OBJECT ORIENTED PROGRAMMING (PCC-CS503)

Unit – 6

Generic Programming

- Generic Programming is a style of computer programming that enables the programmer to write a general algorithm (function/class) which will work with all data types.
- It eliminates the need to create different algorithms for different types of data.
- Algorithms are written in terms of types *to-be-specified-later* that are then *instantiated* when needed for specific types provided as parameters.
- The advantages of Generic Programming are:
    - Code Reusability
    - Avoid Function Overloading
    - Once written it can be used for multiple times and cases.

Template concept

- Templates help implementing generic programming.
- A single class or a function can work for different data types using templates.
- Generic programming is a technique where generic types are used as parameters in algorithms so that they can work for a variety of data types.
- Templates can be represented in two ways:
    - Function templates
    - Class templates

Function template

- Generic functions use the concept of a function template. Generic functions define a set of operations that can be applied to the various types of data.
- The type of the data that the function will operate on depends on the type of the data passed as a parameter.
- For example, Quick sorting algorithm is implemented using a generic function, it can be implemented to an array of integers or array of floats.
- A Generic function is created by using the keyword template. The template defines what function will do.

Syntax:

```
template <class T> T func_name(parameter_list)
{
    // body of function.
}
```

T: It is a placeholder name for a data type used by the function. It is used within the function definition. It is only a placeholder that the compiler will automatically replace this placeholder with the actual data type.

class: A class keyword is used to specify a generic type in a template declaration. It can also be replaced by keyword **typename**.
Example:

```
#include <iostream>
using namespace std;
template<class T, class X>
T add(T a, X b)
{
    T result = a+b;
    return result;

}
int main()
{
  int i =2;
  int j =3;
  float m = 2.3;
  float n = 1.2;
  cout<<"Addition of i and j is: "<<add(i,j);
  cout<<'\n';
  cout<<"Addition of m and n is: "<<add(m,n);
  return 0;
}
```

O/P:

Addition of i and j is: 5
Addition of m and n is: 3.5

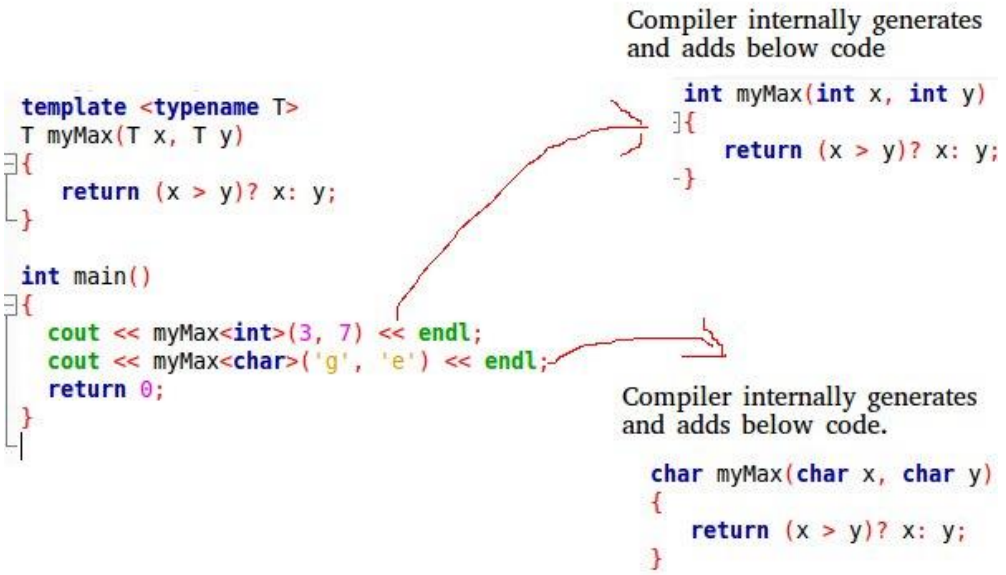
How templates work?

Example 2:

```
#include <iostream>
using namespace std;

// One function works for all data types. This would work
// even for user defined types if operator '>' is overloaded

template <typename T>
T myMax(T x, T y)
{
return (x > y)? x: y;
}

int main()
{
cout << myMax<int> (3, 7) << endl; // Call myMax for int
cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double
cout << myMax<char>('g', 'e') << endl; // call myMax for char
return 0;
}
```

O/P:
7 7 g

Templates are expanded at compiler time. Compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class.



NOTE: We can define more than one generic data type using a comma-separated list.

Class template

Class Template can also be defined similarly to the Function Template. When a class uses the concept of Template, then the class is known as generic class.

Syntax:

```
template<class T> class class_name
{
   .
   .
}
```

T: is a placeholder name which will be determined when the class is instantiated. We can define more than one generic data type using a comma-separated list. The T can be used inside the class body.

Now, we create an instance of a class

Syntax:

```
class_name<type> ob;
```

class_name: It is the name of the class.
type: It is the type of the data that the class is operating on.
ob: It is the name of the object.

Example:

```
#include <iostream>
using namespace std;
```

```
template<class T > class A   //creating the generic class

{
    public:
    T num1 = 5;
    T num2 = 6.5;
    void add()
    {
       std::cout << "Addition of num1 and num2 : " << num1+num2<<std::endl;
    }
};

int main()
{
    A<int,float> d;  //creating instance of the generic class
    d.add();
    return 0;
}
```

O/P:

Addition of num1 and num2 : 11