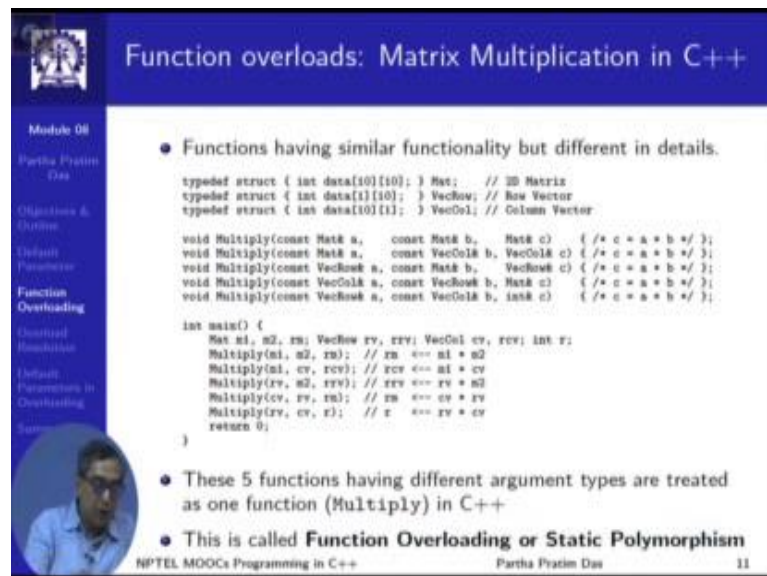


Programming in C++
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 13
Default Parameters and Function Overloading (Contd.)

Welcome to Module 8 of Programming in C++. We have been discussing on this module in the earlier part and we talked about default parameters. Now, we will start discussing the function overloading.

(Refer Slide Time: 00:40)



Function overloads: Matrix Multiplication in C++

- Functions having similar functionality but different in details.

```
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(const Mat& a, const Mat& b, Mat& c) { /* c = a * b */; }
void Multiply(const Mat& a, const VecCol& b, VecCol& c) { /* c = a * b */; }
void Multiply(const VecRow& a, const Mat& b, VecRow& c) { /* c = a * b */; }
void Multiply(const VecCol& a, const VecRow& b, Mat& c) { /* c = a * b */; }
void Multiply(const VecRow& a, const VecCol& b, int& c) { /* c = a * b */; }

int main() {
    Mat m1, m2, r1; VecRow rv, rrv; VecCol cv, rrv; int r;
    Multiply(m1, m2, r1); // r1 <== m1 * m2
    Multiply(m1, cv, rrv); // rrv <== m1 * cv
    Multiply(rv, m2, rrv); // rrv <== rv * m2
    Multiply(cv, rv, r1); // r1 <== cv * rv
    Multiply(rv, cv, r); // r <== rv * cv
    return 0;
}
```

- These 5 functions having different argument types are treated as one function (Multiply) in C++
- This is called **Function Overloading** or **Static Polymorphism**

NPTEL MOOCs Programming in C++ Partha Pratim Das 11

To understand the function overloading, we will first take an example to illustrate why at all something like a function overloading is required and what it could possibly mean. So, here is an example in C. It is an example to multiply matrices multiply vectors with matrices or multiply vectors themselves. So, if you look little carefully into the definitions given here, first we define three types, these are alias. Matrix is a two-dimensional square matrix that we have taken here in this example. Then, the second is vec row. In this vec row is a row vector and vec col is a column vector. So, you can understand these. These you can look at this.

So, these are the kit types and then, what we want is, wherever it is defined by the rules of matrix multiplication, we want to write function to multiply them. The first function multiplies two matrices and returns the result matrix. You can easily see that mat is a square matrix A, mat B is another square matrix and both of them have size 10. So, if I multiply them, I will get another square matrix C as a result by the rules and you understand that since we are writing this in C, the call by value we need to use pointers to get the output from this function.

Now, suppose I also want to define a multiplication which is the second one which is between a square matrix A and a column vector B. If I do that, naturally I will get a column vector C. Now, if we can recap on the rules by which we multiply matrices, the row column mechanism which all of you know and I do not need to elaborate that here. You will know that there is hardly any difference between multiplying two matrices or multiplying a matrix or rather post multiplying a square matrix with a column vector of the same size. Only difference being your result will be a column vector and if you proceed, if we look into the third, it is a pre-multiplication of a matrix by a row vector.

