Subject Name: **Object Oriented Programming Using C++**

Subject Code: **BCA-301 N**

Subject Topic: **Templates**

Abhishek Dwivedi

Assistant Professor

Department of Computer Application

UIET, CSJM University, Kanpur

C++ Templates

- A C++ template is a powerful feature added to C++. It allows you to define the generic classes and generic functions and thus provides support for generic programming. Generic programming is a technique where generic types are used as parameters in algorithms so that they can work for a variety of data types.

Templates can be represented in two ways:

a. Function templates

b. Class templates

- **Function Templates:** We can define a template for a function. For example, if we have an add() function, we can create versions of the add function for adding the int, float or double type values.
- **Class Template:** We can define a template for a class. For example, a class template can be created for the array class that can accept the array of various types such as int array, float array or double array.

Function Template

- Generic functions use the concept of a function template. Generic functions define a set of operations that can be applied to the various types of data.
- The type of the data that the function will operate on depends on the type of the data passed as a parameter.
- For example, Quick sorting algorithm is implemented using a generic function, it can be implemented to an array of integers or array of floats.
- A Generic function is created by using the keyword template. The template defines what function will do.

- **Syntax:**

```
template < class Ttype> ret_type func_name(parameter_list)
{
   // body of function.
}
```

- Where **Ttype**: It is a placeholder name for a data type used by the function. It is used within the function definition. It is only a placeholder that the compiler will automatically replace this placeholder with the actual data type.
- **class**: A class keyword is used to specify a generic type in a template declaration.

```
#include <iostream>
using namespace std;
template<class T> T add(T &a,T &b)
{
   T result = a+b;
   return result;
}
int main()
{
  int i =2;
  int j =3;
  float m = 2.3;
  float n = 1.2;
  cout<<"Addition of i and j is :"<<add(i,j);
  cout<<'\n';
  cout<<"Addition of m and n is :"<<add(m,n);
  return 0;
}
```

- **In the above example, we create the function template which can perform the addition operation on any type either it can be integer, float or double.**

Class Template

- **Class Template** can also be defined similarly to the Function Template. When a class uses the concept of Template, then the class is known as generic class.

template<**class** Ttype>

class class_name

{

.

.

}

- **Ttype** is a placeholder name which will be determined when the class is instantiated. We can define more than one generic data type using a comma-separated list. The Ttype can be used inside the class body.
- Now, we create an instance of a class

  class_name<type> ob;

```
#include <iostream>
using namespace std;
template<class T>
class A
{
   public:
   T num1 = 5;
   T num2 = 6;
   void add()
   {
      cout << "Addition of num1 and num2 : " << num1+num2<<endl;
   }

};

int main()
{
   A<int> d;
   d.add();
   return 0;
}
```

Overloading a Function Template

- We can overload the generic function means that the overloaded template functions can differ in the parameter list.

```
#include <iostream>
using namespace std;
template<class X> void fun(X a)
{
   cout << "Value of a is : " <<a<< endl;
}
template<class X,class Y> void fun(X b ,Y c)
{
   cout << "Value of b is : " <<b<< endl;
   cout << "Value of c is : " <<c<< endl;
}
int main()
{
  fun(10);
  fun(20,30.5);
  return 0;
}
```

Non-type Template Arguments

- The template can contain multiple arguments, and we can also use the non-type arguments In addition to the type T argument, we can also use other types of arguments such as strings, constant expression and built-in types.

```
template<class T, int size>
class array
{
    T arr[size];          // automatic array initialization.
};
```

- In the above case, the non-type template argument is size and therefore, template supplies the size of the array as an argument.

Arguments are specified when the objects of a class are created:

```
array<int, 15> t1;          // array of 15 integers.
array<float, 10> t2;        // array of 10 floats.
array<char, 4> t3;          // array of 4 chars.
```

```
template<class T, int size>
class A
{
   public:
   T arr[size];
   void insert()
   {
      int i =1;
      for (int j=0;j<size;j++)
      {
         arr[j] = i;
         i++;
      }
   }

   void display()
   {
      for(int i=0;i<size;i++)
      {
         std::cout << arr[i] << " ";
      }
   }
};
```

```
int main()
{
   A<int,10> t1;
   t1.insert();
   t1.display();
   return 0;
}
```

Points to Remember

- C++ supports a powerful feature known as a template to implement the concept of generic programming.
- A template allows us to create a family of classes or family of functions to handle different data types.
- Template functions can also be overloaded.
- We can also use non-type arguments such as built-in or derived data types as template arguments.

References:

- www.studytonight.com
- www.tutorialpoint.com
- www.geeksforgeeks.org
- “Object oriented programming in C++” Robert Lafore
- “Object oriented programming with C++”, E.Balagurusamy