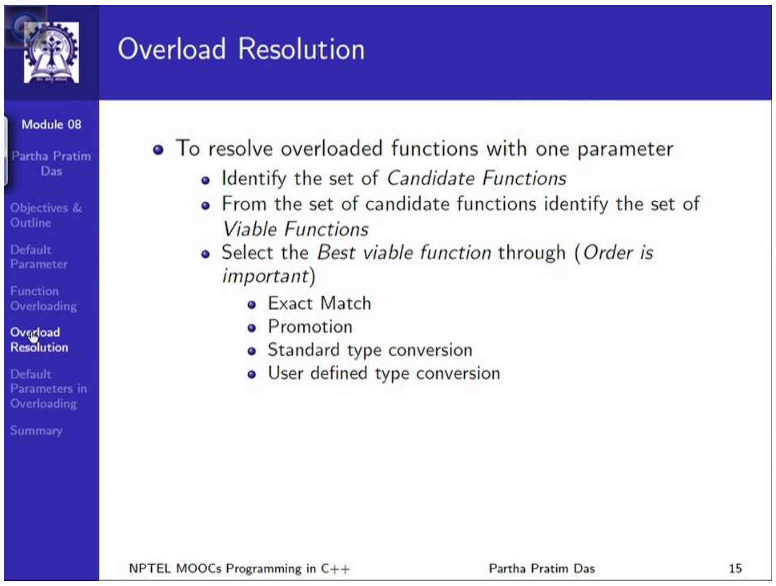
Programming in C++

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

Lecture – 14
Default Parameters and Function Overloading (Contd.)

Welcome to Module 8 of Programming in C++. We have already discussed default parameters and we have discussed function overloading, the basic requirement of function overloading and how functions can be overloaded, that is, multiple functions can be there with the same name in C++. But, as long as they differ by their parameter type, number and types of parameters. And, we have observed that functions cannot be overloaded with the return type.

(Refer Slide Time: 00:54).



Overload Resolution

- To resolve overloaded functions with one parameter
  - Identify the set of *Candidate Functions*
  - From the set of candidate functions identify the set of *Viable Functions*
  - Select the *Best viable function* through (*Order is important*)
    - Exact Match
    - Promotion
    - Standard type conversion
    - User defined type conversion

Module 08
Partha Pratim Das
Objectives & Outline
Default Parameter
Function Overloading
Overload Resolution
Default Parameters in Overloading
Summary
NPTEL MOOCs Programming in C++
Partha Pratim Das
15

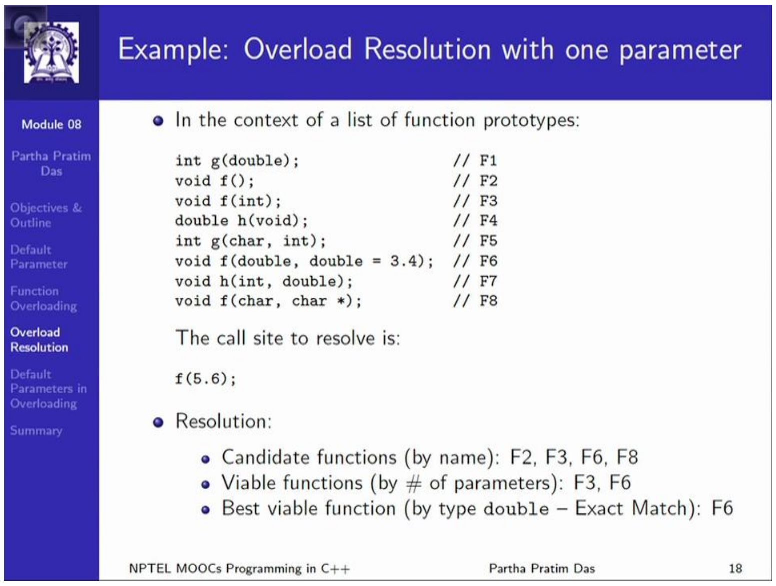
Now, we would go next to this. Give a glimpse of the overload resolution. Overload resolution is a process by which the compiler decides amongst multiple candidate functions, multiple candidate definitions of a overloaded function that exist. The compiler has to decide which particular one it should bind to. This is overload resolution, is a little bit advanced topic in function overloading. So, if you find it little difficult to handle at this stage, you may keep it at a (Refer Time: 01:35) and come back to it later when we discuss about the depths of overloading in C++, later on.

