Introduction to Programming in C
Department of Computer Science and Engineering

In this lecture will see some more pointer arithmetic operators, and we will introduce those by talking about them through a problem.

(Refer Slide Time: 00:11)

More Pointer Operations - Reversing an Array

Write a function to reverse an array

The declaration of the function is

```
void rev_array ( int a[], int n );
```

The function takes an integer array and reverses the array *in place* - that is, without using any extra space.

before calling rev_array

a[0] a[1] a[2] ... a[n-2] a[n-1]

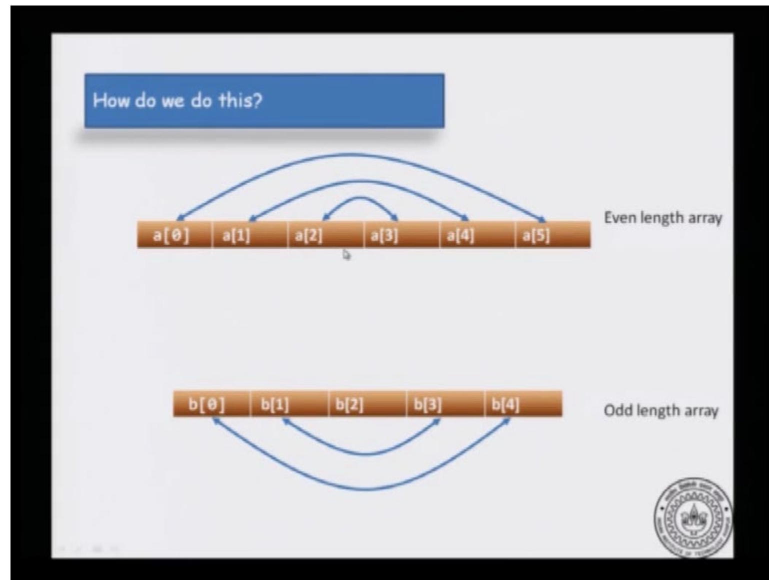
After calling rev_array

a[n-1] a[n-2] ... a[2] a[1] a[0]

So, the problem that I have is that of reversing an array. So, we have to write a function to reverse an array, and let say that the declaration of the function is void rev_array (int a[ ], int n[ ]). Now if you have to reverse an array, what is one way to do it you take the array copy to into another array, and then copy back in the reverse fashion. So, you have an array a[ ] copy all the values in to b[], and now you copy those values back to a in the following way that the b, b is last value will go to a[0], b is second value last value will go to a[1] and so on. Now, let us try to do it slightly more cleverly, we want to take an integer array and reverse the array in place.

That means that essentially using no extra space. So, do not use and extra array in order to reverse it, reverse it within a itself. So, the array before calling reverse array will look like a[0] upto a[n-1] in this way. And after calling reverse array it should look like a[n-1] a[n-2], etcetera up to a[0], and we doing that we should not use an extra array.

(Refer Slide Time: 01:38)



So, how do we do this, let us look at bunch of couple of concrete examples to see how do it by hand, and then we will try to code that algorithm. So, in this there are 2 cases. So, for example, what happens when you having even length array, when you have let say 6 elements. Then a[0] and a 5 have to the exchange, a 5 goes to a[0], a 5 goes to the 0 th location, a[0] goes to the sixth location of the fifth location. And then a[1] and a[4] have to be exchanged, a[2] and a[3] have to be exchanged, after this you should stop, that will correctly reverse the array. Now what happens if the case of in the case of an odd length array, suppose you have an array b which has only 5 elements. In that case to reverse the array you have to exchange b[0] and b[4] b[1] and b[3], and you can stop there, because does no need to exchange b[2] with itself. So, the case of an odd length array, you will end of with n element which does not need to be touched, in the case of an even length array, you have to exchange until you reach the middle of the ((Refer Time: 03:00)).

(Refer Slide Time: 03:05)

```
void rev_array ( int a[], int n )
{
        /* pointer to last element of a */
        int* b = a+n-1;
        while ( b > a ) {
                swap ( a, b );
                a = a+1;
                b = b-1;
        }
}
```

```
void swap ( int* ptra,
            int* ptrb )
{
        int t = *ptra;
        *ptra = *ptrb;
        *ptrb = t;
}
```

OK, but what does b > a mean?

