Лабораторна робота №7. Стандартна бібліотека шаблонів STL, контейнери та ітератори

МЕТА РОБОТИ: засвоїти технологію узагальненого об’єктного програмування з використанням бібліотеки стандартних шаблонів (STL) мови C++ для роботи з вбудованими і користувацькими типами даних.

7.1. Програма роботи

7.1.1. Отримати завдання.

7.1.2. Написати програми відповідних класів, основну та відповідні допоміжні функції, згідно з вказівками до виконання роботи.

7.1.3. Підготувати власні коректні вхідні дані (вказати їх формат і значення) і проаналізувати їх.

7.1.4. Оформити електронний звіт про роботу та захистити її.

7.2. Вказівки до виконання роботи

7.2.1. Студент, згідно з індивідуальним номером, вибирає своє завдання з розд. 7.5 і записує його до звіту.

7.2.2. Оголошення класу (структури), основну та відповідні допоміжні функції необхідно запрограмувати так, як це показано у розд. 7.4.

7.2.3. Власних вхідних даних необхідно підготувати не менше двох комплектів. Їхні значення мають бути коректними, знаходитися в розумних межах і відповідати тим умовам, які стосуються індивідуального завдання.

7.2.4. Звіт має містити такі розділи:

* мету роботи та завдання з записаною умовою задачі;
* коди всіх використовуваних .h і .ccp файлів, а також пояснення до них;
* результати реалізації програми;
* діаграма класів і варіантів використання;
* висновки, в яких наводиться призначення програми, обмеження на її застосування і можливі варіанти удосконалення, якщо такі є.

7.3. Теоретичні відомості

Бібліотека стандартних шаблонів STL (Standard Template Library) є однією із найдосконаліших інструментів програмування мовою С++.

STL є бібліотекою контейнерних класів С++. До складу бібліотеки входять такі компоненти:

* **контейнер** – об’єкт, який містить інші об’єкти, організовані у вигляді послідовностей (колекції об’єктів);
* **ітератор** **–** забезпечує доступ до вмісту контейнера;
* **алгоритм** – функція, яка забезпечує обчислювальну процедуру над елементами контейнера;
* **алокатор** – забезпечує керування динамічною пам’яттю, виділеною для STL – об’єкта;
* **функтом –** інкапсулює функцію в об’єкті для використання іншими компонентами;
* **предикат –** функція булевого типу, яка перевіряє визначені програмістом властивості або відношення між об’єктами;
* **адаптер** – пристосовує компонент бібліотеки для забезпечення різних інтерфейсів.

Ядром бібліотеки є взаємозалежні контейнери, алгоритми та ітератори.

**Контейнери** забезпечують керування колекціями елементів. Контейнери поділяють на базові та асоціативні. До базових контейнерів належать динамічний масив (вектор), список та черга. Асоціативні контейнери складаються з відображення елементів та множин. Асоціативні контейнери містять набір пар значень, одне з яких є ключем, що забезпечує доступ до відповідних йому даних. Колекції асоціативних контейнерів впорядковані за значенням ключа.

У контейнерному класі визначено методи для роботи з інкапсульованими елементами. Наприклад, для вставки, видалення або злиття елементів.

**Ітератори** – це об’єкти, які забезпечують доступ до елементів контейнера. Оголошуються за допомогою слова iterator. Подібні до вказівників, які забезпечують доступ до елементів масиву. Над ітераторами можна виконувати операції, призначені для роботи з вказівниками, наприклад, інкремент, декремент, розіменування тощо.

За призначенням розрізняють такі види ітераторів:

* **однонапрямлений –** для послідовного читання і записування елементів контейнера, може проходити контейнер тільки в одному напрямку;
* **введення –** тільки для послідовного читання елементів контейнера в одному напрямку;
* **виведення –** тільки для послідовного записування значень у контейнер в одному напрямку;
* **двонапрямлений –** для послідовного читання і записування елементів контейнера, може проходити контейнер в обох напрямках;
* **довільного доступу –** для читання і записування елементів контейнера у довільному порядку; забезпечують прямий доступ до елементів контейнера;
* **зворотний –** послідовний двонапрямлений та довільного доступу, який проходить послідовність елементів з кінця;інкремент такого ітератора призводить до переміщення від кінця до початку послідовності елементів.

