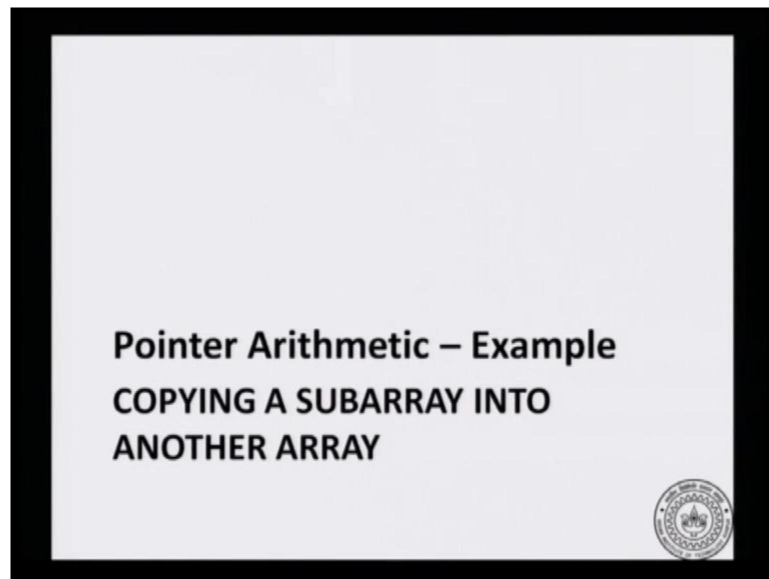
Introduction to Programming in C
Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 35

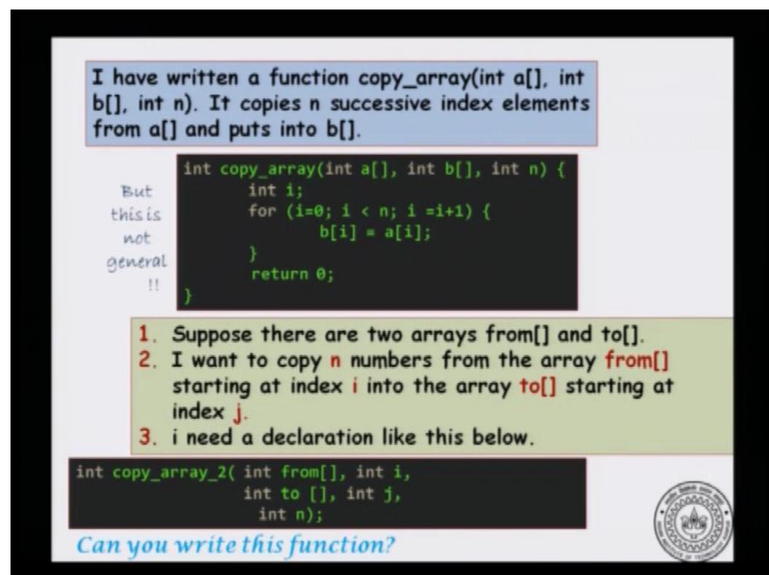
Since pointer arithmetic is a tricky concept let us solve one more problem to try to get comfortable with that notion.

(Refer Slide Time: 00:07)



So, the problem here is copying a sub array into another array. Now, let us explain what that means?

(Refer Slide Time: 00:17)



Suppose, I have written a function copy array which has three arguments an integer array a and integer array b and n which is the size, I want to copy n successive index elements from a and put it into b. So, a is 0 through a n minus 1 have to be copied to b. I can easily write it in the following function has int copy array int a, int b, int n and then I have one variable to keep track of the index and that variable goes from 0 to n, for i equal to 0, i less than n, i equal to i plus 1 and then I simply say b i equal to a i within the loop.

So, this would copy whatever a i is into the location b i. So, once the loop executes, I would have copied n elements from the array a to the array b. But, this is not general and I want to solve the following problem, I have two arrays let us name them from and to and I want to copy n numbers from the array from to 2. But, I have an additional requirement, I want to copy n elements from index i.