Design Logic

1. a initially points to the first element (left end) of the array.
2. b initially points to the last element (right end) of the array.
3. a moves forward, b moves backward.
4. In the loop, exchange *a with *b. Repeat this until a and b cross over, or until a and b become equal.

Let us try to code this up. So, how is the reverse array return first I need... So, remember how we did this by hand, we exchange the 0 th location with the last location, then we exchange the first location within second last location, and so on. So, it is easy to code, if you have two pointers; initially one pointers starts to at the beginning of array, the second pointer is to the last of the array exchange those values, then the first pointer goes to the next location, and the second pointer goes to the previous location, that is how you did it by hand. So, let us try to code that up I will have a pointer *b, which points to the last element of array a + n - 1.

Now, the loop is as follows, I will discuss this in a minute while b < a. So, remember in the example by hand, we had to exchange till we reach the middle of the array. How do you find the middle of the array, I will just write it as b > a, and I will explain it in a minute. So, while this is true that you have not a ((Refer Time: 04:25) the middle of the array. You exchange swap a and b, here we use the swap function which we have seen in the previous lecture. So, for example, it will swap the 0 th element with the n - first element. After that is done you increment a and you decrement b. So, the design logic is that a initially points to that first element of the array - the left end of the array, and b points to the right end of the array.

In general while the algorithm happens then a is moving forward, and b is moving backward. Inside the loop you exchange *a with *b; that is what is accomplish by callings swap(a , b), because swap a and b will dereference goes locations and exchange

the values there. So, do this repeatedly until a and b cross over, because them then a is moving forward and b is moving backward, then the middle of the array is the point where a and b cross over or the point where a and b meet. So, this is the very simple logic for reversing an array. Now I have left one thing unexplained. What do I mean by b > a, b and a are pointers. So, what do I mean by pointer b < pointer a, we need to explain that. We are introducing an operation on operator and two pointers, what does it mean?

(Refer Slide Time: 06:07)

```
void rev_array ( int a[], int n )
{
        /* pointer to last element of a */
        int* b = a+n-1;
        while ( b > a ) {
                swap ( a, b );
                a = a+1;
                b = b-1;
        }
}
```

```
void swap ( int* ptra,
            int* ptrb )
{
        int t = *ptra;
        *ptra = *ptrb;
        *ptrb = t;
}
```

OK, but what does b > a mean?

Relational Comparison between Pointers

Let a and b be pointers to variables of the same type (e.g. int *a; int *b;). Then

1. We can compare pointers using == and !=
2. a==b is true if and only if they are pointing to the same box in memory
3. Otherwise a != b is true.

So, we are seeing a new concept which is relational comparison between two pointers. If a and b are pointers to variables of the same type like int *a, int *b. We can compare them, compare these pointers using =, and not =. This can be done for arbitrary locations a, and b as long as those locations are of the same type. So, a = b is true, if and only if a and b are pointing to the same location; that is natural to expect. Otherwise if they are pointing to different locations a not equal to b is true. Now there is another case, if a is pointing to an integer let say, and b is pointing to a float then equal to and not equal to are undefined. So, notice that even though this behavior looks natural, it is natural only if there pointing to this same type. So, here are operations equal to and not equal to.

(Refer Slide Time: 07:15)

```
void rev_array ( int a[], int n )
{
        /* pointer to last element of a */
        int* b = a+n-1;
        while ( b > a ) {
                swap ( a, b );
                a = a+1;
                b = b-1;
        }
}
```

```
void swap ( int* ptra,
            int* ptrb )
{
        int t = *ptra;
        *ptra = *ptrb;
        *ptrb = t;
}
```

Relational Comparison between Pointers

OK, but what does b > a mean?

Comparing pointers using <, <= etc.

1. We can compare two pointers a and b using <. For this, they must be pointing to locations in the same array.
2. a < b is true if and only if a is pointing to an array location with index less than the index of the array location pointed to by b.
   e.g. We can say a+1 < a+2 in an array int a[10];

What about less than less than or equal to greater than greater or equal to and so on. And this is surprising, because here is something that you do not expect. You cannot compare less than less than   equal to on arbitrary locations in the memory. We can compare a and b using less than for this they must be pointing to the same locations in the array. Earlier when we discussed + and -, we were saying that + and - are well behave the only when you are navigating within an array.

