Лекція 12

Бібліотека STL

12.1 Загальна структура бібліотеки
12.2. Контейнери
12.3. Алгоритми
12.4. Ітератори
12.5. Розподільники, предикати і функтори
12.6. Типова організація класу контейнера

12.1. Загальна структура бібліотеки

Бібліотека STL складає половину стандарту C++ 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 елементі 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.

Алгоритми виконують операції над елементами, що містяться у контейнері. Основними алгоритмами є заповнення контейнера за відповідним правилом, сортування за вказаним критерієм, пошук за заданим критерієм, заміна елементів контейнера. Алгоритми визначені в заголовному файлі <algorithm.h>. Нижче приведені імена деяких найбільше часто використовуваних функцій-алгоритмів STL.

Ітератори — це об'єкти, що реалізують концепцію інтелектуального вказівника 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.

12.2. Контейнери

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.
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, що реалізує концепцію стеку — лінійну структуру даних, організовану за принципом LIFO (останнім увійшов — першим вийшов);
queue	Контейнер, що реалізує концепцію черги — лінійної структури даних, що добавляються і видаляються за принципом FIFO (першим увійшов — першим пішов).
priority queue	Контейнер, що реалізує концепцію черги за пріоритетами, тобто черги, елементи якої відсортовані за пріоритетом.

12.3. Алгоритми

Алгоритми розподіляються на ті, що не модифікують елементи контейнера і ті, що модифікують. Бібліотека STL базується на принципі інкапсуляції та приховання даних. Всі алгоритми працюють з методами контейнерів, нічого не знаючи про те, як вони реалізовані.

Операції, що немодифікують	
`for_earch()`	Виконує операції для кожного елемента послідовності
`find()`	Знаходить перше входження значення в послідовність
`find_if()`	Знаходить перша відповідність предикату в послідовності
`count()`	Підраховує кількість входжень значення в послідовність
`count_if()`	Підраховує кількість виконань предиката в послідовності
`search()`	Знаходить перше входження послідовності як підпослідовності
`search_n()`	Знаходить n-і входження значення в послідовність

Операції, що модифікують	
`copy()`	Копіює послідовність, починаючи з першого елемента
`swap()`	Міняє місцями два елементи
`replace()`	Замінює елементи з заданим значенням
`replace_if()`	Замінює елементи при виконанні предиката
`replace_copy()`	Копіює послідовність, заміняючи елементи із заданим значенням
`replace_copy_if()`	Копіює послідовність, заміняючи елементи при виконанні предиката
`fill()`	Заміняє всі елементи даним значенням
`remove()`	Видаляє елементи з даним значенням
`remove_if()`	Видаляє елементи при виконанні предиката
`remove_copy()`	Копіює послідовність, видаляючи елементи із заданим значенням
`remove_copy_if()`	Копіює послідовність, видаляючи елементи при виконанні предиката
`reverse()`	Змінює порядок проходження елементів на зворотний
`random_shuffle()`	Переміщає елементи відповідно до випадкового рівномірного розподілу (“перетасовує” послідовність)
`transform()`	Виконує задану операцію над кожним елементом послідовності
`unique()`	Видаляє рівні сусідні елементи
`unique_copy()`	Копіює послідовність, видаляючи рівні сусідні елементи

Сортування	
`sort()`	Сортує послідовність з гарною середньою ефективністю
`partial_sort()`	Сортує частина послідовності
`stable_sort()`	Сортує послідовність, зберігаючи порядок проходження рівних елементів
`lower_bound()`	Знаходить перше входження значення у відсортованій послідовності
`upper_bound()`	Знаходить перший елемент, більший чим задане значення
`binary_search()`	Визначає, чи є даний елемент у відсортованій послідовності
`merge()`	зливає дві відсортовані послідовності

Операції над множинами	
`includes()`	Перевірка на входження
`set_union()`	Об'єднання множин
`set_intersection()`	Перетинання множин
`set_difference()`	Різниця множин

Мінімуми і максимуми	
`min()`	Менший з двох елементів
`max()`	Більший з двох
`min_element()`	Найменше значення у послідовності
`max_element()`	Найбільше значення у послідовності

