Templates

The core idea of template is to be able to write a code which at the time of compilation can generate further new code and that generated code get compiled again.

The main motivation of using templates is an attempt to code reuse.

Consider a function, add(), which takes two numbers as parameters, adds these two numbers and returns the added value. If both the numbers are integers then the body of the function will be as follows:

```
int add (int x, int y)
{
    int z;
    z = x+y;
    return z;
}
```

If both the numbers are floats then the body of the function will be as follows:

```
float add (float x, float y)
{
    float z;
    z = x+y;
    return z;
}
```

In both the functions, we can observe that in case of latter, the word 'int' has been replaced with 'float'. Otherwise, everything is same between both the functions. For such cases, C++ provides the concept of templates in which we no need to separately write two functions, one for 'int' and one for 'float'. Here, only one function is written and the data type is passed as

a parameter at the time of calling of the function.
For example,

```
template <class T>
T add (T x, T y)
{
    T z;
    z = x+y;
    return z;
}
int main()
{ int x=2, y=3;
  int z;
  z = add <int> (x,y); /* here the data type 'int' is being passed as a parameter */
  cout << z;

  float a = 2.5, b=3.5;
  float c;
  c = add <float> (a,b);
/* here the data type 'float' is being passes as parameter */
  cout << c;
}
```

In the above program, when 'int' is being passes as a parameter, in this case, the template parameter 'T' in the add() function will be replaced by int and the function add() will behave in such a way that it takes two integers as parameters and returns the sum of these two integeter. Similarly, when float is being passed as a parameter then the template parameter T in the add() function will be replaced with float

Similarly, consider a function max() which returns the maximum value among two variables. In case of two character variables, it returns a value with greater ASCII value.

```
template <class T>
  T max(T x, T y)
  {
     if (x > y)
       return x;
     else return y;
  }
int main()
  {
     int x=2, y=30, z;
     z = max<int>(x,y);
     cout << z;
     float a = 2.5, b = 3.5, c;
        c = max<float>(a,b);
     char cout << c;
     char d = 'm', e = 'N', f;
        f = max<char>(d,e);
        cout << f;
  }
```

~~Function overloading in template~~

Function template with multiple parameters: In case of ~~template~~ parameters in function, we know that there can be be more than one parameter with different data type.

e.g. func(int x, float y)

Similarly, in case of function template also, there can be multiple template parameters.

```
template <class T1, class T2>
void fun(T1 x, T2 y)
{
    cout << "double template";
}

int main()
{
    fun<int, float>(5, 6.5);
}
```

=) In case of passing a single parameter, we can remove the angular bracket while calling the function.

e.g.

```
template <class T>
void fun(T x)
{
    cout << "template";
}
int main()
{
    fun(5);   // here <int> is removed
}
```

But, in case of multiple template parameters, in order to remove the angular bracket,

But, in this case, in case of multiple parameters with single template parameter, the data types of the parameters being passed must be same otherwise we need to use the angular bracket for internal type conversion. e.g.

```
template <class T>
void fun(T x, T y)
{
    cout << x << y;
}
main()
{
    fun(5, 5);   // it will print 5 5
}
```

fun(5, 6.5); /* it will give error bcoz data types of parameters must be same as we used only single template parameter & we are not using angular bracket while calling the function. */

fun 6<int>(5, 6.5); /* it will print 5, 6 here, 6.5 will truncated to 6 since we are using <int>.

}

Function overloading in template

As we know that, primary condition for function overloading is (i) name of the functions must be same.

Similarly, in case of template also, above rules must be followed. Apart from that, in a program, both functions with normal parameters as well as template parameters may exists, forming the situation of function overloading. For example,

```
void fun(T x);     (function with template parameter)
void fun(int y);   (func^n with normal parameter)
```

In such situations, if a function call exactly matches with the function with normal parameter then this function will be invoked and not the function with template parameter.

In case of function with multiple parameters having single template parameter, implicit type conversion will be done with function with normal parameter for function overloading.

```
template < class T >
void fun( T x, T y)
{ cout << "Single temp" << x << y;
}
template < class T1 , class T2 >
void fun ( T1 P, T2 q)
{
cout << "multi temp" << P << q;
}
void fun( int m, int n )
{
cout << "normal" << m << n;
}
int main( )
{
fun( 5, 5); /* normal 5 5 */
fun (6.5, 6.5); /* Single temp 6.5 6.5 */
fun (5, 6.5); /* multi temp 5 6.5 */
}
```

```
template < class T >
void fun( T x, T y)
{ cout << " Single temp" << x << y;
}
template
void fun( int m, int n)
{ cout << "normal" << m << n;
}
```

```
int main( )
{
    fun(10, 10);       /* normal      10  10 */

    fun(9.5, 9.5);     /* Single temp  9.5 9.5 */

    fun(9.5, 10);      /* normal  9  10 */
```

here, implicit type conversion will be done with the function with normal parameters.

```
}
```

=) As of now, we learned how to use template in a function. As we know that, a class consists of member function as well as data member. So, in order to use the concept of template in a class, we not only need to templatize the member functions but also the data member. For example,

```
class abc
{
    int x;
    float y;
    Public:
    void Setdetail (int m, float n)
    {  x=m;
       y=n;
    }
    int square_x ( )
    {  return x*x;
    }
    float square_y ( )
    {  return y*y;
    }
};
```

If we want to use the concept of template in the class "abc" then the class may look as follows:

Date
Page

```
template < class T1, class T2 >
class abc
    {
      T1 x;
      T2 y;
    Public:
      void Setdetail (T1 m, T2 n)
        {
          x = m;
          y = n;
        }
      T1 Square_x ( )
        {
          return x*x;
        }
      T2 Square_y ( )
        {
          return y*y;
        }
    };

int main( )
  {
    abc <int, float> m;
    m.Setdetail (5, 6.5);
    int p;
    p = m.Square_x( );
    cout << p;          /* 25 will be printed */
    float q;
    q = m.Square_y( );
    cout << q;          /* 42.25 will be printed */
  }
```