МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота №12

з дисципліни
«Алгоритмізація та програмування»

Виконав:
студент групи КН-114
Добрій Назарій

Викладач:
Мочурад Л.І

Львів – 2019 р.

Тема: проектування і опрацювання програми з родовими функціями і родовими класами.

Що таке шаблонні функції в C++ та їхня реалізація.

Шаблони (англ. *template*) — засіб мови C++, який призначений для кодування узагальнених алгоритмів, без прив'язки до деяких параметрів: типу даних, розміру буфера та стандартного значення. В C++ можливе створення шаблону функції і шаблону класу. Хоча шаблони надають коротку форму запису ділянки коду, насправді їх використання не скорочує виконаний код, тому що для кожного набору параметрів компілятор створює окремий екземпляр функції або класу.

При створенні функцій іноді виникають ситуації, коли дві функції виконують однакову послідовність дій, але працюють з різними типами даних (наприклад, одна використовує параметри типу *int*, а інша типу *float* або *double*). За допомогою механізму перевантаження функцій можна використовувати одне й те ж ім'я для функцій, що виконують різні дії і мають різні типи параметрів. Однак, якщо функції повертають значення різних типів, слід використовувати для них унікальні імена. Шаблонна функція дає можливість оперувати параметрами різних типів і повертати значення різних типів, що значно полегшує роботу і зменшує кількість змінних у коді. Іншими словами шаблонна функція являє собою набір функцій.

Для чого існують шаблонні функції? Механізм шаблонів в мові C ++ дозволяє вирішувати проблему уніфікації алгоритму для різних типів: немає необхідності писати різні функції для цілочисельних, або призначених для користувача типів - досить скласти узагальнений алгоритм, що не залежить від типу даних, який базується тільки на загальні властивості. Наприклад, алгоритм сортування може працювати як з цілими числами, так і з об'єктами типу «автомобіль».

Існують шаблони функцій і шаблони класів.

Шаблони функцій - це узагальнений опис поведінки функцій, які можуть бути викликані об'єктів різних типів. Іншими словами, шаблон функції (шаблонна функція, узагальнена функція) являє собою сімейство різних функцій (або опис алгоритму). За описом шаблон функції схожий на звичайну функцію: різниця в тому, що деякі елементи не визначені (типи, константи) і є параметризрвані.

Шаблони класів - узагальнений опис користувальницького типу, в якому можуть бути параметризовані атрибути і операції типу. Являють собою конструкції, за

якими можуть бути згенеровані дійсні класи шляхом підстановки замість параметрів конкретних аргументів.