So, here we would just try to give you a basic idea that how to resolve the overloaded functions. And, I am using an example here with just one parameter to keep things simple. These are the basic steps which are to identify a set of candidate functions; that is, you first scan all functions that are been defined. Certainly, a function can be called, provided it has already been defined. So, the compiler scans all definitions that have been received so far, before this call and forms a set of candidate functions.

Then, it scans through the set of candidate functions that is this point scans to the set of candidate function to find a set of viable functions that match the number of parameters and so on. And then, it tries to do the most difficult part, which is, decides on what is the best viable function. And, for doing that it makes use of the following strategies. And, please note that the order is important here. That is, it applies the strategies, not arbitrarily. But, in this order of exact match, promotions, standard type conversion and user defined type conversion.

Now, certainly we cannot explain all of this right here. So, we will come back to it even at a much later stage. For now, I will just show you a simple example of how the resolution is done using exact match and talk a little bit about what is a scope of exact matching and what is the scope of promotion.

(Refer Slide Time: 03:28)



Example: Overload Resolution with one parameter

Module 08

Partha Pratim Das

Objectives & Outline

Default Parameter

Function Overloading

Overload Resolution

Default Parameters in Overloading

Summary

- In the context of a list of function prototypes:

```
int g(double);                 // F1
void f();                      // F2
void f(int);                   // F3
double h(void);                // F4
int g(char, int);              // F5
void f(double, double = 3.4);  // F6
void h(int, double);           // F7
void f(char, char *);          // F8
```

The call site to resolve is:

```
f(5.6);
```

- Resolution:
  - Candidate functions (by name): F2, F3, F6, F8
  - Viable functions (by # of parameters): F3, F6
  - Best viable function (by type `double` – Exact Match): F6

NPTEL MOOCs Programming in C++ Partha Pratim Das 18

So, let us take an example. Let us take an example of overload resolution with one parameter. Let us assume; concentrate on the list on top here. Let us concentrate on this list. Let us assume that our target actually is to find out for this particular call of a function by name f and parameter 5.6 to decide, which of the signatures this function corresponds.

That is, if I have said f 5.6, if the user has said that then which particular function has the user intended to call. So, in the context we are taking that before this call of f 5.6 has been encountered 8 different function headers or function definition has been seen by the compiler. And, just to refer to them conveniently we have called them as F1 to F8. Now, how do you decide from? So, certainly f call of f 5.6 has to match exactly with one of them, so that I can say that at particular function is getting called. So, what are the steps in resolution? so, if you look at here, if you use steps in resolutions, first you form the candidate function.

In this case, the simple way to form the candidate function is to look for the name and find functions, which has a matching name. Here, we do not have any other factors. So, we will just go by name; the name is f here. So, let me look at what are the functions which have name f; this function, this function, this function, this function. So F2, F3, F6, F8. F2, F3, F6 and F8 are my candidate function.

Certainly, if I say f 5.6, I could not have meant this function g, even though g has one parameter, even though g has a type parameter type which is double, which is the type of 5.6. But, certainly I could not have meant that because my names are different. So, my candidate functions give me these four functions. So, now, I focus only on this candidate function.

Now from this, based on this candidate function next screening that I do is based on the number of parameters. So, out of these candidate functions, I see what are the number of parameters. So, if I look at F2, this is 0; if I look at F3, it is 1; if I look at F6, it is 2 or 1 and if I look at this, this is 2. So, these are the required number of parameters for calling the respective candidate functions. And, how many parameters we have here? 5.6 - one parameter.

