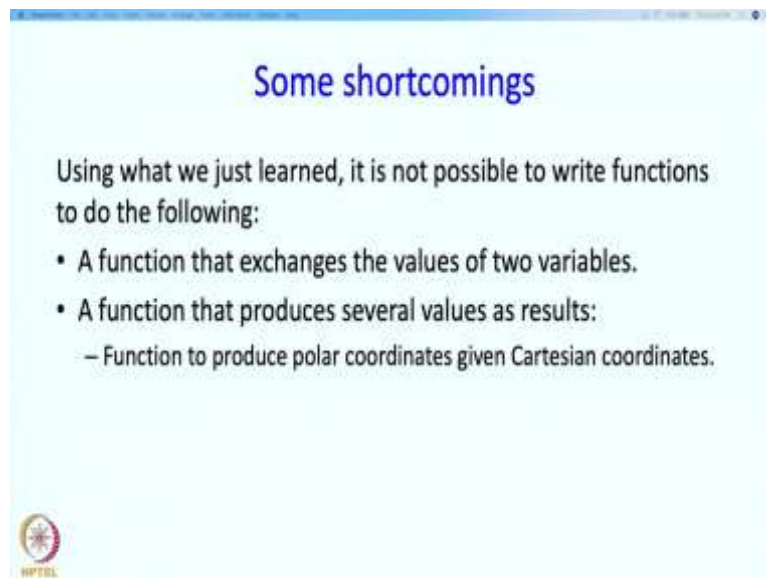


An Introduction to programming through C++
Professor Abhiram G Ranade
Department Of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 10 Part – 3
Functions Reference Parameters

So, in the last two segments we have discussed various aspects of functions. We began by saying where functions should be define and used. And then we talk about how they execute, how they can have their own variables. And then finally we said, how should we be thinking about functions? So, basically we should not be thinking about how a function executes, once we have designed and made sure it works correctly. From that point onwards we should worry about how we use it. So, we should not, when we make a call to a function we should trust that the function is going to return us the correct value so long as we are following the specification. In this segment we are going to discuss something called reference parameters. And, let me began by saying why these are necessary.

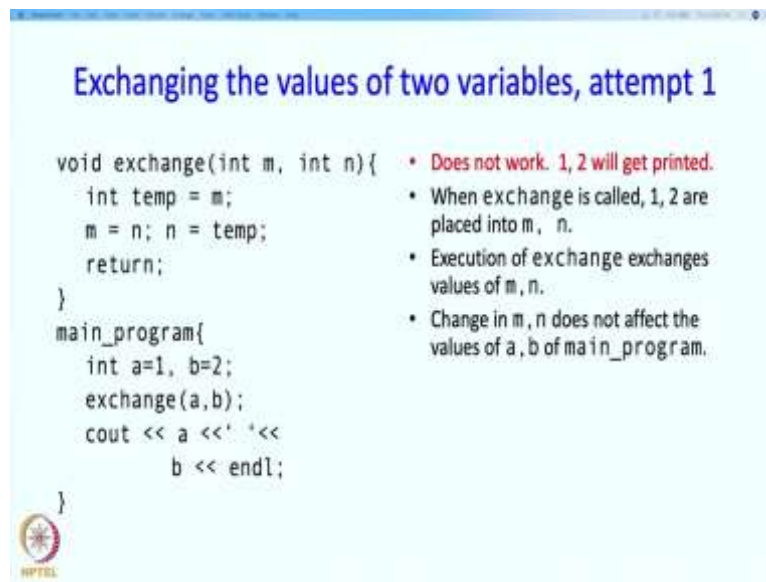
(Refer Slide Time: 01:09)



So, what we have so far has some shortcomings. So, we are not able to write functions to do the following things. So, for example, we cannot write a function that exchanges the values of two variables. So, you may think that often we might want to say look exchange the values of these two variables. So, should not we have a function to do that? Yeah, it seems like a reasonable requirement but, you cannot do that using what I have just taught you. So, we will see that in a minute and we will also see it how to fix it.

