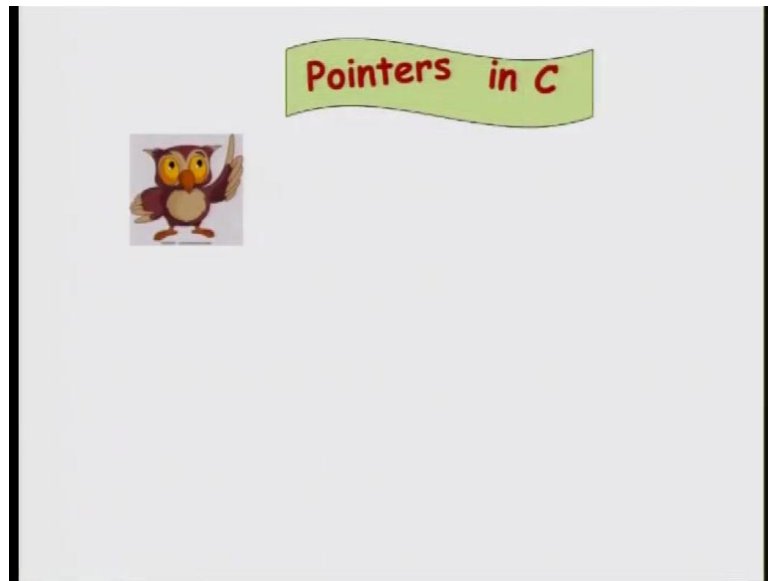


Introduction to Programming in C
Prof. Satyadev Nandakumar
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 32

(Refer Slide Time: 00:09)



In this video, we will start with a topic that is considered one of the trickiest concepts in C. These are what are known as pointers. We will just recap what we know about arrays, because arrays and pointers are very closely related in C.

(Refer Slide Time: 00:23)

The memory allocated to array has two components:

- A consecutively allocated segment of memory boxes of the same type, and
- A box with the same name as the array. This box holds the address of the base (i.e., first) element of the array.

too much theory. Give examples please

OK, Consider the definition.

`int num[10];`

num num[0] num[1] num[2] ... num[9]

This definition for num[10] gives 11 boxes, 10 of type int, and 1 of type address of an int box. *hmm...*

1. We represent the address of a box x by an arrow to the box x . So addresses are referred to as pointers.
2. The contents of an address box is a pointer to the box whose address it contains. e.g., num points to num[0] above.

The memory allocated to any array has two components. First is there are a bunch of consecutively allocated boxes of the same type. And the second component is there is a box with the same name as the array. And this box contains the address of the first element of the array. So, that let us be clear with the help of concrete example. So, let us consider a particular array of size 10 declared as `int num 10`. Conceptually, there are 10 boxes from num 0 through num 9. These are all containing integers. Plus there is an additional eleventh box – num. So, it has the same name as the name of the array. And it contains the address of the first location of the array. So, it contains the address of num 0. These are num 0 through num 9 are located somewhere in memory. So, maybe this is memory location 1000. So, num contains the number 1000, which is supposed to indicate that, the address of the first location in the array is 1000 or num points to the memory location 1000. So, conceptually, this gives 11 boxes, which are 10 integer boxes plus 1 box, which holds the address of the first box.

Now, we represent the address of a box x by an arrow to the box x . So, addresses are referred to as pointers. And this is all there is to C pointers. Pointers in some sense are variables, which hold the addresses of other variables. That is an exact description of the concept of pointers. Now, we will see now what that means and what can we do with pointers.

(Refer Slide Time: 02:23)

What can we do with a box? e.g., an integer box?

```
int num[10];
```

True! But we can also take the address of a box. We do this when we use scanf for reading using the & operator.

ptr would be of type address of int. In C this type is `int *`.

```
int * ptr;  
ptr = &num[1];
```

That's simple. We can do operations that are supported for the data type of the box.

For integers, we can do + - * / % etc. for each of num[0] through num[9].

OK. Say i want to take the address of num[1] and store it in an address variable ptr.

```
ptr = &num[1];
```

But what is the type of ptr? And how do i define ptr?

Let us just step back a minute and say what can we do with a particular box or particular variable in memory, which is an integer. So, that is very simple. For example, you can scan f into that box; you can print the value in that box; you can do arithmetic operations on that box like plus, division, modulo, and so on. And you can do this for each of the boxes from num 0 through num 9, because each of them by itself is in integer. But, we will also see a new operation, which is that, you can take the address of a box. So, we have already done this when we did scan f. So, we mentioned AND of a variable. So, we will see these AND operator in somewhat more detail.

