RESEARCH PAPER | Engineering | Volume : 2 | Issue : 1 | October 2012 | ISSN - 2249-555X

Introduction to Templates

KEYWORDS

Mr. A. N. Shah
P.G. Student,Computer Department, C.U.Shah College of Engineering & Technology, Wadhwan City

Mr. N. N. Shah
Lecturer (E.C.), R.K. University school of Engineering, Rajkot.

ABSTRACT *Templates allow us to define generic class and function. So it supports generic programming. Generic programming is an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structures. A template can be used to create a family of classes or functions. For example, it can be used with class like student etc and it also can be used for function mul() etc. A template is also known as a kind of macro. Whenever template is created for any class it required substituted data types. Since template is define with a parameter for classes or functions so it is sometime called as parameterized classes or functions.*
The main aim of this thesis is to show various ways to use template either with function or class.

INTRODUCTION
Template is one which supports generic class and function. So it supports generic programming. Generic programming is an approach where generic types are used as parameters in algorithms so it supports variety of suitable data types and data structures.

CLASS TEMPLATE
Following is syntax of class template.

```
template <class T>
class class-name

{
//.....
// class members specification
// with anonymous type T
// whenever appropriate
//.....
};
```

As shown in syntax we can give any valid name in place of T. Now consider the following class, class student

```
{
int rno;
public:
void input()
{
cout<<"\n Enter rno:";
cin>>rno; }
void display()
{ cout<<"\n Rno : "<<rno; }
};
```

Here now if I want to change data of rno from int to float then I have to redefine whole class instead if doing this use template. Consider same example using template.

```
#include<iostream.h>
#include<conio.h>
template <class ans>
class student
{
ans rno;
public:
void input()
{
    cout<<"\n Enter rno:";
    cin>>rno;
}
void display()
{    cout<<"\n Rno : "<<rno;   }
};
void main()
{
clrscr();
student <int> s;
s.input();
s.display();
student <float> s1;
s1.input();
s1.display();
getch();
}
```

OUTPUT:

```
Enter rno : 4
Rno : 4
Enter rno : 1.3
Rno : 1.3
```

As shown in example first we have pass integer value and then we have pass float value this can be done easily using template no need to redefine class.

CLASS TEMPLATE WITH MULTIPLE PARAMETERS
We can use more then one generic data types in a class template. Following is an example which illustrates this.

```
#include<iostream.h>
#include<conio.h>
template <class T1,class T2,class T3>
class student
{
    T1 rno;
    T2 name[20];
    T3 per;
 public:
    void input()
    {
            cout<<"\n Enter rno,name,per:";
            cin>>rno>>name>>per;
    }
    void display()
    {
            cout<<"\n Name : "<<name;
            cout<<"\t Rno : "<<rno;
            cout<<"\t Per : "<<per;
    }
};
void main()
{
    student <int,char,float> s;
    s.input();
    s.display();
    getch();
```

Volume : 2 | Issue : 1 | October 2012 | ISSN - 2249-555X

```
}
OUTPUT:
Enter rno,name,per:1 ans 80
Name : ans Rno : 1 Per : 80
```

FUNCTION TEMPLATE

We can also create template for function, following is syntax,

```
template<class T>
return-type function-name(arguments of type T)
{
    //.....
    // Body of function
    // with anonymous type T
    // whenever appropriate
    //.....
}
```

This syntax is much similar as template syntax for class. In place of T we can give any valid name. Following is example.

```
#include<iostream.h>
#include<conio.h>
template <class t>
void display(t a)//function with template
{
    cout<<"\n Value : "<<a;
}
void main()
{
    clrscr();
    display(5);
    display("ans");
    getch();
}
OUTPUT:
    Value : 5
    Value : ans
```

FUNCTION TEMPLATE WITH MULTIPLE PARAMETERS

Like template class we can use more than one generic data type in function template. Following is syntax,

```
template<class T1,class T2>
return-type function-name(arguments of types T1,T2,...)
{   // Body of function          }
```

Following is an example.

```
#include<iostream.h>
#include<conio.h>
template <class t1,class t2>
void sum(t1 a,t2 b)
{
    float c;
    c = a + b;
    cout<<endl<<a<<" + "<<b;
    cout<<"\t = "<<c;
}
void main()
{
    clrscr();
    sum(5,4.3);
    sum(1.4,2);
    getch();
}
OUTPUT:
    5 + 4.3          = 9.3
    1.4 + 2          = 3.4
```

OVERLOADING TEMPLATE FUNCTIONS

A template function may be overloaded either by template functions or ordinary functions of its name. In such case compiler find match as follow.

[1] Call an ordinary function that has an exact match.
[2] Call a template function that could be created with an exact match.
[3] Try normal overloading resolution to ordinary functions and call the one that matches.

An error is generated if no match found. Note that no automatic conversions are applied to arguments on the template functions. Following is example.

```
#include<iostream.h>
#include<conio.h>
template <class t>
void display(t a)
{   cout<<"\n Value [using template]: "<<a;       }
void display(int b)
{ cout<<"\n Value [without using template]: "<<b; }
void main()
{
    clrscr();
    display(5);
    display(2.5);
    display("hi");
    getch();
}
OUTPUT:
Value [without using template]: 5
Value [using template]: 2.5
Value [using template]: hi
```

CONCLUSION

When in any program we need to change our arguments data type then template is best option to use. So by using template each time we do not need to rewrite our class or function to change arguments data types.

REFERENCES

[1] Object-Oriented Programming With C++ by E Balagurusamy | [2] T. Becker. Type erasure in C++: The glue between object oriented and generic programming. In K. Davis and J. Striegnitz, editors, Multiparadigm Programming 2007: Proceedings of the MPOOL Workshop at ECOOP'07, July 2007. | [3] J. C. Dehnert and A. A. Stepanov. Fundamentals of Generic Programming. In Selected Papers from the International Seminar on Generic Programming, pages 1-11, London, UK, 2000. Springer-Verlag. | [4] Peter Pirkelbauer, Sean Parent, Mat Marcus and Bjarne Stroustrup "Runtime Concepts for the C++ Standard Template Library"