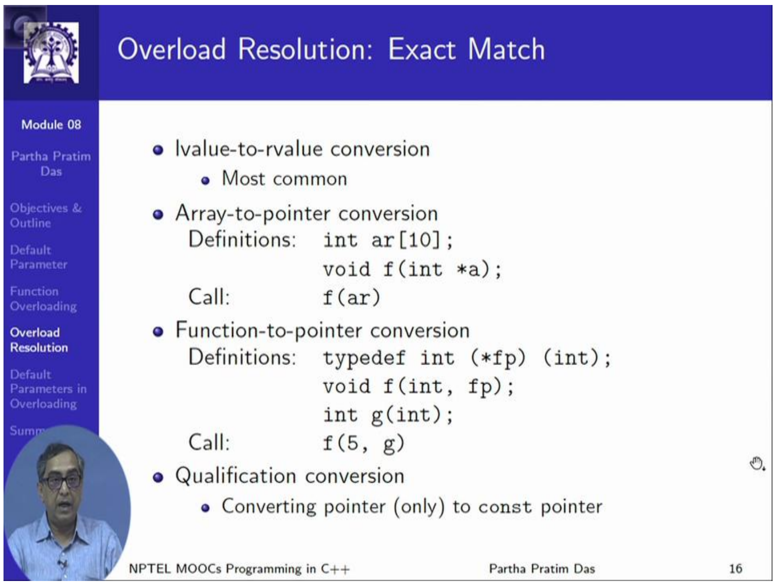
Second, immediately say that this is ruled out. I cannot call F2 by f 5.6. I cannot similarly call this because it requires two parameters. So, my choice voice down to these two; it is either F3 or F6; F3 OR F6. So, I call them the set of viable functions.

Now between F3 and F6, I have to decide which one to use. So, I can think of that I am actually calling the function F3, which needs a int parameter. So, if I have to call function F3, then what is to happen? It has to happen that this parameter actually is a double parameter; 5.6 is a double. So, if have to call it F3, then this double has to get converted to int; which means, some truncation is, have it to happen. And actually the call will turn out to be F5. And then, I will be able to call it. Otherwise, I can call F6, where in it is f 5.6 as a first parameter, second parameter is defaulted. So, the second is; so, this could be the call. So, either this is the call or this is the call.

Now, the exact match say that I will choose this because here the type of the parameter, formal parameter and the type of the actual parameter, these two match. Both of them are double. Whereas between this and this, I need to convert one is double. I need to convert, actually truncate that to main int. So, that is not a preferred one. So the exact match strategy, by using the type double tells me that my dissolved function is F6. So, once I am done, the compiler will resolve that you have called this function.

This is the process of overload resolution. And, it is a, quite a complex process because so many different types could be involved. So, many different cases can happen. And, so while somebody who writes a compiler needs to know it very thoroughly. As a programmer, it is good to have some idea because you may have meant certain overload to be used. If the compiler did uses a different kind of overload, then you are using a wrong function at a right place. So, just let us take a quick look into some of these strategies. Exact match is the first one. That just to recap, these are the different ways to; these are the different ways to resolve the best function, exact match promotion and so on. So, you just are looking at what is a exact match.

(Refer Slide Time: 09:51).



Overload Resolution: Exact Match

Module 08

Partha Pratim Das

Objectives & Outline

Default Parameter

Function Overloading

Overload Resolution

Default Parameters in Overloading

Summ

- lvalue-to-rvalue conversion
  - Most common
- Array-to-pointer conversion

```
Definitions:  int ar[10];
              void f(int *a);
Call:         f(ar)
```

- Function-to-pointer conversion

```
Definitions:  typedef int (*fp) (int);
              void f(int, fp);
              int g(int);
Call:         f(5, g)
```