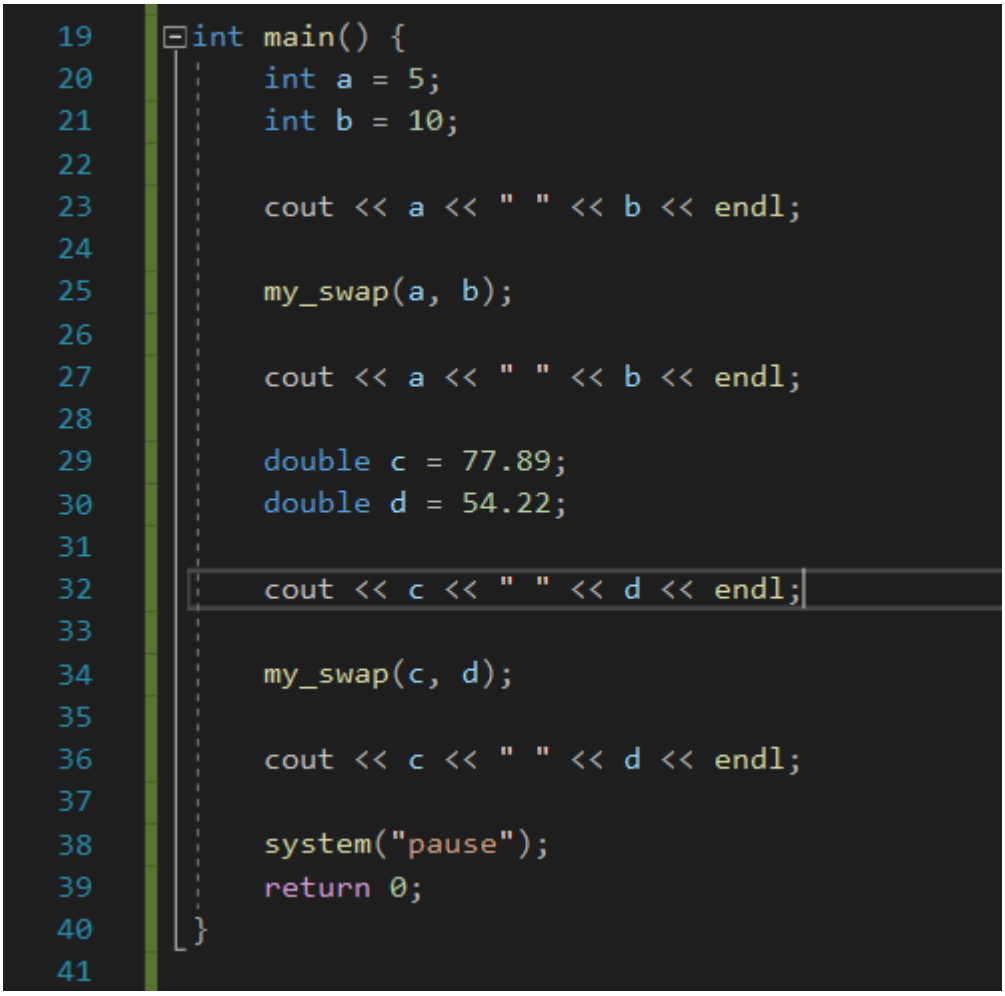
Давайте розглянемо простий приклад. Припустимо, у нас є функція, яка міняє місцями значення двох змінних типу int:

```
1   #include <iostream>
2
3   using namespace std;
4
5   void my_swap(int& first, int& second)
6   {
7       int temp(first);
8       first = second;
9       second = temp;
10  }
11
12  int main() {
13      int a = 5;
14      int b = 10;
15
16      cout << a << " " << b << endl;
17
18      my_swap(a, b);
19
20      cout << a << " " << b << endl;
21
22      system("pause");
23      return 0;
24  }
25
```

Тепер, припустимо, у нас в функції main так само є дві змінні типу double, значення яких теж потрібно обміняти. Функція для обміну значень двох змінних типу int нам не підійде. Напишемо функцію для double:

```
12  void my_swap(double& first, double& second)
13  {
14      double temp(first);
15      first = second;
16      second = temp;
17  }
```

І тепер перепишемо main:



```
int main() {
    int a = 5;
    int b = 10;

    cout << a << " " << b << endl;

    my_swap(a, b);

    cout << a << " " << b << endl;

    double c = 77.89;
    double d = 54.22;

    cout << c << " " << d << endl;

    my_swap(c, d);

    cout << c << " " << d << endl;

    system("pause");
    return 0;
}
```

Як бачите, у нас алгоритм абсолютно однаковий, відрізняються лише типи параметрів і тип змінної temp. А тепер уявіть, що нам ще потрібні функції для short, long double, char, string і ще безлічі інших типів. Звичайно, можна просто скопіювати першу функцію, і виправити типи на потрібні, тоді отримаємо нову функцію з необхідними типами. А якщо функція буде не така проста? А раптом потім ще виявиться, що в першій функції була помилка? Уникнути всього цього можна, наприклад, «шаманством» з препроцесором, але це нам ні до чого, нам допоможуть шаблони.

Отже, опис шаблону починається з ключового слова template за яким в кутових дужках ("<" і ">") слід список параметрів шаблону. Далі, власне йде оголошення шаблонної суті (наприклад функція або клас). Має вигляд:

```
template < template-parameter-list > declaration
```

Тепер давайте напишемо шаблонну функцію my_swap. Виходячи зі згаданої вище структури оголошення шаблону слід, що наша функція буде виглядати так:

```
19  template < typename T >
20  void my_swap(T& first, T& second)
21  {
22      T temp(first);
23      first = second;
24      second = temp;
25  }
```

typename в кутових дужках означає, що параметром шаблона буде тип даних. T - ім'я параметра шаблона. Замість typename тут можна використовувати слово class: template <class T> В даному контексті ключові слова typename і class еквівалентні (особисто мені більше подобається typename, а кому-то class). Далі, в тексті шаблону всюди, де ми використовуємо тип T, замість T буде проставлятися необхідний нам тип.

