STL

Generic programming

Saurabh Shukla (MySirG)

Agenda

1. Generic Programming in C++
2. History
3. Templates

Generic Programming

Generic Programming enables the programmer to write a general algorithm which will work with all data types. It eliminates the need to create different algorithms if the data type is an integer, string or a character.

Generic Programming

Generic programming is a style of computer programming in which algorithms are written in terms of types to be specified later that are then initiated when needed for specific types provided as parameters.

History

Approach of generic programming, pioneered by the ML (Meta Language) Programming language in 1973, permits writing common functions or types that differ only in the set of types on which they operate when used, thus reducing duplication.

Such software entities are known as

- generics in C#, Java, Python, TypeScript, VB.NET, Rust
- parametric Polymorphism in ML, Scala, Julia
- Templates in C++ and D

Template

A template is a simple and a powerful tool in C++.

The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types.

For example, you need to code a sort function for different data types. Rather than writing and maintaining multiple codes, we can write one sort() and pass data type as a parameter.

C++ provides two keywords to support templates

① template ② typename

typename can always be replaced by keyword class

Templates are expanded at compile time
Compiled code may contain multiple copies of function / class

Template

1. Function Template or Generic Function
2. class Template

Function Template

```
template <typename X>
function definition
```

Class template

```
template <typename X>
class definition
```

Multiple typenames

There can be more than one argument to templates.

```
template < typename X , typename Y >
```

Default type parameter

We can specify default value (data type) for the template arguments

```
template < class X, class Y = char >
class Test
{

};
```

```
Test <int, float> t1;
Test <double> t2;
```

Template Specialization

It is possible in C++ to get a special behavior for a particular data type. This is called template specialization.

```
template <class X>
void sort( X arr[], int size)
{
  ...
}
```

Now suppose we want some special behavior for char type:

```
template <>
void sort <char> ( char arr[], int size)
{ ... }
```