

Programming, Data structures and Algorithms
Prof. Shankar Balachandran
Department of Computer Science
Indian Institute of Technology, Madras

Module - 10D

Lecture - 23

Example: swap value of two variables

Why pass by value does not work?

Passing parameters by reference

Arrays as function of parameters

Examples: printing arrays, swapping elements

Welcome back. Earlier I promised that we will look at pass by reference in lot more details. So, in this module, we will look at pass by reference and we will also see the case for passing arrays parameters to functions and how do you program something like that.

(Refer Slide Time: 00:30).

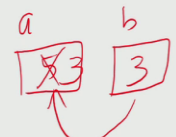
Exchanging Values of Two Variables

- Write a function that exchanges contents of two variables a and b
- Incorrect code:

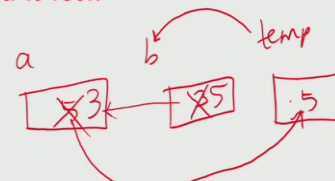
```
a = b;  
b = a;
```

X

value of a is lost!


- Basic code:

```
temp = a;  
a = b;  
b = temp;
```



So, let us start with the very basic example. So, let us say I want to exchange the values of two variables. So, I have two variables, a and b and I want to exchange the values of a and b. So, one way to do that is as follows. So, a equals b and b equals a. This is a very common thing that beginners do take a equals b and b equals a. Unfortunately, this is something that would not work. So, let us take a small example. Let us say a was (00.58).

Let us say a is a variable. So, in memory it has a value 5. B is a variable and in memory it has a value 3. Then, if you do a equals b, it takes 3 and overwrites the value. Here it becomes 3 so, because of this the value of a is lost. So, both a and b will have the same

value instead what will actually work is something like this. So, this is a very basic template for swapping two variables. This is something that you will see very often, and you will also use it very often to swap two variables. So, this is a very straight forward way of doing it. So, instead of two variables, just a and b which we already have. So, let us, say we start with that a equals 5, b equals 3. We also have a new variable called temp in memory, right.

What we are going to do is since we were losing a, we will take a and copy on to temp, right and now if I copy b to a, I lose 5 and I have 3. However, I have remembered the value 5. So, it is not a problem. Now, what I will do is to compute b, I will copy back the value from 10. So, from temp I copied to b and that becomes 5. So, at the end we have a equals 3, and b equals 5. So, this is a basic piece of code, right. So, if I want to swap two variables in the body of a program, I could do this.

(Refer Slide Time: 02:40).

Writing a function swap() for exchange

```
void swap1(int a1, int b1)
{
    int temp = a1;
    a1 = b1;
    b1 = temp;
}

#include<stdio.h>
void main()
{
    int a = 5, b = 6;
    swap1(a,b);
    printf("%d %d",a,b);
}
```

Function

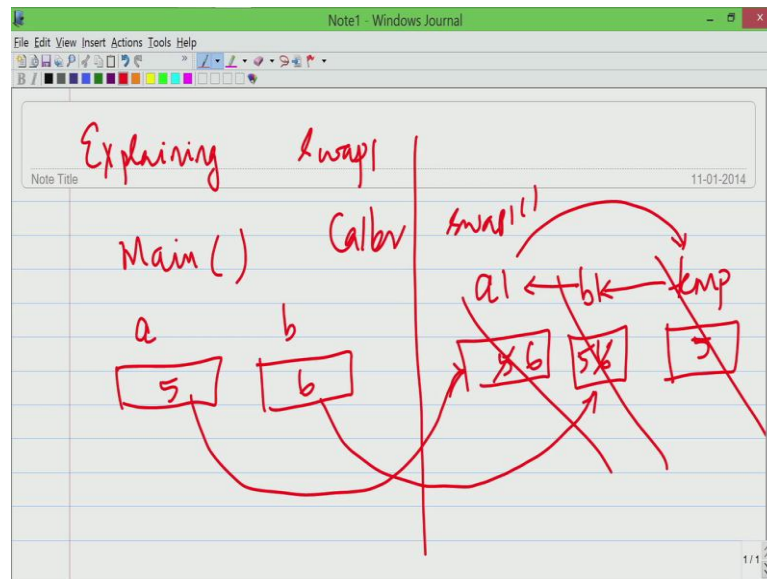
Function Call

5, 6

So, now I want to take this and write a function instead, right. So, I want to take two variables and I want to write a function call swap which will exchange the variables. It looks like it is straight forward. So, we write a function we take two variables a 1 and b 1 and this is the body of code. Let us say I just copied and paste from the previous line. Instead of a and b, I call them a 1 b 1, right. So, clearly if I start with a 1 b 1, at the end of this a 1 and b 1 would be swapped, right. So, if a 1 was 3 and b 1 was 5, right. Sorry, if a 1 was 5 and b 1 was 3 as I had in the example earlier, I would get a 1 to be 3 and b 1 to be 5 so, that is guaranteed. Let us go and look at how the function called would be. So, I have a equals 5 and b equals 6. I am calling this function swap 1 of a, b and then, I am

