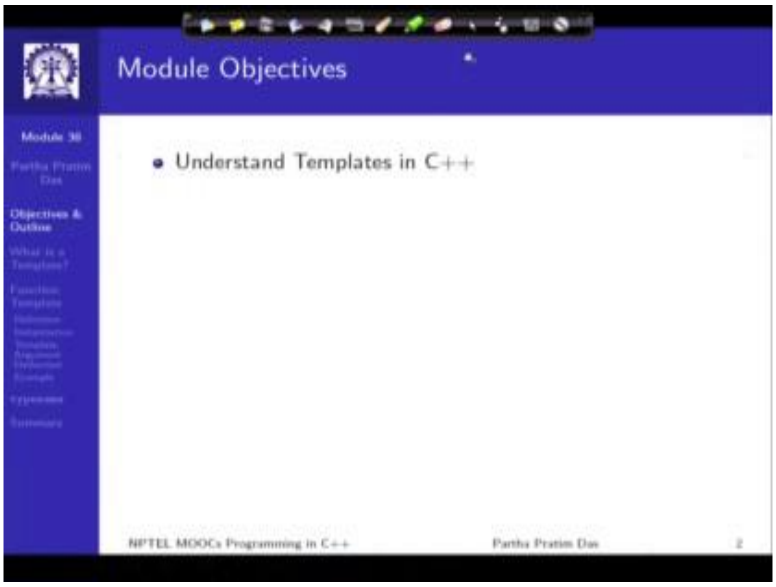
Programming in C++

Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 54
Template (Function Template): Part 1

Welcome to module 38 of programming in C++. We are almost at the end of the course and in this module and the next we introduce very different concepts in C++ programming, which is known as generic programming concepts. And this is or it is also referred to as meta programming concepts. This is done through a feature called templates. We will go through what a templates are and why they are required, but the core idea of template is to be able to write a code, which at the time of compilation can generates for the new code and that generated code then gets complied again. So, templates is not only a program, but it is a meta program which generates other programs

(Refer Slide Time: 01:26)



Module Objectives

Module 38

Partha Pratim Das

Objectives & Outline

What is a Template?

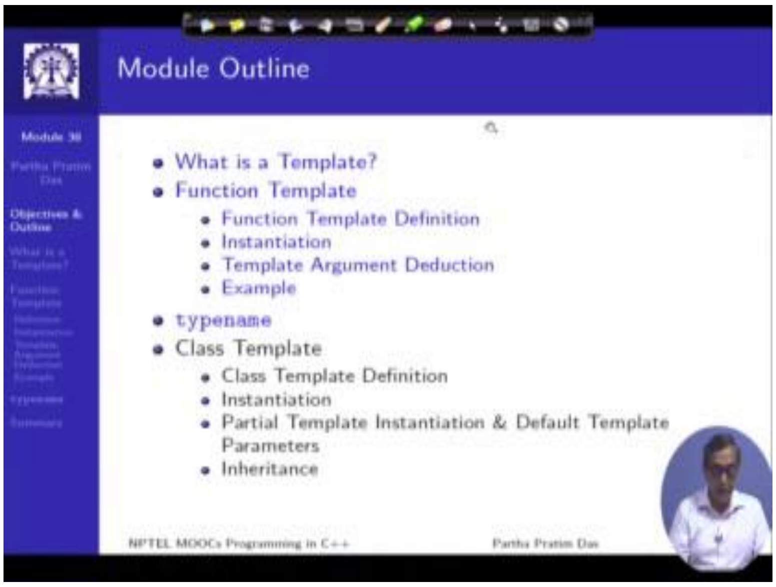
Function Template

- Understand Templates in C++

NPTEL MOOCs Programming in C++ Partha Pratim Das 2

So, we will cover this in two modules, naturally the objective will be to understand templates.

(Refer Slide Time: 01:33)



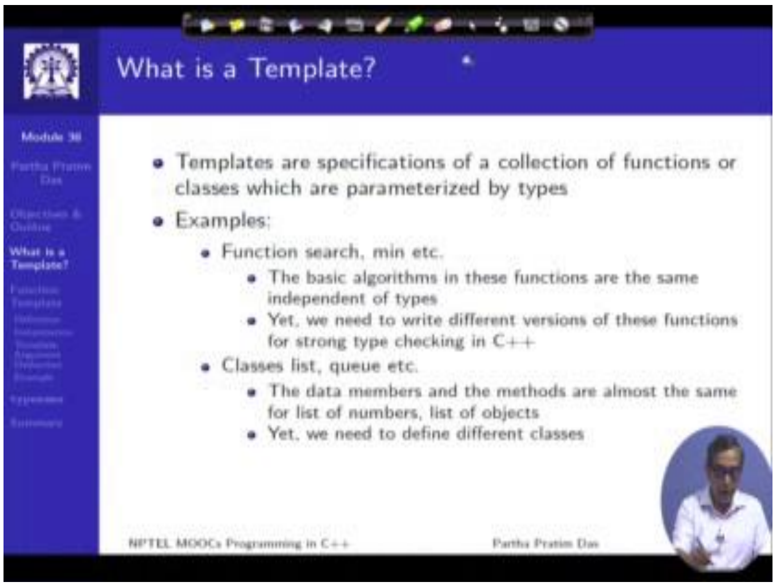
Module Outline

- What is a Template?
- Function Template
  - Function Template Definition
  - Instantiation
  - Template Argument Deduction
  - Example
- `typename`
- Class Template
  - Class Template Definition
  - Instantiation
  - Partial Template Instantiation & Default Template Parameters
  - Inheritance

NPTEL MOOCs Programming in C++ Partha Pratim Das

And this is what are outline would be the blue part is what we will do first which will significantly covered the function template. And in the next module, we will talk about the class template.

(Refer Slide Time: 01:48)



What is a Template?