**Алгоритми –** як правило, параметризовані зовнішні функції, призначені для виконання дій над елементами контейнера, наприклад, ініціалізація, пошук, сортування, заміна елементів.

**Класи бiблiотеки стандартних шаблонів**

|  |  |  |
| --- | --- | --- |
| **Класи** | **Файли включення #include** | **Призначення** |
| vector | <vector> | одновимірний масив елементів |
| valarray | <valarray> | масив для математичних обчислень |
| complex | <complex> | комплексне число |
| list | <list> | лінійний список |
| stack | <stack> | стек |
| queue | <queue> | черга |
| priority\_queue | <queue> | пріоритетна черга |
| deque | <deque> | черга з двома кінцями |
| map | <map> | відображення (карта, словник) без повторень елементів |
| multimap | <map> | відображення з повтореннями елементів |
| set | <set> | множина без повторення елементів |
| multiset | <set> | множина з повтореннями елементів |
| basic\_string | <string> | базовий клас для роботи з рядками |
| string | <string> | спеціалізація basic\_string<char> |
| basic\_istream | <istream> | базовий клас для роботи з потоками введення |
| istream | <istream> | спеціалізація basic\_istream<char> |
| basic\_ostream | <ostream> | базовий клас для роботи з потоками виведення |
| ostream | <ostream> | спеціалізація basic\_ostream<char> |
| basic\_iostream | <iostream> | базовий клас для роботи з потоками виведення |
| iostream | <iostream> | спеціалізація basic\_iostream<char> |
| basic\_ifstream | <ifstream> | базовий клас для роботи з файловими потоками введення |
| ifstream | <ifstream> | спеціалізація basic\_ifstream<char> |
| basic\_ofstream | <ofstream> | базовий клас для роботи з файловими потоками виведення |
| ofstream | <ofstream> | спеціалізація basic\_ofstream<char> |
| basic\_fstream | <fstream> | базовий клас для роботи з файловими потоками введення – виведення |
| fstream | <fstream> | спеціалізація basic\_fstream<char> |

Контейнерні класи бібліотеки шаблонів містять перевантажені конструктори, методи та операції для роботи з інкапсульованими елементами. Це робить можливими різні форми створення та ініціалізації об’єктів, виклики методів з різними списками параметрів.

**Асоціативні контейнери** (associative containers) дозволяють за допомогою ключів отримати швидкий доступ до значень, що зберігаються в них.

У кожному класі-контейнері визначений набір функцій для роботи з ними. Наприклад, список містить функції для вставки, видалення й злиття елементів.

**Ітератори** (iterators) – це об'єкти, які стосовно контейнера відіграють роль покажчиків. Вони дозволяють одержати доступ до вмісту контейнера приблизно так само, як покажчики використовуються для доступу до елементів масиву.

З ітераторами можна працювати так само, як з вказівниками. До них можна застосувати операції \*, інкремента, декремента. Типом ітератора оголошується тип *iterator*, що визначений у різних контейнерах.

Існує п'ять типів ітераторів:

**1.Ітератори введення** (input\_iterator) підтримують операції рівності, розіменування та інкремента: ==, !=, \*i, ++i, i++, \*i++. Спеціальним випадком ітератора введення є istream\_iterator.

**2.Ітератори виведення** (output\_iterator) підтримують операції розіменування, припустимі тільки з лівої сторони присвоювання, та інкремента: ++i, i++, \*i=t, \*i++=t. Спеціальним випадком ітератора виводу є ostream\_iterator.

**3.Односпрямовані ітератори** (forward\_iterator) підтримують всі операції ітераторів уведення/виводу й, крім того, дозволяють без обмеження застосовувати присвоювання: ==, !=, =, \*i, ++i, i++, \*i++.