going to print. So, since this is calling the function, I would expect a and b to be swap. I would expect the print out to give me 6 and 5. Unfortunately, this one gives the result 5 and 6 itself, right. So, a is 5 and b is 6 before the function call. Even after the function call, a and b seems to be 5 and 6 itself. So, we need to think about why this is really happening. So, to do that, we will have to go back to the basic permeation function parameters in c. So, let us look at what is going to happen.

(Refer Slide Time: 04:20).



So, I am in the main function let us say. So, this is for explaining swap 1, right. Swap 1 is the function name. So, we had a which was 5 and b which was 6, and let us say at some point I called swap 1. So, when swap 1 comes, you are going to get an activation record for all the new variables, right. So, what are the new variables? We had a 1, b 1 and temp. So, a 1 and b 1 were actually formal parameters, right and temp was actually declared within the program. So, if you go back, we see that temp is a variable that is declared within the program. A 1 b 1 are actually formal parameters, and the thing with c is we take whenever I have function call, we copy the formal parameters, right. So, I make a copy of the formal parameters 5, gets it copied to a 1 because we call swap 1 of a, b. A gets copy to a 1 and b gets copy to b 1. So, I have 5 and 6.

Now, the function is ready to execute. I have the variables a 1 and b 1, and what is that it we did that we took 5 and made a copy on temp. So, it got copied and then, we took 6 and then, copied to 5. So, a 1 got 6, and we took temp and copied into b 1 that got 5. So, clearly at the end of this a 1 and b 1 got 6 and 5. So, I already mentioned this it swaps a 1 b 1. Let us say now you are done with this. The function returns to the caller. What

would happen? So, this side is the caller and this side is the callee, right. This is swap 1. So, when the callee returns, all the local variables get destroyed. We do not have them anymore. You cannot access them anymore, right and when you came back, you did really change a and b. No, we made copies of a and b on a 1 b 1. A 1 and b 1 got swapped, but they do not change a and b, right. So, this is the common rookie mistake that people make that user call by value here because we called a 1 and b 1 by value. The pass 5 in 6 within the function it swapped, but when you return from the function, it does not swap. Now, how do we fix this problem? So, clearly it does not work. You need to think about something else. We will write this swap function using references..

(Refer Slide Time: 07:22)

Writing swap() using references

```
void swap2(int *a1, int *b1)
{
    int temp = *a1;
    *a1 = *b1;
    *b1 = temp;
}

#include<stdio.h>
void main()
{
    int a = 5, b = 6;
    swap2(&a,&b);
    printf("%d %d",a,b);
}
```

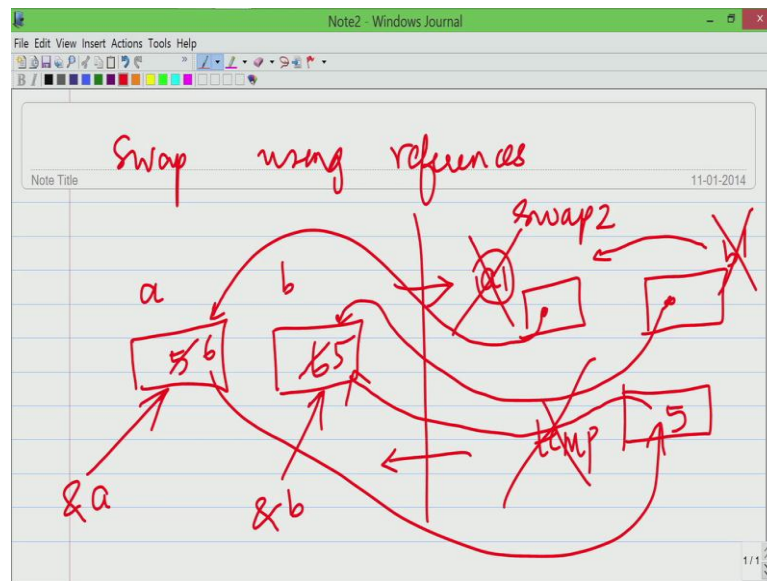
Function

Function Call

6 5

So, as before we will start with functions. So, we have a equals 5, b equals 6. What I am going to do is instead of swap 1 and going to write a new function called swap 2. So, clearly there was a problem. We should not use pass by value. I am going to use pass by reference. So, the way it is going to work, this follows.

