Templates

Introduction

Template is one of the new features of C++. It enables the calling of only one function with generic interface but different types associated with the call.

Templates help in enclosing a class or a function in another canvas called template. The purpose is to define a class or function with generic data types. Then the defined generic type can be substituted with specific types such as int, float, double or char during actual use.

A template can be considered as a kind of macro.

Function Template

To create a function template, we have to enclose the function in a template and declaring a generic data type, which can be called by any name such as T.

```
template <typename T>
void add(T a, T b)
{
   T c;
   c=a+b;
   cout<<"\nThe sum is "<<c;
}
```

The entire piece of code is called a function template. T is a generic data type. The template function is one, which has parameterized at least one of the data types of the arguments. Some compiler will only allow class in place of typename as given below:

```
template <class T>
```

```
#include<iostream>
using namespace std;

template <typename T>
void add(T a, T b)
{
   T c;
   c=a+b;
   cout<<"\nThe sum is "<<c;
}

int main()
{
   add(3,4);
   add(7.0f,2.0f);
   add(3.6,7.3);
}
```

```
#include<iostream>
using namespace std;

template <typename T>
T add(T a, T b)
{
   T c;
   c=a+b;
   return(c);
}
int main()
{
   cout<<add(3,4)<<endl;
   cout<<add(7.0f,2.0f)<<endl;
   cout<<add(3.6,7.3)<<endl;
}
```

Function Template instantiation

The template arguments provide place holders for the data types. A function is instantiated through a function call. Instantiation is substituting the formal parameters with the actual data types. The C++ compiler instantiates a function with one particular data type when the function is called for the first time. The compiler instantiates the function once per data type.

The template is defined with a parameter and substituted with the specified data type at the time of actual use. Hence, the templates are also called parameterized function when it refers to a function and as parameterized class when a template refers to a class.

Function template overloading

The function template can also be overloaded as function overloading. This means that we can have a number of function templates with the same name having a different set of arguments. We can also combine ordinary functions with the same name with overloaded function templates.

Class Template

The purpose of the class template is to create a family of classes with varying types of data members.

```
#include<iostream>
using namespace std;

template <typename T>
class Account
{
   int no;
   T bal;
public:
   void setData(int n,T b)
   {
      no=n;
      bal=b;
   }
   void display()
   {
      cout<<"\nAccount No.: "<<no;
      cout<<"\nBalance    : "<<bal;
   }
};
int main()
{
   Account <int> a1;
   a1.setData(1001,5000);
   Account <double> a2;
   a2.setData(1002,3000.50);
   a1.display();
   a2.display();
}
```

Template for member functions

- To avoid inline functions

```
#include<iostream>
using namespace std;

template <typename T>
class Account
{
   int no;
   T bal;
public:
   void setData(int n,T b);
   void display();
};
```

```
 template <typename T>
 void Account <T> ::setData(int n,T b)
   {
       no=n;
       bal=b;
   }
  template<typename T>
  void Account <T>::display()
   {
       cout<<"\nAccount No.: "<<no;
       cout<<"\nBalance    : "<<bal;
   }
int main()
{
   Account <int> a1;
   a1.setData(1003,3000);
   Account <double> a2;
   a2.setData(1004,5000.50);
   a1.display();
   a2.display();
}
```

Multiple Arguments

```
#include<iostream>
using namespace std;

template <typename T1,typename T2>
T2 add(T1 a, T2 b)
{
   T2 c;
   c=a+b;
   return(c);
}
int main()
{
   cout<<add(3,4.7)<<endl;
   cout<<add(7.0f,2)<<endl;
   cout<<add(3.6,7.3)<<endl;
}
```

```
#include<iostream>
using namespace std;

template <typename T,int size>
class arr
{
   T a[size];
public:
   void getData()
   {
      cout<<"Enter the elements\n";
      for(int i=0;i<size;i++)
      {
         cin>>a[i];
      }
   }
   void display()
   {
      cout<<"The array elements are\n";
      for(int i=0;i<size;i++)
      {
         cout<<a[i]<<"\n";
      }
   }
};

int main()
{
   arr<int,5> a1;
   a1.getData();
   a1.display();
}
```