Similarly, sometimes you might want a function which not, which does not produce just result may be two results. So, suppose you want to produce polar coordinates given Cartesian coordinates. So, given X and Y I want to produce R and theta. So, the return statement enables you to return one value so, how do you return two values? So, that is another problem that we will see how to fix.

(Refer Slide Time: 02:15)



Exchanging the values of two variables, attempt 1

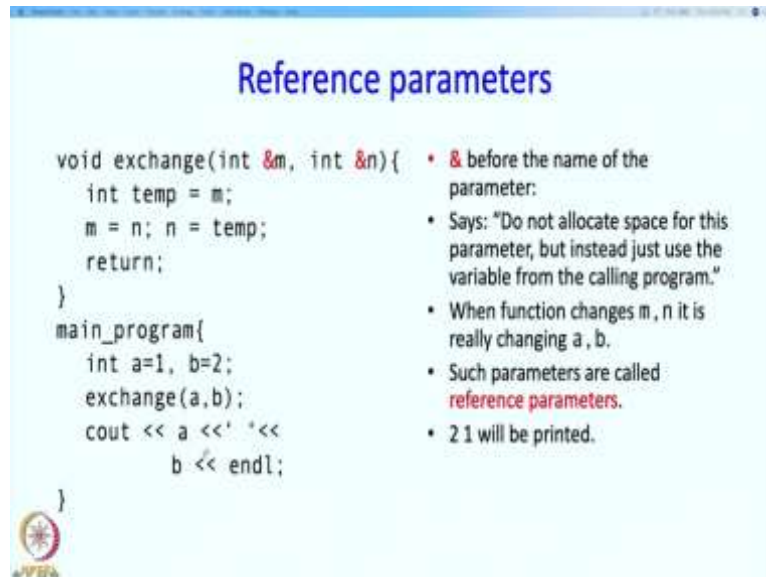
```
void exchange(int m, int n){
    int temp = m;
    m = n; n = temp;
    return;
}
main_program{
    int a=1, b=2;
    exchange(a,b);
    cout << a << ' ' <<
        b << endl;
}
```

- Does not work. 1, 2 will get printed.
- When exchange is called, 1, 2 are placed into m, n.
- Execution of exchange exchanges values of m, n.
- Change in m, n does not affect the values of a, b of main_program.

So, let us come to the problem of exchanging two values. So, the first attempt. So, here is the potential function that I could write. A possible function that could write so, the function could be called exchange. And, it takes two parameters or the call will have two arguments M and N the, values that I want to exchange. So, I place the value from M into temp, then I place M into N, and then I place N into temp. So, at this point clearly the value of M and N are exchanged and, I am not expecting to return anything. And presumably in the main program I can do something like this. So, might I have A is equal to 1 and B is equal to 2, I exchange A B and when I print out I should expect that, hopefully A will be 2 and B will be 1. However, this does not work. 1 and 2 will get printed over here and let us see why that happens. So, when exchange is called 1 and 2 are indeed place into M and N. So, our rule, our usual rule is that when we make a call the value of arguments 1 and 2 are copied into M and N. Now, execution of exchange does exchange value of M and N. But, unfortunately the change in M and N happens in the activation frame of exchange and this does not change anything over here, in the activation frame of the main program. So, when you return the program has same values and so 1 and 2 will get printed. Now, how do you fix this? It turns out that the fix is

actually very-very compact, very slight. It is just a matter of adding two characters but they are kind of important characters so, let us see that.

(Refer Slide Time: 04:30)



Reference parameters

```
void exchange(int &m, int &n){
    int temp = m;
    m = n; n = temp;
    return;
}
main_program{
    int a=1, b=2;
    exchange(a,b);
    cout << a << ' ' <<
        b << endl;
}
```

- & before the name of the parameter:
- Says: "Do not allocate space for this parameter, but instead just use the variable from the calling program."
- When function changes m, n it is really changing a, b.
- Such parameters are called **reference parameters**.
- 2 1 will be printed.

