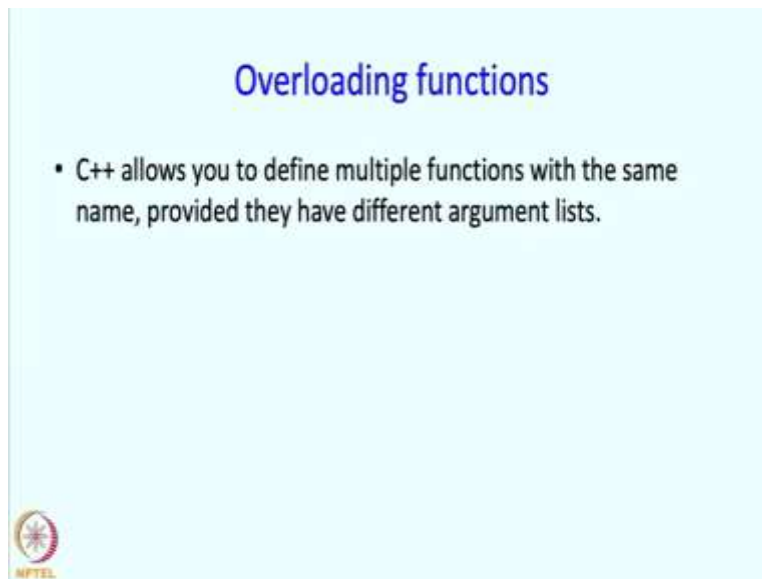


An Introduction to Programming through C++
Professor. Abhiram G. Ranade
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay India.
Lecture 14

Advanced Features of Functions: Function overloading and lecture conclusion


Welcome back, in the last segment we discussed how default values can be given to some parameters in a function that we are defining. In the next segment we are going to discuss another feature which can make it easy we can make it more convenient to write certain functions and that is so called overloading of functions. And in this segment will also conclude this entire lecture sequence.

(Refer Slide Time: 00:47)



So, C++ allows you to define multiple functions with the same names, provided they have different argument lists. So, the term overloading is used and perhaps we should really be saying overload the name, overload in the sense that the same name means some different things and the same name can be used to refer two different functions. So that is what C++ allows.

(Refer Slide Time: 01:17)



Overloading functions

- C++ allows you to define multiple functions with the same name, provided they have different argument lists.
- Consider a function to compute the area of a graphical object.
- It will be nice to give the name Area to the function even though the arguments could be a circle or a rectangle.
- Just do it! It is allowed in C++.

So suppose you want to write a function to compute the area of a graphical object. Now the name area seems like a good name and perhaps you might want to use the same name to find the area of a circle or to find the area of a rectangle after all what you are getting out is the area so why should you use a more complicated name?

Now C++ allows you to do this and there is no explicit new feature that explicit new construct that I am going to talk about you just have to do it that is all. The only requirement is that the parameter lists should be different. So long as the parameter lists are different you can do it and the parameter lists are need to be different.

Because that is how C++ will note that the two functions actually are different, otherwise if you write f of some x and there are two definition of f of x that is not allowed but if you write f of x where x is of type real and if you write f of x where x is of type int, then C++ will know which function to use if you indeed have two separate functions defined. So I am going to show you this file area.cpp in which I have written such functions and let us get to that.

(Refer Slide Time: 2:48)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

double Area(Circle &c){
    double r = c.getRadius();
    return PI*r*r;
}

double Area(Rectangle &r){
    return r.getWidth() * r.getHeight();
}

int main(){
    initCanvas();
    Rectangle r(100,100, 50,70);
    Circle c(100,100,100);

