 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 біліотекою STL»

Варіант №15

**Виконав:**

студент гр. БС-71

Орлівський С. П.

**Перевірив:**

ac. 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:**

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

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

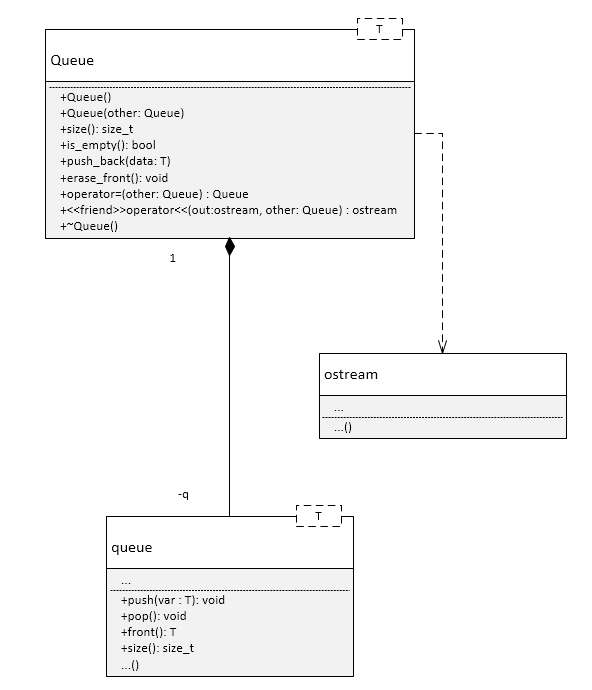
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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. Необхідно використовувати STL.
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:**

****

**Код:**

**queue.hpp**

#pragma once

#include <iostream>

#include <queue>

#include "node.hpp"

template <typename T>

class Queue

{

private:

    std::queue<T> q;

public:

Queue() = default;;

Queue(const Queue<T> &other) = default;;

size\_t size() const;

bool is\_empty() const;

void push\_back(T data);

void erase\_front();

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

template<typename Tt>

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

void clear();

~Queue() = default;;

};

**queue\_methods.hpp**

#include "queue.hpp"

template<typename T>

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

{

return q.size();

}

template<typename T>

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

{

return size();

}

template<typename T>

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

{

    q.push(data);

}

template<typename T>

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

{

if (q.size())

{

        q.pop();

}

else

{

std::cout << "noting to remove somehow\n";

}

}

template<typename T>

void Queue<T>::clear()

{

while (q.size())

{

        q.pop();

}

}

template<typename Tt>

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

{

if (!q.size())

{

out << "queue is empty somehow";

return out;

}

while (q.size())

{

out << q.q.front() << ' ';

        q.erase\_front();

}

    out << '\n';

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;

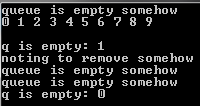
Queue<char> qc;

    std::cin.get();

return 0;

}

**Робота програми:**



**Контрольні запитання**

**1. Опишіть організацію контейнера STL.**

Бібліотека STL складає половину стандарту С++ і містить універсальні шаблонні класи і функції, що реалізують основні алгоритми і структури даних. Бібліотека складається з чотирьох категорій: контейнери, алгоритми, ітератори і функтори.

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. Наприклад, клас, що описує роботу із вектором, має відповідний ітератор для прямого доступу до елементів вектору, а також функції для вставки і видалення тощо.