So, suppose I want to take the address of num 1 and store it in an address variable ptr. So, what I am essentially saying is that, you can say ptr equal to AND of num 1. So, num 1 is an integer box; it is an integer variable; AND of num 1 is the address of that integer in memory. So, you assign it to the variable ptr. But, every variable in C needs to have a type. What is the type of ptr? And how do you declare or define such a type – such a variable? Now, ptr holds the address of an integer. In C, you denote that by saying that, the type of ptr is int star. So, here is a new type that we are seeing for the first time. We are saying ptr is of type int star. Just like you can say that, if I have int a, you can say that, a is of type int. In this case, we can say ptr equal to AND of num 1.

(Refer Slide Time: 04:40)

To see the meaning of `ptr = &num[1]`, let's look at the memory state.

Here is the state after `int num[10]` gets defined.

```
int num[10];
int * ptr;
ptr = &num[1];
```

OK, `ptr` is of type pointer to integer. But what does `ptr = &num[1];` mean?

The statement `int *ptr;` creates a new box of type "address of an int box", more commonly referred to as, of type "pointer to integer".

The statement `ptr = &num[1];` assigns to `ptr` the address of the box `num[1]`. Commonly referred to as: `ptr` now points to `num[1]`.

We have discussed right now we have `int num 10`, `int star ptr`, and `ptr` equal to `&num 1`. So, `ptr` is the pointer to an integer. But, what does `ptr` equal to `&num 1` really mean? So, let us look at the memory status once we declare this array. So, we have `num`, which is the address of the first location. And then we have somewhere in memory, we have 10 consecutive locations corresponding to the array – `num 0` through `num 9`. Now, I declare `int star ptr`. So, I create a box. Now, this box is supposed to hold the address of some integer variable. So, `ptr` is of type address of an integer box or more commonly referred to as pointer to integer. The statement `ptr = &num 1` says that, now, `ptr` points to `num 1` or `ptr` contains the address of `num 1`. And pictorially, we denote an arrow from `ptr` to `num 1` just like I denoted an arrow going from `num` to `num 0`, because the name of the array is a pointer to the first location of the array. The name of the array is a box, which holds the address of the first location of the array. Similarly, `ptr` is a box, which holds the address of `num 1`. So, we say that, `ptr` points to `num 1`. And we denote it pictorially by an arrow.

(Refer Slide Time: 06:11)

OK, i see. The program fragment below results in this memory state.

```
int num[10];
int * ptr;
ptr = &num[1];
```

Suppose I now add the following statement after above fragment

```
scanf("%d", ptr);
```

scanf("%d", &num[i]);

question

The program status is like this – state is like this. Now, suppose I add one more statement after all these three statements; I say scan f percentage d ptr. Now, earlier when we declared an array and we read into an array directly, I said that, you can do the following. I can write scan f percentage d and num 1. So, this will value whatever the user input into the first array using the AND operator. Now, ptr equal to AND num 1. So, it is a reasonable thing to ask – can I say scan f percentage d ptr? There is no AND operator here because ptr is AND num 1. This was our original statement and this is our new statement.

(Refer Slide Time: 07:15)

OK, i see. The program fragment below results in this memory state.

```
int num[10];  
int * ptr;  
ptr = &num[1];
```

num [0] num[1] num[2] ... num[9]

ptr

5

question

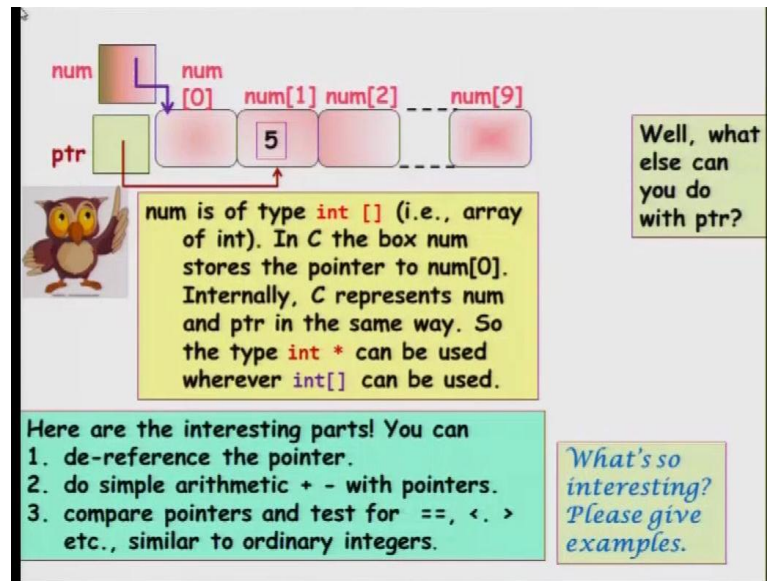
1. Yes! `scanf("%d", ptr)` reads input integer into the box pointed to by the corresponding argument.
2. The box pointed to by `ptr` is `num[1]`.
3. So `num[1]` becomes 5.