- Qualification conversion
  - Converting pointer (only) to const pointer

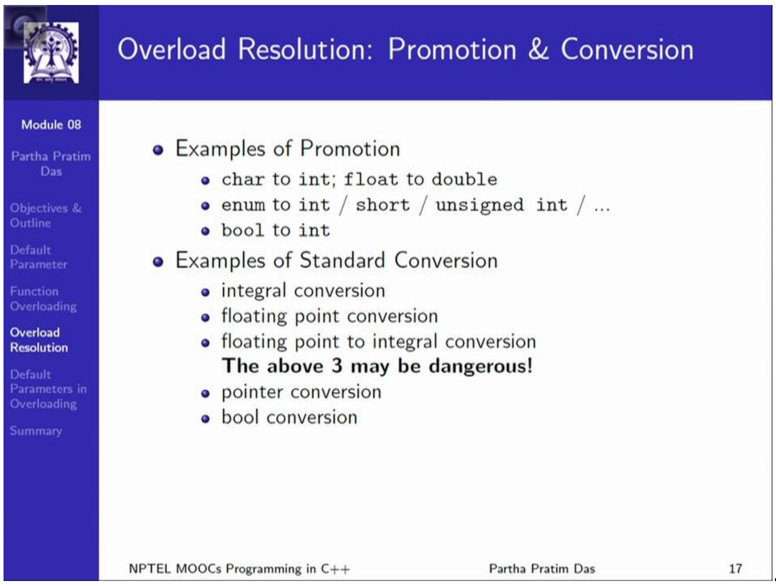
NPTEL MOOCs Programming in C++ Partha Pratim Das 16

So, exact match is; these are different cases of exact match. That is, you can convert an lvalue to an rvalue; that is, if you are using a variable just to a, in case you are not familiar with this terms, l value, l value is left value or basically the value of the variable. And rvalue is, I am sorry this is not, this is wrong, address of a variable. And, rvalue is a value; is a right value. So, it is like if I am writing a assign b, then what I am interested in b is its value of the b. and, what I am interested in a is the location where we a exist where I can go and keep the value of b. So, here I am interested in the lvalue and here I am interested in the rvalue.

So, you may need to convert from lvalue to rvalue. The other possibilities are like this, where you convert and you are trying to pass an array, but the actual function needed a pointer. You can convert arrays (Refer Time: 11:17) to pointer. You can convert function to pointer. For example, this is the function pointer. We have seen what function pointers are. This is a function that I am trying to resolve. Which, second parameter is a function pointer; this particular function pointer. But, what I have passed is actually a function. I did not pass a pointer. So, I should have passed ampersand g, which is a pointer.

But, I have just passed the function name directly. So, this conversion is allowed as a match. And, I can convert pointers to constant pointers. So, these are different cases of exact match that the overload resolution uses. Besides that, you could take promotions and conversions.

(Refer Slide Time: 12:05)



Overload Resolution: Promotion & Conversion

Module 08

Partha Pratim Das

Objectives & Outline

Default Parameter

Function Overloading

Overload Resolution

Default Parameters in Overloading

Summary

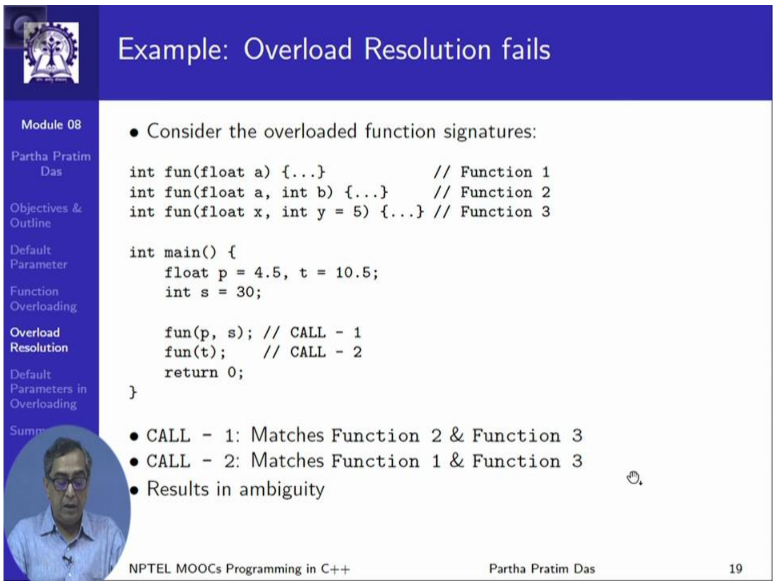
- Examples of Promotion
  - `char` to `int`; `float` to `double`
  - `enum` to `int` / `short` / `unsigned int` / ...
  - `bool` to `int`
- Examples of Standard Conversion
  - integral conversion
  - floating point conversion
  - floating point to integral conversion
    The above 3 may be dangerous!
  - pointer conversion
  - bool conversion

NPTEL MOOCs Programming in C++ Partha Pratim Das 17

.

