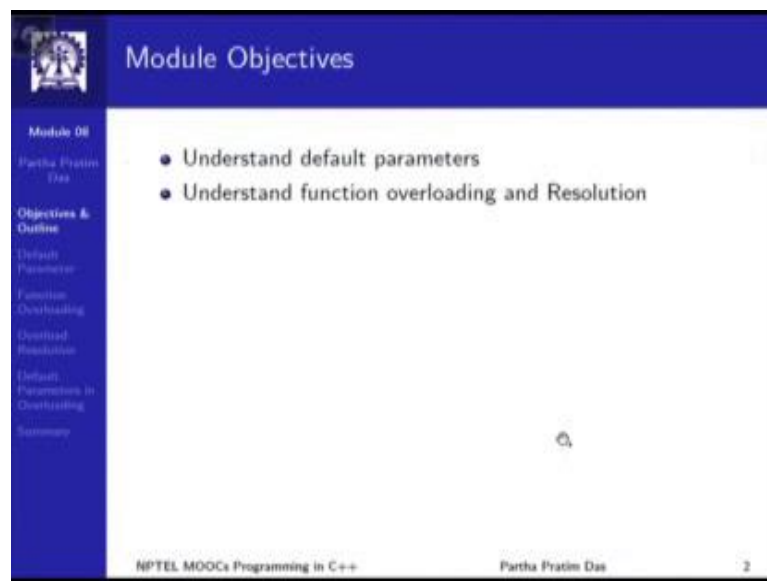


Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 12
Default Parameters and Function Overloading

Welcome to module 8 of Programming in C++. We have been doing the better C features of C++, we have already talked about Const and Volatile and Macros, Inline functions and also the reference variable and call by reference and return by reference mechanisms. These are all better C features of C++. We will now in this module 8, talk about Default Parameters and Function Overloading.

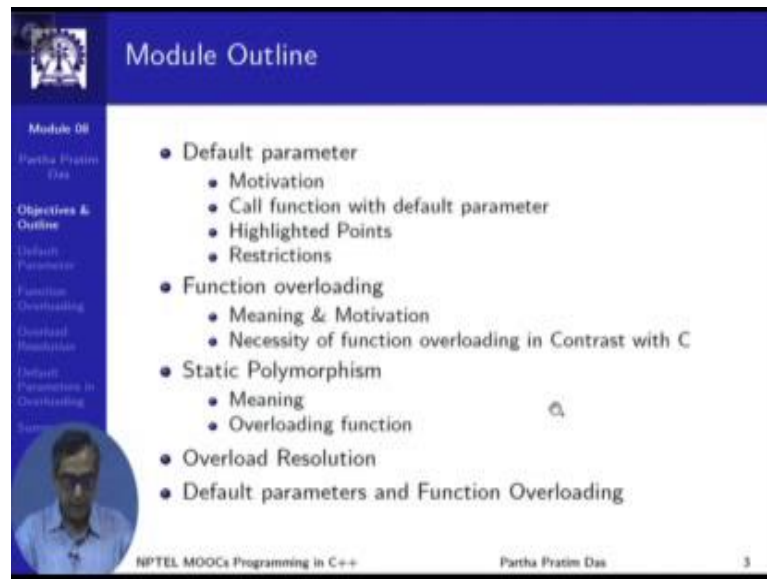
(Refer slide Time: 01:03)



The image shows a presentation slide with a blue header and a white main area. The header contains the text 'Module Objectives'. On the left side, there is a vertical navigation menu with the following items: 'Module 08', 'Partha Pratim Das', 'Objectives & Outline', 'Default Parameter', 'Function Overloading', 'Default Resolution', 'Default Parameters in Overloading', and 'Summary'. The main content area contains two bullet points: '• Understand default parameters' and '• Understand function overloading and Resolution'. At the bottom of the slide, there is a footer with the text 'NPTEL MOOCs Programming in C++', 'Partha Pratim Das', and the number '2'.

There are two very core very important features which go a long way in enabling the object oriented properties of C++ language.

