C++ Class Templates

Templates are powerful features of C++ that allows us to write generic programs. There are two ways we can implement templates:

- Function Templates
- Class Templates

Similar to function templates, we can use class templates to create a single class to work with different data types.

Class templates come in handy as they can make our code shorter and more manageable.

Class Template Declaration

A class template starts with the keyword `template` followed by template parameter(s) inside `<>` which is followed by the class declaration.

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

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

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

Creating a Class Template Object

Once we've declared and defined a class template, we can create its objects in other classes or functions (such as the `main()` function) with the following syntax:

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

For example,

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

```
/*C++ program to demonstrate the use of class templates                // Example 1          */
#include <iostream>
using namespace std;

template <class T>                          // Class template
class Number {
  private:
   // Variable of type T
   T num;

  public:
   Number(T n) : num(n) {}                  // constructor

   T getNum() {
      return num;     }
};
int main() {
   Number<int> numberInt(7);                          // create object with int type
   Number<double> numberDouble(7.7);                  // create object with double type
   cout << "int Number = " << numberInt.getNum() << endl;
   cout << "double Number = " << numberDouble.getNum() << endl;

   return 0;
}
```

Defining a Class Member Outside the Class Template

Suppose we need to define a function outside of the class template. We can do this with the following code:

```
template <class T>
class ClassName {
    ... .. ...
    // Function prototype
    returnType functionName();
};

// Function definition
template <class T>
returnType ClassName<T>::functionName() {
    // code
}
```

Notice that the code `template <class T>` is repeated while defining the function outside of the class. This is necessary and is part of the syntax.

If we look at the code in **Example 1**, we have a function `getNum()` that is defined inside the class template `Number`.

We can define `getNum()` outside of `Number` with the following code:

```
template <class T>
class Number {
    ... .. ...
    // Function prototype
    T getnum();
};

// Function definition
template <class T>
T Number<T>::getNum() {
    return num;
}
```

This program uses a class template to perform addition, subtraction, multiplication and division of two variables `num1` and `num2`.

The variables can be of any type, though we have only used `int` and `float` types in this example.

```
#include <iostream>
using namespace std;

template <class T>
class Calculator {
   private:
    T num1, num2;

   public:
    Calculator(T n1, T n2) {
        num1 = n1;
        num2 = n2;
    }

    void displayResult() {
        cout << "Numbers: " << num1 << " and " << num2 << "." << endl;
        cout << num1 << " + " << num2 << " = " << add() << endl;
        cout << num1 << " - " << num2 << " = " << subtract() << endl;
        cout << num1 << " * " << num2 << " = " << multiply() << endl;
        cout << num1 << " / " << num2 << " = " << divide() << endl;
    }

    T add() { return num1 + num2; }
    T subtract() { return num1 - num2; }
    T multiply() { return num1 * num2; }
    T divide() { return num1 / num2; }
};
```

```
int main() {
    Calculator<int> intCalc(2, 1);
    Calculator<float> floatCalc(2.4, 1.2);

    cout << "Int results:" << endl;
    intCalc.displayResult();

    cout << endl
         << "Float results:" << endl;
    floatCalc.displayResult();

    return 0;
}
```

C++ Class Templates With Multiple Parameters

In C++, we can use multiple template parameters and even use default arguments for those parameters. For example,

```
template <class T, class U, class V = int>
class ClassName {
  private:
    T member1;
    U member2;
    V member3;
    ... .. ...
  public:
    ... .. ...
};
```

```cpp
#include <iostream>
using namespace std;

// Class template with multiple and default parameters
template <class T, class U, class V = char>
class ClassTemplate {
   private:
    T var1;
    U var2;
    V var3;

   public:
    ClassTemplate(T v1, U v2, V v3) : var1(v1), var2(v2), var3(v3)

    void printVar() {
        cout << "var1 = " << var1 << endl;
        cout << "var2 = " << var2 << endl;
        cout << "var3 = " << var3 << endl;
    }
};
```

```cpp
int main() {
    // create object with int, double and char types
    ClassTemplate<int, double> obj1(7, 7.7, 'c');
    cout << "obj1 values: " << endl;
    obj1.printVar();

    // create object with int, double and bool types
    ClassTemplate<double, char, bool> obj2(8.8, 'a', false);
    cout << "\nobj2 values: " << endl;
    obj2.printVar();

    return 0;
}
```

Notice the code `class V = char`. This means that `V` is a default parameter whose default type is `char`.

Inside `ClassTemplate`, we declare 3 variables `var1`, `var2`, and `var3`, each corresponding to one of the template parameters.

In this program, we have created a class template, named `ClassTemplate`, with three parameters, with one of them being a default parameter.

```
template <class T, class U, class V = char>
class ClassTemplate {
  // code
};
```

Notice the code `class V = char`. This means that `V` is a default parameter whose default type is `char`.

Inside `ClassTemplate`, we declare 3 variables `var1`, `var2`, and `var3`, each corresponding to one of the template parameters.

```
class ClassTemplate {
   private:
    T var1;
    U var2;
    V var3;
    ... .. ...
    ... .. ...
};
```

In `main()`, we create two objects of `ClassTemplate` with the code

```
// create object with int, double and char types
ClassTemplate<int, double> obj1(7, 7.7, 'c');

// create object with double, char and bool types
ClassTemplate<double, char, bool> obj2(8, 8.8, false);
```

Here,

Object	T	U	V
`obj1`	`int`	`double`	`char`
`obj2`	`double`	`char`	`bool`

For `obj1`, `T = int`, `U = double` and `V = char`.

For `obj2`, `T = double`, `U = char` and `V = bool`.