    cout <<Area(c)<<' '<<Area(r)<<endl;
    getClick();
}
```

UUU: F1 area.cpp All L17 (C++/I Abbrev) 4:28PM 1.10

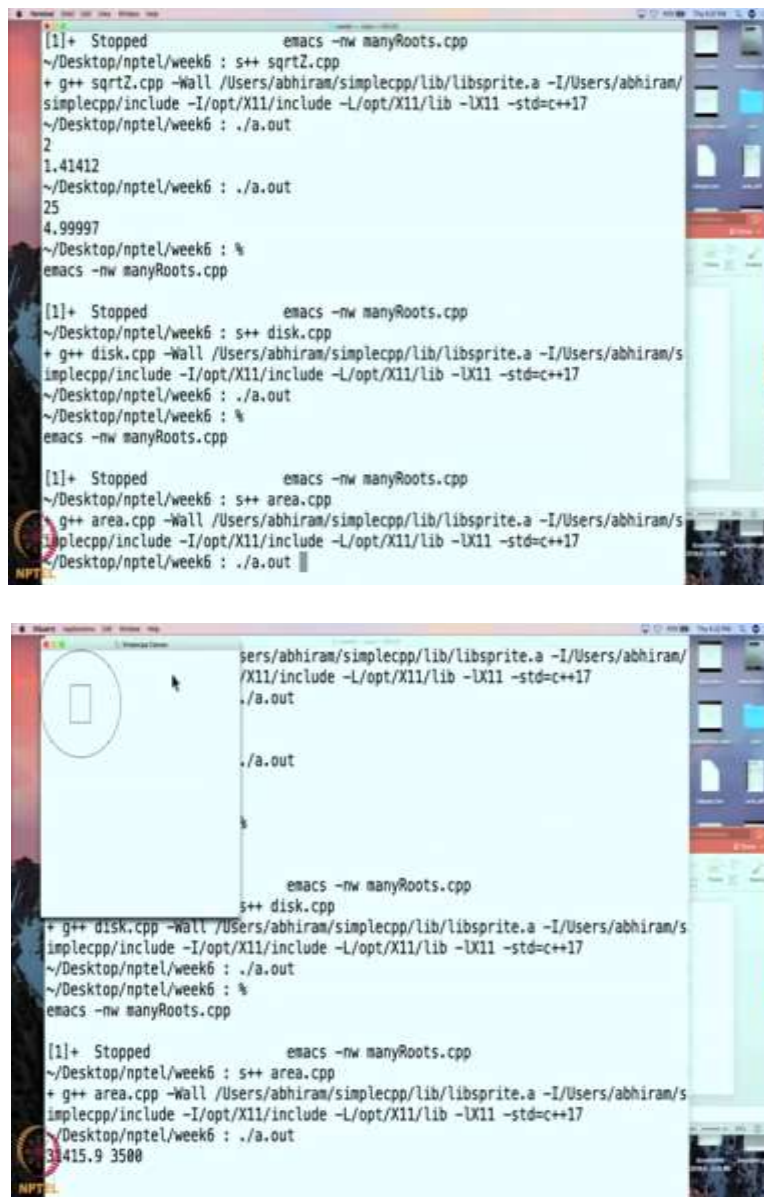
So here is the first definition of area and we are having area function area which is going to be written in double and its taking as argument a circle c. So, remember we said that type of the shape, the name associated with the shape is the shape itself. So, you can pass that shape over here and we do not want to copy that shape and therefore we are passing the reference. But whatever we are passing it has type circle and it's a reference to a circle object.

Now inside this function we can extract the radius of that circle so there is a command getRadius so I can write c.getRadius and that will get me that radius of that circle. And now we know the area of the circle is πr^2 . So this is a simple CPP name, the PI stands for 3.1415192 whatever the expansion is to some large number of decimal places and we just multiply pie by r by r and written the result. So this area, this function expects u to send it a circle a reference to it and it will calculate the area and return that value. Then I have an area of a rectangle so again what is expected over here is a rectangle object should come, rectangle name should come but a graphical object but what is actually coming over here is the reference to it. So this r really refers to the graphical object in the calling function. And, so rectangle can be operated upon by a command getWidth so that the command gets the width of the rectangle, and this getHeight will get the height of the rectangle and if you just take the product you will get the area. So, that is what this area does.

So in this main program I have first something which creates the canvas, then I am going to create a rectangle. This rectangle is centered at 100-100 and it has width 50, height 70. This circle is centered at 100-100 and it has radius 100 as well. Now the finally I have command which will calculate the area of circle as well as area of the rectangle. So, before proceeding further let us see if we can guess what this area should be.

So, the radius of the circle is 100, so πr^2 is going to be π times 10,000. So it should be something like 31,415 something-something so it should be that is what it should be. And the area of the rectangle well the width and the height are 50 and 70 so the area should be 3500. So let us see whether that what the program prints. So let us compile this and run it.

(Refer Slide Time: 06:07)



