TEMPLATES

BY: RICHA JAIN

Generic Programming

It 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.

- Generic classes
- Generic functions

Templates

- A template is a mechanism that allows you to create functions and classes that can be reused with every data types.
- Templates are referred as generic types.
- C++ provides two kinds of templates
  - Class template
  - Function template
- Advantage → avoid repetition of source code.

Templates

- A template can be considered as a kind of macro.
- For actual use, template definition for that class is substituted with the required data types.
- Template is defined with a parameter that would be replaced by a specific data type at the time of actual use of class or function, the templates are sometimes called as **parameterized classes or functions.**

Example

- To add 2 no.(2 int, 2 float, 2 double)
- One method can be – define a different function for each
- Second method can be- define a template with generic datatype as parameter

Syntax of class Template

```
template<class T>
class classname
{
//..............
//Class member specification with
//anonymous type T
 //wherever appropriate.
..............
};
```

T may be substituted by any datatype including user defined datatypes.

Example

```
#include<iostream.h>
template <class T>
class mix
{
T a,b;
public:
mix(T x,Ty)
{
a=x;
b=y;
}
T max()
{
return (a>b?a:b);
};
int main()
{
mix<int> obj(10,20);
Cout<<“max value is”<<obj.max();
}
```

Class template with multiple parameters

Template <class T1, class T2,….>

Class *classname*

{

……..

………(body of the class)

};

Two generic data types in a class

```
#include<iostream>
template<class T1, class T2>
class Test
{
T1  a;
T2  b;
Public:
Test(T1  x, T2  y)
{    a=x;
     b=y;}
void show(){cout<<a<<“and”
<<b;}
};
```

```
int main()
{
Test <float,int> test1(1.23,12);
Test <int ,char> test2(20, ‘z’);
test1.show();
test2.show();
return 0;
}
```

Output:

```
1.23 and 12
20 and z
```

Function Templates

```
template <class T>
returntype functionname(arguments of type t )
{
// body of function with type T
………..
}
```

Example of function template

```
#include <iostream.h>
template<class T>
 void swap(T &x, T &y)
{
    T temp= x;
    x=y;
    y= temp;
}
void fun(int m, int n,float a, float b)
{
cout<<m and n before swap :<<m
    <<""<<n;
    swap(m,n);
cout<< m and n after swap:<<m
<<""<<n;
cout<<a  and b before swap: << a
<<""<<b;
    swap(a,b);
cout<< a and b after swap: <<a
<<""<<b;
}
int main()
{
    fun(10,30,23.32,45.32);
    return 0;
}
```

Function templates with multiple parameters

```
template<class T1,class T2,…..>
returntype functionname(arguments of type t1,t2,…..)
{
   //………
   //body of the function
   //……..
}
```

Function with two generic types

```
#include<iostream.h>
#include <conio.h>
template<class T1,class T2>
void display(T1 x, T2 y)
{
cout<<x<<“”<<y<<“\n”;
}
int main()
{
display(10,25.34);
display(“asdf”,200);
return 0;
}
```

Overloading of template functions

- A template function may be overloaded either by a template function or ordinary function of its name.
- Overloading resolution is accomplished as:
  - Call an ordinary function that has an exact match.
  - Call a template function that could be created with an exact match.
- An error is generated if no match is found. Note that no automatic conversions are applied to arguments on the template functions.

Overloading of template function

```
#include<iostream.h>
#include<conio.h>
Template<class T>
Void display(T x)
{
    cout<<“Template display”<<x<<“\n”;
}
Void display(int x)
{
    cout<<“Explicit display<<x<<“\n”;
}
```

```
Int main()
{
display(100);
display(56.78);
display(‘a’);
return 0;
}
```

Member function templates

```
Template <class T>
returntype classname<T>::functionname(argument_list)
{
//……
//function body
//……
}
```

Example

```
// class templates
 #include <iostream>
using namespace std;
template <class T>
class mypair {
T a, b;
 public:
mypair (T first, T second)
   {a=first; b=second;}
T getmax ();
};
```

```
template <class T>
T mypair<T>::getmax ()
{ T retval;
retval = a>b? a : b;
 return retval; }
 int main ()
 {
 mypair <int> myobject (100,
   75);
cout << myobject.getmax();
   return 0; }
```

OUTPUT: 100