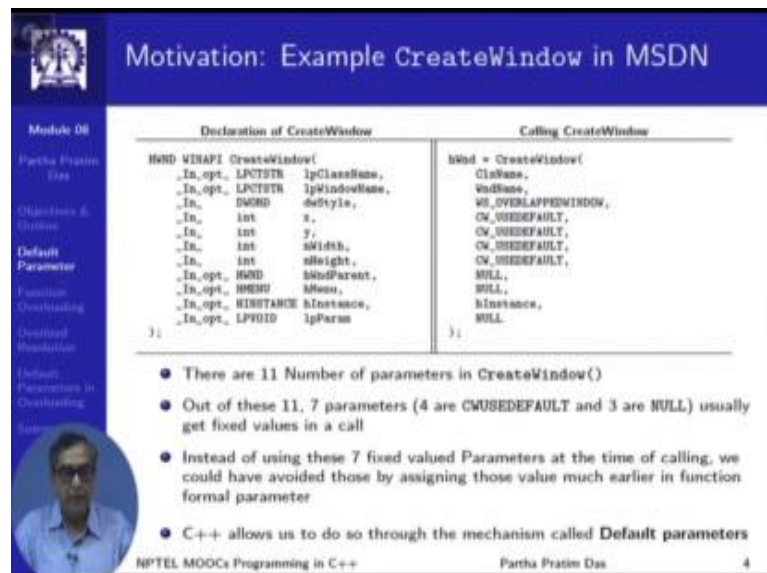
(Refer slide Time: 01:19)



The slide titled "Module Outline" features a blue header with the NPTEL logo and the text "Module Outline". On the left, a vertical sidebar lists navigation options: "Module 06", "Partha Pratim Das", "Objectives & Outline", "Default Parameter", "Function Overloading", "Overload Resolution", "Default Parameters in Overloading", and "Summary". A circular portrait of Partha Pratim Das is positioned below the sidebar. The main content area contains a bulleted list of topics: "Default parameter" (with sub-points: Motivation, Call function with default parameter, Highlighted Points, Restrictions), "Function overloading" (with sub-points: Meaning & Motivation, Necessity of function overloading in Contrast with C), "Static Polymorphism" (with sub-points: Meaning, Overloading function), "Overload Resolution", and "Default parameters and Function Overloading". The footer includes "NPTEL MOOCs Programming in C++", "Partha Pratim Das", and the page number "3".

This will be the overall outline for this module which you continue to see on the left panel of the presentation. So we now get started.

(Refer slide Time: 01:32)



The slide titled "Motivation: Example CreateWindow in MSDN" has a blue header with the NPTEL logo and the text "Motivation: Example CreateWindow in MSDN". The left sidebar is identical to the previous slide. The main content area is divided into two columns: "Declaration of CreateWindow" and "Calling CreateWindow".

Declaration of CreateWindow	Calling CreateWindow
<pre>HWND WINAPI CreateWindow(_In_opt_ LPCTSTR lpClassName, _In_opt_ LPCTSTR lpWindowName, _In_ DWORD dwStyle, _In_ int x, _In_ int y, _In_ int nWidth, _In_ int nHeight, _In_opt_ HWND hWndParent, _In_opt_ HMENU hMenu, _In_opt_ HINSTANCE hInstance, _In_opt_ LPVOID lpParam)</pre>	<pre>hWnd = CreateWindow(ClassName, winName, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, NULL, NULL, hInstance, NULL</pre>

Below the code, a bulleted list explains the motivation: "There are 11 Number of parameters in CreateWindow()", "Out of these 11, 7 parameters (4 are CWUSEDEFAULT and 3 are NULL) usually get fixed values in a call", "Instead of using these 7 fixed valued Parameters at the time of calling, we could have avoided those by assigning those value much earlier in function formal parameter", and "C++ allows us to do so through the mechanism called Default parameters". The footer includes "NPTEL MOOCs Programming in C++", "Partha Pratim Das", and the page number "4".

