**C++ Templates**

A C++ template is a powerful feature added to C++. It allows you to define the generic classes and generic functions and thus provides support for generic programming. 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 Templates:**

We can define a template for a function. For example, if we have an add() function, we can create versions of the add function for adding the int, float or double type values.

**Class Template:**

We can define a template for a class. For example, a class template can be created for the array class that can accept the array of various types such as int array, float array or double array.

## 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 of Function Template

template < class Ttype> ret\_type  func\_name(parameter\_list)

{

    // body of function.

}

**Or**

**template** <**class** T>

T someFunction(T arg)

{

... .. ...

}

Where **Ttype**: 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.

**Example:**

#include <iostream.h>

#include<conio.h>

template<class T>

T add(T &a,T &b)

{

T result = a+b;

return result;

}

void  main()

{

int i =2;

int j =3;

float m = 2.3;

float n = 1.2;

clrscr();

cout<<"Addition of i and j is :"<<add(i,j);

cout<<'\n';

cout<<"Addition of m and n is :"<<add(m,n);

getch();

}

**Output:**

Addition of i and j is :5

Addition of m and n is :3.5

## 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 Ttype>

class class\_name

{

  .

  .

}

**Ttype** 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 Ttype can be used inside the class body.

Now, we create an instance of a class

**class\_name<type> ob;**

**where 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.h>

#include <conio.h>

template<class T>

class A

{

    public:

     T num1 = 5;

     T num2 = 6;

    void add()

    {

        cout << "Addition of num1 and num2 : " << num1+num2<<endl;

    }

};

void  main()

{

    A<int> d;

    d.add();

    getch();

}

**Output:**

Addition of num1 and num2 : 11

# C++ ‘this’ Pointer

The **this** pointer holds the address of current object, in simple words you can say that this [pointer](https://beginnersbook.com/2017/08/cpp-pointers/) points to the current object of the class.

**C++ Example: this pointer**

Here you can see that we have two data members num and ch. In member function setMyValues() we have two local variables having same name as data members name. In such case if you want to assign the local variable value to the data members then you won’t be able to do until unless you use this pointer, because the compiler won’t know that you are referring to object’s data members unless you use this pointer. This is one of the example where you must use **this** pointer.

#include <iostream.h>

#include<conio.h>

class Demo {

private:

int num;

char ch;

public:

void setMyValues(int num, char ch)

{

this->num =num;

this->ch=ch;

}

void displayMyValues( )

{

cout<<num<<endl;

cout<<ch;

}

};

void main(){

Demo obj;

obj.setMyValues(100, 'A');

  obj.displayMyValues();

getch();

}

**Output:**

100

A