**4.Двонаправлені ітератори** (biderectional\_iterator) мають всі властивості forward-ітераторів, а також мають додаткову операцію декремента (--i, i--, \*i--), що дозволяє їм проходити контейнер в обох напрямках.

**5.Ітератори довільного доступу** (random\_access\_iterator) мають всі властивості biderectional-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.

В STL також підтримуються **зворотні ітератори** (reverse iterators). Зворотними ітераторами можуть бути або двонаправлені ітератори, або ітератори довільного доступу, але мають послідовність у зворотньому напрямку.

На додачу до контейнерів, алгоритмів та ітераторів в STL підтримується ще кілька стандартних компонентів. Головними серед них є **розподільники пам'яті**, **предикати** й **функції** **порівняння**.

У кожного контейнера є визначений для нього розподільник пам'яті (**allocator**), що управляє процесом виділення пам'яті для контейнера.

За замовчуванням розподільником пам'яті є об'єкт класу **allocator**. Можна визначити власний розподільник.

У деяких алгоритмах і контейнерах використовується функція особливого типу, названа **предикатом**. Предикат може бути унарним і бінарним. Значення, що повертає: істина або неправда. Точні умови одержання того або іншого значення визначаються програмістом. Тип унарних предикатів **UnPred**, бінарних – **BinPred**. Тип аргументів відповідає типу об'єктів, що зберігаються в контейнері.

Визначено спеціальний тип бінарного предиката для порівняння двох елементів. Він називається **функцією порівняння** (comparison function). Функція повертає істину, якщо перший елемент менше другого. Типом функції є тип **Comp**.

Особливу роль в STL грають об'єкти-функції.

Об'єкти-функції ( це екземпляри класу, у якому визначена операція «круглі дужки» (). У ряді випадків зручно замінити функцію на об'єкт-функцію. Коли об'єкт-функція використовується як функція, то для її виклику використовується operator ().

**Класи-контейнери.** В STL визначені два типи контейнерів: послідовності й асоціативні.

Ключова ідеядля стандартних контейнерів полягає в тому, що коли це представляється розумним, вони повинні бути логічно взаємозамінними. Користувач може вибирати між ними, ґрунтуючись на міркуваннях ефективності й потреби в спеціалізованих операціях. Наприклад, якщо часто потрібен пошук по ключу, можна скористатися **map** (асоціативним масивом). З іншого боку, якщо переважають операції, характерні для списків, можна скористатися контейнером **list**. Якщо додавання й видалення елементів часто проводиться в кінці контейнера, варто подумати про використання черги **queue**, черги із двома кінцями **deque**, стека **stack**. За замовчуванням користувач повинен використати **vector**; він реалізований, щоб добре працювати для самого широкого діапазону завдань.

Ідея обігу з різними видами контейнерів і, у загальному випадку, з усіма видами джерел інформації – уніфікованим способом веде до поняття **узагальненого програмування**. Для підтримки цієї ідеї STL містить множина узагальнених алгоритмів. Такі алгоритми рятують програміста від необхідності знати подробиці окремих контейнерів.