Now, first we will discuss about default parameters and I start with an example from C particularly this is an example from MSDN, the windows programming. If you are

familiar with a graphics programming and windows you would have seen this function, if you have not seen it, it really does not matter. All that I want to show on the left column is the prototype or header of the create window function. You can see the function has a large number of specifically 11 different parameters for its specification in the prototype.

When you need to create a graphics window you will need to specify all these 11 parameters to call the create window function. On the right column I show a typical create window call for creating window. Now, if I considered this particular function you can see two things; one is it has a large number of parameters, 11 parameters and to be able to use this you will typically need to know what these 11 parameters mean and you will need to specify those parameters.

Now, often you just want to create a sample window, you just what to window which has a default background color, default text color, you want it at a default center location in the monitor you wanted it to have a default width and height and so on. Given that you may not be necessary that you will specify all distinct values distinct for your applications in these 11 parameters. So to facilitate that if you looking to the actual call you will see that in the call there are several parameters like, if you look at the parameters here, these are parameters which are basically manifest constants a CW stands for create window use default. It is saying that there is some default value already defined in the library so you can use those values. So basically, you are not supplying those values.

Similarly, if you look into the window parent and the menu or if you looking into the window parameters we are just passing null which are basically again a kind of a default value. Agent stands, the instance of the window if you know window programming you would know is a instance of your application which also kind of gets default values so is the same with the type of window that you are creating that is you are creating a overlapped window and so on.

Even though these 11 parameters are there in the function and you need to specify all of them several of them are given default values by the library or a passed as null and you

actually need to specify few things like what is your window class which defines what a specific structure you are doing and possibly the name of the window to make this call. But yet, you need to need the full range of all eleven parameters to be actually written in the colon.

It would have been good that if you could arrange the parameters in a way so that only the parameters that you need to specify for your call need to be given at the call sight. If the system could understand and take the usually default parameters without your having to specify some default values for them, every time you make that call. So, C++ allows us to do something regarding this and that feature is known as Default Parameter.

(Refer slide Time: 05:50)

```
Program 08.01: Function with a default parameter

#include <iostream>
using namespace std;

int IdentityFunction(int a = 10) { // Default value for the parameter
    return (a);
}

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

    y = IdentityFunction(x); // Usual function call
    cout << "y = " << y << endl;

    y = IdentityFunction(); // Uses default parameter
    cout << "y = " << y << endl;
}

-----
y = 5
y = 10
```

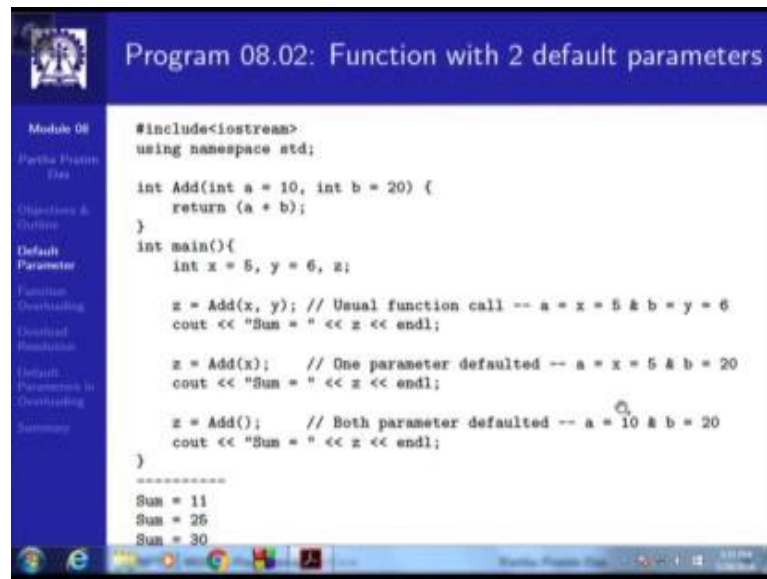
Now, here is an example to just illustrate what default parameters mean. Consider this a function, this is the functionality is not really very important we are calling this has a identity function that is it takes a parameter and simply returns the same parameter back. But what we are interested to highlight what you will need to observe here is a parameter has a value given in the signature of the function as kind of an initial value of the parameter a. Now, we understand what a initial value of a radiable with in a certain scope are what is a meaning of having a kind of initial or default for a parameter. To understand that let us, looking at the call, let us looking to the first call of the function.