If you look into the fourth, it is multiplication of a column vector by a row vector where the result turns to be a square matrix and in the fifth, you get C, the multiplication of a row vector by a column vector which turns out to be a single value. Now, all of this follows the same algorithm for multiplication, but if I want to express them, code them in C language, I will need to provide all different names. If you look into this part, I need to provide all different names to these functions because they are different functions. They take different kinds of, they all take three arguments because conceptually they are taking A and B matrices and multiplying and giving you a result C.

But since these are all different types, I need to have different names given to this function and I will need to remember not only later on when I am using, I not only need to re-understand what different parameters I am passing, but corresponding to that RV is a row vector M2 is a matrix. So, I have to remember that if I am using a row vector and a matrix by name of the function has to be multiplied underscore VR underscore M or something similar that is whereas if I am doing multiplication of a row vector by a column vector, then the name of the function has to be something different VRVC.

So, if you summarize these observations, so what you understand is five multiplication functions that we show here, share the same functionality. They have same algorithm, but just they have five different kinds of argument types and result types and consequently C needs to treat them as five different separate functions. Now, C++ fortunately has an elegant solution for this situation, where different functions which have similar functionality may be they use the same algorithm or maybe they use slightly different variations of the same algorithm, but certainly has to deal with different kinds of data types in the argument can now share their function name and can make the whole programming exercise lot easier.

So, to start with continuing with the same example, now this is again coded in C++. Just look at the major differences. The first if you look into the type def, they are identical. We are dealing with the same types. If you look into the five functions, we have the same five functions so far as the functionality is concerned that is the first one still multiplies two matrices, the second one multiplies a matrix, post multiplies a matrix with a column vector and so on, but if you look into the names, you see something very different. You see that all of them have the same name, something that you could not have done in C.

In C we formally say that function names are global. These are global symbols. So, in the whole of C program, irrespective of how many files in which you write and so on, but in the whole of the program that will run as one unit from call to one main, every function name has to be distinct, every function name. There can be only one function name for every different function that you write for the program, but here you can see that we have five different functions sharing the same name, but of course when you look into their parameters like in here, all the parameters you will find that no two of them have the same set of parameters.

They do not actually need to because what distinguishes the first multiply which multiplies two matrices and gives through another square matrix from the second function which post multiplies a matrix with a column vector to give a column vector or the third one which pre multiplies a matrix with a row vector to give a row vector. All of them have various different, their corresponding different data types for the arguments.

Now added with that the small thing that we have done is what we have learnt in module 6 and 7 that we have learnt to use the call by reference matrices at big objects. So, you would not like to pass them as value and copy them, rather we would like to pass them as reference and to make sure that the input matrices or vectors are not tempered within the function. We make those parameters constant whereas, on the output side we use just the call by reference mechanisms, so that we do not need that indirect pointer part, but this is just a matter of detail with the points to note is all these five have the same name and all these five can be used by the respective actual parameters with the same name.

So, what happens is suppose I am trying to do a post multiplication of a matrix with a column vector, so I have a matrix M1, I have a column vector CV and if I multiply them, what should I get? I should get a column vector. So, let say I am expecting the column vector RCV as a result. So, what I say I just write multiply; I put M1 as the matrix, CV as a column vector and RCV as my result matrix. Now, somehow this means that if I do this and I must call that particular function appropriately which multiplies, post multiplies a square matrix with a column vector.

So, very interestingly if you specify these parameters in this function, then interestingly this will actually call this function. Though there are five functions, all of them called as multiply and I am calling this function also as multiply, but somehow I will be able to understand given the parameters, given the types of the parameters that I am actually interested in the second function with whom the types of these parameters, the type of this parameter is mat which matches.


Here the type of the second parameter is parameter CV is vec call which matches here and type of the third parameter RCV is again vec call which matches here. We find that this is only function out of the 5, where these 3 parameters match in their type and the compiler would be accordingly able to call this the second function here from this particular invocation.

Similarly, if I look into this, then depending on the types of these 3 parameters, the compiler would be able to actually call this function because RV is vec row, R CV is vec call and R is int RVCVR vec row vec call int will only specifically call the last function

which is exactly what it should exactly work. So, these 5 functions have different argument types, but they are treated as one function having a common name in C++ and this feature is very powerful which will enable us to do lot of things called function overloading or as we will slowly understand an alternate name, more formal name for it is static polymorphism. So, we just saw the motivation of y.