So, the earlier code solve the problem from index 0 in general I want to copy from index i of from in the elements into the locations starting at index j in 2. So, the earlier function assume that i and j were both 0. In the general function I want arbitrary i and arbitrary j. So, I need a declaration like the following, I have int copy array 2. So, this is the second function I am writing and I have from i to j and then n is the number of elements to copy. So, what I have to do is from i from i plus 1. So, on up to from i plus n minus 1 have to be copied to 2 of j 2 of j plus 1 so, on up to 2 of j plus n minus 1.

So, for the purposes of this lecture let us just assume that from and to are big enough. So, that you will never ever over suit the arrays by taking i plus n minus 1 and j plus n minus 1. Can you write this function? Now; obviously, you can write a separate function to solve this. Now, the trick is can you use the copy array function, the copy array functions copied n elements starting from index 0 of a to index n minus 1 to the array b starting at index b of 0 to b of n minus 1. So, that is what is it means.

And this should be strange, because if you think about it in a mathematical way, you are saying that a general function is being solved in terms of s particular functions. So, you are reducing a general case to a special case at sounds a bit strange. But, we can do this with pointer arithmetic.

(Refer Slide Time: 03:58)

```
int copy_array_2( int from[], int i,
                  int to [], int j,
                  int n);

int copy_array(int a[], int b[], int n) {
        int i;
        for (i=0; i < n; i =i+1) {
                b[i] = a[i];
        }
        return 0;
}

int copy_array_2( int from[], int i, int to[], int j, int n) {
        copy_array(from+i,to+j,n);
        return 0;
}
```

OK! You can write the new function using your own function

So, let us try to solve with using our own function, and here is the guess that I am making and then I will justify that this works. So, I will just say here is my function from i, to j n that will be defined as copy array from plus i to plus j comma n. So, copy array will start from a of 0 and copy n elements after wards to b of 0 so, on up the b of n minus 1. And that function I am calling using the address from plus i two plus j and n, n is the number of elements I want to copy. Now, I will justify that this works.

(Refer Slide Time: 04:54)

Problem: Given two int arrays f[] and t[], copy n numbers starting from index i in f[] to index j in t[]. So t[j] = f[i], t[j+1] = f[i+1], ..., t[j+n-1] = f[i+n-1].

```
void copy_array( int a[],
          int b[], int n) {
    int i;
    for (i=0; i<n; i =i+1) {
          b[i] = a[i];
    }
    return 0;
}

int copy_array_2( int f[], int i,
      int t[], int j, int n) {
      copy_array(f+i,t+j,n);
      return 0;
}
```

State: execution at start of copy_array_2

f	f[0]				f[4]				f[8]		f[10]
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

t	t[0]							t[7]			t[10]
	1	2	-3	6	-5	4	6	-9	11	81	42

i = 2, j = 4, n = 5

So, here is a problem that I have and I want t of j equal to f of f i, t of j plus 1 equal to f of i plus 1. So, on up to t of j plus n minus 1 equal to f of i plus n minus 1. So, let us try to see what happens in this function? Suppose, I call copy array 2 from name, using the

arrays f, t, i, j, n, n and that function nearly calls the old copy array function, using f plus i, t plus j and n. So, this state of execution at the start of copy array 2, let say that f is an array with say 10 elements and t is an array with say 10 elements arbitrary and what I want is I also assume that i is 2 and j is 4.

So, I want to copy 5 elements starting from the second location or the third location in f f of 2 onwards to the fifth location in t onwards. So, here is what I want to… So, I want to copy this minus 1, minus 1, minus 1, minus 1, minus 1, to t of 4 onwards. So, I will copy them here, so, 5 elements are to be copied.

(Refer Slide Time: 06:33)

Let see; how our, function is able to do this. So, t plus 4 is this location f plus 2 is just location. So, what I am calling is the old copy array function with f plus i,. So, f is the address of the first location of the array. Therefore, f plus 2 using pointer arithmetic is the second integer box after that. So, it is pointing to f of 2, similarly t plus 4 t plus j in this case is pointing to the fourth location after the location pointer 2 by t, t is an array. So, t points to the first location in the array therefore, t plus 4 will point to the fifth location in the array.

