Objective

- To understand the concept of templates in C++.
- To implement function templates for generic operations.
- To demonstrate overloading of function templates.

Theory:

Function Template

Theory:

A function template is used to write a single function definition that works with different data types. The compiler generates the appropriate function based on the type of arguments passed.

Syntax:

```
template <typename T>
T functionName(T arg1, T arg2) {
   // Function body
}
```

Overloading Function Template

Theory:

Function templates can be overloaded like regular functions. You can define multiple versions of a function template or combine a template with a non-template function to customize behavior for specific data types.

Syntax:

```
template <typename T>
void functionName(T a) {
   // Generic version
}

void functionName(int a) {
   // Specialized version for int
}
```

Class Template

A class template defines a class with generic data types. When an object is created from the class, the actual type is specified. This enables the same class to handle various data types without code duplication.

Syntax:

```
template <class T>
class ClassName {
   T data;
public:
   ClassName(T val) { data = val; }
   void display();
};
```

Derived Class Template

Theory:

A class template can serve as a base class, and the derived class can also be a template or a specific type. This is useful for extending generic behavior in inheritance hierarchies.

Syntax:

```
template <typename T>
class Base {
protected:
   T data;
public:
   Base(T d) : data(d) {}
};

template <typename T>
class Derived : public Base<T> {
   T extra;
public:
   Derived(T d1, T d2) : Base<T>(d1), extra(d2) {}
};
```