12.4. Ітератори

Ізольованого типу ітератора не існує, тому що він визначається у різних контейнерах залежно від організації самого контейнера. 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.
2. 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.
3. 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
```

4. Двоспрямовані ітератори мають усі властивості односпрямованих ітераторів, а також мають додаткову операцію декремента (`--i`, `i--`, `*i--`), що дозволяє їм проходити контейнер в обох напрямках.
5. 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.

12.5. Розподільники, предикати і функтори

Крім до контейнерів, алгоритмам і ітераторів у STL підтримується ще кілька стандартних компонентів. Головними серед них є розподільники пам'яті, предикати і функції порівняння.

У кожного контейнера мається визначений для нього розподільник пам'яті, що керує 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, називана предикатом. Предикат може бути унарним і бінарним. Значення, що повертається: істина або хибність. Точні умови одержання того чи іншого значення визначаються програмістом. Тип аргументів відповідає типу об'єктів, що зберігаються в контейнері. Визначено спеціальний тип бінарного предиката для порівняння двох елементів. Він називається функцією порівняння. Функція повертає істину, якщо перший елемент менше другого.

Особливу роль у STL грають об'єкта-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 функцію, то для її виклику використовується operator().

```
class less
 {
 public:
  bool operator()(int x, int y)
  {
    return x < y;
   }
 };
```

У STL визначені два типи контейнерів: послідовності й асоціативні. Ключова ідея для стандартних контейнерів полягає в тім, що коли це представляється розумним, вони повинні бути логічно взаємозамінними. Користувач може вибирати між ними, ґрунтуючись на розумінні ефективності і потреби в спеціалізованих операціях. Наприклад, якщо часто потрібно пошук по ключу, можна скористатися map (асоціативним масивом). З іншого боку, якщо переважають операції, характерні для списків, можна скористатися контейнером list. Якщо додавання і видалення елементів часто виробляється в кінці контейнера, варто подумати про використання черги queue, черги з двома кінцями deque, стека stack. За замовчуванням користувач повинний використовувати vector ; він реалізований, щоб добре працювати для самого широкого діапазону задач.

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

12.6. Розподільники, предикати і функтори

Класі контейнера, як правило, містить оголошення наступних типів.

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 как 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()	Указує на перший елемент
end()	Указує на елемент, що розтащований за останнім
rbegin()	Указує на перший елемент у зворотній послідовності
rend()	Указує на елемент, що розташований за останнім у зворотній послідовності

Доступ до елементів	
front()	Посилання на перший елемент
back()	Посилання на останній елемент
operator [](i)	Доступ по індексу без перевірки
at(i)	Доступ по індексу з перевіркою

Включення елементів	
insert(p, x)	Додавання x перед елементом, на який указує p
insert(p, n, x)	Додавання n копій x перед p
insert(p, first, last)	Додавання елементів з [first:last] перед p
push_back(x)	Додавання x у кінець
push_front(x)	Додавання нового першого елемента (тільки для списків і деку)

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 p
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 привласнюються елементи контейнера x
assign(n, x)	Присвоювання контейнеру n копій елементів x (не для асоціативних контейнерів)
assign(first, last)	Присвоювання елементів з діапазону [first:last]

Асоціативні операції	
operator [](k)	Доступ до елемента з 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, великим k
equal_range(k)	Знаходить lower_bound (нижню границю) і upper_bound (верхню границю) елементів із ключем k

Приклад вектора

```
#include <iostream>
#include <vector>

using namespace std ;

void print(vector<int> &array);

int main()
{
   const int SIZE = 10;
   vector<int> array;    // Порожній вектор

   // Заповнюємо вектор за допомогою функції-члена push_back()

   for (int i = 0; i < SIZE; i++)  array.push_back(i);

   // Виводимо вектор на екран
   cout << "Вихідний вектор" << endl;
   print(array);

   // Видаляємо 5-й елемент вектора

   array.erase(array.begin() + 4);
   cout << "Після видалення числа 4" << endl;
   print(array);

   // Видаляємо діапазон [6,9)
   array.erase(array.begin()+5, array.begin()+8);
   cout << "Після видалення чисел 6, 7 й 8 << endl;;
   print(array);

   // Виводимо на печатку зарезервований розмір вектора
   int arrayMaxSize = array.max_size();
   cout << "Максимальний розмір вектора = " << arrayMaxSize;

   // Виводимо на друк розмір вектора
```

```
   int arraySize = array.size();
   cout << "Розмір вектора = " << arraySize;

   // Виводимо на друк ємність вектора
   int arrayCapacity = array.size();
   cout << "Ємність вектора = " << arrayCapacity;

   // Вставляємо в діапазон [1,5) чотири п'ятірки й виводимо вектор на печатку
   array.insert(array.begin()+1,4,5);
   cout << "Після вставки чотирьох п'ятірок у діапазон [1,5)" << endl;
   print(array);

   // Видаляємо останній елемент
   cout << "Після видалення останнього елемента" << endl;
   array.pop_back();
   print(array);

   // Повідомляємо новий вектор. Копіюємо в нього вміст вектора array
   vector<int> newArray(array.begin(),array.end());
   cout << "Вектор newArray" << endl;
   print(newArray);