This function overloading would be easy. We saw that in situation where we had to give five different function names to five different functions while their core functionality, core algorithm are still same and we had to remember all these names distinctly, we in C++ we can use function overloading to just use the same name and depending on our collar or use of the function, the appropriate function will get called appropriate function will be bound to the particular call we have seen that. So, now we go for what and see how we can do different kinds of function overloading in C++.

(Refer Slide Time: 13:27)


Program 08.03/04: Function Overloading

Module 08
 Partha Pratim Das
 Questions & Answers
 Default Parameters
 Function Overloading
 Operator Overloading
 Default Parameters in Overloading
 Summary

- Define multiple functions having the same name
- Binding happens at compile time

Same # of Parameters	Different # of Parameters
<pre>#include <iostream> using namespace std; int Add(int a, int b) { return (a + b); } double Add(double c, double d) { return (c + d); } int main() { int x = 5, y = 6, z; z = Add(x, y); // int Add(int, int) cout << "int sum = " << z; double u = 3.5, t = 4.25, u; u = Add(u, t); // double Add(double, double) cout << "double sum = " << u << endl; return 0; }</pre>	<pre>#include <iostream> using namespace std; int Area(int a, int b) { return (a * b); } int Area(int c) { return (c * c); } int main(){ int x = 10, y = 12, z = 5, t; t = Area(x, y); // int Add(int, int) cout << "Area of Rectangle = " << t; int z = 5, u; u = Area(z); // int Add(int) cout << " Area of Square = " << u << endl; return 0; }</pre>
<pre>int sum = 11 double sum = 7.75</pre> <ul style="list-style-type: none"> Same Add function Same # of parameters but different types 	<pre>Area of Rectangle = 12 Area of Square = 25</pre> <ul style="list-style-type: none"> Same Area function Different # of parameters

NPTEL MOOCs Programming in C++
Partha Pratim Das
12

So, in function overloading, we define multiple functions having the same name because unless you have more than one function having the same name, that issue of overloading does not arise and the second is the binding happens at compile time binding is a formal term to say that given the function call, how do you decide which particular function will

actually be called and that process is known as binding. This is what the compiler does. So, this is called the binding compile type.

(Refer Slide Time: 14:13)

Program 08.03/04: Function Overloading

- Define multiple functions having the same name
- Binding happens at compile time

Same # of Parameters	Different # of Parameters
<pre>#include <iostream> using namespace std; int Add(int a, int b) { return (a + b); } double Add(double c, double d) { return (c + d); } int main() { int x = 5, y = 6, z; z = Add(x, y); // int Add(int, int) cout << "int sum = " << z; double s = 3.5, t = 4.25, u; u = Add(s, t); // double Add(double, double) cout << "double sum = " << u << endl; return 0; }</pre>	<pre>#include <iostream> using namespace std; int Area(int a, int b) { return (a * b); } int Area(int c) { return (c * c); } int main(){ int x = 10, y = 12, z = 5, t; t = Area(x, y); // int Add(int, int) cout << "Area of Rectangle = " << t; int z = 5, u; u = Area(z); // int Add(int) cout << "Area of Square = " << u << endl; return 0; }</pre>
<pre>int sum = 11 double sum = 7.75</pre>	<pre>Area of Rectangle = 12 Area of Square = 25</pre>

• Same Add function
• Same # of parameters but different types

• Same Area function
• Different # of parameters

Now, let us look into this. On the two columns, we will show two different kinds of instances here. We show the overloading of a function at where the first instance here, you just look at specifically focus on the parameters. The add takes two integer parameters and returns an integer in the second case, it takes two double parameters and returns a double. So, we are trying to write an add function which conceptually should add two numbers, but in C it does matter or in C++ also, it does matter as to whether the numbers it is adding are int or they are double or they are something else.

Now, I can write these two functions together in my same code and then, I am using it at two different places here. I am using it add xy, where x is an int and y is an int. So, basically I am using the calling add with two int parameters. Therefore, this call will get bound to this definition of add which is what works for two int parameters whereas, if you look into the second call of the add function which uses s and t of time double, our parameter types are double and double the compiler will figure out that this call is actually for the second definition of the add function and will bind it there.

Accordingly the first one will print a sum 11 and the second call will print a sum 7.75. As we all know that it does matter for addition has to what type of data you are using because if I am adding two integers, I have one kind of addition. If I am adding doubles, I have a different kind of addition. So, in this part we have seen that add the function add has in both cases. The same number of parameters, but the types of the parameters are different and based on that we are able to resolve as to what a particular call is talking about, what a particular from the call, what particular function of the different alternatives that have been overloaded needs to be actually invoked, need to be actually bound. Now, it is not necessary for overloading.