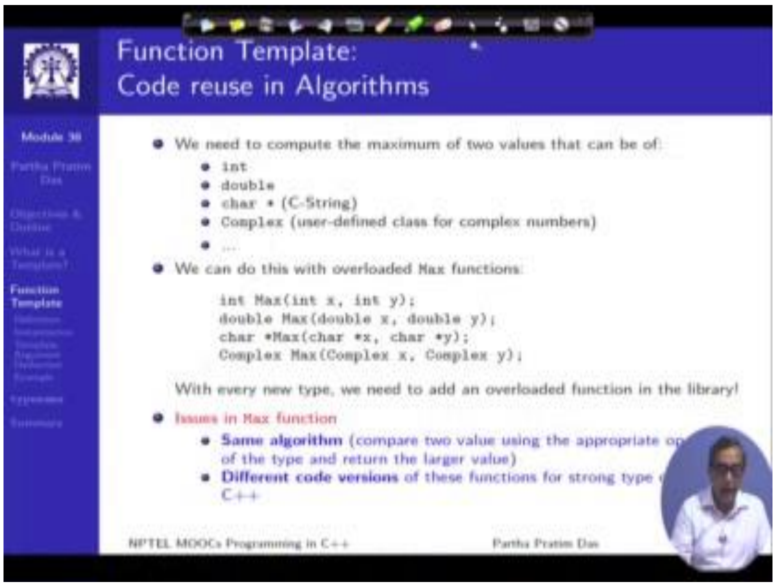
- Templates are specifications of a collection of functions or classes which are parameterized by types
- Examples:
  - Function search, min etc.
    - The basic algorithms in these functions are the same independent of types
    - Yet, we need to write different versions of these functions for strong 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

NPTEL MOOCs Programming in C++ Partha Pratim Das

Now, what is Template? Templates are specifications of a collection of a function or classes which are parameterized by types, this is the important part. They are parameterized by type which is a complete a new concept now coming in. In that so far we always knew that any code that we write in C++ must have a fixed type is a strongly type language whether it is int or it is double, or it is char star, or it is a user-defined type, it must be known. But here we are now seeing that the type can be also a parameter and the motivation comes from the fact that we want to increase the code reused in C++.

There are several algorithms which are generic in terms of algorithmic strategy like finding maximum or minimum of two numbers or searching a set of data items which can be compared, but depending on the specific type their codes become different. Similar things happen with a lot of class designs if we talk about stack all that we know is it is a LIFO structure, last infrastructure, but depending on the specific type of elements that we have to deal with the stacks using a character elements, a stack using an integer element, a stack using some user defined type elements will all be different. So, template is an attempt in a trying to combine all this into a single binding a code.

(Refer Slide Time: 03:25)



Function Template:
Code reuse in Algorithms

Module 38

Partha Pratim Das

- 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!

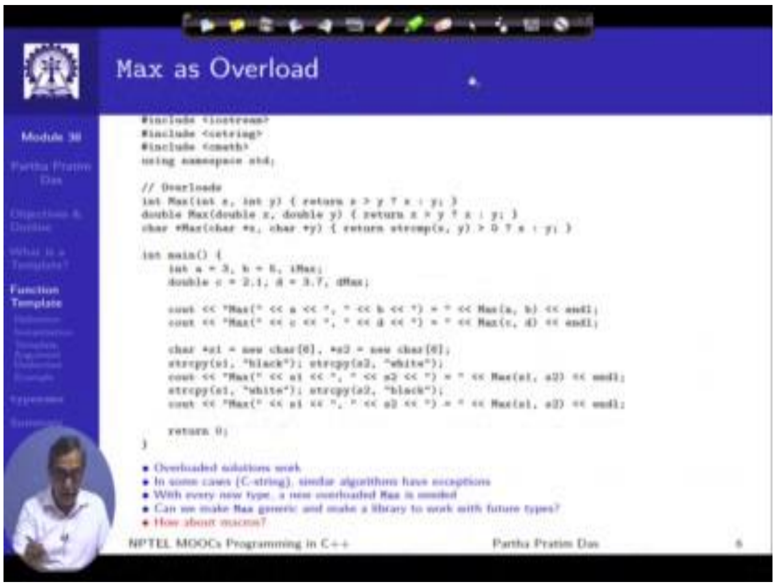
- Issues in Max function
  - **Same algorithm** (compare two value using the appropriate op
    of the type and return the larger value)
  - **Different code versions** of these functions for strong type
    C++

NPTEL MOOCs Programming in C++ Partha Pratim Das

So, let us just understand this little bit in more depth by taking examples. Suppose, I want to compute a maximum of two values, and the values could be integers value, there could be double values they could be strings, so in C string, they are they are char star they could be complex numbers and so on they could be so many others. Now, if I want to so a max will have a signature of max, I have one value, I have other value, I have the results. So, this is a signature of max which is kind of would come in something like this. If it i am doing it for int, if I am doing it for double it will look like this for strings for complex and so on. So, certainly we know that unlike in C were writing such functions were really difficult because you could not reuse the function name for different types C++ allow us to overloads. So, with the overloading, we can actually write a number of overloaded functions all of which will be max and we can just use that.

Now, if we look into these overloaded functions then we see few we observe a few things; one is they have the same algorithm the way to find out max is to do compare two values find out which one is larger and then returns that larger value that is the basic you know algorithm for. So, the algorithm is same in respective of which overload which we use, but since C++ is strongly typed for every version, we need to have a separate code we need to have a separate version of code which will work the code for int will not work for double and so on. So, that is a lot of you know code duplications that need to be done. So, that is basic motivation for looking at some solution to optimize all this.