First call of the function is usual we are calling the identity function with a actual parameter x which currently has a value 5, so we are passing that parameter here and therefore it will return the same value and as you output from this line and you will get first line of the output at this point. There is no surprise till this point.

Now, let us focus on the second call of the same function. You will be surprised to note that now I can call this function without passing any parameter. We have so far in C always known that the number of actual parameters and the number of formal parameters must match between the call and the prototype of the function. The actual parameters and the formal parameters must match in their order, in their data type and so on. But here, the function actually has one parameter, but when I call here I am not specifying that parameter. This is what the default parameter feature is. What the C++ does? Is since you have made the call without any parameter and since the function is already defined with a parameter value default value 10 here it will assume that you have as if called the function with this default value.

So, if you default a parameter then you actually get two options; one as you get here where you can make the function call exactly in the way you have been making function calls earlier specifying if actual parameter, or you can make a choice using the default value of the parameter and not need to specify an actual parameter you can just skip that the system will understand that you are using the parameter and specify the value which is given in the default in the function deceleration itself. So this is the basic feature of default parameter. Let us now proceed and take a look into another example.

(Refer slide Time: 09:13)



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

int Add(int a = 10, int b = 20) {
    return (a + b);
}

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

    z = Add(x, y); // Usual function call -- a = x = 5 & b = y = 6
    cout << "Sum = " << z << endl;

    z = Add(x);   // One parameter defaulted -- a = x = 5 & b = 20
    cout << "Sum = " << z << endl;

    z = Add();   // Both parameter defaulted -- a = 10 & b = 20
    cout << "Sum = " << z << endl;
}

-----
Sum = 11
Sum = 26
Sum = 30
```

In the last example we showed one function which has a default parameter and now we are showing that it is not necessary that you will have only one default parameter you can actually have up to default parameters or any arbitrary number of default parameters also. We just show a function here, please do not become concerned about the functionality of this the algorithm for this simply take two parameters adds them and returns them just look at the definition of this function here, and it has two parameters both int a and int b. And what we have done is each of the parameters have been defaulted with an initial value. So with that when we use this function and make a call if you look at the first call here, the first call is a usual function call, x and y are two actual parameters. We call add x y, so x corresponds to a gets copied there it call by value you can see y is copied to b and function is called with the values of x and y that is 5 and 6 and therefore when you print the sum here we get the first output that is the sum turns out to be 11.

Now, look at the second call of the same function. In the second call of this function, we provide the first parameter x but we have not provided the second parameter, which means that though the function actually had two parameters we are calling it with one parameter. What will happen? This one parameter that we have provided that will correspond to x will correspond to the formal parameter a, so whatever is the value of x

will get copied in the call by value to the formal parameter a. But the second parameter has not been provided, so the system will understand that you are using the default value of the second parameter, that is the second parameter will be taken to be 20 because a default value is given in the second parameter is 20. Therefore, in this case the function will proceed with a being 5 and b being 10 and it will compute and you can see from the output a plus b will compute to be 25 and that is what will be printed.

We can extend this further. This is what we do in the third instance of the call where we have not provided any actual parameter. So, the two parameters of the function a, and b both are taken their default values that a is 10 and b is 20 and the function here it is not really visible, but the function here prints 30 as the some result. We can see that it is not necessary that I should default only one parameter, but it possible that I can default an arbitrary number of parameters in a function. And this is what will be very useful in writing functions particular with large number of parameters where a good set of parameters may often take a default value to work with.

(Refer slide Time: 12:56)

The slide is titled "Default Parameter: Highlighted Points" and features a blue header with a logo on the left. A sidebar on the left lists navigation options: "Module 08", "C++ Primer", "Day", "Objectives & Outline", "Default Parameter", "Function Overloading", "Operator Overloading", "Default Parameters in Overloading", and "Summary". The main content area contains a bulleted list of points:

- C++ allows programmer to assign default values to the function parameters
- Default values are specified while prototyping the function
- Default parameters are required while calling functions with fewer arguments or without any argument
- Better to use default value for less used parameters
- Default arguments may be expressions also

Handwritten in red ink at the bottom of the slide are two function prototypes: `void f(int = 2+3);` and `void f(int = 2+n);`. The second prototype has an arrow pointing from the `n` to the `=` sign, indicating that the default value is an expression.

We will highlight the basic points about default parameters that is C++ allows programmers to assign default values to the function parameters, you have seen this. Default values are specified while prototyping the function. Prototyping the function that

is what is meant is you write the default values in the function signature as we have seen. Default parameters are required while calling the function with fewer arguments or without any argument, so we have seen how one or two parameters as been used with their default value when they were missing in the call site.

As a practice certainly it cannot be said as to which parameters should be default and which parameters should be not default. But as a practice it will be good to use default values for less used parameters, while you can actually use default value for all the parameters if you so desire. The default arguments, when you call the function the default arguments or even the default values can also be expression as long as they can be computed at the compilation time.

I can have a default values so I can have a default value like 2 plus 3 for as a default parameters. This will be valid for a function I can have a default value like 2 plus 3 what their compiler will do it will compute 2 plus 3 while it is compiling and it is said that default value as 5. But please note that I cannot have a default value which is actually an expression at the time of compilation that is I cannot have something like say 2 plus n unless n is defined to be a constant value, constant integer. Because if n is variable then the compiler does not know at the time of compile your program as what default value are you providing.

(Refer slide Time: 15:18)

Restrictions on default parameters

- All parameters to the right of a parameter with default argument must have default arguments (function `f`)
- Default arguments cannot be re-defined (function `g`)
- All non-defaulted parameters needed in a call (call `g()`)

