

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 write a generic function that can be used for different data types.

A single function template can work with different data types at once but, a single normal function can only work with one set of data types.

Normally, if you need to perform identical operations on two or more types of data, you use function overloading to create two functions with the required function declaration.

However, a better approach would be to use function templates because you can perform the same task writing less and maintainable code.

Examples of function templates are `sort()`, `max()`, `min()`, `printArray()`.

Syntax of Function Template

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

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.

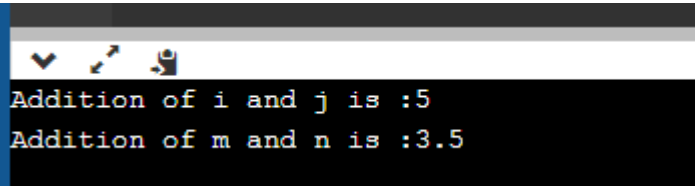
Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

main.cpp

Ctrl+S

```
1  #include <iostream>
2  using namespace std;
3  template<class T> T add(T &a,T &b)
4  {
5      T result = a+b;
6      return result;
7  }
8  }
9  int main()
10 {
11     int i =2;
12     int j =3;
13     float m = 2.3;
14     float n = 1.2;
15     cout<<"Addition of i and j is :"<<add(i,j);
16     cout<<"\n";
17     cout<<"Addition of m and n is :"<<add(m,n);
18     return 0;
19 }
```



```
✓ ↗ 🐞
Addition of i and j is :5
Addition of m and n is :3.5
```

Function Templates with Multiple Parameters

We can use more than one generic type in the template function by using the comma to separate the list.

```
template<class T1, class T2,.....>
```

```
return_type function_name (arguments of type T1, T2....)
```

```
{
```

```
    // body of function.
```

```
}
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 template<class X,class Y> void fun(X a,Y b)
4 {
5     std::cout << "Value of a is : " <<a<< std::endl;
6     std::cout << "Value of b is : " <<b<< std::endl;
7 }
8
9 int main()
10 {
11     fun(15,12.3);
12
13     return 0;
14 }
```

```
Value of a is : 15
Value of b is : 12.3
```

Overloading a Function Template

We can overload the generic function means that the overloaded template functions can differ in the parameter list.

main.cpp

```
1 #include <iostream>
2 using namespace std;
3 template<class X> void fun(X a)
4 {
5     std::cout << "Value of a is : " <<a<< std::endl;
6 }
7 template<class X,class Y> void fun(X b ,Y c)
8 {
9     std::cout << "Value of b is : " <<b<< std::endl;
10    std::cout << "Value of c is : " <<c<< std::endl;
11 }
12 int main()
13 {
14     fun(10);
15     fun(20,30.5);
16     return 0;
17 }
```

input

```
Value of b is : 20
Value of c is : 30.5
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

Restrictions of Generic Functions

Generic functions perform the same operation for all the versions of a function except the data type differs. Let's see a simple example of an overloaded function which cannot be replaced by the generic function as both the functions have different functionalities.

```
void fun(double a)
{
    cout<<"value of a is : "<<a<<"\n";
}
void fun(int b)
{
    if(b%2==0)
    {
        cout<<"Number is even";
    }
    else
    {
        cout<<"Number is odd";
    }
}
```

In the above example, we overload the ordinary functions. We cannot overload the generic functions as both the functions have different functionalities. First one is displaying the value and the second one determines whether the number is even or not.

Class Templates Like function templates, class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

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.

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

type: It is the type of the data that the class is operating on.

ob: It is the name of the object.

```
main.cpp
1  #include <iostream>
2  using namespace std;
3  template<class T>
4  class A
5  {
6      public:
7          T num1 = 5;
8          T num2 = 6;
9          void add()
10         {
11             std::cout << "Addition of num1 and num2 : " << num1+num2<<std::endl;
12         }
13     };
14 };
15
16 int main()
17 {
18     A<int> d;
19     d.add();
20     return 0;
21 }
```

CLASS TEMPLATE WITH MULTIPLE PARAMETERS

We can use more than one generic data type in a class template, and each generic data type is separated by the comma.

Syntax

template<class T1, class T2,>

class class_name

```
{
    // Body of the class.
}
```

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

```

main.cpp
1  #include <iostream>
2      using namespace std;
3      template<class T1, class T2>
4      class A
5      {
6          T1 a;
7          T2 b;
8      public:
9          A(T1 x,T2 y)
10         {
11             a = x;
12             b = y;
13         }
14         void display()
15         {
16             std::cout << "Values of a and b are : " << a<< " ,"<<b<<std::endl;
17         }
18     };
19
20     int main()
21     {
22         A<int,float> d(5,6.5);
23         d.display();
24         return 0;
25     }

```

input

Values of a and b are : 5 ,6.5

Nontype Template Arguments

The template can contain multiple arguments, and we can also use the non-type arguments. In addition to the type T argument, we can also use other types of arguments such as strings, function names, constant expression and built-in types. **Let's see the following example:**

```

template<class T, int size>
class array
{
    T arr[size];        // automatic array initialization.
};

```

In the above case, the nontype template argument is size and therefore, template supplies the size of the array as an argument.

Arguments are specified when the objects of a class are created:

1. array<int, 15> t1; // array of 15 integers.
2. array<float, 10> t2; // array of 10 floats.
3. array<char, 4> t3; // array of 4 chars.

Points to Remember

Join Our Telegram Group to Get Notifications, Study Materials, Practice test & quiz:

<https://t.me/ccatpreparations> Visit: <https://ccatpreparation.com>

- C++ supports a powerful feature known as a template to implement the concept of generic programming.
- A template allows us to create a family of classes or family of functions to handle different data types.
- Template classes and functions eliminate the code duplication of different data types and thus makes the development easier and faster.
- Multiple parameters can be used in both class and function template.
- Template functions can also be overloaded.
- We can also use nontype arguments such as built-in or derived data types as template arguments.