(Refer Slide Time: 05:19)



Max as Overload

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

// Overloads
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;

    return 0;
}
```

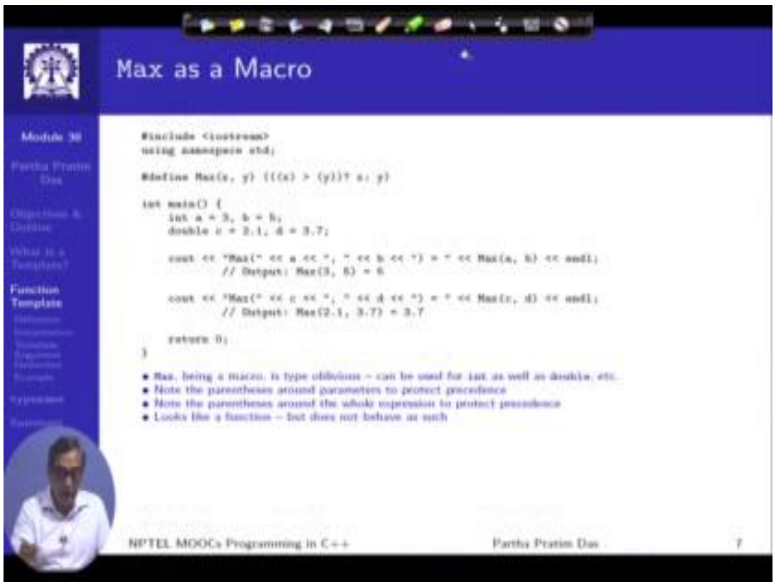
- Overloaded solutions work
- In some cases (C-string), similar algorithms have exceptions
- With every new type, a new overloaded Max is needed
- Can we make Max generic and make a library to work with future types?
- How about macros?

NPTEL MOOCs Programming in C++ Partha Pratim Das 6

So, first is if we will look at max as overload. So, these are the different overload. You can see that this is overload for int, this for double, this is for char star, c string now naturally if you c strings are represented by pointer to character. And in that you are not really interested to compare the pointers, but you want to compare the strings. So, here your algorithm is little bit different, but it falls in the same category or structure. And with that, you can you know print the max value of a different integers pairs, double pairs, string pairs and so on.

So, this will work, but the catch in this as the problem here is the solution works and in some cases exceptional I mean different kind of code need to be written, but the main problem is if I have a new type, I will need to write the max all over again, I have to overload and write that. So, every time I add a new type to my system if it can be compared and I want to do a max, I have to write a separate code. So, the point is can we make the reuse stronger can we make max generic, so that it not only will work with the currently known types it will also work for the types that will be defined in future. So, that is an interesting problem. So, can we do that for, you will immediately come and say that is possible you in C already that is called macros.

(Refer Slide Time: 06:50)



Max as a Macro

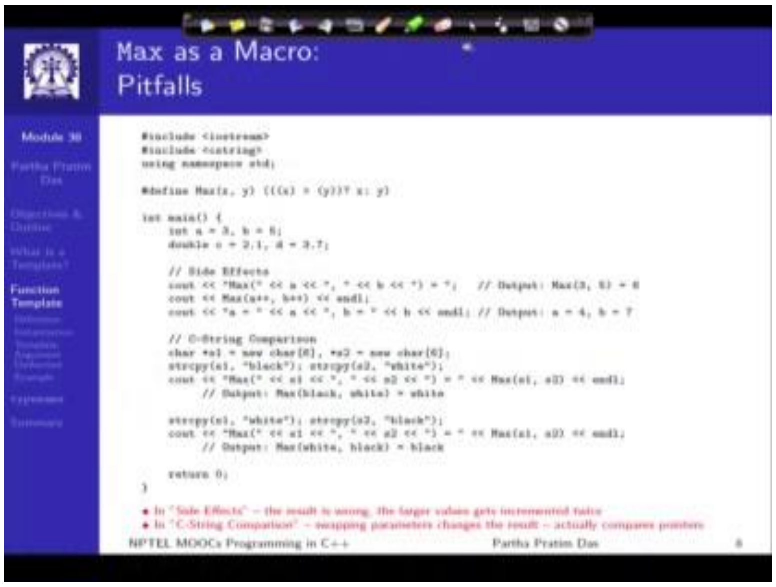
```
#include <iostream>
using namespace std;

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

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

    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

So, lets us look back into macros. So this is a macros, this is a macros. I can write the max function in terms of a macro like this and if I do this for int double. I do not really when I write the macro, I do not need really need to know what the type is whether it is int or it is double. I can simply write this because it gets replace at the point, where the macro is instantiated and therefore, the corresponding expression works. So, this is this is fine it looks like a function, but it does not really behave like a function. But just you remind you that the moment you write macros your into lot of lot of different problems, one a certainly is the way you write the macro you see that the parameters are parenthesized here.

The reason is unless you do that then x itself could be an expression which will get replaced here and that if that expression has operators which have a precedence, which is lower than this greater then operator then you will have a whole of this expression mixed up. Similarly the whole expression needs a parenthesis around it because otherwise for example, if you want to use it here as I did. So, here this whole expression is coming up if you do not have these then the precedence of this question mark colon and the precedence of the output indirection operators are in the wrong order. So, it tries to output only this part and then put a question mark column on the top of that. So, these are these are some of nuances you have live with, but then it look likes can we can we live this solution.

(Refer Slide Time: 08:21)



Max as a Macro: Pitfalls

Module 38

Partha Pratim Das

Objectives & Outline

What is a Template?

Function Template

Definition

Instantiation

Template Argument Deduction

Example

typename

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;
        // Output: Max(black, white) = white