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

* vector - Контейнер, що реалізує прямий доступ до послідовно розташованих елементів (аналог масиву).
* list - Контейнер, що реалізує концепцію двозв’язного списку, тобто забезпечує доступ до елементів(вузлів), що розташовані у довільних комірках пам’яті і пов’язані між собою відповідними посиланнями (вказівниками на вузли).
* deque - Контейнер, що реалізує концепцію двосторонньої черги, тобто дозволяє вставляти і видаляти елементи з обох кінців.
* set - Контейнер, що реалізує концепцію множини елементів, які не повторюються і розташовані у певному порядку.
* multiset - Контейнер, що реалізує концепцію множини елементів, які можуть повторюватися і розташовані у певному порядку (впорядковано).
* map - Контейнер, що реалізує концепцію асоціативного масиву, тобто забезпечує доступ до елементів за ключами.
* multimap - Контейнер, що реалізує концепцію асоціативного масиву, забезпечуючи доступ до елементів за ключами, які можуть повторюватися
* stack – Контейнер-адаптер, що реалізує концепцію стеку — лінійну структуру даних, організовану за принципом LIFO (останнім увійшов — першим вийшов);
* queue – Контейнер-адаптер, що реалізує концепцію черги — лінійної структури даних, що добавляються і видаляються за принципом FIFO (першим увійшов — першим пішов).
* priority queue – Контейнер-адаптер, що реалізує концепцію черги за пріоритетами, тобто черги, елементи якої відсортовані за пріоритетом.

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

Контейнер може містити об'єкти будь-якого типу, крім посилань. Конструктор копіювання, деструктор та оператор присвоєння повинні бути у відкритому доступі.

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

Контейнери представляють собою типи даних з STL, які можуть містити дані.

Приклад: vector як динамічний масив.

Адаптери представляють собою типи даних з STL, які адаптують інший контейнер (наприклад, вектор або ліст) для надання певного інтерфейсу.

Приклад: stack - забезпечення інтерфейсу стека над обраним контейнером

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

Ітератори — це об'єкти, що реалізують концепцію універсальної обгортки над вказівником і дозволяють одержати доступ до елементів контейнера. Оскільки ітератор є аналогом вказівника, до нього можна застосовувати ті ж самі операції, що і до звичайного вказівника: розіменування, інкремента, декремента і порівняння.

Існує п'ять категорій ітераторів:

1. Ітератори вводу призначені для запису елементів у контейнер. Вони підтримують операції рівності, розіменування та інкремента. ==, !=, \*i, ++i, i++, \*i++

Спеціальним випадком ітератора вводу є клас istream\_iterator.

1. Ітератори виводу призначені для читання елементів із контейнера. Вони підтримують операції розіменування, що можна виконувати лише у лівій частині присвоювання, та інкремента. ++i, i++, \*i = t, \*i++ = t

Спеціальним випадком ітератора вводу є клас ostream\_iterator.

1. Односпрямовані ітератори призначені для перебору елементів контейнера у визначеному напрямку. Вони підтримують всі операції ітераторів вводу/виводу і, крім того, дозволяють без обмеження застосовувати присвоювання. ==, !=, =, \*i, ++i, i++, \*i
2. Двоспрямовані ітератори мають усі властивості односпрямованих ітераторів, а також мають додаткову операцію декремента (--i, i--, \*i--), що дозволяє їм проходити контейнер в обох напрямках.
3. Ітератор прямого (довільного) доступу мають усі властивості двоспрямованих ітераторів, а також підтримують операції порівняння й адресної арифметики, тобто безпосередній доступ по індексі. i += n, i + n, i -= n, i - n, i1 - i2, i[n], i1 < i2, i1 <= i2,

i1 > i2, i1 >= i2

У бібліотеці STL реалізовані також зворотні ітератори, роль яких можуть відігравати або двоспрямовані ітератори, або ітератори довільного доступу, але обходячи послідовність у зворотному напрямку.

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

Операції «++», «\*», «==», «!=», «=» можна виконувати над усіма типами ітераторів.

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

Ітератори довільного доступу мають усі властивості двоспрямованих ітераторів. Крім того, вони можуть забезпечити довільний доступ. Зокрема, вони виконують операції арифметики ітераторів (за аналогією з арифметикою звичайних вказівників). Можна складати і віднімати зсув, обчислювати різниці і порівнювати ітератори за допомогою операцій порівняння, наприклад < і >. Ітератори контейнерних класів vector, deque, array і string є ітераторами довільного доступу.