**Огляд операцій**

Типи

**value\_type** тип елемента

**allocator\_type** тип розподільника пам'яті

**size\_type**  тип індексів, лічильника елементів і т.д.

**iterator** поводиться як value\_type\*

**reverse\_iterator** переглядає контейнер у зворотному порядку

**reference** поводиться як value\_type&

**key\_type** тип ключа (тільки для асоціативних контейнерів)

**key\_compare** тип критерію порівняння (тільки для асоціативних контейнерів)

**mapped\_type** тип відображеного значення

Ітератори

**begin()** вказує на перший елемент

**end()** указує на елемент, що випливає за останнім

**rbegin()** вказує на перший елемент у зворотній послідовності

**rend()** вказує на елемент, що випливає за останнім у зворотній послідовності

Доступ до елементів

**front()** посилання на перший елемент

**back()** посилання на останній елемент

**operator[]** (i)доступ по індексі без перевірки

**at**(i) доступ по індексі з перевіркою

Включення елементів

**insert(**p,x**)** додавання х перед елементом, на який указує р

**insert(**p,n,x**)** додавання n копій х перед р

**insert(**p,first,last**)** додавання елементів з [first:last] перед р

**push\_back(**x**)** додавання х у кінець

**push\_front(**x**)** додавання нового першого елемента (тільки для списків і черг із двома кінцями)

Видалення елементів

**pop\_back()** видалення останнього елемента

**pop\_front()** видалення першого елемента (тільки для списків і черг із двома кінцями)

**erase(**p**)** видалення елемента в позиції р

**erase(**first,last**)** видалення елементів з [first:last]

**clear()** видалення всіх елементів

Інші операції

**size()** число елементів

**empty()**  контейнер порожній?

**Capacity** пам'ять, виділена під вектор (тільки для векторів)

**reserve(**n**)** виділяє пам'ять для контейнера під n елементів

**resize(**n**)** змінює розмір контейнера (тільки для векторів, списків і черг із двома кінцями)

**swap(**x**)** обмін місцями двох контейнерів

**==**, **!=**, **<** операції порівняння

Операції присвоювання

**operator=(**x**)** контейнеру присвоюються елементи контейнера х

**assign(**n,x**)** присвоювання контейнеру n копій елементів х (не для асоціативних контейнерів)

**assign(**first,last**)** присвоювання елементів з діапазону [first:last]

Асоціативні операції

**operator[](**k**)** доступ до елемента із ключем k

**find(**k**)** знаходить елемент із ключем k

**lower\_bound(**k**)**  знаходить перший елемент із ключем k

**upper\_bound(**k**)**  знаходить перший елемент із ключем, більшим k

**equal\_range(**k**)** знаходить lower\_bound (нижню границю) і upper\_bound (верхню границю) елементів із ключем k

Контейнер *vector-вектор*.

Вектор vector в STL визначений як динамічний масив з доступом до його елементів по індексу.

*template<class T,class Allocator=allocator<T>>class std::vector{...};*

де *T* – тип призначених для зберігання даних.

*Allocator* задає розподільник пам'яті, що за замовчуванням є стандартним.

Приклад

*vector<int> a;*

*vector<double> x(5);*

*vector<char> c(5,'\*');*

*vector<int> b(a); //b=a*

Для будь-якого об'єкта, що буде зберігатися у векторі, має бути визначений конструктор за замовчуванням. Крім того, для об'єкта повинні бути визначені оператори < і ==.

Для класу вектор визначені наступні оператори порівняння: ==, <, <=, !=, >, >=.

Крім цього, для класу vector визначається оператор індексу [].

1. Нові елементи можуть включатися за допомогою функцій

*insert(), push\_back(), resize(), assign().*

1. Існуючі елементи можуть видалятися за допомогою функцій

*erase(), pop\_back(), resize(), clear().*

1. Доступ до окремих елементів здійснюється за допомогою ітераторів

*begin(), end(), rbegin(), rend(),*

1. Маніпулювання контейнером, сортування, пошук у ньому тощо можливо за допомогою глобальних функцій файлу ( заголовка <algorithm.h>).

Асоціативні контейнери (масиви).

Асоціативний масив містить пари значень. Знаючи одне значення, що називається **ключем** (key), ми можемо дістати доступ до іншим, називаним **відображеним значенням** (mapped value).

Асоціативний масив можна представити як масив, для якого індекс не обов'язково повинен мати цілочисельний тип:

V& operator[](const K&) повертає посилання на V, що відповідає K.

Асоціативні контейнери - це узагальнення поняття асоціативного масиву.

Асоціативний контейнер **map** − це послідовність пар (ключ, значення), що забезпечує швидке одержання значення по ключі. Контейнер map надає двонаправлені Ітератори.

Асоціативний контейнер map вимагає, щоб для типів ключа існувала операція “<”. Він зберігає свої елементи відсортованими по ключі так, що перебір відбувається один по одному.

Специфікація шаблона для класу map:

template<class Key,class,class Comp=less<Key>,class Allocator=allocator<pair> >

class std::map

Визначено операцію присвоювання:

map& **operator=**(const map&);

Визначено наступні операції: ==, <, <=, !=, >, >=.