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

    return 0;
}
```

- 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

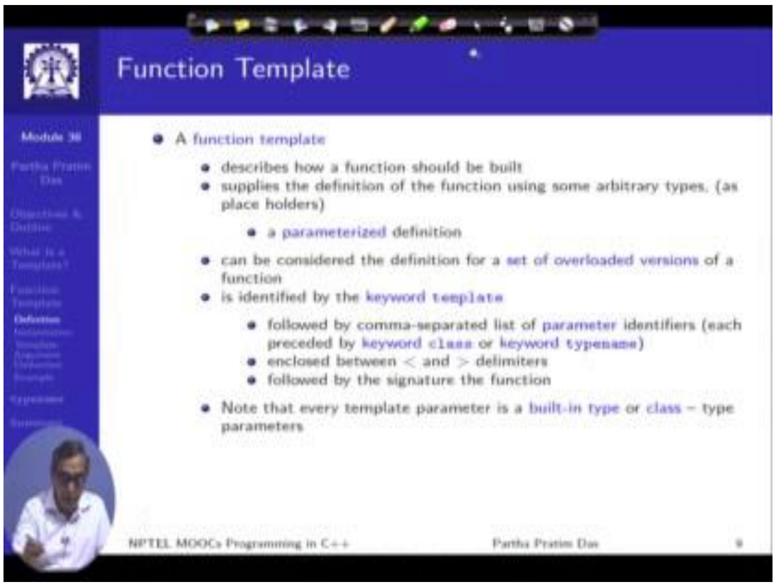
NPTEL MOOCs Programming in C++ Partha Pratim Das 6

Now, so I would remind you to some of the pitfalls this is one. So, I would like to given a, and b, I would like write max a plus plus, b plus plus thinking that this will give me the maximum value from the original value if a, and b and then it will give me the incremented value. So, the original values being 3 and 5, I am expecting an output 5 naturally, because of the original values. But if you actually work this out, you will find that the output of max is given as 6.

And after this is completed, the incremented values, the first value is incremented ones, the second will be incremented twice or basically the larger value gets incremented twice, because it turns out to be the expression is basically a plus plus, b plus plus question mark, a plus plus ,b plus plus this is what you actually got. So, depending on whichever was larger possibly in this case, this was larger. So, b plus plus got incremented twice. So, this is that is kind of a, so this is quite counter intuitive to this.

Then you have issue of if you have two character pointers in want to compare them, and you will simply use one pointer keeps use a string black, other gives a string white. Certainly, in this part of the code, if you look carefully all the time doing is I have just stopped the order in which they are invoked; s1 and s2, and the result become different. The reason is simply, because here when we write the macro, it basically compares the value of the pointers not the string that are pointed to by those pointers. So, if we change that order this will be basically change.

(Refer Slide Time: 10:11)



Function Template

Module 38

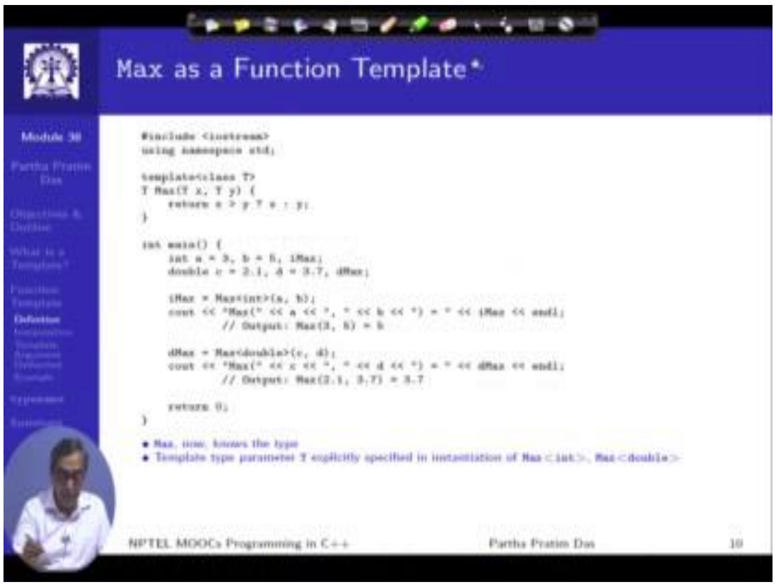
Partha Pratim Das

- 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

NPTEL MOOCs Programming in C++ Partha Pratim Das 9

So, this kind of you know pitfalls make macros quite unusable and we lead to what is known as templates. Templates are nothing but functions were the type is parameterized. So, it is a parameterized definition. So, functions have multiple parameters, arguments, formal arguments, and a written type, so all of these or sum of these could be made into parameterized types. So, that you do not really well you write the function, you do not really say what the type is, but rather you put a type variable in place of that. So, this is a description of what the syntax how the syntax should be written we will skip over, because once you see the example, it will become obvious us to how you write that.

(Refer Slide Time: 10:50)



Max as a Function Template*