For example, I can convert character to integer, enum to integer, bool to integer. Or, I can convert between other kinds of integral conversions float to double, double to float; all of that various kinds, various types of a pointers can be converted. So, all these promotions and conversions are also used for overload resolution. This is just to give you the glimpse. We will come back to this much later when we talk about variety of other overloading that are possible in C++. So, this is just to give you a basic idea.

(Refer Slide Time: 12:43).



Example: Overload Resolution fails

Module 08

Partha Pratim Das

Objectives & Outline

Default Parameter

Function Overloading

Overload Resolution

Default Parameters in Overloading

Summ

- Consider the overloaded function signatures:

```
int fun(float a) {...}              // Function 1
int fun(float a, int b) {...}      // Function 2
int fun(float x, int y = 5) {...} // Function 3

int main() {
    float p = 4.5, t = 10.5;
    int s = 30;

    fun(p, s); // CALL - 1
    fun(t);    // CALL - 2
    return 0;
}
```

- CALL - 1: Matches Function 2 & Function 3
- CALL - 2: Matches Function 1 & Function 3
- Results in ambiguity

NPTEL MOOCs Programming in C++ Partha Pratim Das 19

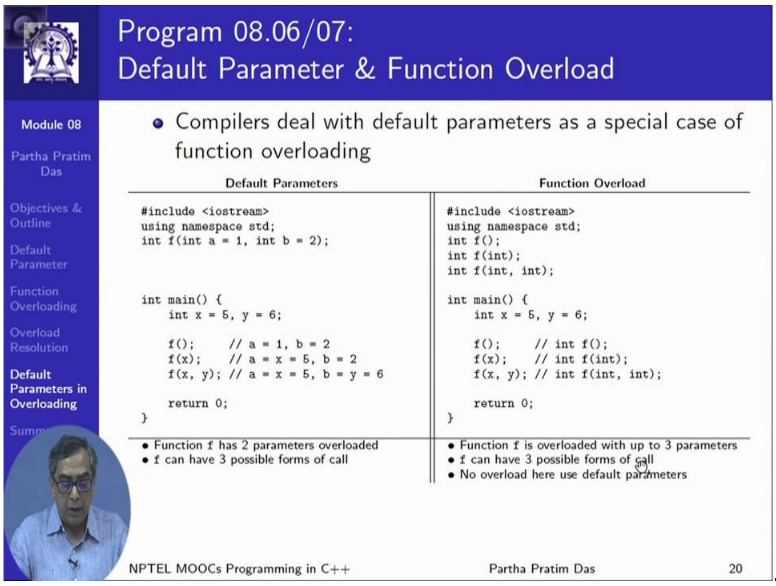
Now, I will end. Before I end, I will also show that a overload resolution may fail in different cases. And, you will have to be careful about those. For example, here look at these three functions; the Function 1 has one float, Function 2 has two parameters, Function 2 also has two parameters with one defaulted. And, this is the use that you are trying to do. So, if you try to resolve at this first CALL-1, which is taking two parameters, then what are your candidates are? All three functions are your candidates; Function 1, 2, 3 because the name matches.

Now, what is viable? Viable are you looking at the number of parameters. It is Function 2 and Function 3. Now between these two, Function 2 and function 3, what is the best one? It is difficult to say what is the best one because p is a float here, which matches the first parameter type of both these functions. If we look into the second, which is s, is int, which matches the second parameter type of both these functions. So, it is not possible for you to resolve between function 2 and function 3. So, if we have such a call, then the compiler will come back and tell you that it is ambiguous. The compiler will say, “I am unconfused, I do not know what to do”. So, these are different cases, where resolution overload resolution will fail.

The second one is another case, where we are just using one parameter for function. So, naturally your candidates again are all three. But, your viable one set, certainly two does not remain viable because here you are using just one parameter. Function 2 would have needed two parameters. So, both of these become viable. But again, if both are these are viable, you cannot easily resolve between them because you would, this could call this by a conversion, by a promotion of int to float; we just seen. Or, it could call 3 by assuming this is defaulted.