Suppose I now add the following statement after above fragment

<code>scanf("%d", ptr);</code>	Input
and input is :	5
Does <code>num[1]</code> become 5?	

And the answer is yes, you can do it. Suppose the input is 5, does num 1 become 5? So, scanf percentage d ptr really does work like scanf percentage d and num 1. So, it reads the value input by the user and it looks up ptr. So, it is an address. So, it goes to that address and stores it there. So, now, we can clarify a long standing mystery, which is the AND operator in the case of scanf. So, we can say that, scanf second argument is a pointer; which says where should I put the input by the user? For example, if I have float variable and I scanf as percentage f and then sum address of a float variable, it is done similar to reading an integer into an integer variable. What scanf takes is an address of int variable or float variable as it may be. If you have a percentage d, then it takes a pointer to an integer variable and takes the input value by the user and puts it into that address. So, as far as scanf is concerned, it does not matter whether you gave it as AND num 1 or whether you initialized ptr to AND num 1 and then gave ptr. It is an address and it will put the integer input by the user into that location. So, num 1 indeed does become 5.

(Refer Slide Time: 08:55)

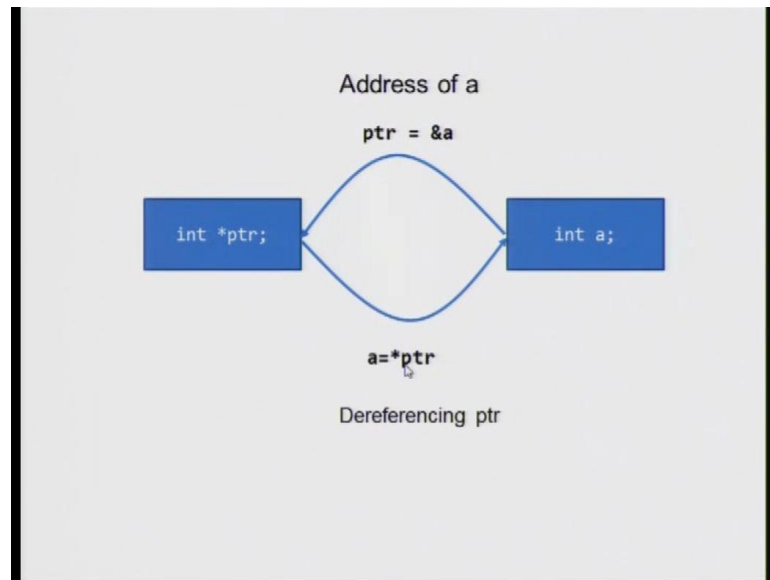


So, the location of the num 1 is now containing the value 5. Now, what else can you do with a pointer? Num is an array; it is of type int square brackets – pair of square brackets. And in C, the box num contains the address of the first location of the array, which is num 0. So, internally, as far as C is concerned, the address of num 0 is just like address of any other integer location. So, the type int star can be interchanged with int square bracket. So, you can think of num itself as just a pointer to an integer; or, you can say that, it is a pointer to an array; which gives you the additional information that, the next 10 values are also integers. If you just say pointer to an integer, the next location may be something else. But, internally as far as C is concerned, an array name num can also be treated as pointer to an integer.