```
[1]+ Stopped emacs -nw manyRoots.cpp
~/Desktop/npTEL/week6 : s++ sqrtZ.cpp
+ g++ sqrtZ.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/
simplecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week6 : ./a.out
2
1.41412
~/Desktop/npTEL/week6 : ./a.out
25
4.99997
~/Desktop/npTEL/week6 : %
emacs -nw manyRoots.cpp

[1]+ Stopped emacs -nw manyRoots.cpp
~/Desktop/npTEL/week6 : s++ disk.cpp
+ g++ disk.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week6 : ./a.out
~/Desktop/npTEL/week6 : %
emacs -nw manyRoots.cpp

[1]+ Stopped emacs -nw manyRoots.cpp
~/Desktop/npTEL/week6 : s++ area.cpp
+ g++ area.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week6 : ./a.out

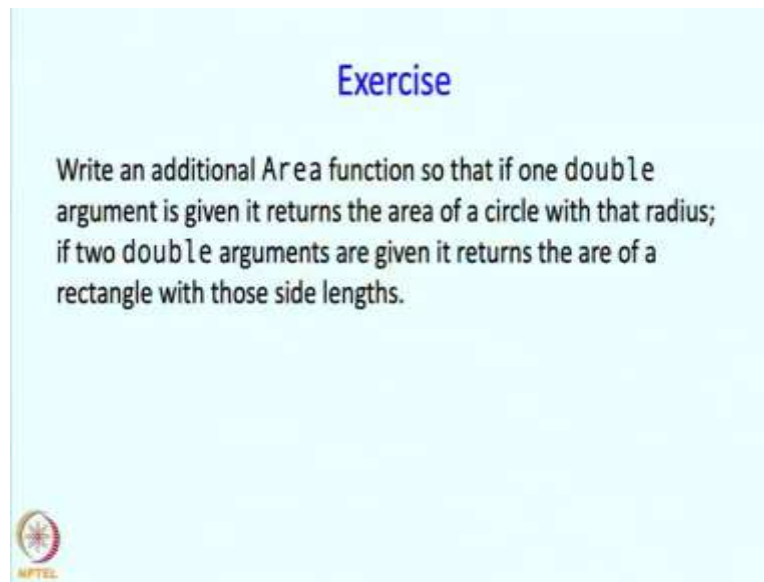
~/Desktop/npTEL/week6 : %
emacs -nw manyRoots.cpp

s++ disk.cpp
+ g++ disk.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week6 : ./a.out
~/Desktop/npTEL/week6 : %
emacs -nw manyRoots.cpp

