Lab 12 – Template in c++

Lab Outcomes:

- **Understanding Function Templates and Type Deduction**: Gain an understanding of function templates in C++, and how they allow for writing generic functions that can operate on different data types. Learn how the C++ compiler automatically deduces the data type of template parameters based on the arguments passed during function calls.
- **Implementing and Using Function Templates**: Develop the ability to define and implement function templates, such as those for performing arithmetic operations or comparing values, while allowing the function to work with multiple data types. Learn how to call a function template and rely on the compiler's type deduction feature for type safety and flexibility.
- **Exploring Class Templates and Object Creation**: Learn how to define class templates in C++, allowing the creation of classes that work with different data types. Understand how to instantiate objects of a class template by specifying the data type explicitly (e.g., `Box<int>`, `Box<double>`) and explore the usage of member functions inside these class templates.
- **Applying Class Templates for Data Storage and Operations**: Develop skills in using class templates to store and manipulate data of various types. For example, create generic container classes (like `Box`, `Stack`, or `List`) that can store items of different types and implement common operations (e.g., adding, removing, or retrieving data) in a flexible and reusable manner.

Templates allow you to write a generic program. It is just like a class. As we discussed earlier that class is a template or a blueprint that create an object.

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.

TYPES:

1) Function Templates
2) Class Templates

FUNCTION TEMPLATES:

Both function overloading and templates are examples of polymorphism features of OOP. Function overloading is used when multiple functions do quite similar (not identical) operations, templates are used when multiple functions do identical operations.

Function templates are templates that define a family of functions. They allow you to write a function that can work with any data type.

Key Points about Function Templates:

Definition and Syntax:

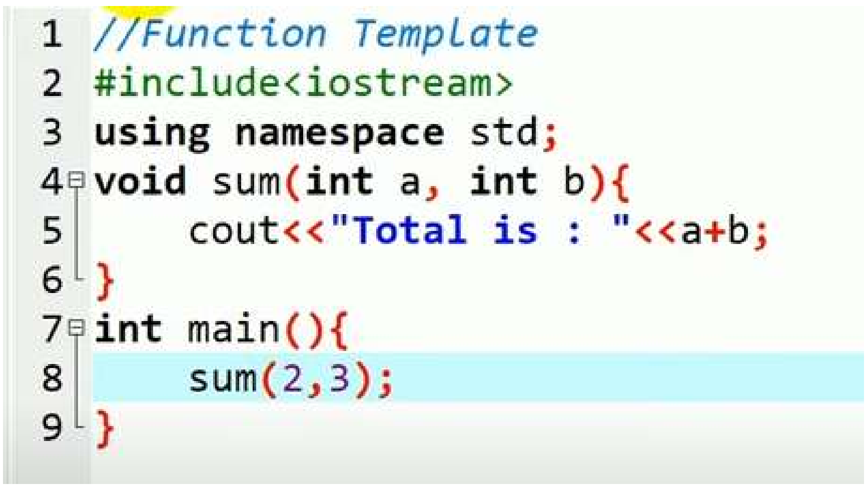
SYNTAX:

```
template <class T>

T add(T a, T b) {

   return a + b;

}
```

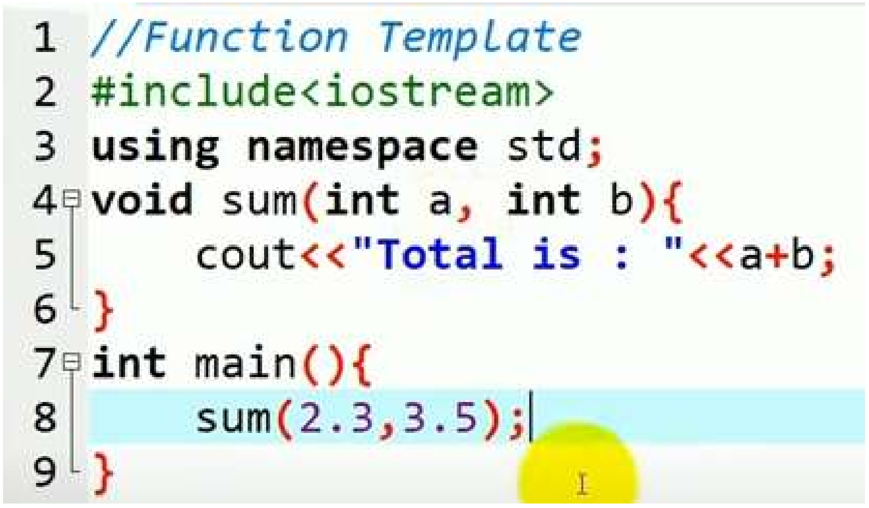
```
template <class T,class T1>

T add(T a, T1 b) {

   return a + b;

}
```