So, these are the two &s that you have to add if, you add those two &s then this code will do what, you expected to do. So, let me explain what these &s do and why they produce the behavior that we wanted. So, if I write an & before the name of the parameter, then it means the following things. It says, "Did you create the activation frame for this function." "Do not allocate spaces for this parameter, but instead make this name refer to a variable, the corresponding variable from the following program." So, in this case this means that this M, there will no variable allocated for M in activation frame of exchange, but instead M will refer to the corresponding argument, so A. So, whatever you do with M in this will actually be done using A. So, A is just going to be, M is just going to be an alternate name or a reference for this A. So, that is why this is called a reference parameter. So, M will really refer to A, M will not have the value of A but M will refer to A. And N is also a similar variable, so N will also refer to B and it will not be a new value. So, when this exchange call happens nothing is actually copied. So, instead these parameters are thought of as referring to the variables in the calling program. So, now when the function changes M and N it is really changing A and B. Because everything over here, so these M's and N's over here are really A's and B's, so, they are just alternate names, so if I say, put an & over here, this is just an alternate name for whatever name you had in call. So, such parameters are called reference parameters and, in fact, in this case what you expect 2 and 1 will get printed.

(Refer Slide Time: 07:01)

Remark

- If a certain parameter is a reference parameter, then the corresponding argument is said to be "passed by reference".
- Now we can write a program that computes polar coordinates given cartesian coordinates
 - We use two reference parameters.
 - Called function stores the polar coordinates in the reference parameters.
 - These changes can be seen in the main program.
- There are other ways of returning 2 values – study later.



If a certain parameter is a reference parameter, then the corresponding argument is said to be passed by reference. And now this reference parameter enables us to not only exchange values, but also compute polar coordinates given Cartesian coordinates and somehow convey the values back to the calling program.


So, basically the idea is that we are going to use two reference parameters and the called function will store the polar coordinates in the reference parameters. And this changes can be seen in the main program. And by the way, this is not the only way of returning two values, we just see later that there are other ways as well, but this is one of the ways.

(Refer Slide Time: 07:50)

Cartesian to polar

```
void CtoP(double x, double y,
         double &r, double &theta){
    r = sqrt(x*x + y*y);
    theta = atan2(y, x); //arctan
    return;
}
main_program{
    double x=1, y=1, r, theta;
    CtoP(x,y,r,theta);
    cout << r << ' ' << theta << endl;
}
```

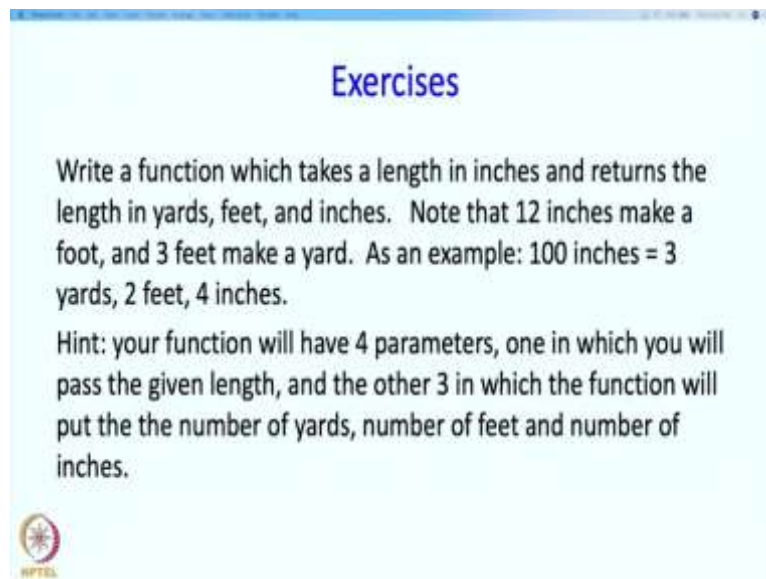
- r, theta in CtoP are reference parameters,
- changing them in CtoP changes the value of r, theta in the main program.
- Hence sqrt(2) and pi/4 (45 degrees) will be printed.