Тепер давайте напишемо функцію main:

```
27  int main() {
28      int a = 5;
29      int b = 10;
30
31      cout << a << " " << b << endl;
32
33      my_swap<int>(a, b);
34
35      cout << a << " " << b << endl;
36
37      double c = 77.89;
38      double d = 54.22;
39
40      cout << c << " " << d << endl;
41
42      my_swap<double>(c, d);
43
44      cout << c << " " << d << endl;
45
46      system("pause");
47      return 0;
48  }
49
```

Як бачите, після імені функції в кутових дужках ми вказуємо тип, який нам необхідний, він то і буде типом T. Шаблон - це лише макет, за яким компілятор самостійно буде генерувати код. При вигляді такої конструкції: my_swap <тип>

компілятор сам створить функцію my_swap з необхідним типом. Це називається інстанціювання шаблону. Тобто при вигляді my_swap <int> компілятор створить функцію my_swap в якій T поміняє на int, а при вигляді my_swap <double> буде створена функція з типом double. Якщо десь далі компілятор знову зустріне my_swap <int>, то він нічого генерувати не буде, тому що код даної функції вже є (шаблон з даними параметром вже інстанціювати).

Таким чином, якщо ми інстанціруем цей шаблон три рази з різними типами, то компілятор створить три різні функції.

Висновок типу шаблону виходячи з параметрів функції.

Насправді, ми можемо викликати функцію my_swap **НЕ** вказуючи тип в кутових дужках. У ряді випадків компілятор може це зробити за вас.

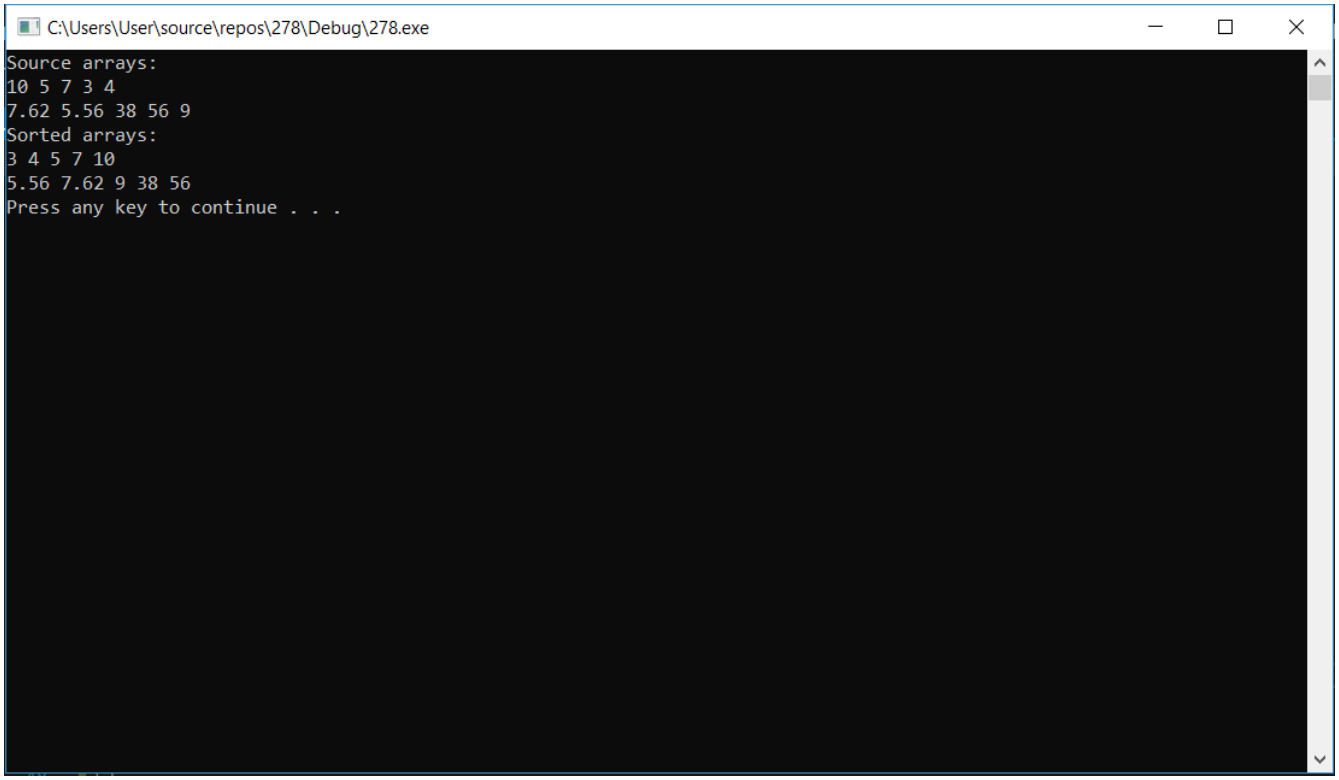
Розглянемо виклик функції без вказівки типу:

```
14      int a = 5;
15      int b = 10;
16
17      my_swap(a, b);
18
```