В map зберігаються пари ключ/значення у вигляді об'єктів типу **pair**.

Створювати пари ключ/значення можна не тільки за допомогою конструкторів класу pair, але й за допомогою функції **make\_pair**, що створює об'єкти типу pair, використовуючи типи даних як параметри.

Типова операція для асоціативного контейнера - це асоціативний пошук за допомогою операції індексації ([]).

mapped\_type& **operator[]**(const key\_type& K);

Множина **set** можна розглядати як асоціативні масиви, у яких значення не грають ролі, так що ми відслідковуємо тільки ключі.

template<class,class Cmp=less<T>,class Allocator=allocator<T>>class std::set{...};

Множина, як і асоціативний масив, вимагає, щоб для типу T існувала операція “менше” (<). Воно зберігає свої елементи відсортованими, так що перебір відбувається один по одному.

**Приклад програми:**

Розробити телефонну книгу на основі класу Особа (Person) та контейнерного класу Множина з повтореннями елементів (multiset). Ввести прізвище Особи та визначити номер її телефону. Визначити кількість осіб, що мають одинакове прізвище та ім’я, і вивести на екран номери їхніх телефонів:

#include <iostream>

#include <set>

#include <string>

using namespace std;

// Клас Особа

class Person {

string LastName; // Прізвище

string FirstName; // Ім’я

long PhoneNumber; // Номер телефону

public:

// Конструктор

Person (string LN= “ ”, string FN= “ ”, long PN=0):

LastName (LN), FirstName (FN), PhoneNumber (PN) { }

// Перевантажена операція new

void \* operator new (size\_t, void \* ptr)

{ return ptr; }

// Перевантажена операція ==

friend bool operator == (const Person& P1, const Person& P2)

{ return (P1.LastName == P2.LastName) && (P1.FirstName == P2.FirstName) ? true:false; }

// Перевантажена операція <

friend bool operator < (const Person& P1, const Person& P2)

{ if (P1.LastName == P2.LastName)

return (P1.FirstName < P2.FirstName) ? true:false;

return (P1.LastName < P2.LastName) ? true:false; }

// Перевантажена операція потокового введення

friend istream& operator >> (istream& is, Person& P)

{ is >> P.LastName >> P.FirstName >> P.PhoneNumber;

return is; }

// Перевантажена операція потокового виведення

friend ostream& operator << (ostream& os, Person& P)

{ os << P.LastName << ‘\t’ << P.FirstName << ‘\t’ << P.PhoneNumber << endl;

return os; }

};

void main ( )

{

// Множина з повтореннями елементів

multiset <Person, less <Person> > S;

// Об’єкт класу Person

Person P;

cout << “Введіть дані про власників телефонів” <<endl;

while (cin>>P) // Введення до натиснення Ctrl/Z

S.insert (P);

// Очищення потоку введення від ознаки EOF(Ctrl/Z)

cin.clear ( );

cout << “Число записів =”<<S.size ( )<<endl;

cout << “Склад множини”<<endl;

multiset <Person, less <Person> > :: iterator i;

for (i = S.begin (); i != S.end (); ++i)

cout <<\*i<<endl;

cout<< “Пошук номера телефону”<<endl;

string LN, FN;

cout<< “Введіть прізвище особи”<<endl;

cin>>LN;

cout<< “Введіть ім’я”<<endl;

cin>>FN;

// Повторна ініціалізація об’єкта

new (&P) Person (LN, FN, 0);

cout<< “Кількість осіб з таким прізвищем та іменем”<<endl;

cout<<S.count (P)<<endl;

cout<< “Всі записи, що відповідають запиту”<<endl;

for (i = S.lower\_bound (P); i!=S.upper\_bound (P); ++i)

cout<<\*i<<endl;

