Лекція 5. STL та контейнери

Версія 9 березня 2023 р.

Анотація

Жодна справжня програма не може обійтись без якогось рішення для зберігання інформації. Об'єкти що використовуються для цього часто звуться контейнерами. Контейнери різняться за способом зберігання та звертання до данних, але не повинні залежити від конктерного типу цих данних, максимум, накладати певні обмеження. В C++ базові контейнери входять в склад Стандартної Бібліотеки Шаблонів (Standard Teamplate Library, STL). Це лекція знайомить читача з ними.

Зміст

1 `std::array`

Найпримітивнішим контейнером в STL є `std::array`. Він дозволяє зберігати набір об'єктів одного типу. Його розмір зазначається при створюванні і не змінюється протягом життя. Приклад використання:

```
#include <array>
#include <string>
```

```cpp
#include <iostream>

int main() {
   std::array<float, 3> floats = {1.0f, 2.0f, 3.0f};
   std::array<std::string, 3> strings = {"one", "two",
       "three"};

   std::cout << "The first number is " << floats[0] <<
       std::endl;
   std::cout << "The last number is " <<
       floats[floats.size()-1] << std::endl;

   // better way
   std::cout << "The first string is " << strings.front() <<
       std::endl;
   std::cout << "The last string is " << strings.back() <<
       std::endl;

   return 0;
}
```

На практиці, `std::array` використовується рідко – найбільше для зберігання невеличкої групки параметрів, яка задається один раз при створенні і не змінює розмір. Гнучкість, якої не вистачає `std::array` для більш широкого використання додає `std::vector`.

2 `std::vector`

Найбільш вживаним контейнером STL є беззаперечно `std::vector`. Якщо просто, то це `std::array`, який може змінювати свій розмір. В деякому сенсі, `std::vector` – це обгортка над `std::array` – він зберігає данні[1] в чомусь подібному на `std::array`[2], але автоматично "змінює" його розмір за нас, тобто створює більший або менший масив і копіює елементи туди. Для оптимізації цього процесу введено поняття ємності (capacity), тобто кількості виділеної пам'яті, яку можна займати перед тим, як треба звертатися за новою пам'яттю. Можна вважати це як розмір того `std::array`, в який ми записуємо. У той самий час, зберігається термін розміру та метод `size()` який повертає кількість елементів в векторі. На практиці, нам часто не потрібно напряму працювати з ємністю вектора, тільки якщо ми заздалегідь

[1] Які доречі зберігаються послідовно в пам'яті комп'ютера.
[2] Який можна отримати за допомогою методу `data()`.

маємо уявлення про те, скільки елементів нам буде потрібно зберігати. В такому випадку, ми можемо за один раз виділити необхідну кількість за допомогою методу reserve. На практиці це може мати суттєвий вплив на час виконання програми.

Рекомендуємо ознайомитись з документацією, де ви знайдете повний список методів, доступних для роботи з `std::vector`. Зазначимо тут тільки основне, а саме

- звертання до елементів відбувається так само, як і для `std::array`[3],
- видалити всі елементи можна за допомогою метода `clear()`,
- нові елементи зручно додавати за допомогою `push_back()`, або `emplace_back()`. Останній – це рекомендований підхід, оскільки дозволяє створити об'єкт в середині вектора, уникаючи зайві копії.[4] Ці методи додають елементи в кінець списку, що є найбільш ефективним, якщо нам не важливий їх порядок. Якщо ж треба поставити елемент на конкретну позицію, дивіться в бік `insert()`,
- для перевірки пустоти контейнера існує метод `empty()`[5].

3 `std::map`

`std::map` – це відсортований асоціативний контейнер.[6] Він містить в собі пари ключ-значення. [7] Його ключі мають бути унікальними. [8] Оскільки, як вже було зазначено, `std::map` – це відсортований контейнер, є вимога до типу його ключів, а саме, для них повинно бути задано `operator<`. При дотримання цієї умови, ви легко можете використовувати власні об'єкти, як `key_type` для `std::map`.

Сортування пар, це додаткова операція, яку `std::map` робить автоматично. Завдяки цьому досягається логарифмічна складність операцій вставки, пошуку, та видалення. Існує ще одна різновидність контейнерів, котра має той самий інтерфейс взаємодії, але зберігає дані інакше, а саме, в вигляді

[3]Зверніть увагу на метод `at()`, який, на відміну від оператора прямокутних дужок, перевіряє чи індекс не більший за розмір вектора.

[4]Насправді, різниця може бути більш тонкою, ніж може здатись з цього твердження. Зверніться до літератури та посилань за повнішою картиною.