(Refer Slide Time: 07:51)



So, I am going to write how to swap using references, ok. I am going to call this function swap 2. So, let us see how I could do this? I have a, which was 5 and b, which was 6 and if I had passed values, they would have been a problem. So, instead what I am going to do is, I am going to take the pointer of a, and pass it on, right. So, let us say I have pointer of a, and that could have been ampersand of a, right. We already know that ampersand of b is ampersand of b, right. So, the address of a and address of b, I can always get it. What I am going to do is I am going to pass that to the function. The function is going to take two pointers now, and what is going to do is follow, right. So, I have these two pointers. So, I have a pointer. Now, I am going to call that a 1. So, this actually now going to point to 5, and I am going to have another pointer called b 1 and that is going to point to 6. So, I copied ampersand of a into a1 and ampersand of b into b1. So, a 1 and b 1 are not integers. They are pointers to integers.

So, now what I am going to do is, I am going to have another local variable called temp because you already know that we cannot swap two variables without these extra things, right. So, there are ways to do that using only two variables, but in the method that we saw, we declared another variable called temp. Now, what I am going to do is, I want to swap not the contents of a 1 and b 1. I want to swap that content pointed to a 1 with a content pointed to by b 1. So, what is a 1 pointing to it is pointing to 5, and b 1 is pointing to 6. I want to swap the contents of that in essence I want to change the contents of a and b. Now, how I am going to do that is, I will keep a copy of the contents pointed to by a 1. What is a 1 pointing to? It is 5. I will keep a copy of contents of the memory

location pointed to by a 1. Then, what I will do is, I will take the contents pointed to by b 1. So, I want to take that content pointed to by b 1 and move it to location pointed by a 1. So, what is b 1 pointing to? It is pointing to 6. I am going to take that, right and move it to the location pointed to by a 1.

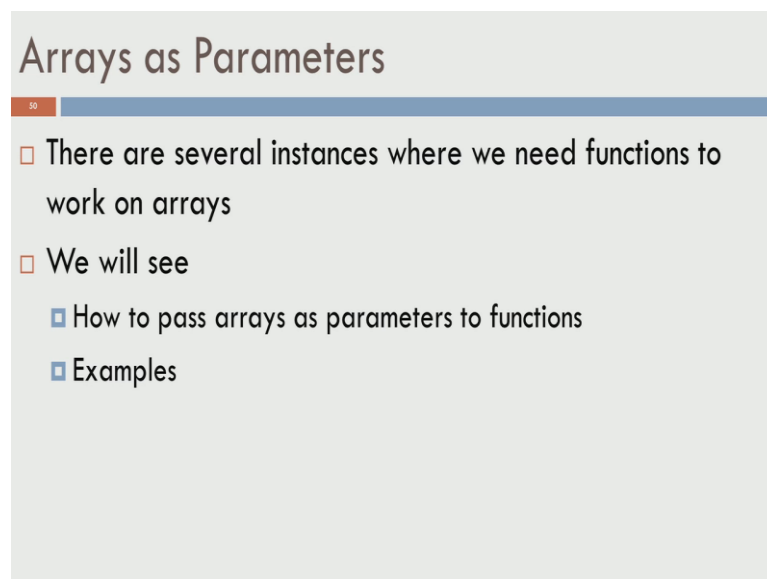
Where is a 1 pointing? A 1 is pointing to this location. I will move it here and finally, I still have to restore it. So, I have to get the values of wherever a 1 was pointing that has to change. So, that has changed. You have to see what contents of b 1 must be. B 1 is pointing to this location which had a value 6. So, I have to take this value 5 and copy into that. So, that would change the values of this memory location and when I write it as before a 1 gets destroyed. So, it was of pointer, but it was a formal parameter. It gets destroyed. B 1 was a formal parameter. It gets destroyed. Temp is a formal parameter that gets destroyed. So, I have written from the callee back to the caller, but that point already a, and b are 6 and 5, right. So, the key thing was instead of passing the values 5 and 6, if we pass the pointers to a and pointers to b, I can use the de-referencing operator that we talked about earlier and get back to the values and use that. So, that is what we are going to do. So, I am going to pass address of a and address of b, and I am going to tired print, right. So, it is function call and let us think about how the function should look like. So, swap 2 is not returning any value, right and you have two pointers that have been passed on. So, we need some mechanism by which we will take to pointers and I do not expect any return values. I will not have any return. So, the function would look something like this, right.

