



## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition  
Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

# Module 38: Programming in C++

## Template (Function Template): Part 1

Instructors: Abir Das and Sourangshu Bhattacharya

Department of Computer Science and Engineering  
Indian Institute of Technology, Kharagpur

`{abir, sourangshu}@cse.iitkgp.ac.in`

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**



# Module Recap

## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

### Objectives & Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

`typename`

Module Summary

- Discussed exception (error) handling in C++
- Illustrated `try-throw-catch` feature in C++ for handling errors
- Demonstrated with examples



# Module Objectives

## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

### Objectives & Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

`typename`

Module Summary

- Understand Templates in C++
- Understand Function Templates



# Module Outline

## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

### Objectives & Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

## 1 What is a Template?

## 2 Function Template

- Definition
- Instantiation
- Template Argument Deduction
- Example

## 3 typename

## 4 Module Summary



# What is a Template?

## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

`typename`

Module Summary

## What is a Template?



# What is a Template?

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition  
Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

- Templates are specifications of a *collection of functions or classes which are parameterized by types*
- Examples:
  - Function search, min etc.
    - ▷ The basic algorithms in these functions are the same independent of types
    - ▷ Yet, we need to write different versions of these functions for strong type checking in C++
  - Classes list, queue etc.
    - ▷ The data members and the methods are almost the same for list of numbers, list of objects
    - ▷ Yet, we need to define different classes



# Function Template

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

`typename`

Module Summary

## Function Template



# Function Template: Code reuse in Algorithms

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

- We need to compute the maximum of two values that can be of:
  - `int`
  - `double`
  - `char *` (C-String)
  - `Complex` (user-defined class for complex numbers)
  - ...
- We can do this with overloaded `Max` functions:

```
int Max(int x, int y);  
double Max(double x, double y);  
char *Max(char *x, char *y);  
Complex Max(Complex x, Complex y);
```

With every new type, we need to add an overloaded function in the library!

- **Issues in `Max` function**
  - *Same algorithm* (compare two values using the appropriate operator of the type and return the larger value)
  - *Different code versions* of these functions for strong type checking in C++





# Max as Overload

## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

```
#include <iostream>
#include <cstring>
#include <cmath>
using namespace std;
// Overloads of Max
int Max(int x, int y) { return x > y ? x : y; }
double Max(double x, double y) { return x > y ? x : y; }
char *Max(char *x, char *y) { return strcmp(x, y) > 0 ? x : y; }

int main() { int a = 3, b = 5, iMax; double c = 2.1, d = 3.7, dMax;
    cout << "Max(" << a << ", " << b << ") = " << Max(a, b) << endl;
    cout << "Max(" << c << ", " << d << ") = " << Max(c, d) << endl;

    char *s1 = new char[6], *s2 = new char[6];
    strcpy(s1, "black"); strcpy(s2, "white");
    cout << "Max(" << s1 << ", " << s2 << ") = " << Max(s1, s2) << endl;
    strcpy(s1, "white"); strcpy(s2, "black");
    cout << "Max(" << s1 << ", " << s2 << ") = " << Max(s1, s2) << endl;
}
```

- Overloaded solutions work
- In some cases (C-string), similar algorithms have exceptions
- With every new type, a new overloaded [Max](#) is needed
- Can we make [Max](#) generic and make a library to work with future types?
- [How about macros?](#)



# Max as a Macro

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

```
// Max as a macro
```

```
#define Max(x, y) (((x) > (y)) ? x : y)
```

```
int main() {
    int a = 3, b = 5;
    double c = 2.1, d = 3.7;
```

```
    cout << "Max(" << a << ", " << b << ") = " << Max(a, b) << endl; // Output: Max(3, 5) = 5
```

```
    cout << "Max(" << c << ", " << d << ") = " << Max(c, d) << endl; // Output: Max(2.1, 3.7) = 3.7
```

```
    return 0;
```

```
}
```

- **Max**, being a macro, is type oblivious – can be used for **int** as well as **double**, etc.
- Note the parentheses around parameters to protect precedence
- Note the parentheses around the whole expression to protect precedence
- Looks like a function – but does not behave as such



# Max as a Macro: Pitfalls

## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

#define Max(x, y) (((x) > (y)) ? x : y)

int main() { int a = 3, b = 5; double c = 2.1, d = 3.7;
    // Side Effects
    cout << "Max(" << a << ", " << b << ") = ";    // Output: Max(3, 5) = 6
    cout << Max(a++, b++) << endl;
    cout << "a = " << a << ", b = " << b << endl; // Output: a = 4, b = 7

