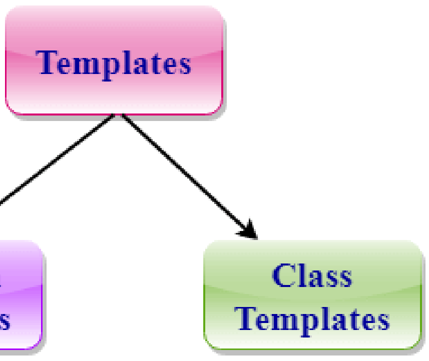
Templates in C++

A template is a simple yet very powerful tool in C++. The simple idea is to pass the data type as a parameter so that we don’t need to write the same code for different data types. For example, a software company may need to sort() for different data types. Rather than writing and maintaining multiple codes, we can write one sort() and pass the datatype as a parameter.

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.

Let's see a simple example of a function template:

```
 #include <iostream>

using namespace std;

template<class T> T add(T &a,T &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);



}
```

Let's see a simple example:

```
#include <iostream>
using namespace std;
template<class X,class Y> void fun(X a,Y b)
{
    cout << "Value of a is : " <<a<< endl;
    cout << "Value of b is : " <<b<< endl;
}
int main()
{
   fun(15,12.3);
}
```

Let's understand this through a simple example:

```
#include <iostream>
using namespace std;
template<class X> void fun(X a)
{
    cout << "Value of a is : " <<a<< endl;
}
template<class X,class Y> void fun(X b ,Y c)
{
    cout << "Value of b is : " <<b<< endl;
   cout << "Value of c is : " <<c<< endl;
}
```

```
int main()
{
  fun(10);
  fun(20,30.5);
}
```

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.

Let's see a simple example:

```
#include <iostream>
using namespace std;
template<class T>
class A
{
   public:
   T num1 = 5;
   T num2 = 6;
   void add()
   {
      cout << "Addition of num1 and num2 : " << num1+num2<<endl;
   }


};
```

```
int main()
{
   A<int> d;
   d.add();
}
```

Let's see a simple example when class template contains two generic data types.

```
#include <iostream>
    using namespace std;
    template<class T1, class T2>
   class A
   {
       T1 a;
       T2 b;
       public:
      A(T1 x,T2 y)
     {
        a = x;
        b = y;
      }
        void display()
       {
             cout << "Values of a and b are : " << a<<" ,"<<b<<endl;
        }
    };
```

```
 int main()
{
     A<int,float> d(5,6.5);
     d.display();
}
```