Templates: code sharing (Genericity)

INTRODUCTION TO CODE SHARING

There are many programming situations where same set of operations are applied on different data types.

For example, the following function exchanges the contents of two parameters of type int.

```
void exchange ( int & oldVal , int & newVal)
   {
int temp;

   temp     =  oldVal;
   oldVal  =  newVal;
   newVal = temp;
    }
```

Code-sharing is implemented in C++ through templates.

- The pertinent question arises that “Can’t we write a **generic** function that can accept all types of parameters?”.
- If yes, then a tremendous amount of typing effort can be saved. This will not only increase the productivity of the programmer but also reduce the size of the program.
- However the programmer must demarcate the program code into two categories: type dependent and type independent code.

TEMPLATES

A template in C++ is used to create generic functions and classes.

The format of template declaration is given below:

template < class ugType >

1. where template: is a keyword
2. < : is the standard left angled bracket or sign for less than operator
3. class : is a keyword
4. ugType : user defined generic type
5. > : is the standard right angled bracket or sign for greater than operator

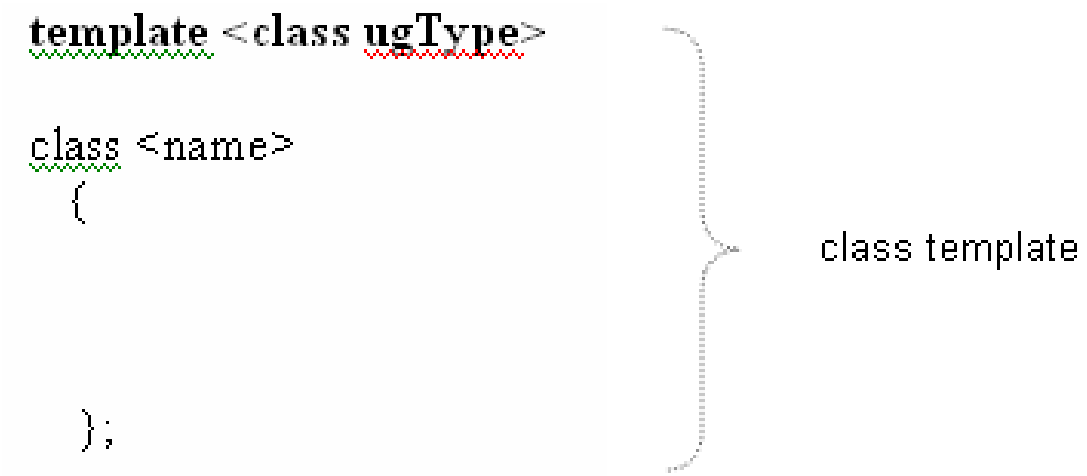
```
template <class myType>          // declare a function template

    void exchange ( myType & oldVal , myType & newVal)
      {
    myType temp;

      temp     =  oldVal;
      oldVal  =  newVal;
      newVal = temp;
       }
```

1. **We have used a generic type called ‘myType’ to declare the variables in the generic function ‘exchange’.**
2. **This generic type (myType) would get substituted at run time by whichever type of data is employed by the programmer.**
3. **Thus, templates use run time polymorphism to bind the data with their types.**

GENERIC CLASSES

Similar to functions, we can also create generic classes in the form of class templates.

The format of a **class template** is given below:



where

template: is a keyword

< : is the standard left angled bracket or sign for less than operator

class : is a keyword

ugType : user defined generic type

> : is the standard right angled bracket or sign for greater than operator

The template declaration followed by class declaration is jointly called as a **class template.**

TEMPLATES WITH MORE THAN ONE GENERIC PARAMETER

A template can have more than one generic parameter as shown by the following declaration:

```
template <class genType1, class genType2>
```

1. The above template declaration is using two generic types: genType1 and genType2.
2. The advantage of this feature of C++ is that a function template or class template can be called for more number of generic parameters.

SUMMARY

1. **A generic function and class can accept parameters of different types.**
2. **It allows code sharing with a view to reduce typing effort, increase productivity, and reduce the size of the program.**
3. **The code of a program can be comfortably divided into two parts: type dependent and type independent.**
4. **The ‘generic type’ gets substituted at runtime by the type of data supplied to the generic function or class.**
5. **A template declaration followed by a class declaration is jointly called a ‘class template’.**
6. **The name of class juxtaposed with a type in angled brackets is called a ‘template class’.**
7. **A template can have more than one generic parameter.**