So far all the examples we have seen either the five multiply functions or the add functions here, the number of parameters have been same in all cases, but this is not mandatory for function overloading. So, look at the right column.

The next example here we show two functions area. The first one is meant to compute the area of a rectangle. So, it has to take two parameters of width and heights multiply them and written that whereas, the second one takes only one parameter because it is supposed to compute the area of a square. So, you just take that and squares that number and written it. So, between these two functions, the name is same. This has two parameters whereas, this as one parameter and we will see below that we can still work with this given if you call area xy, then we know that there are one function.

One of these function, this one has two parameters, the parameter x as here is int, parameter y is also int. So, it is a int int call. So, two parameters, both of them should int. They are indeed int here. So, this call will actually invoke the first area function.

In contrast if we look into the second call, there is one parameter z which is of type int. This will get bound to the second area function which has one parameter. So, appropriately different functions as overloaded will get called even when there are different numbers of parameters. Note that though the numbers of parameters are different, in terms of the first parameter here the type is a same.

In the earlier cases, it is the number that could give you the number of parameters where same in the earlier cases. Just the different types were giving you the clue in terms of which is the right function to call here. The numbers are different even though the type is overlapping. You still are able to resolve, were still able to bind the right function for the right call. So, when you talk about function overloading, we are basically talking about situations where there is more than one function which needs to be used and these functions are somewhat related in their functionality.

Now, certainly you will not overload a function to do square root with the function to sort ten numbers. I mean you could always call them that let say this I use a name my func and I overload my func. If the parameter is double, then it will find square root and if the parameter to my func is an array and then number of the elements of the array and then it would sort. It is according to function overloading rules, you will be able to do that, but certainly that will be a disaster use of this feature.

So, when we overload the basic conceptualization is that all overloaded functions must have very related functionality. They should be talking about very similar algorithms, but the types of the parameters that they use, number of parameters that they use for doing the computation has to be distinct, so that I can give the same name to all of these functions and that is mechanism by which function overloading works.

(Refer Slide Time: 21:16)

Program 08.05: Restrictions in Function Overloading

- Two functions having the same signature but different return types cannot be overloaded

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

int Area(int a, int b) { return (a + b); }
double Area(int a, int b) { return (a + b); }
// Error C2666: 'double Area(int,int)': overloaded function differs only by return type
// from 'int Area(int,int)'
// Error C3871: 'Area': redefinition; different basic types

int main() {
    int x = 10, y = 12, a = 5, r;
    double f;

    t = Area(x, y);
    // Error C2668: '=': unable to resolve function overload
    // Error C3861: 'Area': identifier not found

    cout << "Multiplication = " << t << endl;

    f = Area(y, x); // Errors C2668 and C3861 as above
    cout << "Multiplication = " << f << endl;