   // Заповнюємо вектор newArray сімками
   newArray.insert(newArray.begin(),newArray.size(),7);
   cout << "Після вставки сімок" << endl;
   print(newArray);

   // Дописуємо у вектор newArray уміст вектора newArray
   newArray.assign(array.begin(),array.end());
   cout << "Після присвоєння вектора array" << endl;
   print(newArray);

   // Стираємо вектор newArray
   newArray.clear();
   cout << "Спроба вивести порожній вектор" << endl;
   print(newArray);

   // Порівняння векторів
   array.clear();
   for (i = 0; i < SIZE; i++)
   {
       array.push_back(i);
       newArray.push_back(i+1);
   }
   cout << "Вектор array:    " << endl;
   print(array);
   cout << "Вектор newArray: " << endl;
   print(newArray);
   if(array == newArray) cout << "array == newArray" << endl;
   if(array >  newArray) cout << "array >  newArray" << endl;
   if(array >= newArray) cout << "array >= newArray" << endl);
   if(array <  newArray) cout << "array <  newArray" << endl);
   if(array <= newArray) cout << "array <= newArray" << endl);

   return 0;
}

void print(vector<int> &array)
{
   if (array.empty())
   {
       cout << "Вектор порожній";
       return;
   }


   vector<int>::iterator iter;
```

```
    for (iter = array.begin(); iter != array.end(); iter++)
         cout << *iter  << endl;
         cout <<  endl ;
}
```

Результат