And, this is another promotion again from int to float. So, both of them has some, same kind of, you know, efforts, same kind of complexity. And therefore, this will result in ambiguity. So, whenever you write overloaded functions, you will have to be careful that such cases are not there in your code or if the compiler refuses to compile this and says that there is an ambiguity, please look for some of these situations that may exist in your code right now.

(Refer Slide Time: 15:40)



Program 08.06/07: Default Parameter & Function Overload

Module 08

Partha Pratim Das

Objectives & Outline

Default Parameter

Function Overloading

Overload Resolution

Default Parameters in Overloading

Summ

- Compilers deal with default parameters as a special case of function overloading

Default Parameters

```
#include <iostream>
using namespace std;
int f(int a = 1, int b = 2);


int main() {
    int x = 5, y = 6;

    f();      // a = 1, b = 2
    f(x);     // a = x = 5, b = 2
    f(x, y); // a = x = 5, b = y = 6

    return 0;
}
```

- Function f has 2 parameters overloaded
- f can have 3 possible forms of call

Function Overload

```
#include <iostream>
using namespace std;
int f();
int f(int);
int f(int, int);

int main() {
    int x = 5, y = 6;

    f();      // int f();
    f(x);     // int f(int);
    f(x, y); // int f(int, int);

    return 0;
}
```

- Function f is overloaded with up to 3 parameters
- f can have 3 possible forms of call
- No overload here use default parameters

NPTEL MOOCs Programming in C++ Partha Pratim Das 20

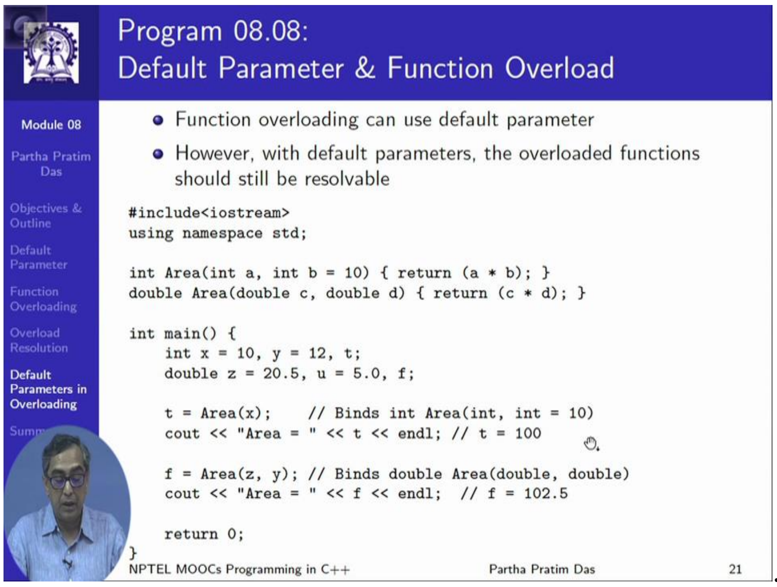
Now, we have looked at default parameters and we have looked at function overloading. Now, you can just observe that the reason we have put them together in the same module is the fact that basically they are; at the core they are basically the same feature, except for maintaining that default parameter value, in case of the default parameters. The whole mechanism in the language is the same. So, if we look at these two code side by side from left and right columns here and if you look at the main function, the main function here is identical. It has, this is exactly the same code on both of these. What, here we have one function with two parameters defaulted and here we have three functions overloaded.

Now in terms of call, all these three calls resolve to the same function. Except that, it depends on how many parameters you are using. And here, these resolve to the respective functions, depending on this is the simple resolution; because in each one of these case, as you can figure out there only be, all three will be in the candidate set. And, in every case there will be only one function in the viable set. So, there is nothing to look for.

So, basically what you can think of that if you default a set of, the set of parameters in the function, then you always are creating two choices for a default parameter that either it may exist there in the call site or it may not exist in the call site. Depending on that, you are as if generating two different signatures for the same function. So, what I am saying is if I; this means, I have, once we, once I write this function; that means, that I can call int with int int, which basically is this overload. It also means that I can skip the second parameter. So, I can call it with just one int, which is this overload.