So, when I say f plus 2 f plus 2 is a pointer 2 here and t plus 4 is a pointer 2 here and I am calling copy array function with these as the arguments and n is the number of elements I want to copy.

(Refer Slide Time: 07:45)

```
int copy_array_2(int f[], int i,
        int t[], int j, int n) {
    copy_array(f+i,t+j,n);
    return 0;
}
```

```
int copy_array(int a[],
        int b[], int n) {
    int i;
    for(i=0; i<n; i =i+1) {
        b[i] = a[i];
    }
    return 0;
}
```

So, here is the state just before I call, copy array function. Now, for example, this particular box has several names, the most common name for it will be f of 3. But, I can also write it as star of f plus 3, this says jump 3 integer boxes after f and then dereference that address. Now, if you are comfortable with the notion that let us say f of i is the same as star of f plus i.

If you are comfortable with that notion, then you should be easy to see that f plus 2 of 1 is just star of f plus 2 plus 1, It is the same formula that I am using and this happens to be f plus 3, which happens to be f of 3. So, star of f plus 3 would be f of 3 and so, on. So, this formula that f of i is the same as dereferencing the address f plus i, f of i is star of f plus i is applicable even for more strange looking expressions.

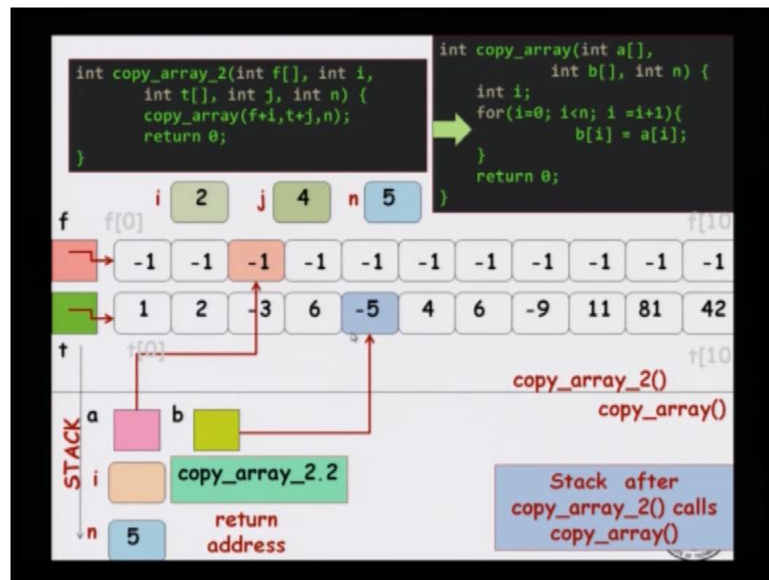
(Refer Slide Time: 09:26)

Now, here is a trivia about c, that because of the way it is defined. So, if you say that f of i is the same as star of f plus i, then you could think that this is the same as star of i plus f. So, I can write this as i of f never do this, but it will actually work. So, f of i you can also write it as i of f. For example, 3 of f and it will also work, because internally c translates it to star of f plus i and we know that star of f plus i is a same as star of i plus f. So, never do this, but this helps you to understand that f of i is being translated by c into this format.

(Refer Slide Time: 10:27)

So, now, that we know this, similarly you can argue about star of t plus 4 and t plus 4 of zero and. So, on all of them refer to the same box.