```
#include <iostream>

void f(int, double = 0.0, char *);
// Error C2548: 'f': missing default parameter for parameter 3

void g(int, double = 0, char * = NULL); // OK
void g(int, double = 1, char * = NULL);
// Error C2672: 'g': redefinition of default parameter : parameter 3
// Error C2672: 'g': redefinition of default parameter : parameter 2

int main() {
    int i = 5; double d = 1.2; char c = 'b';

    g(); // Error C2660: 'g': function does not take 0 arguments
    g(1);
    g(1, d);
    g(1, d, &c);
    return 0;
}
```

Now, next we will try to see that is it that I can default the parameters of a function in any way that I desire, or there are some basic rules or restrictions that we will need to follow. We discuss some basic rules of a default parameters. The first rule is very important it says that all parameters to the right of a parameter with default argument must all have default arguments.

For this rule, look at this particular function. We are talking about function `f` which has three parameters, `int`, `double`, and pointer to character. In this case we have provided a default value to the second parameter, why we have not provided any default value to the third parameter which is `char star`. If you do that then the compiler will give you an error and in this case I have shown the error from one compiler, which says that `f` has missing default parameter for parameter three that is for this parameter.

So, it says that if I write any function and if I have one parameter, two parameter, three parameter, and if some parameter is defaulted then all parameters that I write after this default parameter all of them once this is the defaulted the next one has to be defaulted the next one will also have to be defaulted all of them till the end we will have to be defaulted. Otherwise, compiler will not be able to dissolve as to which parameters you are actually using. So that is the first rule, that once we default a parameter to it is right

all parameters that exist must get default values

Second rule as in here, is default parameters cannot be redefined. You look at say these two lines. Here, first in the first of this we have defined a function f which has three parameters again, int, double and char star, and two parameters have been defaulted the second parameter to 0 and third parameter to a null pointer, this is naturally. Later on we are again talking about that same function prototype g, but if you look in to these two values we are now using a different default value for the second parameter. So this is not permitted, you can default the parameter value of a function only once.

Once you have defaulted then you cannot again specify the default parameter value. Now you think that this is a the ambiguity is from the fact that since you have defaulted double two zero and then you have defaulted double two one if you looking to this carefully that you have used two default values and that is the reason the compiler is giving this error which is shown here as parameter error on the parameter two, but incidentally compiler actually does not look at the actual value of the default that you have provided.

To understand this, please focus your attention to the default value of the third parameter. Here the third parameter was initially in the first case was given to be null and the second case it is redefined to null again so you are actually using the same default value in the next time. But even then the compiler gets confused and will tell you that the default parameter for parameter three has been redefined, it was already defined and it is being defined again and this will not be allowed. It does not matter whether you are redefining a default value by the same value that you are defined earlier or your are defining a default value using a different value that you are used earlier in both cases this will not be permitted. So default arguments cannot be redefined for functions were you are using the default parameters, this is the rule number two.

(Refer slide Time: 20:22)

Restrictions on default parameters

- All parameters to the right of a parameter with default argument must have default arguments (function f)
- Default arguments cannot be re-defined (function g)
- All non-defaulted parameters needed in a call (call g())