It also means that I may not specify any parameter. Just call it without parameter, which is this overload. So underline, default parameter is basically nothing other than a special case of function overloading. With the added feature that it also carries the overload value, which is the initial, the value of the parameter along with it. Otherwise, depending on the function site, all site like this, all like this, which function will be bound is again a overload resolution problem.

(Refer Slide Time: 18:31)



Program 08.08: Default Parameter & Function Overload

- Function overloading can use default parameter
- However, with default parameters, the overloaded functions should still be resolvable

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

int Area(int a, int b = 10) { return (a * b); }
double Area(double c, double d) { return (c * d); }

int main() {
    int x = 10, y = 12, t;
    double z = 20.5, u = 5.0, f;

    t = Area(x);    // Binds int Area(int, int = 10)
    cout << "Area = " << t << endl; // t = 100

    f = Area(z, y); // Binds double Area(double, double)
    cout << "Area = " << f << endl;  // f = 102.5

    return 0;
}
```

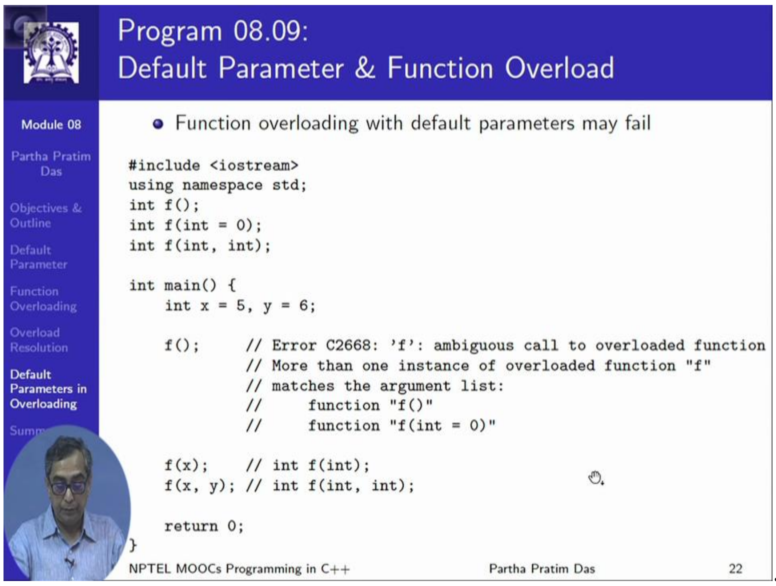
NPTEL MOOCs Programming in C++ Partha Pratim Das 21

So, now naturally you can use default parameters with function overloading. Some of the earlier examples, I have already used that. But, here I am just explicitly wanted to discuss it that I can, between the overloaded functions themselves, some parameters could be default. But, as long as you can resolve the whole set of functions with the default parameters from the call site, these usages are fine.

So, here is an example again here. So, the area function, we had this function, we had seen earlier. So, here both of them have two parameters. They are different types. And, in this case one parameter is defaulted. So, if we make this call, since, only this function can be called with one parameter. This will get resolved here. If we call this, then z is of type double and y is of type int. So, you will not be able to call the first function. We will call the second function because the first parameter matches as double. And, it is easier to convert an int to a double than converting a double to a int because if you convert a double to a int, you lose information.

If you convert int to double, then you do not lose information. So, that is the promotion strategy which will tell you that the bindings will happen as this will bind here and this will bind here. That is what we have shown at this point. So, bottom line is what we are trying to illustrate is default parameters and function overloading can be mixed together as long as the whole thing can be resolved. And, of course it can fail.

(Refer Slide Time: 20:25)



Program 08.09: Default Parameter & Function Overload

Module 08

Partha Pratim Das

Objectives & Outline

Default Parameter

Function Overloading

Overload Resolution

Default Parameters in Overloading

Summ

- Function overloading with default parameters may fail

