Object Oriented Programming using C++

Topic : Templates in C++

Templates

- Templates support generic programming, which allows to develop reusable software components such as function, class, etc.
- Supporting different data types in a single framework.
- A template in C++ allows the construction of a family of template functions and classes to perform the same operation on different data types.
- The templates declared for functions are called function templates and those declared for classes are called class templates.
- It allows a single template to deal with a generic data type T.

Function Templates

- There are several functions of considerable importance which have to be used frequently with different data types.
- The limitation of such functions is that they operate only on a particular data type.
- It can be overcome by defining that function as a function template or generic function.
- Syntax:

```
template <class T, …..>
returntype function_name (arguments)
{
      ……   // body of template function
      ……
}
```

Ex : Multipe swap functions

```
#include<iostream.h>
Void swap(char &x, char &y)
{        char t;
         t = x;    x = y;   y = t;
}
Void swap(int &x, int &y)
{        int t;
         t = x;    x = y;   y = t;
}
Void swap(float &x, float &y)
{        float t;
         t = x;    x = y;   y = t;
}
```

```
Void main()
{
    char ch1, ch2;
    cout<<“\n Enter values  : ”;
    cin>>ch1>>ch2;
    swap(ch1,ch2);
    cout<<“\n After swap ch1 = 
”<<ch1<<“ ch2 = ”<<ch2;
    int a, b;
    cout<<“\n Enter values  : ”;
    cin>>a>>b;
    swap(a,b);
    cout<<“\n After swap a = 
          ”<<a<<“ b = ”<<b;
    float c, d;
    cout<<“\n Enter values  : ”;
    cin>>c>>d;
    swap(c,d);
    cout<<“\n After swap c =
          ”<<c<<“ d = ”<<d;
}
```

Output:

```
Enter values : R K
After swap ch1 = K
ch2 = R
Enter values : 5 10
After swap a = 10
b = 5
Enter values : 20.5 99.3
After swap c = 99.3
d = 20.5
```

Generic fuction for swapping

```
#include<iostream.h>
Template<class T>
Void swap(T &x, T &y)
{       T t;
        t = x;    x = y;   y = t;
}
Void main()
{
    char ch1, ch2;
    cout<<“\n Enter values  : ”;
    cin>>ch1>>ch2;
    swap(ch1,ch2);
    cout<<“\n After swap ch1 =
”<<ch1<<“ ch2 = ”<<ch2;
     int a, b;
    cout<<“\n Enter values  : ”;
    cin>>a>>b;
    swap(a,b);
    cout<<“\n After swap a =
          ”<<a<<“ b = ”<<b;
    float c, d;
    cout<<“\n Enter values  : ”;
    cin>>c>>d;
    swap(c,d);
    cout<<“\n After swap c =
          ”<<c<<“ d = ”<<d;
}
```

output :

same as previous example

Function and Function Template

- Function templates are not suitable for handling all data types, and hence, it is necessary to override function templates by using normal functions for specific data types.

```
Ex: #include<iostream.h>
    #include<string.h>
    template <class T>
    T max(T a, T b)
    {    if(a>b)
              return a;
          else
              return b;
    }
    char *max(char *a, char *b)
    {    if(strcmp(a,b)>0)
              return a;
```

```
          else
              return b;
    }
    void main()
    {
          char ch,ch1,ch2;
          cout<<“\n Enter two
                  char value : ”;
          cin>>ch1>>ch2;
          ch=max(ch1,ch2);
          cout<<“\n max value ”
                <<ch;
    int a,b,c;
    cout<<“\n Enter two int
        values : ”;
    cin>>a>>b;
    c=max(a,b);
    cout<<“\n max value : ”<<c;
    char str1[20],str2[20];
    cout<<“\n Enter two str
        values : ”;
    cin>>str1>>str2;
    cout<<“\n max value : ”
        <<max(str1,str2);
}
```

```
Output :
Enter two char value : A  Z
Max value : Z
Enter two int value : 12  20
Max value : 20
Enter two char value :
Tejaswi Rajkumar
Max value : Tejaswi
```

- In the above example if we not use the normal function, when a statement call such as,

  max(str1,str2)

- It is executed, but it will not produce the desired result. The above call compares memory addresses of strings instead of their contents.
- The logic for comparing strings is different from comparing integer and floating point data types.
- It requires the normal function having the definition but not the function template.
- We can use both the normal function and function template in a same program.

Overloaded Function Templates

- The function template can also be overloaded with multiple declarations.
- It may be overloaded either by functions of its mane or by template functions of the same name.
- Similar to overloading of normal functions, overloaded functions must differ either in terms of number of parameters or their types.

```
Ex : #include<iostream.h>
      template <class T>
      void print(T data)
      {   cout<<data<<endl; }
      template <class T>
      void print(T data, int
                    ntimes)
      {   for(int i=0;i<ntimes;i++)
               cout<<data<<endl;
      }
     void main()
     {   print(1);
         print(1.5);
         print(520,2);
         print("OOP is Great",3)
     }
```

```
Output :
1
1.5
520
520
OOP is Great
OOP is Great
OOP is Great
```

Class Templates

- Class can also be declared to operate on different data types. Such class are called class templates.
- A class template specifies how individual classes can be constructed similar to normal class specification.
- These classes model a generic class which support similar operations for different data types.
- Syntax :

```
template <class T1, class T2, …..>
class class_name
{
    T1 data1;  // data items of template type
    void func1 (T1 a, T2 &b); // function of template
                                          argument
    T func2 (T2 *x, T2 *y);
}
```

Example :

```
Class charstack
{     char array[25];
      unsigned int top;
      public:
          charstack();
          void push(const char
                         &element);
          char pop(void);
          unsigned int getsize
                     (void) const;
};
Class intstack
{     int array[25];
      unsigned int top;
       public:
          intstack();
          void push(const int
                         &element);
          int pop(void);
          unsigned int getsize
                     (void) const;
};
Class doublestack
{     double array[25];
      unsigned int top;
      public:
          doublestack();
          void push(const
          double &element);
          double pop(void);
          unsigned int getsize
                     (void) const;
};
```

- In the previous example, a separate stack class is required for each and every data types. Templates declaration enables subatitution of code for all the three declarations of stacks with a single template class as follows :

```
template<class T>
class datastack
{     T array[25];
       unsigned int top;
      public:
            doublestack();
            void push(const double &element);
            double pop(void);
            unsigned int getsize (void) const;
};
```

Inheritance of Class Template

Use of templates with respect to inheritance involves the followings :

- Derive a class template from a base class, which is a template class.
- Derive a class template from a base class, which is a template class, add more template members in the derived class.
- Derive a class from a base class which is not a template, and template member to that class.
- Derive a class from a base class which is a template class and restrict the template feature, so that the derived class and its derivatives do not have the template feature.

The syntax for declaring derived classes from template-based base classes is as :

```
template <class T1, …..>
class baseclass
{
    // template type data and functions
};
template <class T1, …..>
class derivedclass : public baseclass <T1, ….>
{
    // template type data and functions
};
```