Лекція 4. Функції в C++.

Версія 5 березня 2023 р.

Анотація

Ця лекція направлення на ознайомлення з, або повторювання, властивостями, особливостями і правилами роботи з функциями в C++.

Зміст

1 Що таке функції в програмуванні?

Функції – це окремі сутності, що інкапсулюють логіку, яка (i) може бути логічно абстрагована в окремий блок від навколишньої логіки, (ii) виконує одне завдання, (iii) може повторюватись декілька разів в рамках програми.

Функції можуть існувати як самі по собі (так звані standalone functions), тоді компілятор їх знаходить за простором імен та типами і кількістю вхідних аргументів, так і бути членами якогось класу, і їх можна викликати через об'єкт цього класу, або через оператор області видимості (для статичних функцій).

Щоб функцію можна було викликати, вона (а точніше її інтерфейс) повинна бути задекларована. Сама логіка, код який треба виконати при її

виклику, зберігається в її тілі, і буде доступний компілятору, якщо додати до процесу компіляції файл з дефініціями.

2 Структура функцій в C++

Сігнатура функції в C++ повністю задає її інтерфейс. Вона має наступний вигляд

```
ReturnType FuncName(ParamType1 arg1, ParamType2 arg2) {
   // Function code goes here...
   return return_value;
}
```

Зліва направо вона складається з: типу повернутого значення, назви функції, у круглих дужках – список вхідних аргументів з їх типами, у фігурних дужках – тіло функції.[1]

Тип значення, що повертається, може бути автоматично виведений компілятором при використанні ключового слова auto, але, для більших функцій з нетривіальними типами, або при використанні поліморфізму, або при написанні перевантажених функцій, може втрачатися виразність та ясність для читача вашого коду. Функція в C++ може повертати тільки одне значення, або нічого, тоді як тип треба вказати ключове слово void. Для повернення декількох значень одного типу можна використати масиви, std::vector, та ін., якщо ці значення різних типів, варто подивитися в бік std::pair, std::tuple, або задати свій тип за допомогою struct або class. Функції, що повертають декілька значень за допомогою std::pair або std::tuple, можуть використовуватись наче ті, що повертають декілька значень (в C++17), наприклад

```
#include <iostream>
#include <tuple>

// auto is resolved as std::tuple<std::string, int>
auto GetData() {
   return std::make_tuple("orders", 2);
}

int main() {
   auto [key, value] = GetData();
   std::cout << "I have found " << value << key << std::endl;
```

[1]Також, функції можуть приймати необов'язкові атрибути, але вони не часто використовуються на практиці.

```
    return 0;
}
```

Повертати значення можна як копію, посилання, або вказівник. Ці стратегії мають суттєві наслідки, і детальніше ми подивимось на них в наступних лекціях. Безпечніше всього повертати значення як копію, тим паче, що в окремих випадках компілятори можуть провести оптимізацію і повторно використати старе значення, замість того, щоб робити нову копію, а стару видалити.

Назва функцій повинна точно та лаконічно описувати *що*, а не *як*, вона робить те, що робить. Іноді, вона має бути достатньо довгою і це нормально. Поганим маркером є, наприклад, якщо вам хочеться додати сполучник 'і' (and в англійський мові) в назву функції, бо згадайте: функція має виконувати тільки одну справу – задумайтесь, чи варто розділити вашу функцію на дві менші під-функції. Також часто маркером того, що функція багато на себе бере та має складну, як на таку структурну одиницю логіку, це те, що вона бере вхідний параметр типу bool. Тіло функції зберігає в собі код, що виконується при виклику цієї функції. Воно задає свою область значень (scope), для якого виконуються звичні правила. Гарно написана функція не потребує коментарів, тому що її код достатньо короткий і виразний, щоб не заплутати програміста, а її назва точно описує її завдання.

Вхідні аргументи – це головний метод отримання інформації ззовні для функції.[2] Змінну типу Type можна передати функції трьома основними способами:

1. як копію: `Type arg`
2. як посилання, що дозволяє функції змінювати її значення у зовнішньому світі: `Type& arg`
3. як константне посилання, що не дозволяє змінювати її значення, ні всередині функції, ні ззовні: `const Type& arg`

Пам'ятайте, що, з точки зору C++, в цих трьох випадках, змінна arg має три різні типи всередині функції. У більшості випадків, якщо це можливо, найкраще передавати дані саме через константне посилання. Таким чином, ми, по-перше, не копіюємо самі дані, а просто передаємо їх адресу в пам'яті, а по-друге, не дозволяємо небажані зміни, які можуть вплинути на подальше їх використання поза цією функцією. Згадаємо мимохідь,

[2]Інший валідний спосіб, це атрибути класу для методів цього класу. Все інше, насамперед глобальні змінні, приховують важливі канали комунікації, призводять до потенціальної великої кількості помилок, які важко відловити, і не повинні використатись, окрім дуже рідкісних випадків. Звичайно, тут поки що не йде мова про таку взаємодію з зовнішнім світом, як читання з файлів, баз даних, вводу користувача.

що вхідні параметри можуть приймати значення за замовченням, що може здатись зручним, але зловживання цим дуже не заохочується.

3 `inline` функції

Якщо у вас є якась (зокрема невеличка) функція, і питання швидкості для вашої програми є в пріоритеті – наприклад, вона виконується на кластері і через неї проходить велика кількість даних – тоді вам може стати у нагоді ключове слово `inline`. Воно говорить компілятору, що йому дозволяється, якщо це можливо, вставити код функції на місце її виклику. Таким чином, коли виконання програми досягне цієї точки, не потрібно буде шукати в скомпільований код цієї функції (не то щоб це було дуже важкою процедурою само по собі), а можна буде одразу почати його виконання. Таким чином, ми трохи можемо виграти в часі при виконанні, зберігаючи ясність вихідного коду, що використовує під-рутини (subroutines). Деякі компілятори можуть провести цю оптимізацію (якщо вона ввімкнута при компіляції) для вас навіть якщо ви не задекларували функцію як `inline`. Деякі читачі можуть відмітити схожість `inline` функцій з макросами (macros). Оскільки використання останніх фактично під забороною, варто дивитися в бік перших, якщо є таке бажання.

4 Перевантаження функцій

Як вже було сказано, сігнатура функції повністю задає її інтерфейс. Компілятор, коли шукає функцію, що ви використовуєте в своїй програмі, послуговуються не тільки її назвою, а ще і кількістю вхідних параметрів, разом з їх типами. Зауважте, що він не відрізняє дві функції тільки по типу їх повернутого значення. Таким чином, якщо ви хочете написати, наприклад, сімейство функцій для порівняння значень, можна перевикористати одне і те саме ім'я

```
bool IsEqual(int left, int right) {
  return left == right;
}

bool IsEqual(const std::string& left, const std::string&
    right) {
  return left == right;
}
```

Зауважте, що для цього прикладу доречніше використовувати шаблонні

функції, про які мо поговоримо згодом, а вдаватися до перевантаження тільки тоді, коли поведінка відрізняється в залежності від типу вхідного значення.

Завдання

Пересвідчитесь, що ви маєте впевнене розуміння функції, бо вони є основою роботи з будь-якою мовою програмування. Передивиться рекомендовану літературу, зупиняючись на темах, що є незрозумілими. Ознайомтесь з розділом про функції Google Style Guide.

Література

- Stroustrup, Bjarne. The C++ programming language. Pearson Education, 2013 (Глава 12).
- CPPReferece
- Google C++ Style Guide, Functions