    return 0;
}
```

NPTEL MOOCs Programming in C++ Partha Pratim Das 13

In C++, there are some restrictions in terms of function overloading function. Overloading is primarily decided based on the signature of the function as we have discussed that in every time case. We are saying there are in C, how you resolve between two functions. They must have different names, different functions. In C++, two functions or more than two functions can have the same name. How do you resolve? You resolve based on their signature, based on the number of parameters, based on the types of the parameters we have seen.

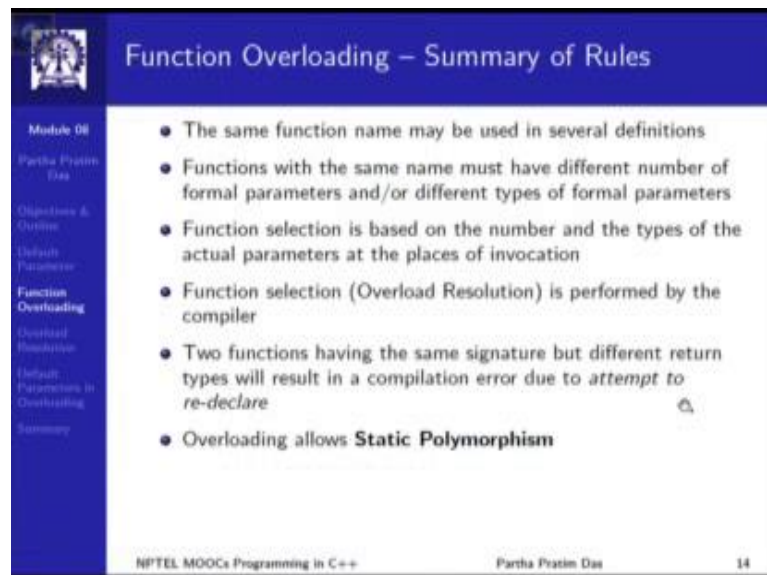
Now, the question is if I have two functions which have let us check this particular case here, I have two functions. Again area one of them, both of them use the same number of parameters to both of them use the same types of parameters int int and int int, but their return types are different. Now, please note such kind of functions cannot be used together. This is not possible in terms of function overloading.

So, perhaps this shown here, this is one typical error message from a compiler showing you that here it is saying it takes the first function that is when you tries to compile, the second function when tries to look into that, it says that is why it is talking about the second function which this double return type says overloaded function differs only by return type. It differs only by return type and that is not what is allowed and that is why

this is considered to be a redefinition as if you are redefining the same function as if you would have given this if you would have compiled this example with C.

You would also find a very similar error message because this is a redefinition message. The first message is what is typical from the function overloading point of view that you cannot overload two functions unless their number of parameters differs or the number of parameters has come, but still the types of the parameters are different. At least for one parameter if they are just if their signatures are just different in the return type, then that overloading is not permitted.

(Refer Slide Time: 23:56)



The slide is titled "Function Overloading – Summary of Rules" and features a blue header and a dark blue sidebar on the left. The sidebar contains a navigation menu with the following items: "Module 04", "Partha Pratin Das", "Objectives & Outline", "Default Parameter", "Function Overloading" (highlighted in white), "Default Parameter in Overloading", and "Summary". The main content area is white and contains a bulleted list of six rules. The footer of the slide includes "NPTEL MOOCs Programming in C++", "Partha Pratin Das", and the number "14".

- The same function name may be used in several definitions
- Functions with the same name must have different number of formal parameters and/or different types of formal parameters
- Function selection is based on the number and the types of the actual parameters at the places of invocation
- Function selection (Overload Resolution) is performed by the compiler
- Two functions having the same signature but different return types will result in a compilation error due to *attempt to re-declare*
- Overloading allows **Static Polymorphism**

So, please keep this restriction in mind while you deal with, while you write overloaded functions to sum up in terms of the rules that we have for function overloading will say that the same function name may be used in several definitions. That is a basic overloading concept. The function with the same name must have different number of formal parameters or different types of formal parameters we have seen examples of both. Function selection is based on the number and types of the actual parameters. We have seen examples here as well.

Then, this function selection as it is done by the compiler is known as overload resolution. What it means that you have multiple functions overloaded. Multiple functions by the same name and you are given one call site; you are given one call where you are trying to figure out which of this multiple candidates should be used. That process is called process of resolution for the overloading, something that C++ compiler performs and overtime you will understand that it is actually extremely complex process and extremely powerful process and by that process, the compiler tries to decide which particular function of the given overloaded versions you are going to use or you intend to use in the call site if the compiler feels to do that.

If two functions have the same signature but different return types or for some other reason the parameter types being different, but not distinct enough for the compiler to resolve. If compiler fails to do that, then the compiler will tell you that you are attempting to re-declare or you would tell that I am confused and will say that there is ambiguity in that and in such cases, you will have to do something different to make the function overloading work or avoid function overloading and to write the code in a different form.

Overloading in a form gives you static polymorphism. We are not immediately going into depths of static polymorphism. We will talk about polymorphism later in the context of various C++ features. The code features here I would just like to explain the two words.

Polymorphism means, poly means many and morph means to change. So, polymorphism is accommodating multiple changes or multiple forms into one. So, here the polymorphism means that the same function by name, but it is depending on the parameter type, different behavior, different algorithms and different interfaces. So, you are trying to basically decide on this multiple form and then, you say that this is static. What static here means that you want to do this all in the compile type that is the compiler should be able to decide between these multiple forms and tell you exactly which of these different forms, polymorphic forms you are trying to use at your function call. So, a mind together this kind of decision or overload mechanism is known as static polymorphism.

Of course, there are other forms of polymorphism which are typically called dynamic or run type polymorphism which we will discuss after when we get into discussing the object base part of the C++ language.