Наша шаблонна функція приймає параметри типу T &, грунтуючись на шаблоні, компілятор бачить, що Ви передаєте в функцію аргументи типу int, тому може самостійно визначити, що в даному місці мається на увазі функція my_swap з типом int. Це **deducing template arguments**. Тепер давайте напишемо приклад складніше. Наприклад, програму сортування масиву (будемо використовувати сортування «бульбашкою»). Природно, що алгоритм сортування один і той же, а ось типи елементів в масиві будуть відрізнятися. Для обміновання значень будемо використовувати нашу шаблонну функцію my_swap. приступимо:

```cpp
#include <iostream>

using namespace std;

template < typename T >
void my_swap(T& first, T& second)
{
    T temp(first);
    first = second;
    second = temp;
}

template < class ElementType >
void bubbleSort(ElementType* arr, size_t arrSize)
{
    for (size_t i = 0; i < arrSize - 1; ++i)
        for (size_t j = 0; j < arrSize - 1; ++j)
            if (arr[j + 1] < arr[j])
                my_swap(arr[j], arr[j + 1]);
}

template < typename ElementType >
void out_array(const ElementType* arr, size_t arrSize)
{
    for (size_t i = 0; i < arrSize; ++i)
        cout << arr[i] << ' ';
    cout << std::endl;
    for (size_t i = 0; i < arrSize; ++i)
        cout << arr[i] << ' ';
    cout << std::endl;
}

int main()
{
    const size_t n = 5;
    int arr1[n] = { 10 , 5 , 7 , 3 , 4 };
    double arr2[n] = { 7.62 , 5.56 , 38.0 , 56.0 , 9.0 };
    cout << "Source arrays:\n";
    out_array(arr1, n);
    out_array(arr2, n);

    bubbleSort(arr1, n);
    bubbleSort(arr2, n);

    cout << "Sorted arrays:\n";
    out_array(arr1, n);
    out_array(arr2, n);

    system("pause");
    return 0;
}

```

Реалізація програми:



Як бачите, компілятор сам генерує out_array для необхідного типу. Так само він сам генерує функцію bubbleSort. А в bubbleSort у нас застосовується шаблонна функція my_swap, компілятор згенерує і її код **автоматично**. Зручно, чи не так?

Шаблони та STL

У комплекті з компілятором Вам надається стандартна бібліотека шаблонів (Standart Template Library). Вона містить безліч шаблонних функцій і класів. Наприклад, клас двусвязного списку (list), клас «пара» (pair), функція обміну двох змінних (swap), функції угруповань, динамічно розширюваний масив (vector) і т.д. Все це - шаблони і Ви можете їх використовувати. Для невеликого прикладу візьмемо vector:

```cpp
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

int main()
{
    vector <int> arr;
    arr.push_back(5);
    arr.push_back(7);
    arr.push_back(3);
    arr.push_back(8);
    cout << "Source vector:\n";
    for (int i = 0, size = arr.size(); i < size; ++i)
        cout << arr[i] << ' ';
    cout << std::endl;

    sort(arr.begin(), arr.end());

    cout << "Sorted vector:\n";
    for (int i = 0, size = arr.size(); i < size; ++i)
        cout << arr[i] << ' ';
    cout << std::endl;
}
```

Висновок:

Шаблони допомагають описати в одній функції значення параметрів не створюючи багато функцій з різними параметрами, це набагато пришвидшує та оптимізує роботу програми. Шаблони це занадто великий і потужний інструмент і описати все в одній статті неможливо. Це було лише невелике введення в світ шаблонів. Заглиблюючись в шаблони, Ви здивуєтеся тому, який потужний це інструмент і які можливості він надає.