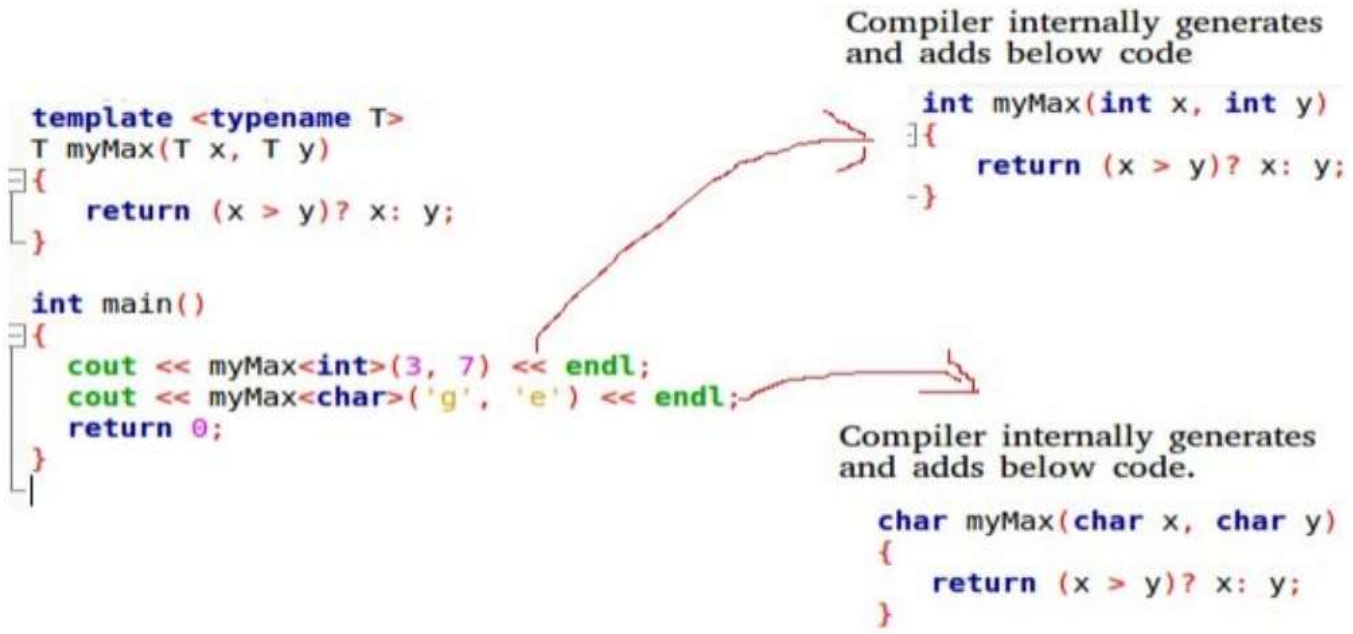
Templates

- Templates is one of the C++ features which enable us to define generic classes and functions and thus provides support for 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, a class template for an array class would enable us to create arrays of various data types such as int array and float array.
- Similarly, we can define a template for a function, say mul(), that would help us create various versions of mul() for multiplying int, float and double type values.
- Since a template is defined with a parameter that would be replaced by a specified data type at the time of actual use of the class function, the templates are sometimes called parameterized classes or functions.

Working of Templates



Templates can be represented in two ways:

1. Function template : used to create a family of functions with different argument types.

2. Class template : a class defines something that is independent of the data type.

Function Templates

General format

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

Example

```
#include<iostream>
using namespace std;
template<class T>
void swap(T &x, T &y)
{
        T temp = x;
        x = y;
        y = temp;
}
void func(int m, int n, float a, float b)
{
 cout<<“m and n before swap:” <<m << “  “ <<n<<“\n”;
swap(m,n);
cout<<“m and n after swap:” <<m << “  “ <<n<<“\n”;
cout<<“a and b before swap:” <<a << “  “ <<b<<“\n”;
swap(a,b);
cout<<“a and b after swap:” <<a << “  “ <<b<<“\n”;
}
```

```
int main()
{
        func(100,200,11.22,33.44);
        return 0;
}
```

Output

```
m and n before swap : 100 200
m and n after swap : 200 100
a and b before swap : 11.22 33.44
a and b after swap : 33.44 11.22
```

Class Templates

General format

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

```
#include<iostream>
using namespace std;
template<class T>
class Number
{
  T num;
  public:
    Number(T n): num(n) { }
    T getNum(){
        return num;
   }
};
```

```
int main()
{
  Number<int> numberInt(7);
  Number<double> numberDouble(7.7);
  cout<<“ int Number = “ << numberInt.getNum() <<endl;
  cout<<“ double Number = “ << numberDouble.getNum() <<endl;
  return 0;
}
```

Output

```
int Number = 7
double Number = 7.7
```