    // C-String Comparison
    char *s1 = new char[6], *s2 = new char[6];
    strcpy(s1, "black"); strcpy(s2, "white");
    cout << "Max(" << s1 << ", " << s2 << ") = " << Max(s1, s2) << endl; // Max(black, white) = white

    strcpy(s1, "white"); strcpy(s2, "black");
    cout << "Max(" << s1 << ", " << s2 << ") = " << Max(s1, s2) << endl; // Max(white, black) = black
}
```

- In "Side Effects" – the result is wrong, the larger values gets incremented twice
- In "C-String Comparison" – swapping parameters changes the result – actually compares pointers



# Function Template

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

- A **function template**
  - describes how a function should be built
  - supplies the definition of the function using some arbitrary types, (as place holders)
    - ▷ a **parameterized** definition
  - can be considered the definition for a **set of overloaded versions** of a function
  - is identified by the **keyword template**
    - ▷ followed by comma-separated list of **parameter** identifiers (each preceded by **keyword class** or **keyword typename**)
    - ▷ enclosed between **<** and **>** delimiters
    - ▷ followed by the signature the function
  - Note that every template parameter is a **built-in type** or **class** – type parameters



# Max as a Function Template

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

template<class T>
T Max(T x, T y) {
    return x > y ? x : y;
}

int main() {
    int a = 3, b = 5, iMax;
    double c = 2.1, d = 3.7, dMax;

    iMax = Max<int>(a, b);
    cout << "Max(" << a << ", " << b << ") = " << iMax << endl; // Output: Max(3, 5) = 5

    dMax = Max<double>(c, d);
    cout << "Max(" << c << ", " << d << ") = " << dMax << endl; // Output: Max(2.1, 3.7) = 3.7
}
```

- **Max**, now, knows the type
- Template type parameter **T** explicitly specified in instantiation of **Max<int>**, **Max<double>**



# Max as a Function Template: Pitfall "Side Effects" – Solved

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

template<class T>
T Max(T x, T y) {
    return x > y ? x : y;
}

int main() {
    int a = 3, b = 5, iMax;

    // Side Effects
    cout << "Max(" << a << ", " << b << ") = ";
    iMax = Max<int>(a++, b++);
    cout << iMax << endl; // Output: Max(3, 5) = 5

    cout << "a = " << a << ", b = " << b << endl; // Output: a = 4, b = 6
}
```

- **Max** is now a proper function call – no side effect



# Max as a Function Template: Pitfall "C-String Comparison" – Solved

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

template<class T> T Max(T x, T y) { return x > y ? x : y; }

template<> // Template specialization for 'char*' type
char *Max<char *>(char *x, char *y) { return strcmp(x, y) > 0 ? x : y; }

int main() { char *s1 = new char[6], *s2 = new char[6];
    strcpy(s1, "black"); strcpy(s2, "white");
    cout << "Max(" << s1 << ", " << s2 << ") = " << Max<char*>(s1, s2) << endl;
    // Output: Max(black, white) = white

    strcpy(s1, "white"); strcpy(s2, "black");
    cout << "Max(" << s1 << ", " << s2 << ") = " << Max<char*>(s1, s2) << endl;
    // Output: Max(black, white) = white
}
```

- Generic template code does not work for C-Strings as it compares pointers, not the strings pointed by them
- We provide a specialization to compare pointers using comparison of strings
- Need to specify type explicitly is bothersome



# Max as a Function Template: Implicit Instantiation

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

template<class T> T Max(T x, T y) { return x > y ? x : y; }

int main() { int a = 3, b = 5, iMax; double c = 2.1, d = 3.7, dMax;
    iMax = Max(a, b); // Type 'int' inferred from 'a' and 'b' parameters types
    cout << "Max(" << a << ", " << b << ") = " << iMax << endl;
        // Output: Max(3, 5) = 5

    dMax = Max(c, d); // Type 'double' inferred from 'c' and 'd' parameters types
    cout << "Max(" << c << ", " << d << ") = " << dMax << endl;
        // Output: Max(2.1, 3.7) = 3.7
}
```

- Often template type parameter T may be inferred from the type of parameters in the instance
- If the compiler cannot infer or infers wrongly, we use explicit instantiation





# Template Argument Deduction: Implicit Instantiation

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition  
Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

- Each item in the template parameter list is a template argument
- When a template function is invoked, the values of the template arguments are determined by seeing the types of the function arguments

```
template<class T> T Max(T x, T y);  
template<> char *Max<char *>(char *x, char *y);  
template <class T, int size> T Max(T x[size]);
```