Defined using the template keyword followed by a template parameter list.

The template parameter list is enclosed in angle brackets (< >) and specifies the types (or values) the template can operate on.

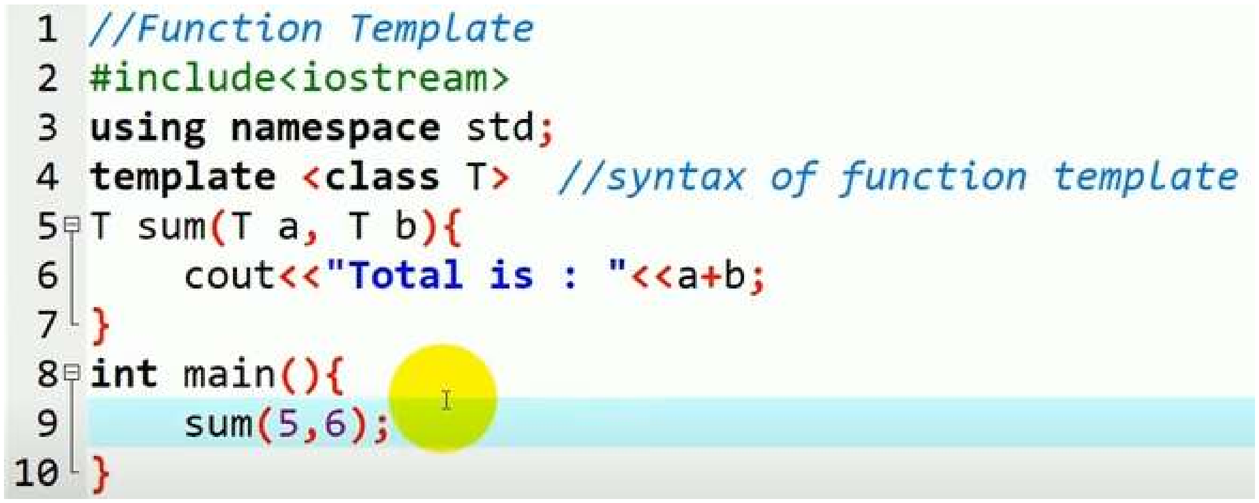


```
//Function Template
#include<iostream>
using namespace std;
void sum(int a, int b){
     cout<<"Total is : "<<a+b;
}
int main(){
     sum(2,3);
}
```



```
//Function Template
#include<iostream>
using namespace std;
void sum(int a, int b){
     cout<<"Total is : "<<a+b;
}
int main(){
     sum(2.3,3.5);
}
```

Now with the help of Templates which makes it generic, no need to explicitly mention the data type here.



```
//Function Template
#include<iostream>
using namespace std;
template <class T>  //syntax of function template
T sum(T a, T b){
    cout<<"Total is : "<<a+b;
}
int main(){
    sum(5,6);
}
```

Usage:

When you call a function template, the compiler automatically deduces the type from the arguments you pass. For example:

int result = add(3, 5);   // T is deduced to be int

double result = add(3.5, 4.2);   // T is deduced to be double

Type Deduction:

The compiler deduces the type of the template parameters based on the function arguments.

CLASS TEMPLATES:

Class templates are templates that define a family of classes. They allow you to define a class that can work with any data type.