```
Вихідний вектор
0 1 2 3 4 5 6 7 8 9
Після видалення числа 4
0 1 2 3 5 6 7 8 9
Після видалення чисел 6, 7 й 8
0 1 2 3 5 9
Максимальний розмір = 1073741823
Розмір вектора = 6
Ємність вектора = 6
Після вставки чотирьох п'ятірок у діапазон [1,5)
0 5 5 5 5 1 2 3 5 9
Після видалення останнього елемента
0 5 5 5 5 1 2 3 5
Вектор newArray
0 5 5 5 5 1 2 3 5
Після вставки сімок
7 7 7 7 7 7 7 7 7 0 5 5 5 5 1 2 3 5
Після присвоєння вектора array
0 5 5 5 5 1 2 3 5
Спроба вивести порожній вектор
Вектор порожній
Вектор array:    0 1 2 3 4 5 6 7 8 9
Вектор newArray: 1 2 3 4 5 6 7 8 9 10
array <  newArray
array <= newArray
```

12.7. РЕЗЮМЕ

- Бібліотека STL складає половину стандарту C++ 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 елементі 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.
- *Алгоритми* виконують операції над елементами, що містяться у контейнері. Основними алгоритмами є заповнення контейнера за відповідним правилом, сортування за вказаним критерієм, пошук за заданим критерієм, заміна елементів контейнера. Алгоритми визначені в заголовному файлі <algorithm.h>.
- *Ітератори* — це об'єкти, що реалізують концепцію інтелектуального вказівника 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.
- Концепцію контейнера реалізують класи **`vector`**, **`list`**, **`deque`**, **`set`**, **`multiset`**, **`map`**, **`multimap`**, **`stack`**, **`queue`** і **`priority queue`**.
- Алгоритми розподіляються на ті, що не модифікують елементи контейнера і ті, що модифікують. Бібліотека STL базується на принципі інкапсуляції та приховання даних. Всі алгоритми працюють з методами контейнерів, нічого не знаючи про те, як вони реалізовані.
- До алгоритмів, що не модифікують елементи контейнера, належать **`for_earch()`**, **`find()`**, **`find_if()`**, **`count()`**, **`count_if()`**, **`search()`** і **`search_n()`**.
- До алгоритмів, що модифікують елементи контейнера, належать **`copy(),swap()`**, **`replace()`**, **`replace_if()`**, **`replace_copy()`**, **`replace_copy_if()`**, **`fill()`**, **`remove()`**, **`remove_if()`**, **`remove_copy()`**, **`remove_copy_if()`**, **`reverse()`**, **`random_shuffle()`**, **`transform()`**, **`unique()`** і **`unique_copy()`**.

- До алгоритмів, що упорядковують елементи контейнера, належать **sort()**, **partial_sort()**, **stable_sort()**, **lower_bound()**, **upper_bound()**, **binary_search()** і **merge()**.
- До алгоритмів, що виконують операції над множинами, належать **includes()**, **set_union()**, **set_intersection()** і **set_difference()**.
- До алгоритмів, що знаходять мінімальні і максимальні елементи, належать **min()**,**max()**, **min_element()** і **max_element().**
- Ізольованого типу ітератора не існує, тому що він визначається у різних контейнерах залежно від організації самого контейнера. Існує п'ять категорій ітераторів.
- Ітератори вводу призначені для запису елементів у контейнер. Вони підтримують операції рівності, розіменування та інкремента: `==, !=, *i, ++i, i++, *i++`.
- 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`.
- 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`.
- Двоспрямовані ітератори мають усі властивості односпрямованих ітераторів, а також мають додаткову операцію декремента (`--i, i--, *i--`), що дозволяє їм проходити контейнер в обох напрямках.
- Ітератор прямого (довільного) доступу мають усі властивості двоспрямованих ітераторів, а також підтримують операції порівняння й адресної арифметики, тобто безпосередній доступ по індексу: `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.
- Розподільник пам'яті керує 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, називана предикатом. Предикат може бути унарним і бінарним. Значення, що повертається: істина або хибність. Точні умови одержання того чи іншого значення визначаються програмістом. Тип аргументів відповідає типу об'єктів, що зберігаються в контейнері. Визначено спеціальний тип бінарного предиката для порівняння двох елементів. Він називається функцією порівняння. Функція повертає істину, якщо перший елемент менше другого.
- Особливу роль у STL грають об'єкта-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 функцію, то для її виклику використовується operator().
- Класі контейнера, як правило, містить оголошення наступних типів.
- У кожному контейнері визначені службові типи: **value_type**, **allocator_type**, **size_type**, **iterator**, **reverse_iterator**, **reference**, **key_type**, **key_compare** і **mapped_type.**
- У кожному контейнері визначені члени, що повертають ітератори: **begin()**, **end()**, **rbegin()** і **rend()**.
- У кожному контейнері доступ до елементів здійснюється функціями-членами **front()**, **back()**, **operator [](i)** і **at(i)**.
- Включення елементів у контейнер здійснюється функціями-членами **insert(p, x)**, **insert(p, n, x)**, **insert(p, first, last)**, **push_back(x)** і **push_front(x)**.
- Видалення елементів у контейнер здійснюється функціями-членами. **pop_back()**, **pop_front()**, **erase(p)**, **erase(first, last)** і **clear()**.
- До службових операцій контейнера належить функції **size()**, **empty()**, **capacity()**, **reserve(n)**, **resize(n), swap(x)**, а також **==, !=, <**.
- Контейнери можна присвоювати за допомогою функцій-членів **operator =(x)**, **assign(n, x)** і **assign(first, last)**.
- В асоціативних контейнерах реалізовані такі операції: **operator [](k)**, **find(k)**, **lower_bound(k)**, **upper_bound(k)** і **equal_range(k).**

12.8. КОНТРОЛЬНІ ПИТАННЯ

1. Назвіть чотирьох категорії сутностей, що складають бібліотеку STL.
2. Що таке контейнери? Наведіть приклади.

3. Опишіть організацію контейнера.
4. Що таке стандартні алгоритми? Наведіть приклад.
5. Що таке ітератори? Які операції можна виконувати над усіма типами літераторів?
6. Які класи належать до категорії контейнерів?
7. На які групи розподіляються алгоритми?
8. Які алгоритми належать до групи тих, що не модифікують елементи контейнера?
9. Які алгоритми належать до групи тих, що модифікують елементи контейнера?
10. Які алгоритми належать до групи тих, що упорядковують елементи контейнера?
11. Які алгоритми належать до групи тих, що виконують операції над множинами?
12. Які алгоритми належать до групи тих, що що знаходять мінімальні і максимальні елементи контейнера?
13. Назвіть п’ять категорій літераторів.
14. Що таке ітератор вводу? Які операції він дозволяє виконувати?
15. Що таке ітератор виводу? Які операції він дозволяє виконувати?
16. Що таке односпрямований ітератор? Які операції він дозволяє виконувати?
17. Що таке двоспрямований ітератор? Які операції він дозволяє виконувати?
18. Що таке ітератор довільного доступу? Які операції він дозволяє виконувати?
19. Що таке розподільник пам’яті?
20. Що таке предикат?
21. Що таке об’єкт-функція (функтор)?
22. Назвіть службові типи, визначені в контейнерах.
23. Назвіть функції-члени, що повертають ітератори.
24. Назвіть функції-члени, що здійснюють доступ до елементів.
25. Назвіть функції-члени, що здійснюють включення елементів у контейнер.
26. Назвіть функції-члени, що здійснюють видалення елементів з контейнеру.
27. Назвіть службові операції над контейнерами.
28. За допомогою яких операція можна виконувати присвоювання контейнерів?
29. Які операції реалізовані в асоціативних контейнерах?