Шаблони в мові C++.

Шаблони (template – English) – це засіб мови C++, призначений для створення кодів узагальнених алгоритмів або класів, не прив'язаних до конкретного типу параметрів.

Навіщо потрібні шаблони?

C++ дозволяє визначати об'єкти (зокрема змінні, функції) лише конкретних типів або із конкретними типами параметрів. Але в багатьох випадках алгоритми обробки таких об'єктів не залежать від їх типу, наприклад, алгоритм сортування або обробка динамічних структур, подібних до бінарного дерева, стеку чи зв'язаного списку – всі вони працюють ідентично для всіх типів даних. Для реалізації подібних алгоритмів програміст може обрати один із наступних шляхів:

- ✔реалізувати один і той самий алгоритм для кожного потрібного типу даних;
- ✔використати абстрактний базовий клас, в якому визначити даний алгоритм і заміщати його відповідним чином у похідних класах;
- ✔використовувати засоби препроцесора і створити макровизначення.

Спробуйте оцінити вади та переваги кожного із цих шляхів.

Шаблони призначені саме для розв'язання цієї проблеми, і оскільки вони є вбудованими засобами мови, для них забезпечується належна підтримка контролю типів.

Шаблон являє собою **функцію** або **клас**, що можуть бути реалізовані для одного чи навіть кількох абстрактних типів даних, які ***невідомі на момент компіляції цього коду***. В момент виклику у шаблон передаються конкретні типи даних, для яких **генерується** функція чи клас із відповідними типами даних.

З іншого боку не варто забувати, що в результаті ми приходимо до певної філософської небезпеки, адже таким чином породжуються програми, які створюють інші програми – ми маємо справу із так званим ***метапрограмуванням***.

1. Шаблони функцій.

Шаблон функції (її інколи називають ще ***родовою функцією***)– це узагальнене визначення цілого сімейства функцій, які можуть бути викликані для даних різних типів.

Визначення шаблону починається із службового слова **`template`**, після якого у кутових дужках міститься список параметрів шаблону – обов'язково непорожній.

При визначенні параметрів шаблону використовується службове слово **`class`**, яке втім не має жодного відношення до поняття класу в мові C++, або службове слово **`typename`**, затверджене лише нещодавно у стандарті мови. Після нього вказується ідентифікатор параметру шаблону.

Розглянемо для прикладу функцію, яка повертає максимум з двох своїх параметрів.

```
// шаблон функції. Type – це параметр шаблону,
// який є типом параметрів
template <typename Type>
Type maxi (Type a, Type b)
{   return (a < b)? b : a;  }
int main( )
{
 int m = 123, n = 10;
 double x = 1.0e-2, y = 1.0e+2;
 char v = 'A', w = 'B';
 cout <<"integer after max:"<< maxi (m, n)<<
endl;
 cout <<"double after max:"<< maxi (x, y)<<
endl;
 cout <<"char after max:" << maxi (v, w) <<
endl;
 return 0;
}
```

Що відбувається при виклику функції, визначеної шаблоном?

Коли компілятор знаходить звертання до функції **`maxi`**, тип даних, що передається у **`maxi`** при виклику, підставляється замість ідентифікатору **`Type`** у всьому коді визначення шаблону, і компілятор створює завершену повноцінну функцію, яка компілюється і викликається. Цей процес називається ***конкретизацією*** (instantiation) шаблону. Таким чином, шаблони дійсно відіграють роль генераторів програмних кодів.

```
// В цьому прикладі шаблон має кілька параметрів:
// Type1 та Type2 – параметри шаблону функції.
// Службове слово class (або typename) вказане
// перед кожним параметром!
template <class Type1, class Type2>
Type1 maxi (Type1 a, Type2 b)
{
   return (a < (Type1) b)? (Type1) b : a;
}
int main( )
{
  cout <<"maxi(double, int):"<< maxi (2.5, 4) <<
  endl;
  cout <<"maxi(int, double):"<< maxi (4, 20.5) <<
  endl;
  cout <<"maxi(int, char):"<< maxi (5, '0')<< endl;
  system("PAUSE");
  return 0;
}
```

