

Templates

Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates.

Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

The concept of templates can be used in two different ways:

- Function Templates
- Class Templates

Function Templates

A function template works in a similar to a normal [function](#), with one key difference.

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.

How to declare a function template?

A function template starts with the keyword **template** followed by template parameter/s inside `< >` which is followed by function declaration.

```
template <class T>  
T someFunction(T arg)  
{  
    ... ..  
}
```

In the above code, `T` is a template argument that accepts different data types (int, float), and **class** is a keyword.

You can also use keyword `typename` instead of class in the above example.

When, an argument of a data type is passed to `someFunction()`, compiler generates a new version of `someFunction()` for the given data type.

```
// template function
template <class T>
T Large(T n1, T n2)
{
    return (n1 > n2) ? n1 : n2;
}

int main()
{
    int i1, i2;
    float f1, f2;
    char c1, c2;

    cout << "Enter two integers";
    cin >> i1 >> i2;
    cout << Large(i1, i2) << " is larger." << endl;

    cout << "Enter two floating-point numbers:";
    cin >> f1 >> f2;
    cout << Large(f1, f2) << " is larger." << endl;

    cout << "Enter two characters:";
    cin >> c1 >> c2;
    cout << Large(c1, c2) << " has larger ASCII value.";

    return 0;
}
```


Class Templates

Like function templates, you can also create class templates for generic class operations.

Sometimes, you need a class implementation that is same for all classes, only the data types used are different.

Normally, you would need to create a different class for each data type OR create different member variables and functions within a single class.

This will unnecessarily bloat your code base and will be hard to maintain, as a change in one class/function should be performed on all classes/functions.

However, class templates make it easy to reuse the same code for all data types.

How to declare a class template?

```
template <class T>
class className
{
    ... ..
public:
    T var;
    T someOperation(T arg);
    ... ..
};
```

In the above declaration, `T` is the template argument which is a placeholder for the data type used.

Inside the class body, a member variable `var` and a member function `someOperation()` are both of type `T`.

How to create a class template object?

To create a class template object, you need to define the data type inside a `< >` when creation.

```
className<dataType> classObject;
```

For example:

```
className<int> classObject;  
className<float> classObject;  
className<string> classObject;
```