```
#include <iostream>
using namespace std;
int f();
int f(int = 0);
int f(int, int);

int main() {
    int x = 5, y = 6;

    f();      // Error C2668: 'f': ambiguous call to overloaded function
              // More than one instance of overloaded function "f"
              // matches the argument list:
              //      function "f()"
              //      function "f(int = 0)"

    f(x);     // int f(int);
    f(x, y);  // int f(int, int);

    return 0;
}
```

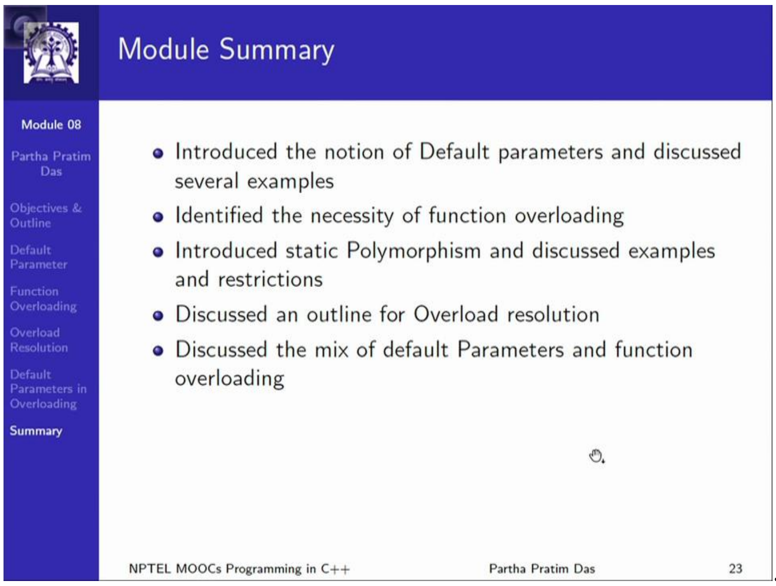
NPTEL MOOCs Programming in C++ Partha Pratim Das 22

For example, this is, again in some way we had seen this earlier. But again, just to specifically note that here are three functions which are overloaded; one this is a zero parameter, this is one parameter and this is two parameters. So, given that these three calls are certainly resolvable. It will, that in every case of the viable function set will have only one function each; will be able to do that. But, the moment you put a default parameter to one of this overloaded functions, you have a problem because now this function can have one parameter or zero parameter.

So, when I write a function call with zero parameter, your viable set will have both of these in place. And, certainly there are no parameters at all. There is no match that you can try out because if there is no parameter, then what type you are going to match between the type of a actual parameter and the type of the formal parameter. And therefore, the compiler cannot resolve. And, this is again from a typical compiler. This is the kind of error message that you will get. And, just see what the compiler is saying; is ambiguous call to overloaded function. And, it says that it cannot resolve between these two functions.

So, while you will use default parameters with overload, will have to remember that you cannot only look the overloading, nor you can just look at the default parameter. You have to look at the width default parameters, what are the different overloads that are getting created and whether those different overloads, whether they can be actually resolved by the outline of the mechanism that we have explained.

(Refer Slide Time: 22:17)



Module Summary

Module 08

Partha Pratim Das

Objectives & Outline

Default Parameter

Function Overloading

Overload Resolution

Default Parameters in Overloading

Summary

- Introduced the notion of Default parameters and discussed several examples
- Identified the necessity of function overloading
- Introduced static Polymorphism and discussed examples and restrictions
- Discussed an outline for Overload resolution
- Discussed the mix of default Parameters and function overloading

NPTEL MOOCs Programming in C++ Partha Pratim Das 23

So, this is; these are the features. So, at the end I would just like to sum up that in this module we have tried to address a major aspect of C++ language that is a function overloading. First, we have taken in look at the notion of default parameters, which later on we have explained. It is a special case of function overloading. Default parameters make it very easy to write functions with large number of parameters to provide their default values and make the function libraries easier to use. And, function overloading is another feature which allows us to write multiple functions with the same function name, as long as their parameter types differ.

So, we had looked into variety of nuances in this regard. And, we have talked of basic static polymorphism. And most importantly, we have also discussed the mechanism by which I mean, we have discussed the outline for the mechanism by which different functions can be resolved for the overload.

With this we will end this module. And, in the next one we will extend the same concept. And, see that similar overloading concept can be applied in case of operators also in C++, which will be known as operator overloading.