Крім параметрів-типів шаблони можуть містити і звичайні параметри, як наприклад, у наступному прикладі:

```
// шаблон функції для пошуку мінімуму в
// масиві містить і звичайний параметр -
// змінну size для визначення кількості
// елементів в масиві
template <class Type, int size>
Type min_( Type (&r_array) [size] )
// зверніть увагу - для "звичайної" функції
// було б досить порожніх квадратних дужок!
{
    Type min_val = r_array[0];
    for ( int i = 1; i < size; i++ )
        if ( r_array[i] < min_val )
            min_val = r_array[i];
    return min_val;
}
```

В такому випадку конкретизація шаблону буде відбуватись не тільки в залежності від типу елементів масиву, але й в залежності від кількості цих елементів.

```
template <class Type, int size>
Type min_( Type (&r_array) [size] );
int main( )
{
  double arr_d  [4] = {1.5, -2.12, 3.5, 4.9};
  int arr_i [] = {1, 2, 3, 4, 0, -10, 22};
// конкретизація: Type -> double, size -> 4
  cout << "\nMin = " << min_ (arr_d) << endl;
// конкретизація: Type -> int, size -> 7
  cout << "Min = " << min_ (arr_i) << endl;
  return 0;
}
```

Слід зазначити, що **родові функції** (**шаблони**) можуть перевантажуватись. Причому може існувати кілька шаблонів функцій з різними наборами параметрів, а може також бути створена звичайна функція з іменем шаблона.

В останньому випадку перевантажена функція може перекривати («затіняти») шаблонну функцію, яку компілятор створив би для даного конкретного виклику. Крім того, перевантаження шаблонів може привести до складних і неоднозначних виборів функцій-кандидатів. Отже, можливість перевантаження шаблонів слід використовувати вкрай обережно або не використовувати взагалі.

2. Шаблони класів.

Шаблон класу (або ***родовий клас***) – як і у випадку родової функції він містить всі необхідні алгоритми обробки даних, а конкретні типи даних підставляються в момент створення екземпляру даного класу. Таким чином, шаблон класу породжує ціле сімейство класів із спільною логікою функціонування для різних типів даних.

Синтаксис визначення шаблону класу наступний:

```
template <class Type> class class_id
{   // визначення класу
};
```

Функції-члени класу автоматично стають шаблонами функцій, хоча для них і не вказується явно службове слово **template**. У випадку, коли така функція лише декларується у класі, а визначається поза ним, використовується достатньо складний синтаксис:

```
template <class Type> тип_результату_функції
class_id <Type> :: func_id (параметри_функції)
{   // тіло функції
}
```

При створенні екземпляру шаблону класу конкретні значення аргументів шаблону вказуються в кутових дужках:

```
class_id <int> c_i;
class_id <double> c_d;
```

Для даного прикладу буде сгенеровано екземпляр **c_i** класу **class_id**, в якому в ролі типу **Type** виступатиме тип **int**, та екземпляр **c_d** того самого класу **class_id**, але із типом **double** замість типу **Type**.

Якщо шаблон має не один параметр, а більше, значення аргументів шаблону при створенні екземпляру мають однозначно відповідати списку параметрів шаблону.

Замість службового слова **class**, що використовується у списку параметрів шаблону, так само можна використовувати його синонім **typename**.

Деякі ***правила визначення шаблонів***:

- ✔Шаблони функцій не можуть бути віртуальними.
- ✔Шаблони класів можуть містити статичні елементи, дружні функції та класи.
- ✔Шаблони класів можуть бути похідними як від звичайних класів, так і від шаблонів, а також бути базовими класами і для шаблонів, і для звичайних класів.

В розпорядженні користувачів потужна бібліотека стандартних шаблонів (**`STL – Standard Template Library`**), яка містить багато шаблонів класів.

Бібліотека стандартних шаблонів мови С++.

Ядро бібліотеки стандартних шаблонів складають 3 основні елементи:

- ✔контейнери
- ✔алгоритми
- ✔ітератори.

Контейнер – об’єкт бібліотеки стандартних шаблонів, призначений для збереження даних. Деякі контейнерні класи наведені у таблиці нижче.

Контейнер	Опис	Заголовок
`bitset`	множина бітів	`<bitset>`
`queue`	черга	`<queue>`
`list`	лінійний список	`<list>`
`set`	множина, в якій кожний елемент унікальний	`<set>`
`stack`	стек	`<stack>`
`vector`	вектор	`<vector>`
`map`	асоційований список для збереження пар ***ключ-значення***	`<map>`