[1]+ Stopped emacs -nw manyRoots.cpp
~/Desktop/npTEL/week6 : s++ area.cpp
+ g++ area.cpp -Wall /Users/abhiram/simplecpp/lib/libsprite.a -I/Users/abhiram/s
implecpp/include -I/opt/X11/include -L/opt/X11/lib -lX11 -std=c++17
~/Desktop/npTEL/week6 : ./a.out
31415.9 3500
```

So indeed the program has drawn the square and the circle, and this whole thing is a 500 by 500 canvas so it is centered at 100-100 as you might expect. And let us see what area has it printed out. Yes, it has printed out 31415.9 and you may see the 3.14159 is the value of PI so that is just multiplied by 10,000. And the area of the rectangle has indeed been printed to 3500 and that is correct because the width is 50 and height is 70 so let us click to stop that program.

(Refer Slide Time: 7:00)



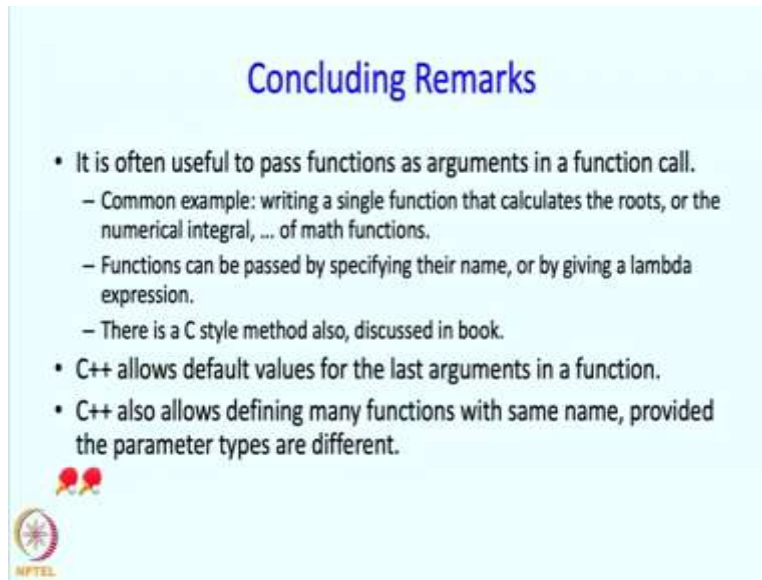
Exercise

Write an additional Area function so that if one double argument is given it returns the area of a circle with that radius; if two double arguments are given it returns the area of a rectangle with those side lengths.

So, here is the quick exercise, so the area function that we wrote was taking shapes as arguments. But you can have additional area functions and you are asked here to write an area function so that if one double argument is given it should be interpreted as requesting the area of the circle with that radius. And if two double arguments are given it should be interpreted as requesting the area of a rectangle with those sidelines. So the code itself is not complicated but the point is that if you can define these functions which take, which have different argument lists, in this case the argument list is a double and for the first one and two doubles for the second ones, and in fact you can have all the four functions existing at the same time.


Because the argument for the very first one was a circle, and for the second one was a rectangle, and then the third one is a single double, and the fourth one is two doubles. So all such, all four can be defined and depending upon what arguments you pass to the appropriate function will be executed. So that concludes this entire lecture sequence so let me make some remarks.

(Refer Slide Time: 08:34)



Concluding Remarks

- It is often useful to pass functions as arguments in a function call.
 - Common example: writing a single function that calculates the roots, or the numerical integral, ... of math functions.
 - Functions can be passed by specifying their name, or by giving a lambda expression.
 - There is a C style method also, discussed in book.
- C++ allows default values for the last arguments in a function.
- C++ also allows defining many functions with same name, provided the parameter types are different.



So, first I should observe that it is often useful to pass functions as arguments in a function call and there are many examples of this. So, common examples are writing a single function that calculates the roots so I will write a single function that calculates the roots and to it pass the function whose root I need, so I do not have to write a separate function to calculate the root of different mathematical functions and this goes not only for roots, but also for numerical integrals or you might want some series.

So for solving series also you can do this and there we saw that you can pass functions by specifying their name, or by giving the lambda expression and one method that I have not discussed in this lecture sequence is a C style method.

So, the C language from which C++ has originated had a different method and that method is a little bit more complicated and if you are using C++ you might as well use the nicer methods. And I did not discuss it today but for completeness it is discussed in the book.

Though, in this course we are not expected to know it. Then we said that C++ allows default values for the last arguments in a function and then we also said that C++ allows defining many functions with the same name provided the parameter types are different. So that concludes this lecture sequence and as always I will suggest to you

that please look at the chapter and the problems given at the end of the chapter and please solve them for practice, thank you.