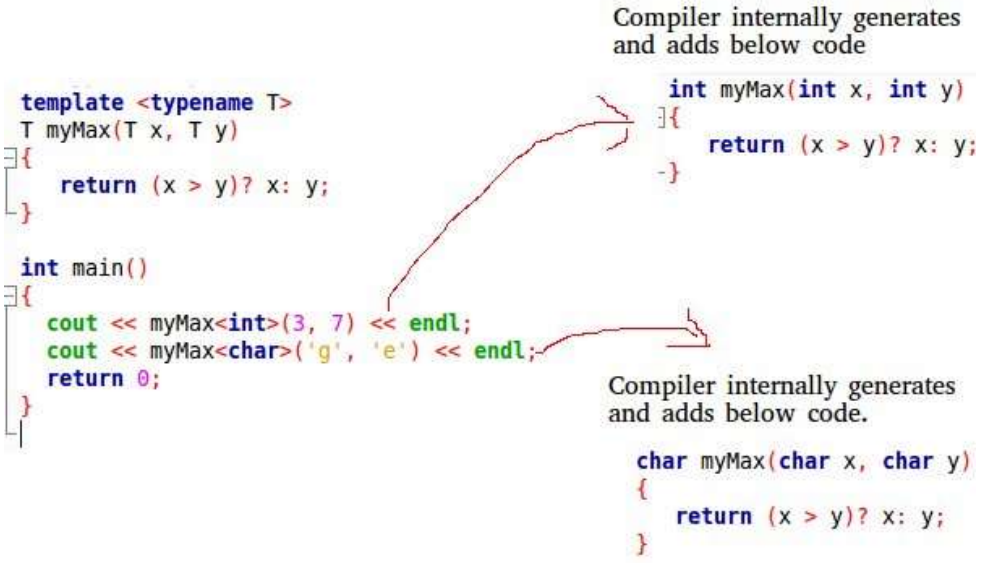
TEMPLATES

- Templates is a concept which enables us to define generic classes and functions and thus provide support for generic programming.
- Generic programing is an approach where generic types are used as parameters in algorithm so that they work for a variety of suitable data types and data structures.
- Example – class template for an array would enable us to create array of various data types.
- Thus templates can be used in applications wherein we require the code to be usable for more than one data types. Templates are also used in applications where code reusability is of prime importance.



IMPLEMENTATION

- Templates can be implemented in two ways:
  - As a function template
  - As a class template

FUNCTION TEMPLATE

- Function Template is just like a normal function, but the only difference is while normal function can work only on one data type and a function template code can work on multiple data types.
- While we can overload a normal function to work on various data types, function templates are always more useful as we have to write the only program and it can work on all data types.
- Syntax

```
template<class T>
 T function_name(T args){
 ……
 //function body
 }
```

CLASS TEMPLATES

- Like in function templates, we might have a requirement to have a class that is similar to all other aspects but only different data types.
- In this situation, we can have different classes for different data types or different implementation for different data types in the same class. But doing this will make our code bulky.
- The best solution for this is to use a template class. Template class also behaves similar to function templates. We need to pass data type as a parameter to the class while creating objects or calling member functions.

CLASS TEMPLATE

```
template <class T>
class className{
…..
public:
                    T memVar;
                    T memFunction(T args);
};

className<int> classObejct1;
className<float> classObject2;
className<char> classObject3;
```

TEMPLATE INSTANTIATION

- The templates are written in a generic way, which means that it's a general implementation irrespective of the data type. As per the data type provided, we need to generate a concrete class for each data type.
- For Example, if we have a template sort algorithm, we may generate a concrete class for sort<int>, another class for sort<float>, etc. This is called instantiation of the template.
- We substitute the template arguments (actual data types) for the template parameters in the definition of the template class.
- Example -

template <class T>

class sort {};

When we pass <int> data type, the compiler substitutes the data type <int> for 'T' so that the sorting algorithm becomes sort<int>.

TEMPLATE SPECIALIZATION

- While programming using templates, we might be faced with a situation such that we might require a special implementation for a particular data type. When such a situation occurs, we go for template specialization.

- In template specialization, we implement a special behavior for a particular data type apart from the original template definition for the other data types.