```
#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

    return 0;
}
```

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

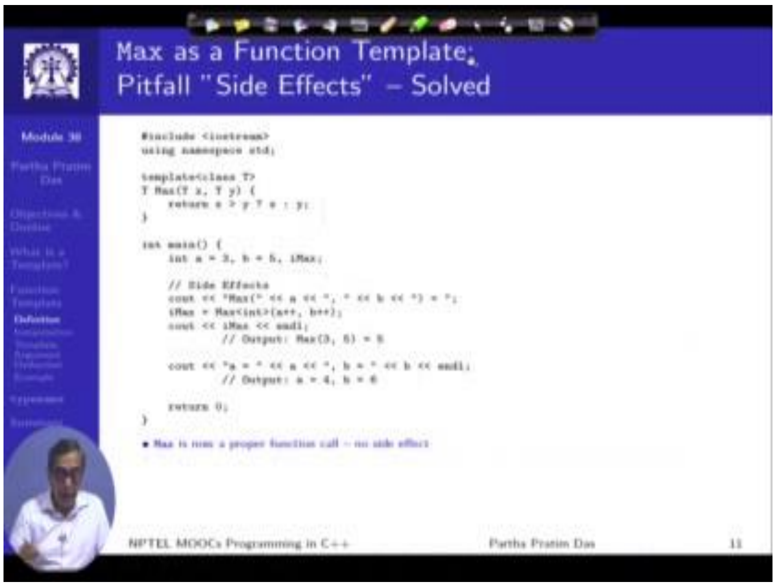
NPTEL MOOCs Programming in C++ Partha Pratim Das 10

So, why you want to write the max function against, the max if I write it with int then it looks like int x, int y it will written an int type and the code certainly is returned x greater than y question mark x column column y. So, this is my code of max all that we are doing now is we are saying that we write this without any specific type int, but rather we will use a placeholder in place of int, which is type T and so that is what we write here. And the fact that we are using a placeholder is specified by this template is a keyword and within the corner bracket you write class T this, defines that T is a type variable T is a placeholder for a type and then I can use it.

So, what it does is something interesting, once I it have written that then I can use the max function like this I can, I will write max within corner bracket I will put a type and then I will show the original name of the function is max that is templatized name and this is the instantiated name, which means that if I when I put this int it means that used the code of max, but take t to be int. And generate this whole code of the function taking t to be int and then you call on to that function. So, when I later on say max double. So, what it does it takes again generates another code were t is taken to be double a similar code of max is generated and that is compiled and that particular.

So, basically once we instantiated int and double actually what I get, when I execute are two overloaded functions both of which are called max both of which takes two parameters and written one value and all of these for a for a one particular function is have the same types. But one is for two integers written integers one is for two doubles and written the double, so that is, what is the basic job of the templates. So, here a earlier we were having to write this overloads earlier we were having to a macro for these overloads, but now we can written one template a function a with the parameterized type and actually get that required functions generated whenever we need them.

(Refer Slide Time: 13:26)



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

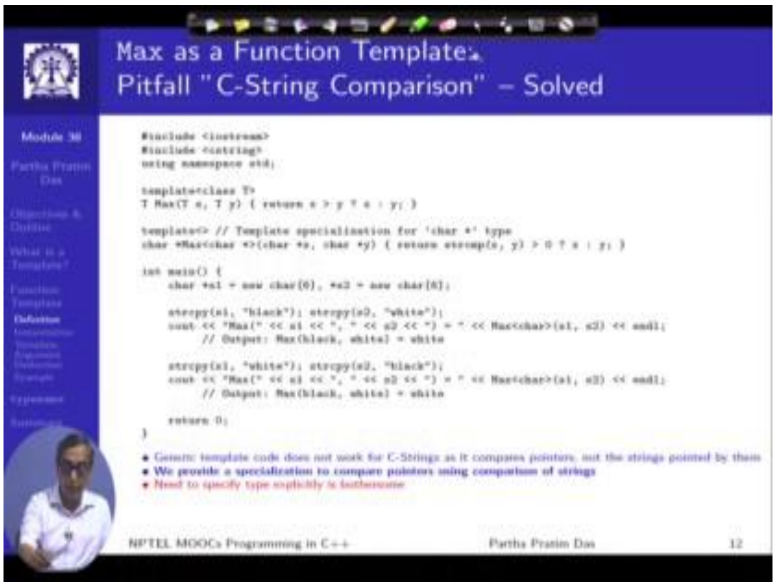
```
#include <iostream>
using namespace std;

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

    return 0;
}
```

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

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

So, if we do this then a certainly the problems get solved, we can now go and write the same code you will find that you are getting the desired effect. Now there is I mean there is nothing special in this, because this is become truly a function. So, this will actually take the value of a take the value of b call the function and after that increment a, and b individually and that will certainly give the correct results. So, are this does not have the issue that macros was showing.

(Refer Slide Time: 13:50)



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

```
#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

    return 0;
}
```

- 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

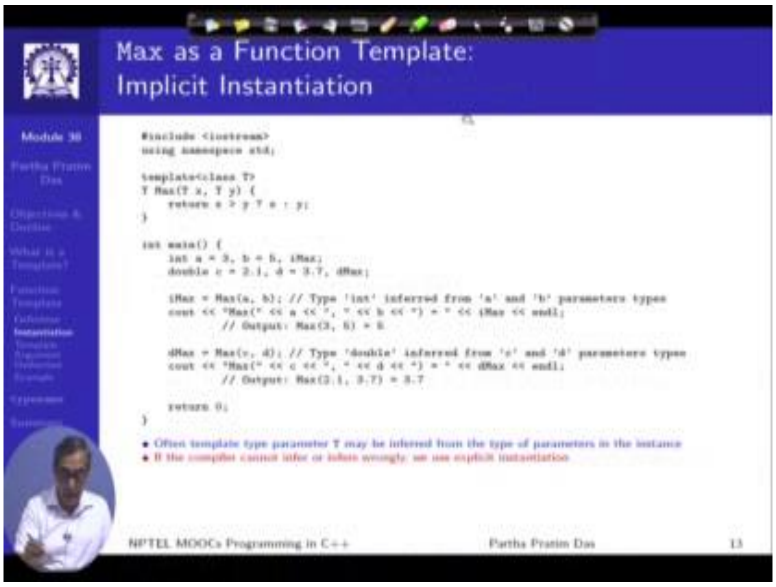
NPTEL MOOCs Programming in C++ Partha Pratim Das 12

Similarly if we want to work with say, also use this templates to compare to c strings and we saw that comparing c strings are difficult, because a we want when we want to compare just if we compare the two pointers then you will not get the same results. So, if we just take this as char star take t as char star I say max char star, then whatever I get will not be a correct thing, because it will give me the same issue as the macros was showing that we will just compare the pointers. So, what can I do is something very, very interesting is I can do what is known as templates specialization, that is I can say that this is a definition of the template max for a type variable t for any t what so ever. But if the t is specifically char star then I have a different definition of this function.

So, you will look at this, this is a general template the generic or the primary templates as it is called and this is specialized templates you here are actually replacing t by char star, we are replacing t by char star. And since we have replaced c this is become the written type this is become the name of the function because showing that you have replaced t by char star this is t replaced by char star and so on. And since the template had only one template parameter and that has been replaced. So, now, the template specialization list as no parameter at all showing that it is being specialized here. So, what it means that, now if will invoke instantiate max int, it will invoke this version of the template function with t being int. But if we invoke a as we do here max this should be char star actually it not char char star, if we do max char star there is small type which I will correct later on.

So, that will actually invoke the specialized template version of this function it will not invoke this one. So, in this version now we take care of the fact that we are not interested to compare the pointers where interested to compare the function that are pointed to by the pointers. So, we do strcmp on these two pointers compare them and give the results. So, it will it will take care of very easily it will take care of the issue that macros are showing, which was not possible to do in terms of the macro we could not we can gave only one definition of a macro not a multiple once. But here we can actually provide a specialized definition for the template to take care of such issue.

(Refer Slide Time: 16:46)



Max as a Function Template: Implicit Instantiation