So, I have void because it is not returning anything. I have swapped to as the function name and as I said earlier, we are passing pointers and the way to receive the pointers is int star a 1 and int star b 1. So, we have two pointers, two integers. One is called a 1 and one is called b 1. Now, I was talking about a new variable called temp. So, this is actually local to swap. It is an automatic local variable. What I am going to do is I am going to take a 1, look at the contents of whatever is pointed to by a 1. So, that is done by using star, right. So, a 1 is pointer star. A 1 gives the contents of the location pointer to by a 1, and I am going to keep a copy of that. Then, I am going to take b 1, look at the contents of the location pointed to by b 1 and make that the value of the contents of the location pointed by a 1. Finally, I take temp which is a local variable and make a copy of that on to contents of location pointed by b 1 and when you return back from here, a 1 b 1 and temp are destroyed. So, we met copies of the pointer.

So, it is not really destroying a and b, it is only destroying the copies of the pointers that we had for a and b. It is not even destroying copies of a and b. It is destroying the copies of the pointers for a and b, and it destroys temp which was a local variable. It does not matter because by now we are actually changed the contents of 6 and 5. So, a would become a 6, and b would become a 5. So, at this point if you print f these two things, you would get 6, 5. So, this is called swap using references. So, what we have actually done is we have taken two variables, and we want the variables to the values of the variables to change. We are not going to work on just the copies. We want actual contents of the variables to change and whenever you have anything of this sort where you want the contents of some variables to change and if you want a function which is dedicated to doing that, you have to pass pointers.

So, reiterate that let us say I have some variables and instead of doing the work on the variables locally, let us say I want to work on it by doing some delegations to a function. I want to pass on to a function, but I expect this piece of code to actually change the contents of the variable. If I am going to only print them on the screen, right. It does not matter. I could have just printed using a function. I do not need pointers, but I want a and b to exchange their values. The moment you write a function, for that you have to use pointers to do it. You cannot just copy the contents and do operations on that because whatever you do on the copies will not reflect on the original variable that you have.

(Refer Slide Time: 16:11)



The slide is titled "Arrays as Parameters" and contains a list of bullet points. The title is in a large, dark font at the top. Below the title, there is a small red square with the number "50" inside. The main content consists of three bullet points: a square bullet point followed by "There are several instances where we need functions to work on arrays", a square bullet point followed by "We will see", and two indented square bullet points: "How to pass arrays as parameters to functions" and "Examples".

Arrays as Parameters

- There are several instances where we need functions to work on arrays
- We will see
 - ▣ How to pass arrays as parameters to functions
 - ▣ Examples

Now, let us look at the larger picture. So, I showed you how to do pass by reference for this small thing. You also did this earlier for quotient and remainder, right. There are

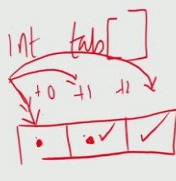
several cases where we need functions to work on arrays, right. So, let us say I give a list of numbers in an array, and I want you to find out the largest of the numbers or I give you list of numbers in array. I want to put it in decreasing order or increasing order and so, on and I want to a function to do that, right. So, I can always go and write the whole thing in the main programme itself, but if I am going to delegate duty and if I make it readable, I am going to write function, in that case we will also need to understand how to pass arrays as parameter to functions, not just individual variable. We also need to learn how to pass arrays.

(Refer Slide Time: 16:56)

Printing Arrays

```
void main(void) {  
    { int a[5]={1,2,3,4,5};  
      int b[] = {-3,-2,-1,0,1,2,3}; }  
    → print1(a, 5);  
    → print1(b, 7);  
}
```

```
void print1(int tab[], int N) {  
    int i;  
    for (i=0; i<N; i++)  
        printf("%d ",tab[i]);  
}
```



The diagram shows a variable declaration `int tab[]` with a red box around it. Below it, a horizontal array is represented with three cells. The first cell contains a red dot and is labeled with `+0` above it. The second cell contains a red dot and is labeled with `+1` above it. The third cell contains a red dot and is labeled with `+2` above it. Red checkmarks are placed above each of the three cells. Arrows point from the `tab[]` variable to the first cell, and from the `+0`, `+1`, and `+2` labels to their respective cells.

