні алгоритми – це алгоритми які визначені в бібліотеці STL. До них відносяться наступні категорії: алгоритми пошуку, алгоритми сортування, видалення елементів, операції відношення, кучі, злиття.

Наприклад, для пошуку можна застосувати функцію find(begin, end, what), яка шукає перший елемент зі значенням what на проміжку begin-end, де begin, end – ітератори відповідного контейнера.

Функція sort(start, end) сортує елементи від start до end в порядку зростання.

1. **Які алгоритми вимагають впорядкованості?**

Впорядкованості потребують алгоритми: binary\_search (бінарний пошук), lower\_bound (знаходить першу позицію у відсортованому діапазоні, перед якою можна вставити елемент, не порушуючи впорядкованості), upper\_bound(знаходить останню позицію у відсортованому діапазоні, перед якою можна вставити елемент, не порушуючи впорядкованості), equal\_range(знаходить найбільший піддіапазон в відсортованому діапазоні, в якому елемент можна вставити перед будь-яким ітератором в цьому діапазоні без порушення впорядкованості), set\_union(формує множину, яка являється обєднанням двох діапазонів), set\_intersection(формує множину, яка являється перетином двох діапазонів), set\_difference(формує множину, яка являється різницею між двома діапазонами), set\_symmetric\_difference(формує послідовність, яка являє собою симетричну різницю двох діапазонів), merge(злиття двох відсортованих діапазонів), inplace\_merge(різновид злиття).