```
#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

    return 0;
}
```

- 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

NPTEL MOOCs Programming in C++ Partha Pratim Das 13

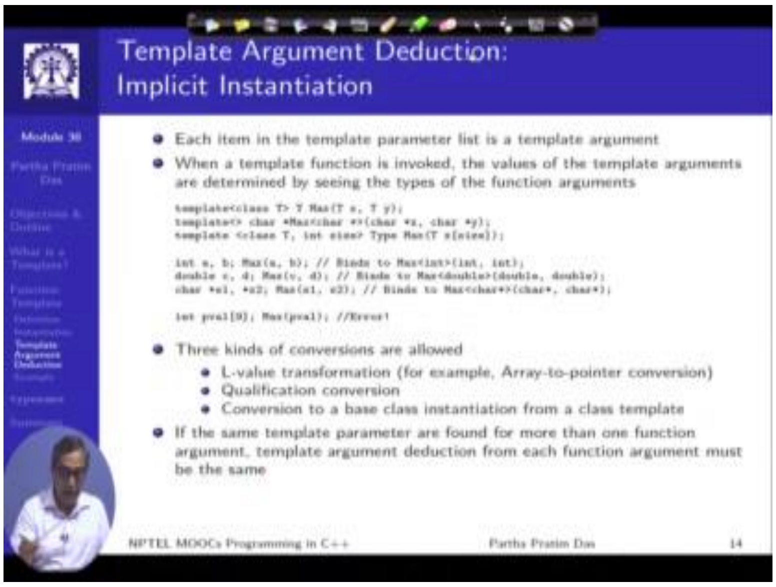
So, this is a really so templates are a good way to solve this. But certainly what will bother you is now the way you invoke the templates is changed. Now you have to invoke them as and the instantiation as we say as to say what the type is or for doing it as a double you have to say what the type is now as it. So, happens that C++ does allow you to skip this in many cases. You may not require to specify the type in the instantiation provided the types of the parameters can tell you what this template variable type must be. So, the idea is like this that let us say if you are looking into here, the type is not specified here looking at max ab. So, what can you get from max ab, if max ab has to match this then you know that the first hypothesis is this is has got a type int, this has got a type int. So, these are first parameters. So, this is the placeholder for x.

So, if this has to match then T should be int these a second parameter if this has to match then T as to be int, and they are consistent, so if I just put T as int and generate a version of the function which T as int that will match up with this instance. Whereas, when I do max cd then c has double, so is d. So, here making T double will actually make it match the definition of the template. So, earlier we needed to write max int ab which is called the explicit instantiation. Here, we were writing max double cd which is the explicit instantiation we were saying that what types we want, we can skip this now because that can be inferred from the parameter types.

Now, it is possible that in some cases you would not be able to do that. For example, a in this context, I can say that if you have max ac suppose you want to call max ac and your expectation is this is int, this is double. And certainly int can be implicitly converted to double, but in this case the compiler will not be the template instantiation will not be able to deduce the template type from this. Because from here it gets T as int; from here it gets T as double, so this is contradictory it has to get the same type for both.

But you will able to still call this, for example, if you say max double ac, how will that work because if you say max double if you are explicit in the instantiation then T is double, which means that this is double as well as this is double. Whereas actually this is int this is double the second one is double it directly matches this is int, an int can always be cast implicitly to double as you know. So, it will cast implicitly and do that. So, it says that it is possible that not in every case you will be able to do and implicit instantiation, but in many cases, you would be able to do that. In cases, you cannot because of you want to mean something different or you want to mix up certain types and do some conversion, you will have to use the explicit instantiation.

(Refer Slide Time: 20:20)



Template Argument Deduction: Implicit Instantiation

- 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> Type 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!
```

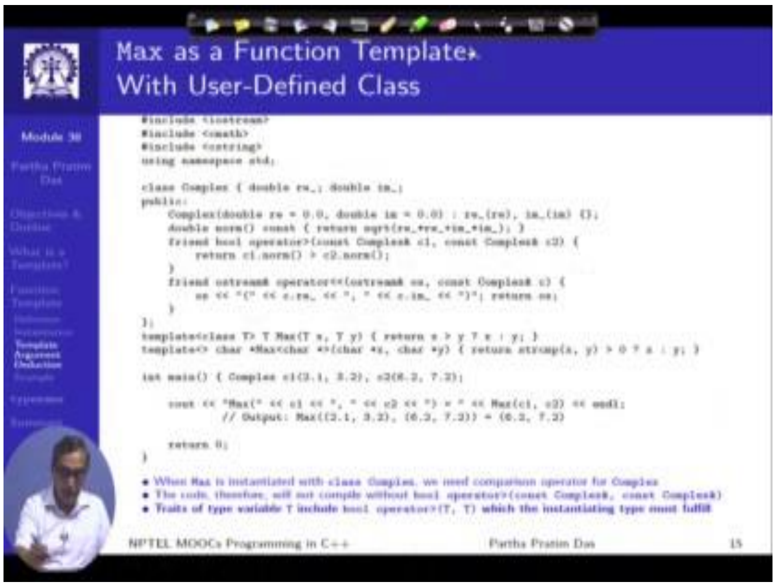
- Three kinds of conversions are allowed
  - L-value transformation (for example, Array-to-pointer conversion)
  - Qualification conversion
  - Conversion to a base class instantiation from a class template
- If the same template parameter are found for more than one function argument, template argument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

NPTEL MOOCs Programming in C++ Partha Pratim Das 14

So, here I have shown a couple of examples of things that you can do implicitly of the different max forms that you had seen. And you will have to be you will be able to figure out has to how they are deducing the type. Normally, when you use templates and three kinds of conversions are allowed for this types; one is called l value transformation that is in place of an array, you could present the pointer and vice versa.

You can do a qualification conversion which is basically conversion using a non-const value in place of a const value and so on. And you can do a base class instantiation from a class template which is basically doing an up-cast conversion in the template. So, these are the conversions are allowed; otherwise, the template arguments are will have to otherwise follow the strict matching of types and it will not deduce anything. It will say that I am confused I am failing and then you will have to explicitly specify the template conversion argument conversion.