Key Points about Class Templates:

Definition and Syntax:

```
template <class T>

class className {

 private:

   T var;

   ... .. ...                              // Class body
```

```
  public:
    T functionName(T arg);
    ... .. ...
};
```

Defined using the template keyword followed by a template parameter list, similar to function templates.

The template parameter list is enclosed in angle brackets (< >) and specifies the types (or values) the template can operate on.

Usage:

When you create an object of a class template, you need to specify the type explicitly. For example:

Box<int> intBox(123);   // T is specified as int

Box<double> doubleBox(456.78); // T is specified as double

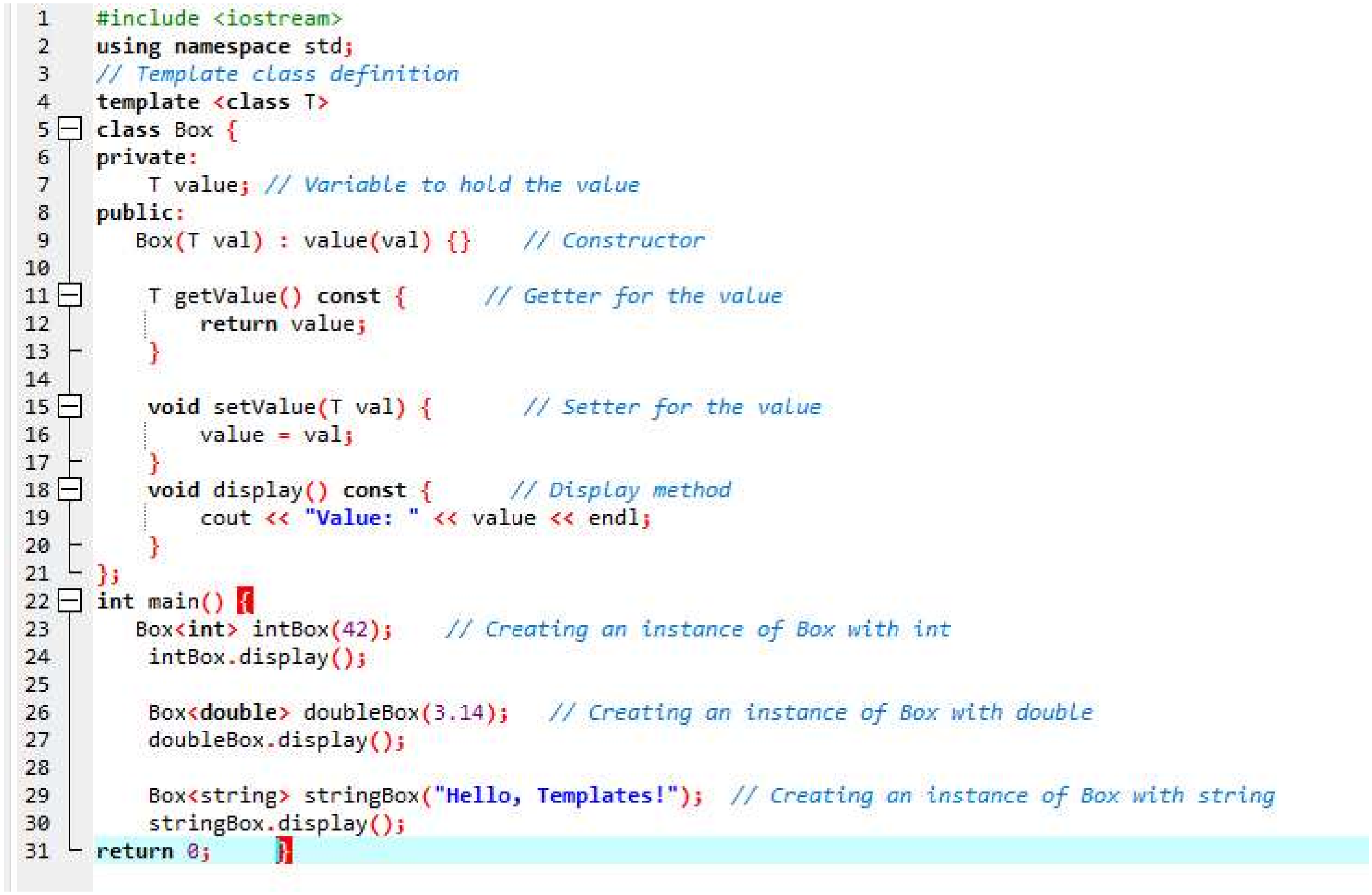
Explicit Type Specification:

You must explicitly specify the type when you create an instance of a class template. Unlike function templates, there is no automatic type deduction.

Use Case:

- Use function templates when you need functions that operate on generic types.
- Use class templates when you need classes that can store or operate on generic types.

Example



```cpp
#include <iostream>
using namespace std;
// Template class definition
template <class T>
class Box {
private:
    T value; // Variable to hold the value
public:
   Box(T val) : value(val) {}    // Constructor

    T getValue() const {      // Getter for the value
        return value;
    }

    void setValue(T val) {       // Setter for the value
        value = val;
    }
    void display() const {      // Display method
        cout << "Value: " << value << endl;
    }
};
int main() {
   Box<int> intBox(42);    // Creating an instance of Box with int
    intBox.display();

    Box<double> doubleBox(3.14);   // Creating an instance of Box with double
    doubleBox.display();

    Box<string> stringBox("Hello, Templates!");  // Creating an instance of Box with string
    stringBox.display();
return 0;     }
```

TASKS:

Question 1: Function Template

Write a function template findMax that takes two arguments of any type and returns the larger of the two. Demonstrate its usage in a main function by comparing two integers, two doubles, and two strings.

Requirements:

The function template findMax should be able to compare and return the maximum of two values of any type.

In the main function, call findMax with pairs of integers, doubles, and strings, and print the results.

Example Output:

```
Max of 3 and 7 is: 7

Max of 4.5 and 2.3 is: 4.5

Max of "apple" and "banana" is: banana
```

Question 2: Class Template

Create a class template Circle that represents a circle with a radius of any numeric type (e.g., int, float, double). The class should have a constructor to initialize the radius, a method to calculate the area of the circle, and a method to calculate the circumference of the circle. Demonstrate the usage of the class with an integer radius and a double radius in a main function.

Requirements:

The class template Circle should have a member variable radius of type T.

Include methods getArea and getCircumference that return the area and circumference of the circle, respectively.

Demonstrate the usage of the class template with both int and double types in the main function.

Example Output:

```
Circle with int radius 5:

  Area: 78.5398

  Circumference: 31.4159


Circle with double radius 5.5:
```

Area: 95.0332

Circumference: 34.5575

Question 3: Simple Calculator Using Class Templates that add, subtract, multiply and divide of integer and floating numbers.

Lab #12 Marks distribution

	ER1	**ER6**	**ER8**
Task	3points	3 points	4 points

Lab # 12 Rubric Evaluation Guideline:

#	**Qualities & Criteria**	**0 < Poor <=0.5**	**0.5 < Satisfactory <= 1.5**	**1.5 < Excellent <=2**
ER1	**Task Completion**	Minimal or no program functionality was achieved.	Some tasks were completed, but the program has errors or incomplete functionalities.	All tasks were completed, and the program runs without errors.
#	**Qualities & Criteria**	**0 < Poor <=0.5**	**0.5 < Satisfactory <= 1.5**	**1.5 < Excellent <=2**
ER6	Program Output	Output is inaccurate or poorly presented.	Output is mostly accurate but may lack labels, captions, or formatting.	Output is clear, accurate, and well presented with labels, captions, and proper formatting.
#	**Qualities & Criteria**	**0 < Poor <= 1.5**	**1.5< Satisfactory <= 3**	**3< Excellent <= 4**
ER8	Question & Answer	Answers some questions but not	Answers most questions confidently	Answers all questions confidently and

		confidently or based on lab task knowledge.	and based on lab task knowledge.	demonstrates a deep understanding of the given lab task.