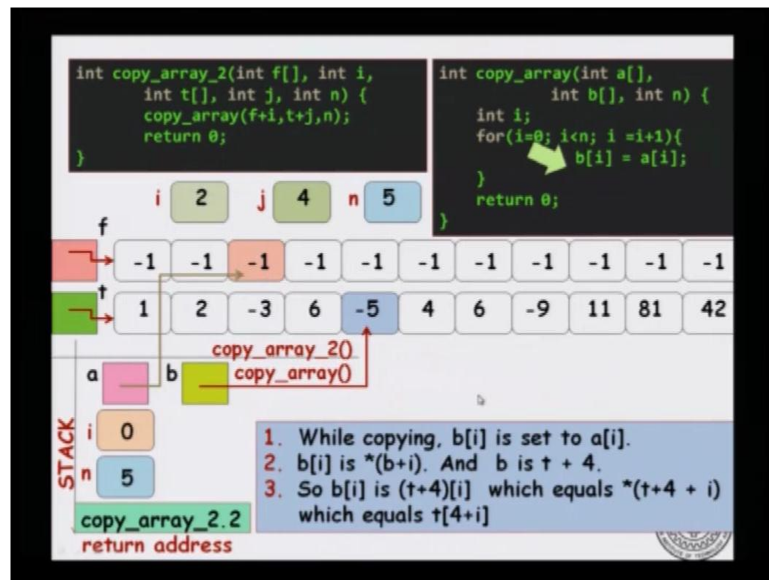
(Refer Slide Time: 10:36)



Now, let us see what happens when we call copy array. So, we have the stack space for copy array 2 and copy array 2 calls copy array. The formal parameters are a and b, a copies the address it was passed to, it was passed the address f plus i. So, a points to f plus 2, similarly b points to t plus 4,. So, b points to this. Now, as for as copy arrays concerned it is not too bothered by the fact that, it was not passed the absolute first address of the array. It will think that whatever address it has been passed is the start of an array and it will work from there.

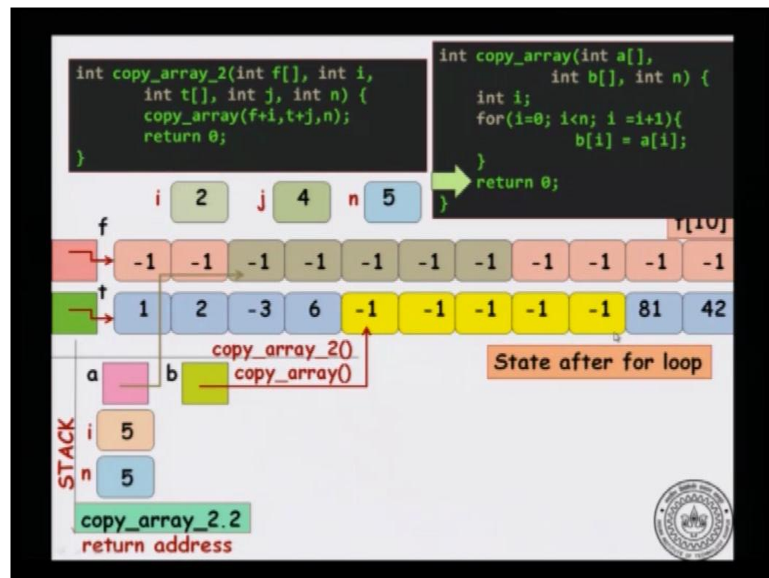
So, copy array does not bother the fact that, I was given the second element of the array, rather than the first element of the array and so, on, it will work as though the array started from there. And here is where we are exploiting that fact so, you now copy n elements from this location to this array, so, n is 5.

(Refer Slide Time: 12:06)



Now, when you execute that you will have copy array 2, should copy 5 elements starting from here to 5 locations here. So, that is what it will actually do.

(Refer Slide Time: 12:18)



And when you execute the loop, it will start from this location copied to t plus j then f plus i plus 1 will be copied to t plus j plus 1 and so, on. So, it will copy these five locations to here in the t array.

(Refer Slide Time: 12:42)

And after you do this copy array returns and every variable that was allocated to copy array is erased. But, then because of pointers it was actually working with the arrays in copy array 2. So, even when you erase all the memory allocated to copy array, once you return these arrays would have been changed. These five locations starting from f plus 2 have been copied to these five locations starting from t plus 4. So, changes made to b by copy array, is still maintained after you returns to the calling function copy array 2.