Similarly, when we are comparing 2 pointers using greater than greater than or equal to less than less than or equal to, then they should all be point then a and b should be pointing to the same array, different locations in this same array. If that is true then a < b, if a is pointing to a location which is before b in the same array. Similarly a < or = b yes true if a is pointing to a location which is b or before b, and so on. So, for example, we can say that if you have an array int a[10], then a + 1 < a + 2, that is clearly true. Because a + 1 is pointing to the location one in array and a + 2 is pointing to location two in the array.

(Refer Slide Time: 08:57)

```
void rev_array ( int a[], int n )
{
        /* pointer to last element of a */
        int* b = a+n-1;
        while ( b > a ) {
                swap ( a, b );
                a = a+1;
                b = b-1;
        }
}
```

```
void swap ( int* ptra,
            int* ptrb )
{
        int t = *ptra;
        *ptra = *ptrb;
        *ptrb = t;
}
```

Relational Comparison between Pointers

OK, but what does b > a mean?

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]

ptra ptrb

Here, ptra < ptrb is true

So, if you have an array for an example let say a[0] through a[9], and ptra is pointing to location one, and ptrb is pointing to location three. Then ptra < ptrb here the comparison is well defined and it is true.

(Refer Slide Time: 09:20)

```
void rev_array ( int a[], int n )
{
        /* pointer to last element of a */
        int* b = a+n-1;
        while ( b > a ) {
                swap ( a, b );
                a = a+1;
                b = b-1;
        }
}
```

```
void swap ( int* ptra,
            int* ptrb )
{
        int t = *ptra;
        *ptra = *ptrb;
        *ptrb = t;
}
```

Relational Comparison between Pointers

OK, but what does b > a mean?

a[0] a[1] a[2] a[3] a[4]

b[0] b[1] b[2] b[3]

ptra ptrb

Here, ptra < ptrb is **undefined** (they are not pointing to the elements in the same array.)

But on the other hand let say that ptra is pointing to a[1], and ptrb is pointing to b[1]. In this case ptra < ptrb is undefined, because their pointing to two different arrays. So, may be in memory a is lead out before b and so on, but that is not what the < or = operation is supposed to do. It is suppose to compare pointers only within the same array.

(Refer Slide Time: 09:55)

So, with this understanding let us understand how the reverse array works. So, in the first iteration, you have an array a[ ], let say that array is 101, 21, and so on, it has 6 locations. And we will run through that trace of the execution for an even length array, and I would encourage you to create and odd length array, and trace to the executions to ensure that the code works for odd length arrays as well. So, in this lecture will do it for an even length array. So, a is initially pointing to the beginning of the array, b is pointing to the end of the array, a + n - 1 will go to the end of the array. Now b < a that is true. So, we will enter the loop and in the first iteration we will swap a and b.

(Refer Slide Time: 10:49)

So, it will go to the swap function, and this is the swap function that actually works from the previous video. So, you can assume that a[0] will be swapped with a[1]. So, they were initially 101 and 0, and after swap they will be 0 and 101. Come once that happens a advances by one integer location b goes back by one integer location. So, this the state after the first iteration. In the second iteration you start with a at 21, and b at - 101, again b < a, so is swap.

(Refer Slide Time: 11:41)

```
void rev_array ( int a[], int n )
{
        /* pointer to last element of a */
        int* b = a+n-1;
        while ( b > a ) {
                swap ( a, b );
                a = a+1;
                b = b-1;
        }
}
```

```
void swap ( int* ptra,
            int* ptrb )
{
        int t = *ptra;
        *ptra = *ptrb;
        *ptrb = t;
}
```


So, 21 - 101 becomes - 101 and 21, so they are swapped. And you advance a by 1 and you take back b by 1, again b < a.

(Refer Slide Time: 12:02)

So, you go to the third iteration. In the third iteration this the state of the beginning of the iteration, and the swap these contents. So, you swap - 1 and 121 it becomes this becomes the state of the array, and once that is done b over suits. So, b b goes before a, and a go goes after b, swap denoted there with two colored arrows. So, here is the b arrow, it goes to location 121, and a arrow goes to location - 1. When this happens b is now < a. So, this means that you have cross the middle of the array. Therefore, you should stop now. So, now b < a, and the loop terminates, and we have seen that this correctly reverse the array. So, here is how the reverse array works, we have seen the concept of relational comparison operators using pointers. How do make sense when they are pointing to locations with them the same array, and how that can be used to write code to using arrays.