```
int a, b; Max(a, b);           // Binds to Max<int>(int, int);  
double c, d; Max(c, d);       // Binds to Max<double>(double, double);  
char *s1, *s2; Max(s1, s2);   // Binds to Max<char*>(char*, char*);
```

```
int pval[9]; Max(pval);       // Error!
```

- Three kinds of conversions are allowed
  - L-value transformation (for example, Array-to-pointer conversion)
  - Qualification conversion
  - Conversion to a base class instantiation from a class template
- If the same template parameter are found for more than one function argument, template argument deduction from each function argument must be the same



# Max as a Function Template: With User-Defined Class

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

```

#include <iostream>
#include <cmath>
#include <cstring>
using namespace std;

class Complex { double re_; double im_; public:
    Complex(double re = 0.0, double im = 0.0) : re_(re), im_(im) { };
    double norm() const { return sqrt(re_*re_+im_*im_); }
    friend bool operator>(const Complex& c1, const Complex& c2) { return c1.norm() > c2.norm(); }
    friend ostream& operator<<(ostream& os, const Complex& c) {
        os << "(" << c.re_ << ", " << c.im_ << ")"; return os;
    }
};

template<class T> T Max(T x, T y) { return x > y ? x : y; }
template<> char *Max<char *>(char *x, char *y) { return strcmp(x, y) > 0 ? x : y; }

int main() { Complex c1(2.1, 3.2), c2(6.2, 7.2);
    cout << "Max(" << c1 << ", " << c2 << ") = " << Max(c1, c2) << endl;
        // Output: Max((2.1, 3.2), (6.2, 7.2)) = (6.2, 7.2)
    }

```

- When **Max** is instantiated with **class Complex**, we need comparison operator for **Complex**
- The code, therefore, will not compile without **bool operator>(const Complex&, const Complex&)**
- Traits of type variable **T** include **bool operator>(T, T)** which the instantiating type must fulfill



# Max as a Function Template: Overloads

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition  
Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

template<class T> T Max(T x, T y) { return x > y ? x : y; }

template<> char *Max<char *>(char *x, char *y) // Template specialization
{ return strcmp(x, y) > 0 ? x : y; }

template<class T, int size> T Max(T x[size]) { // Overloaded template function
    T t = x[0];
    for (int i = 0; i < size; ++i) { if (x[i] > t) t = x[i]; }

    return t;
}

int main() {
    int arr[] = { 2, 5, 6, 3, 7, 9, 4 };
    cout << "Max(arr) = " << Max<int, 7>(arr) << endl; // Output: Max(arr) = 9
}
```

- Template function can be overloaded
- A template parameter can be non-type (`int`) constant



# Swap as a Function Template

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

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

template<class T> void Swap(T& one, T& other) { T temp;
    temp = one; one = other; other = temp;
}

int main() { int i = 10, j = 20;
    cout << "i = " << i << ", j = " << j << endl;
    Swap(i, j);
    cout << "i = " << i << ", j = " << j << endl;

    string s1("abc"), s2("def");

    cout << "s1 = " << s1 << ", s2 = " << s2 << endl;
    Swap(s1, s2);
    cout << "s1 = " << s1 << ", s2 = " << s2 << endl;
}
```

- The traits of type variable **T** include  
default constructor (**T::T()**) and  
copy assignment operator (**T operator=(const T&)**)
- Our template function cannot be called **swap**, as **std** namespace has such a function



# typename

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

**typename**

Module Summary

# typename



# typename Keyword

Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

**typename**

Module Summary

- Consider:

```
template <class T> f (T x) {  
    T::name * p;  
}
```

- What does it mean?

- `T::name` is a *type* and `p` is a *pointer* to that type
- `T::name` and `p` are *variables* and this is a *multiplication*

- To resolve, we use **keyword typename**:

```
template <class T> f (T x) { T::name * p; } // Multiplication
```

```
template <class T> f (T x) { typename T::name * p; } // Type
```

- The keywords **class** and **typename** have almost the same meaning in a template parameter
- **typename** is also used to tell the compiler that an expression is a type expression



# Module Summary

## Module 38

Instructors: Abir  
Das and  
Sourangshu  
Bhattacharya

Objectives &  
Outlines

What is a  
Template?

Function  
Template

Definition

Instantiation

Template Argument  
Deduction

Example

typename

Module Summary

- Introduced the templates in C++
- Discussed function templates as generic algorithmic solution for code reuse
- Explained templates argument deduction for implicit instantiation
- Illustrated with examples