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

Параметрами для алгоритмів, як правило, служать ітератори. Алгоритмом все одно до типу переданого йому ітератора. Головне, щоб останній підпадав під певну категорію. Приміром, якщо параметром алгоритму має бути односпрямований ітератор, то підставлений ітератор повинен бути або односпрямованим, або двонаправленим, або ж ітератором довільного доступу.

Є п'ять категорій ітераторів в залежності від операцій, визначених для них: введення (input iterators), виведення (output iterators), послідовні (forward iterators), двонаправлені (bidirectional iterators) і довільного доступу (random access iterators.)

Наприклад:

ітератор введення (це типу навіть просто вказівник)

void printValue(int num)

{

cout << num << "";

}

int main()

{

int init[] = {1, 2, 3, 4, 5};

for\_each(init, init + 5, printValue);

}

ітератор виведення

int main()

{

int init1[] = {1, 2, 3, 4, 5};

int init2[] = {6, 7, 8, 9, 10};

vector v(10);

merge(init1, init1 + 5, init2, init2 + 5, v.begin());

copy(v.begin(), v.end(), ostream\_iterator(cout, ""));

}

послідовний ітератор – алгоритм replace

template

void replace (ForwardIterator first, ForwardIterator last, const T& old\_value,

const T& new\_value)

{

while (first != last)

{

if (\*first == old\_value) \*first = new\_value;

++first;

}

}

На двонапрямлених алгоритмах базуються різні алгоритми, які виконують реверсивні операції, наприклад reverse

main(void)

{

int init[] = {1, 2, 3, 4, 5};

reverse(init, init + 5);

copy(init, init + 5, ostream\_iterator(cout, "

"));

}

Як правило, всі складні алгоритми, що вимагають розширених обчислень, оперують ітераторами довільного доступу.

int main(void)

{

const int init[] = {1, 2, 3, 4, 5};

vector v(5);

typedef vector::iterator vectItr;

vectItr itr ;

copy(init, init + 5, itr = v.begin());

cout << \*( itr + 4 ) << endl;

cout << \*( itr += 3 ) << endl;

cout << \*( itr -= 1) << endl;

cout << \*( itr = itr - 1) << endl;

cout << \*( -itr ) << endl;

for(int i = 0; i < (v.end() - v.begin()); i++)

cout << itr[i] << space;

cout << endl;

}

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

В Стандартній бібліотеці C++, алгоритми це компоненти, що виконують алгоритмічні операції над контейнерами і іншими послідовностями Стандарт C++ містить деякі стандартні алгоритми зібрані в файлі заголовку <algorithm>.

Приклади:

* OutputIterator copy(InputIterator source\_begin, InputIterator source\_end,OutputIterator destination\_begin)
* void fill(ForwardIterator destination\_begin, ForwardIteratordestination\_end, T value)
* InputIterator find(InputIterator begin, InputIterator end, T search\_object) (повертає ітератор знайденого об'єкту або end, якщо об'єкт не знайдено)
* const T& max(const T& a, const T& b) повертає більший з двох аргументів
* ForwardIterator max\_element(ForwardIterator begin, ForwardIterator end) знаходить максимальний елемент в наборі
* const T& min(const T& a, const T& b) повертає менший з двох аргументів
* ForwardIterator min\_element(ForwardIterator begin, ForwardIterator end) знаходить мінімальний елемент в наборі

Також є алгоритми, у яких операції не не змінюють послідовності (count, search), операції що змінюють послідовність (наприклад, replace), сортування (sort, stable\_sort), бінарний пошук (upper\_bound), купа ( make\_heap, push\_heap).

**10. Які алгоритми вимагають впорядкованості?**

Алгоритми, які вимагають впорядкованості: бінарний пошук, злиття, lower\_bound(upper\_bound), difference, merge, ), inplace\_merge, equal\_range, set\_union, set\_intersection, set\_difference, set\_symmetric\_difference.