[5]Він є універсальним для всіх контейнерів та рядків.

[6]Це аналог `dict` (словника) в мові Python, або `struct` в MatLab.

[7]В деякому сенсі буквально – якщо ітерувати по `std::map` за допомогою range for-loop, то елементом буде об'єкт типу `std::pair<key_type, mapped_type>`, що за сумісництвом э його `value_type`.

[8]Інша різновидність асоціативного контейнера в STL – `std::multimap` дозволяє мати однакові ключі.

хеш таблиці. Мова йде про `std::unordered_map`. У цьому випадку, контейнер повинен мати змогу отримати хеш `key_type`'у. Вона має в середньому константну складність згаданих операцій, але в гіршому випадку – лінійну. Тому, в залежності від способу взаємодії з даними та їх кількості, швидшим може виявитись або один, або інший тип.

Рекомендуємо ознайомитись з документацією, де ви знайдете повний список методів, доступних для роботи з `std::map`. Зазначимо тут тільки основне, а саме

- як і для інших контейнерів, є імплементованими методи `empty()` та `size()`,
- звертання до елементів відбувається трьома основними способами, які відрізняються поведінкою у випадку відсутності пари з заданим ключем:
  - `operator[]`, який повертає створений за замовченням (пустий) елемент типу `mapped_type`,
  - `at()`, який кидає виняток типу `std::out_of_range`,
  - `find()`, який повертає ітератор, що дорівнює `end()`.
- видалити всі елементи можна за допомогою метода `clear()`,
- є так само декілька методів для додавання елементів, рекомендуємо почати з `emplace()`, або `try_emplace()`, якщо не бажано заміняти існуюче значення. Альтернативою є `operator[]` та `insert()` та інші,
- перевірити наявність ключа можна за допомогою метода `count()`: `m.count(key) > 0`, або `m.contains(key)` починаючи з C++20,
- для ітерації по елементах, використовуйте range for-loop (починаючи з C++17):

```
std::map<char, int> my_dict{{'a', 27}, {'b', 3}};
for (const auto& [key, value] : my_dict)
   std::cout << key << " has value " << value <<
       std::endl;
```

4 Ітератори

Якщо узагальнити, то ітератори – це уніфікований інтерфейс роботи з елементами контейнерів в C++. Алгоритми та функції стандартної бібліотеки C++ працюють саме з ітераторами, а не з об'єктами конкретного типу, що дозволяє їм бути універсальними. Також, якщо ви пишете свій контейнер, або обгортку, то правильно задані ітератори автоматично дозволить вам використовувати ваші структури з функціями в `std`.

Ітератори поведінкою і механізмом взаємодії схожі з вказівниками, конкретніше

- щоб отримати елемент, на який посилаються, використовується оператор `*`: `*iter`,
- методи та атрибути доступні через оператор стрілки `->`,
- щоб перейти на наступний елемент, зручно використовувати оператор `++`: `iter++`,
- а взагалі, для навігації по масиву можна використовувати оператор `+`, або `-` та ціле число, наприклад, просунутись на два елементи вперед можна за допомогою `iter = iter + 2`[9], на три назад – `iter = iter - 3`[10],
- працює з range for-loops,
- задані оператори порівняння, `==`, `!=`, `<`.

Для початку роботи, найчастіше, нам необхідно отримати ітератор початку, або кінця. Зробити це можна за допомогою методів `begin()` та `end()`, відповідно. Однак, якщо це можливо, краще використовувати варіацію цих методів, що повертає їх константну версію, яка не дозволяє змінювати елемент, на який пони посилаються, а саме `cbegin()` та `cend()`. За аналогією, існують методи, що повертають зворотні ітератори для проходу по контейнеру в зворотньому напрямку: `rbegin()/crbegin()` та `rend()/crend()`.

Типовими алгоритми, що часто використовуються, які задані в файлі заголовку `algorithm`, та які можуть дати уяву про роботу з ітераторами є:

- std::sort
- std::find
- std::fill
- std::count
- std::count_if
- std::for_each
- std::all_of
- std::transform
- std::accumulate
- std::max
- std::min_element
- std::minmax_element
- std::clamp

[9] iter += 2
[10] iter -= 3

Завдання

Переконайтесь, що ви маєте впевненість при використанні об'єктів з STL.

Література

- Stroustrup, Bjarne. The C++ programming language. Pearson Education, 2013 (Глави 30-33).
- CPPReference, Containers.
- CPPReference, Iterators.
- CPPReference, Algorithms.