```
#include <iostream>

void f(int, double = 0.0, char *);
// Error C2648: 'f': missing default parameter for parameter 3

void g(int, double = 0, char * = NULL); // OK
void g(int, double = 1, char * = NULL);
// Error C2672: 'g': redefinition of default parameter : parameter 3
// Error C2672: 'g': redefinition of default parameter : parameter 2

int main() {
    int i = 5; double d = 1.2; char c = 'b';

    g(); // Error C2660: 'g': function does not take 0 arguments
    g(i);
    g(i, d);
    g(i, d, &c);
    return 0;
}
```

Handwritten red annotations on the slide include: a circle around the double parameter in the function signature of g, a red arrow pointing to the g() call with the text "g(i, &c)", and a red "X" over the g(i, &c) call with the text "double = 0.0".

The third rule which says that since you are using default parameters you can choose to specify them at the time of a call or you may not specify them at the time of a call. So, if we look in the context of function g, let say this a particular definition which is a certainly a correct definition then we can see that these three calls are valid that is I can call the function with one int parameter g i which will you be taken as a first or I can call it with the first two parameters i and d which is int and double or I can call it with all the three parameters. All three of them are working. Suppose, if you try to call the function g without any parameter then that will be an error, because if you have to call this function without any parameter then the first parameter should have been defaulted which in this case has not been done.

So, it says that if some parameters of a function are given default values then at least all the non default parameters must be specified as actual parameters and the default parameters in case of the default parameters you have a choice and have you can specify none of them or some of them, but if you specify some of them then you will always have to follow the rule that is you will have to specify them in the left to right order. What I mean is suppose here, I am specifying if you look into this particular specification let me clean up and show again, suppose you are looking at this call here the function has three parameters but we are specifying this two. So the first of them i

here is certainly non default parameter which is mandatorily required and the second parameter d that you specify is taken to be double.

Now, suppose you tried to do a call like `i ampersand c` thinking that this call will mean that `i` will go for the `int`, this will forgo for the third parameter `char star` and the double value in the middle will be taken as `0.0`. This will not be possible, this is an error the reason is the compiler has no way to know that between the two default parameters double and char star which one you are specifying here and which one you are specifying here. What it has to assume that if you are using these parameters then you can use the default values in a particular order that is the in call the parameters actually given has to match from left to right it is only after a point you can skip all the parameters to the right of that formal parameter, but not skip something from the middle. So, these are some of the basic rules that the default parameters have to follow.

(Refer slide Time: 24:03)

Restrictions on default parameters

- Default parameters should be supplied only in a header file and not in the definition of a function

```
// Header file: myFunc.h
void g(int, double, char = 'a');
```

```
// Source File: myFunc.cpp
#include <iostream>
using namespace std;
#include "myFunc.h"

void g(int i, double d, char c) {
    cout << i << " " << d << " " << c << endl;
}
```