So, here is the Cartesian to polar program. As you can see X and Y are ordinary parameters there are passed by value. R and theta on the other hand are reference parameters. So, what does that mean? So, when this call CtoP happens the values 1 and Y are copied into X and Y, but the values of R and theta are not copy. Now, this function is going to execute and plays some value in R and this R is really the same value of R because this is a reference parameter. So, this value will actually get stored in the activation frame of the main program and in particular, in the variable R in the activation frame of the main program, and similarly, this theta, because it is the reference parameter and because the fourth argument here is the theta of this main program, this theta is just a name for this theta. And therefore, whatever is calculated over here, will get stored in the activation frame of the main program in the theta variable over here.

So, as a result what gets printed is going to be the R which was calculated over here and theta which was calculated over here. So, changing them in CtoP change is the value of R theta in the main program and hence, what is get calculated, so square root of a 1 and 1, 1 square plus 1 square, so square root 2 will be printed for R and, atan 2 of 1,1 is 45 degrees so pi by 4 will get printed over here.


(Refer Slide Time: 09:33)



Exercises

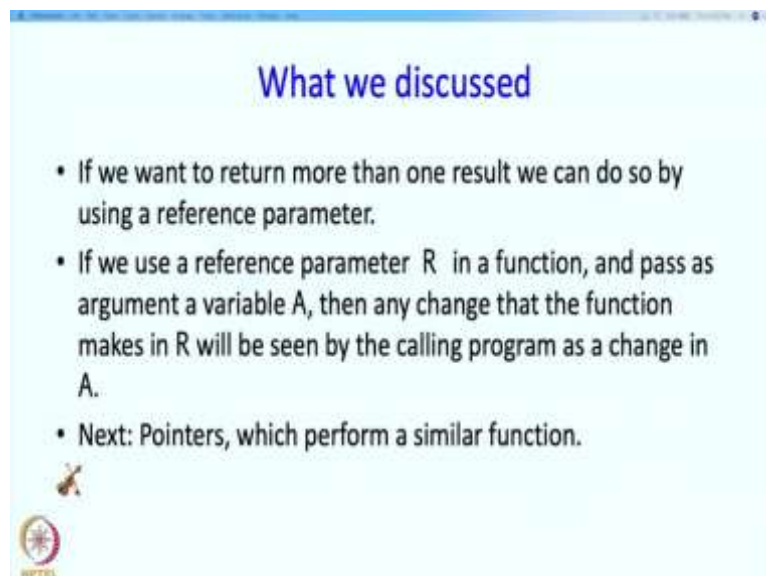
Write a function which takes a length in inches and returns the length in yards, feet, and inches. Note that 12 inches make a foot, and 3 feet make a yard. As an example: 100 inches = 3 yards, 2 feet, 4 inches.

Hint: your function will have 4 parameters, one in which you will pass the given length, and the other 3 in which the function will put the the number of yards, number of feet and number of inches.




Alright so, here is a quick exercise, so write a function which takes length and inches and returns the length in yards, feet, and inches. So, this is the so called a British system which is not really in use, but which does get used once in a while. So, you should note that 12 inches make a foot and 3 feet make a yard. So, as an example, if you have 100 inches then that is equivalent to 3 yards 2 feet and 4 inches. So, that is what you are supposed to print. So, clearly you are expected to be returning 3 values. And, therefore you should have 4 parameters 1 in which will pass the given length and, other 3 in which the function will put the number of yards, number of feet and number of inches. Alright so, what did we discuss in this segment.

(Refer Slide Time: 10:33)



What we discussed

- If we want to return more than one result we can do so by using a reference parameter.
- If we use a reference parameter R in a function, and pass as argument a variable A , then any change that the function makes in R will be seen by the calling program as a change in A .
- Next: Pointers, which perform a similar function.



So, we said that if you return more than 1 results, we can do so by using reference parameter. If we use a reference parameter R in some function and pass as argument a variable A, then any change that the function makes in R, will be seen by the calling program as a change in A. In the next segment we are going to discuss pointers which will perform a similar function, but pointers are quite useful in other places as well so we will see that shortly. Let us take a break.