TEMPLATES

What is Template???

- *Template* is an added feature which creates and defines generic class and generic functions, and hence it's also referred as *generic programming*.

Function Templates…

Syntax:

```
template <class user-defined data type>
return type function name(list of arguments)
{
        Body of the function
}
```

Example:

```
template<class T>
void fun()
{
        cout << x ;
}
```

Find the Square of a number

```
template <class T>
T square(T x)
{
    return x*x;
}
```

```
int main()
{
    int a = 10, b ;
    b = square( a );
    cout<< b;
    float p = 2.5, q;
    q = square( p );
    cout<<q;
}
```

Output:

100

6.25

Swapping of two values

```
template<class T>
void swap(T &x , T &y)
{
      T temp;
      temp=x;
      x=y;
      y=temp;
}
```

```
int main()
{
      int a = 10 ,  b = 20;
      cout<<“Before swapping:: ”<<a<<“ “<<b;
      swap(a,b);
      cout<<endl;
      cout<<“After swapping:: ”<<a<<“ ”<<b;
      cout<<endl;
      float m = 12.35 , n = 7.28;
      cout<<“Before swapping:: ”<<m<<“ ”<<n;
      swap(m , n);
      cout<<endl;
      cout<<“After swapping:: ”<<m<<“ ”<<n;
```

Output:

Before Swapping::10 20

After Swapping::20 10

Before Swapping::12.35 7.28

After Swapping::7.28 12.35

Function Templates with multiple parameters

Syntax:

```
template<class T1,class T2,… >

return type function name(Arguments type of T1,T2,… )
{
        //Body of the function
}
```

Example…

```
template<class T1,class T2>

void stdinfo (T1 sno , T2 mark)
        {
        cout<<sno<<"  "<<mark;
        }
```

```
int main()
{
     stdinfo("S001",72);
     stdinfo(12.55,67);
}
```

Output-

S001 72

12.55 67

Class Template

Syntax:

```
template<class T>
class Class – Name
{
        private:
                <M . V. >;
        public:
                <M . F . >;
};
```

Sum of two data using generic class

```
template<class T>
class Sample
{
      T x , y , z;
      public:
      void get()
      {
        cout << “Enter two values of x and y::”
      <<endl;
            cin>>x>>y;
      }
      void sum()
      {
                  z=x+y;
                  cout<<z;
      }
};
```

```
void main()
{
      Sample<int> s1;
      Sample<float> s2;

      s1.get();
      s1.sum();

      s2.get();
      s2.sum();
}
```

Class Template with Multiple Parameters

Syntax:

```
template<class T1,class T2,…>
class Class - Name
{
        //Body of the Class
};
```

Example…

```
template<class T1,  class T2>
class Sample
{
    T1  x ; T2  y;
    public:
    Sample(T1 a, T2 b)
    {
            x = a ; y = b;
    }
    void show()
    {
            cout<<x<<"    "<<y;
    }
};
```

```
void main()
{
    Sample<int , float>s1(22,12.33);
    Sample<float, char>s2(12.44,'c');

    s1.show();
    s2.show();
}
```

OverLoading of Template Functions

A template function may be overloaded either by template functions or ordinary functions of it name. In such situations the overloading resolution is done as follows:

1) Call an ordinary function that has an exact match.
2) Call a template function that could be created with an exact match.

Example

```
template<class T>
void disp( T  x )
{
    cout<<“Generic Function  ”<<x;
}

void  disp(int y)
{
    cout<<“Ordinary Function  ”<<y;

}
```

```
void main()
{
    disp(100);
    disp(13.22);
    disp(‘Z’);
}
```

Output-

Ordinary Function 100

Generic Function 13.22

Generic Function Z

Member Function Template

When we create a class template we can define the member function of the class outside by using scope resolution operator.

Syntax:

```
template<class T>
return type classname<T>::function name (Argument List)
{
        //Body of the class
}
```

Example

```
template<class T>
class Sample
{
    T x , y , z;
    public:
    void get( T, T );
    void show();
};
template<class T>
void Sample <T>::get(T a , T b)
{
    x = a;
    y = b;
}
```

```
template<class T>
void Sample <T>::show()
{
    cout <<x + y ;  endl ;
}
Void main()
{
    Sample<int>s1;
    Sample<float>s2;
    s1.get(7 , 5);
    s1.show();

    s2.get(12.33,66.88);
    s2.show();
}
```

Usage of friend function in Template

```
template<class T>

Class Sample
{
    T  x ;
    friend void disp(Sample<T>)
};
template<class T>
void disp(Sample<T>obj)
{
    T y ;
    cout<<“Enter the Value::”;
    cin>> obj . x;
    y= obj . x ;
    cout<< y ;
}
```

```
int main()
{
    Sample<int> s1;
    Sample<char> s2;
    disp (s1);
    disp (s2);
}
```