(Refer Slide Time: 21:28)



Max as a Function Template: With User-Defined Class

```
#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)

    return 0;
}
```

- 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

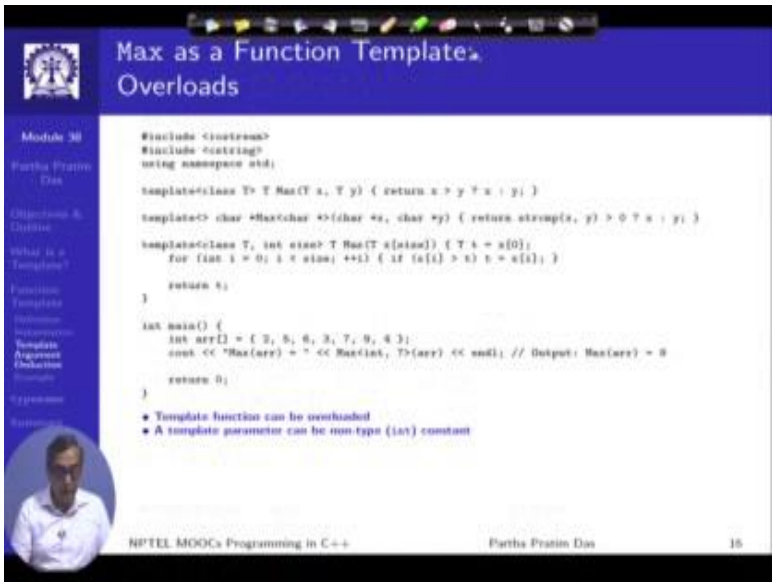
NPTEL MOOCs Programming in C++ Partha Pratim Das 15

Now, continuing with our max example, I show here that you can use the similar template for user-defined types also complex is a user-defined type here. And I have variables of complex type and I am using max on them with implicit instantiation. Now, of course, you will have to keep in mind that if you do this, then certainly which version this version is specifically for char star. So, if you are doing calling this then it is inferring c 1 is complex; c 2 is complex, so it is inferring T is complex for this instantiation. So, it is basically looking at this version. If it is looking at this version, if it is looking at this version, that means, that it is looking at complex, so this is how the generated function will look like complex x, complex y return then x greater than y question mark x y right. This is what it will return.

Now, if it is doing that then what it will require what is x greater than y. It is x is complex, y is complex, so x greater than y will have to be defined for the complex type, if the complex type does not have a x greater than y, that is if the greater than operator has not been overloaded in the complex class. You will not be able to put this instantiation. So, for that, I had to provide a overloading opera of operator greater than in the complex class, if you remove this you will find that this code will not compiled because the it will deduce T to be complex and then it will say that it requires a greater than operator which does not exist in the complex class. And therefore, this templatized version cannot be instantiated for complex.

So, these are you will have to give the parameter type t there are certain basic operations and operations that T will have to satisfy. So, that if you want to pass some user-defined type want to instantiate with some user-defined type your user-defined type will have to satisfy those properties. For example, the other one is certainly it returns by value. So, this will have to your complex class need to have a copy constructor. Now, here I have not provided one. So, it is assumed that since this does not have dynamically allocated memory it will get a free constructor, free copy constructor from the compiler and that that itself will be used. So, in this way you could use any of the user-defined types also provided they satisfy this trait of T that is the required operations of the type variable T, if the satisfy that then you will be able to use that in terms of instantiating your template.

(Refer Slide Time: 24:32)



Max as a Function Template: Overloads

```
#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) { return strcmp(x, y) > 0 ? x : y; }

template<class T, int size> T Max(T x[size]) { 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

    return 0;
}
```

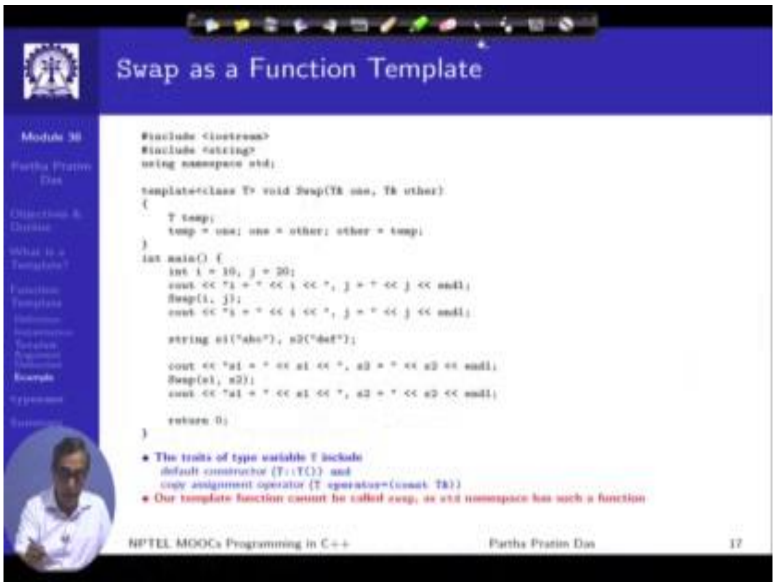
- Template function can be overloaded
- A template parameter can be non-type (int) constant

NPTEL MOOCs Programming in C++ Partha Pratim Das 16

And finally, you can explicitly overload a template function also. So, as we have explained that the template itself is an arbitrary number of overloaded functions because for every type T this is overloaded function of T. Similarly, this is an explicit specialization, this is a specialization for char star right, there could be specialization for other types also these are also overloads. But I could have a more classical overload like this for example, here where I have a max which takes an array as a single parameter and that array is specified by a size. So, you look at this is an array and I am trying to call this max this certainly does not match with this template function, it does not match with this template function both of them need two parameters, but this has a single parameter. So, these are overload of the max template function, and this itself is another template.