So, let us see a simple example. So, we will start with things which are only reading the arrays, but not changing the contents of the arrays. So, let us look at this simple example called print 1, right. So, it is a function which is going to take an array and print the values that is all, nothing more right. So, I have two arrays in the main function a and b. So, a is an array of size 5, and b is an array of size 7 and this function takes an array and the number of elements in the array, it prints all of it. So, if you forget this part, right now, let us look at the body of the loop. The body of the loop if I have n for i equals 0, i less than n i plus print tab of i, So, I have some one-dimensional table I want to print the contents of that. I can do it definitely. So, it is not a problem, right. Now, what I want to point out is the mechanics or how to do this, right. So, if you look at the function call, it has print 1 a, 5. So, this point you are passing a, which is the variable name that we use for the array.

What it really does is, it sense of pointer to the first element of a. So, to print 1 the caller

sense a pointer to the first element of a, and this on the other side in tab of. So, we have this int tab of left and right bracket. So, what this does is, it receives the pointer and we can do tab of i because you have received the pointer and from there if I say i equals 0. So, it is that pointer plus 0 steps away from it. Then, when i become 1, it is that pointer plus one more step away from it and then, two steps away from it and so, on. So, that is what we are going to do, right. So, we get a copy of the pointer to this and then, if you know how to access the contents of this, I can print it. Then, I will get one step away from it. So, tab of 1 is one step away from tab of 0. I can look at the contents and print it. Then, tab of 2 is one step away from that I can print it and so, on. So, this is a one way of passing arrays to function.

So, what we have is we have two arrays a and b. So, at this point you are passing pointing to a pointed to a of 0 to print 1. In this line you pass pointer of b. So, you pass pointer of b of 0 to the print 1. So, local variable is tab is formal parameter and n is also a formal parameter. So, the formal parameter receives pointer of a of 0 the first time and it takes n equals 5. So, it will print five entries. The second time this formal parameter tab takes the value which is the address of b of 0, and it will take n which is 7 and print seven times, right. So, this is one way of doing..

(Refer Slide Time: 20:31)

Printing Arrays (2)

```

void main(void) {
    int a[5]={1,2,3,4,5};
    int b[] = {-3,-2,-1,0,1,2,3};
    print2(a,5);
    print2(b,7);
}

void print2(int *tab,int N) {
    int * ptr;
    for (ptr=tab; ptr<tab+N; ptr++)
        printf("%d ", *ptr);
}

```

No change

a

ptr

ptr

ptr

ptr

ptr

ptr

ptr

ptr

ptr

ptr

So, there is another way of doing the same thing. So, in all these examples the main function, there is no change. You watch out for what is happening in the print alone. So, this is the second version of print. So, the first change we have is, we have int star tab, right. So, instead of int tab. So, in the previous example we had int tab of array index like

right. So, instead we have `int star tab int n`. So, this is as it is before and we have `int star pointer ptr`. What we are going to do is we are going to take `ptr equals tab` which means `tab` the first invocation of `print 2` will have a pointer to `a` of `a` of `0`. You make a copy of that to `ptr`. So, let us look at the sequence of things. So, in `print 2 a, 5` when you call this, the caller passes the pointer to `a` of `0`. So, `a` of `0`, `a` of `1` and so, on, right. So, let us say that is the pointer it passes that to `tab`.

So, you make a copy of that to `tab`, right and in this loop you have another variable called `ptr` which is also a pointer data type, and this pointer data type makes another copy of that is `ptr` makes another copy of `tab`. Now, if you do `star ptr` where is it going to point? So, `tab` had a copy of pointed to `a` of `0`, `ptr` is copy of `tab`. So, `ptr` also points to `a` of `0`, right to `star ptr` will print `a` of `0`. Then, we come back and look `ptr plus plus`. So, since this is an integer pointer, this is supposed to point to the next valid integer. So, `ptr` would have been pointing to `a` of `0`. Now, `ptr` will point to `a` of `1`, right and then, `star ptr` will print `a` of `1` and we keep doing this till `ptr` is less than `tab plus n`, right. So, this is one way of doing this. Same thing I already talked to you about this notion of arithmetic in `tab`, right. So, `tab plus n` means take `n` step away from wherever `tab` is pointing to `a` of `0`. So, we are looking at `a` of `0` and if I walked five steps away from it. So, `tab` would be pointing to something beyond `a` of `4`. So, as long as you are less than that, you are good. We can keep printing. So, this is `print 2`. Let us look at `print 3`. In `print 3`, we do not use this `ptr` anymore. In fact, if you look at this `print 2`, all we did was we made up copy of `tab` to `ptr`. So, `tab` itself is a copy of pointer to `a` of `0` and we made up copy of that and we use that for the iteration.