}

7.4. Індивідуальні завдання

1. Написати програму – список групи студентів. Для цього побудувати асоціативний список map для збереження імен, проіндексованих прізвищами студентів. За заданим прізвищем знайти та вивести на екран ім’я студента.
2. Написати шаблон класу, який використовує шаблонний клас vector для реалізації стекової структури даних. Продемонструвати операції занесення елемента у стек, вилучення елемента зі стека, копіювання стека, виведення вмісту стека.
3. Створити клас Employee про облік робочого часу працівників: кількість відпрацьованих годин, погодинна оплата. Визначити метод для визначення величини зарплати працівника. Згенерувати випадкові значення відпрацьованого часу та значення погодинної оплати. Зберегти ці значення у контейнері vector <Employee\*>. Обчислити сумарні витрати компанії на оплату праці.
4. Заповнити контейнер multiset<int> випадковими цілими числами від 0 до 20. Підрахувати кількість повторень кожного елемента множини.
5. Написати програму обліку книг бібліотеки на основі контейнерного класу multimap. Дані про книги складаються з прізвища та ініціалів автора, назви книги, року видання. Необхідно забезпечити додавання інформації про нові надходження книг, видалення даних про списані книги, пошуку та виведення всіх книг заданого автора або року видання.
6. Згенерувати послідовність цілих випадкових чисел та зберегти їх у масиві. Ініціалізувати контейнер valarray <int> вмістом цього масиву. Обчислити суму, мінімальне та максимальне значення, середнє арифметичне цілих чисел засобами контейнера valarray.
7. На основі контейнера stack побудувати стек з інформацією про успішність студентів (прізвище, ідентифікаційний номер, рейтинг). Занести у стек дані про групу студентів. Знайти у стеку дані про одного із студентів. Очистити стек до заданого прізвища.
8. Написати програму обліку продукції на складі. Дані про наявну на складі продукцію задаються у списку асортименту list, який побудовано на основі ідентифікаторів виробів. Інший список містить інформацію про замовлену продукцію. Якщо на склад надходить новий виріб, то дані про нього заносять у список асортименту. Потім переглядають список замовлень. Якщо замовлення може бути задоволено, то відбувається корегування обох списків.
9. На основі контейнера string побудувати клас для роботи з масивом рядків символів. Заповнити масив введеними з клавіатури рядками. Відсортувати масив рядків символів. Підрахувати кількість слів у кожному рядку.
10. На основі контейнера vector побудувати двовимірний масив цілих чисел. Виконати введення та виведення масиву, пошук максимального та мінімального елементів.
11. Написати програму – алфавітний покажчик у вигляді списку впорядкованих за алфавітом слів, кожному з яких відповідають номери рядків тексту, у яких зустрічається задане слово. Для цього розібрати заданий текси на слова і побудувати індексований словами контейнерний клас multimap.За заданим словом вивести номери рядків, що містять це слово.
12. На основі контейнера stack побудувати стек із чисел. Занести дані у стек з імовірністю µ, а зчитувати – з імовірністю η . Для заданих ймовірностей визначити середній максимально необхідний розмір стека.
13. Створити шаблонний клас матриці на основі вектора vector<vector<T>>. Визначити операції введення та виведення матриці. Створити операції: додавання матриць, множення матриць, множення матриці на вектор. Продемонструвати роботу шаблону для типів int та float.
14. Написати програму, яка отримує з командного рядка ім’я текстового файла. Відкрити файл та прочитати його по словах. Зберегти слова у vector<string>. Перетворити усі слова до верхнього регістра, відсортувати слова, видалити повторення слів та вивести результат.
15. На основі контейнера set побудувати множину з унікальними елементами. Продемонструвати операції об’єднання, різниці, перетину, відношення, перевірки входження елемента у множину, введення – виведення множин.

7.5. Контрольні запитання

1. Які засоби містить стандартна бібліотека шаблонів?
2. Чи всі контейнери працюють з ітератором довільного доступу?
3. Для чого використовується контейнер STL?
4. Який зв'язок між контейнерами, літераторами і алгоритмами?
5. Що таке предикат і що він повертає?
6. Що потрібно додати до програми, щоб створити контейнер користувацького типу і працювати з ним?
7. Поясніть як працює контейнер з двоспрямованим послідовним доступом? Наведіть приклад.
8. Що таке асоціативний контейнер? Наведіть приклад.