```
// Application File: App.cpp
#include <iostream>
#include "myFunc.h"
// void g(int, double, char 'a');
void g(int i, double f = 0.0, char ch); // OK a new overload
void g(int i = 0, double f, char ch); // OK a new overload
int main() {
    int i = 5; double d = 1.2; char c = 'b';

    g(); // Prints: 0 0 a
    g(1); // Prints: 5 0 a
    g(1, d); // Prints: 5 1.2 a
    g(1, d, c); // Prints: 5 1.2 b
}
```

Handwritten red annotations:
- A red arrow points from the function signature in the header file to the function definition in the source file.
- A red circle highlights the function signature in the application file.
- A red bracket groups the two function declarations in the application file.
- A red note says: `g(int, double = 0.0, char = 'a');`

Here, we show another, this is not exactly a rule by the definition of the language but I just show this as the way the practice of how you should be doing using the default parameters. So, he is saying that default parameters should be supplied only in a header file and not in the definition of the function, that is if you are doing a default parameters then this is as if you have a header file where you have specify a default this is the

definition file the source file where you have defined the function it have provided the body for that function. So what we are saying is you should never specify any default parameter here or some are else where you are possibly using this function let say here.

Now, to understand let us see what is happening, this function has been define the prototype is given in the header and this is an application code this one is an application code which is using that function so it includes that header which means this particular function definition, this particular function prototype which has been included. Now suppose in the body of your application you want to write something like this, surprisingly this all will be accepted these are all valid things. So what you are saying here is, you are saying that in from the header the third parameter was already defaulted, now you are saying my second parameter is also defaulted so now I have as if my function g, so these two together actually means that my function g has two of it is parameters defaulted and so on.

Similarly, in the third case here we show that even the first parameter has been defaulted. The default parameter feature allows you to do this, but the difficulty is while you are using you will not know which a part of the default definition is in the header file and part of the default definitions are in your source file, implementation file. So you will not know at any single point as to what parameters are defaulted and to which values they are defaulted. So all defaults that are to be used if you have to do something like this all of them should be moved to the header files so that at one point you can see what default parameters exist and what are their values otherwise this mechanism of this structure of writing the code with default parameters can get really very confusing.

This is a restriction from practice point of view language does allow it so if you look at the end of I have shown that with this definitions what you can achieve you can actually now call the function g with four different forms because all three parameters eventually have been defaulted, but it would always be better to default them at one place so that in one signature itself instead of defaulting them successively in three different signatures split between multiple files this will become very confusing for the purpose of use.

(Refer slide Time: 27:57)

Function overloads: Matrix Multiplication in C

- Similar functions with different data types & algorithms

```
typedef struct { int data[10][10]; } Mat; // 2D Matrix
typedef struct { int data[1][10]; } VecRow; // Row Vector
typedef struct { int data[10][1]; } VecCol; // Column Vector

void Multiply_M_M (Mat a, Mat b, Mat* c) { /* c = a * b */ }
void Multiply_R_VC (Mat a, VecCol b, VecCol* c) { /* c = a * b */ }
void Multiply_VR_M (VecRow a, Mat b, VecRow* c) { /* c = a * b */ }
void Multiply_VC_VR (VecCol a, VecRow b, Mat* c) { /* c = a * b */ }
void Multiply_VR_VC (VecRow a, VecCol b, int* c) { /* c = a * b */ }
```

```
int main() {
    Mat m1, m2, m; VecRow vr, vr1; VecCol vc, vc1; int r;
    Multiply_M_M (m1, m2, &m); // m <-- m1 * m2
    Multiply_R_VC (m1, vc, &vc1); // vc1 <-- m1 * vc
    Multiply_VR_M (vr, m2, &vr1); // vr1 <-- vr * m2
    Multiply_VC_VR (vc, vr, &m); // m <-- vc * vr
    Multiply_VR_VC (vr, vc, &r); // r <-- vr * vc
    return 0;
}
```

- 5 multiplication functions share same functionality but different argument types
- C treats them as 5 separate functions
- C++ has an elegant solution

NPTEL MOOCs Programming in C++ Partha Pratim Das 10

So, we have seen the default parameters feature of a C++ and we have seen how to use it and what are the restrictions for using that?