(Refer Slide Time: 23:50)

Printing Arrays (3)

```
void main(void) {
    int a[5]={1,2,3,4,5};
    int b[] = {-3,-2,-1,0,1,2,3};
    print3(a,5);
    print3(b,7);
}
```

```
void print3(int *tab,int N) {
    int i;
    for (i=0; i<N; i++, tab++)
        printf("%d ", *tab);
}
```

We do not have to do that. Instead what we are going to do is, we have tab which is pointing to a of 0. So, as before we have a and this is a of 0 and pointed to that is copied to tab when you call the function. So, tab actually is also going to 0.2 a of 0 and star tab would be a of 0 first and then, this is actually running a loop n times, right. So, this is not using any pointer arithmetic for stopping the loop, instead we know that we want to print n of these variables on the screen. So, just run the loop n times. So, this check is actually on an integer, right. So, we have i equals 0, i less than and i plus plus will run this loop n times starting from i equal to zero to i equal to n minus 1, and each time what you doing is the first time i equals to 0. Star tab will be pointing to a of 0 you print. So, tab is pointing to a of 0. Star tab will actually print a of 0 and then, you do tab plus plus which means you are actually pointing to the next location.

The next time when you come around, star tab would be a of 1. You print that and you increment tab to go here and so, on. So, that increment is happening here. So, there is pointer arithmetic. This tab plus plus is actually saying where ever you are pointing to the start pointing to the next location where ever you are. So, it is start with a of 0. Next time we will point a of 1 and so, on, right and once a is printed, when you come back here tab gets destroyed and n gets destroyed because these are in the callers. They get destroyed. I also get destroyed. Now, you start with b, b, 7. At that point you have this array called b which has 7 entries and you are actually passing the pointer of b to tab. So, this is in the caller side in the callee side. So, you have tab. It starts with a copy of b of c, right. So, this is the setup, right. So, we saw three ways of doing this. So, this is one most

likely way in which we will do things. This print f print 3 is the most likely way in which you will actually do things, right.

(Refer Slide Time: 26:25)

```
void swap3 (int array[], int i, int j)
{
    int temp;
    temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```

So, now, let us go back to now looking at how to use arrays and do some simple things. So, in this example we have a small function which takes care of swapping, right. So, we have swap 3. We have two versions of swap. Earlier swap 1 which is actually incorrect, swap 2 which was taking to pointers and swap the contents of the locations, where the pointers pointing here. What we are going to do is, we are going to take an array and two indices i and j. What we want is a of i and a of j to be swapped. So, that is the goal of this. So, the way to do that is very similar to what we did earlier. We have this new thing called temp. Temp keeps a copy of array of i. Array of i copies, array of j and array of j copies temp, and this i claim is correct, right. So, even though it looks like it is very if i replace array of i by a and array of j by b, it is looking like a swapping local variable, but you are actually passing the pointer to the local variables. Array of i is not the simple value, right. It actually goes to what was pointed to by the thing called array. I will take the contents of that, put int temp, right. So, this is very similar to pass by reference only that two pointers are pointed to the ith location, pointed to the jth location.

So, you are accessing the contents of the location pointed to by what is happening in the ith location and j. You can think of it as the index which gives you the pointed to the jth location of the array. You get that pointer; you access the contents and so, on. So, this is a very simple way of swapping the contents of two variables in an array. So, we are passed array by reference. We are actually passing pointers here. We did not copy the contents

of arrays on to formal parameter list. You only passed a pointer to the beginning of the array and if you have i, j , these are 2 indices. So, array of i will go i steps away from the beginning of array. Array of j will go j steps from the beginning of the array, and you are swapping the contents of these two locations. So, you get the swap 2 elements. So, something like this is very useful if you want to do some things like order the elements in an array or find out the largest or smallest value in an array and so, on.

So, this brings us to the end of this module as well. So, we will see how to look at using this swap function to do finding out the n th element in an array or the largest element in an array, and see how to do swapping.

Thank you very much.