And what is interesting about it has a class type and this also as another template parameter which is called non-type parameter, this is not a here you accept a type and here you get a more traditional function kind of time, this is called a non-type parameter. So, this is possible only if this is an int type. So, you can pass a constant value here in terms of the template you can pass a constant value here. So, here I am using this non-type parameter to pass the size of the array. So, we will say that the size of the array is 7. So, we will we are specifying that, so the array if you see this is 7. So, I am passing this here. So, this will tell the template function that the size is 7, so that size is being used within the code of the template function. So, in this way, you can use non-type parameters and as well as explicitly overload the template function with other templates or other non template functions as well.

(Refer Slide Time: 26:43)



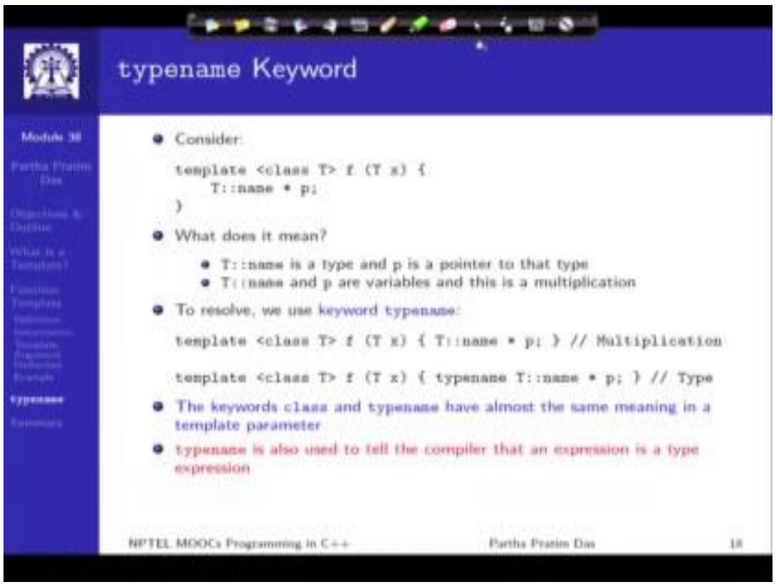
Swap as a Function Template

```
#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;

    return 0;
}
```

- 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

So, this is just for your practice and understanding. I will not go through these; it is here where we have written the swap code. Swap is something that you need to do every time and we have seen that. With the use of reference parameter swap really becomes efficient to write in C++. And now you can make it more efficient by actually writing a swap for any class what so ever except the fact that certainly swapping need copying values. So, if we have to swap then the copy operators are copy constructor in the copy assignment operator has to be there in the available trait of the function parameter a type parameter T.

(Refer Slide Time: 27:23)



typename Keyword

Module 38

Partha Pratim Das

Objectives & Outline

What is a Template?

Function Template

typename

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

NPTEL MOOCs Programming in C++ Partha Pratim Das 18

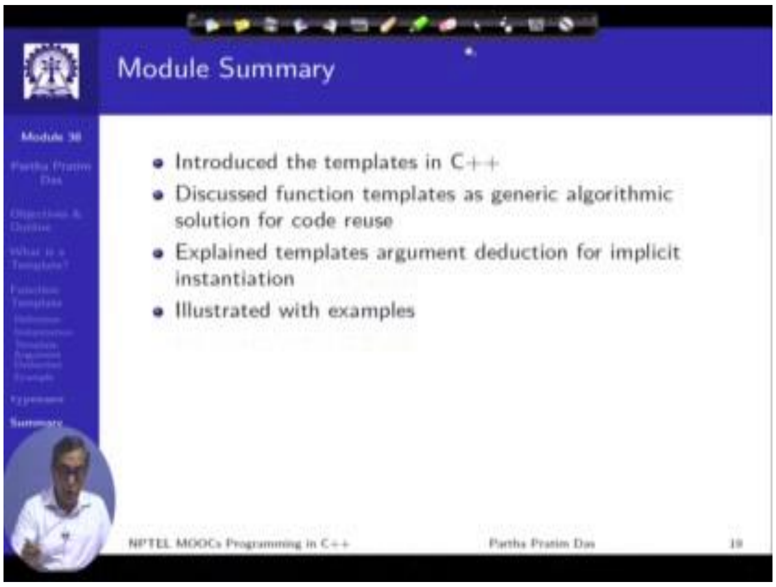
Ah finally, let me introduce another keyword typename, like we are writing class here in state C++ also allows us to write the keyword typename. You can interchangeably write class and typename, there is basically no difference except for one case which is illustrated here suppose you have written this. Now, the question is what is this code mean? You can read it in two ways one is you can think of that this whole thing is name of a variable. So, T is some class name or some you know namespace name. So, the T colon colon name basically turns out to be the name of a variable and this is another variable. So, this is a multiplication, or it could be this is name is basically within t class t name is a typedef. So, it is a type. So, this is if this is a type then this p is a declaration or pointer declaration of type T colon colon name.

So, from the syntax you cannot figure out which one your meaning that is what the C++

gets confusing. So, C++ has to provide this specific keyword typename to resolve for example, if you just write this then the default interpretation is this is a multiplication. So, this is a variable, this is another variable. But if you want to say that this is a type, and p this is a declaration then you have to write typename in front of it. And then you will have to write this where in it will be understood that this is a typename and this is a name of a variable declaration, a pointer variable declaration in this case.

So, whenever you need to use specifically say in the template that you are talking about a type you use this typename keyword, you cannot use a class here to mean the something, because that will get confused with the nested class because if you have a class within a scope then it means another different class definition. So, the new keyword was required, but otherwise in every other context class and typename can be used interchangeably.

(Refer Slide Time: 29:21)



Module Summary

Module 38

Partha Pratim Das

Objectives & Outline

What is a Template?

Function Template

typename

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

NPTEL MOOCs Programming in C++ Partha Pratim Das 19

So, with this, we are at the end of today's module. We have introduced templates in C++ and specifically we have discussed about function templates and shown how they are advantages over simple overloads and use of macros in C. And how do they allow you to write a generic code meta programming code which given the instantiation which could be implicit or explicit can generate the actual typed function code and that can be use subsequently in your program.