Now, here are some other interesting things that you can do with pointers. Whenever you declare a data type, you also define what all operations can you do with a date type. So, 2 and 3 are fairly simple; we have already seen it with integers, floating points and so on. You can do simple arithmetic plus and minus with pointers. You cannot do star and divide. You cannot do that. But, you can do plus and minus. Similarly, if you have two pointers, you can test for equality, you can test for less than, you can test for greater than and so on as though you are comparing ordinary integers. So, 2 and 3 are what we have seen before; except that, in 2, you cannot do multiply and divide and modulo. All these

things are not done with pointers. But, addition and subtraction can be done. But, there is a new operation, which is dereferencing a pointer. We have not seen this operation before with earlier data types.

(Refer slide Time: 11:10)



What is dereferencing? Let me pictorially represent what it does. Suppose you have `int star ptr` and `int a`. So, `a` is an integer variable and `ptr` is a pointer to `int`. If I want to store the address of an `int ptr`, I do it as follows: `ptr equal to AND a`. So, this means that, take the address of `a` and store it in `ptr`. So, now, you can say that, `ptr` points to `a`. Now, I can also think of a reverse operation; which is `ptr` contains some address. Go look up that address; so that will be an `int`. And store that value in `int`.

So, that is what is known as the star operator – `a equal to star ptr`. This means that, `ptr` is an integer pointer. So, `ptr` will point to a location, which contains an integer. Star `ptr` will take the contents of that location and store it in `a`. So, this is known as the dereferencing operator. So, the address operator takes an integer variable and stores the address in a pointer. The dereferencing operation takes a pointer; looks up that address; and stores the value in `a`. So, you can visualize the AND operator and the star operator as sort of reverse operations of each other. AND takes an integer and takes the address of that; star takes a pointer and takes the value of the address pointed to that.

(Refer Slide Time: 12:52)

De-referencing a pointer `ptr` gives the box pointed to by `ptr`. The de-referencing operator in C is `*`. *Hmm...*

```
printf("%d", *ptr);
```

Output 5

Since `ptr` points to `num[1]`, `*ptr` is the box `num[1]`. Printing it gives the output 5.

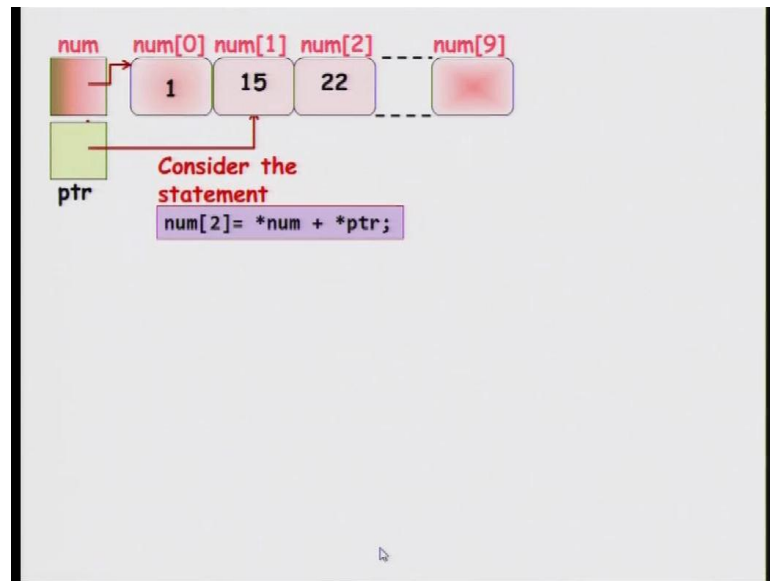
Consider statement `*ptr = *ptr + 5;`

This will add 5 to the value in box pointed by `ptr`. So `num[1]` will become $5+5 = 10$

Dereferencing a pointer therefore gives you the value contained in the box pointed to by the pointer. The dereferencing operator is star. So, if I say `printf("%d", *ptr);`, what it will do is – look up the location pointed to by `ptr`. In this case, it is this integer box. The contents of that box is 5 and it will be printed. So, the output will be 5. Not for example, the content of `ptr`. So, the content of `ptr` may be like 1004; it will not print 1004; but what it is supposed to do is look up the location 1004; it contains the value 5; print that value. So, `*ptr` is the box `num[1]`. And printing it gives you the output 5.

Now, can I consider a statement like `*ptr = *ptr + 5;`? This is perfectly legal. What this will do is `*ptr` is an integer value. It is equal to 5, because look up this location `ptr`; that is an integer; take that value; which will be 5. So, this will be 5 plus 5 – 10. And where do you store it? You store it in the integer variable corresponding to `*ptr`. The integer variable corresponding to `*ptr` is `num[1]`. So, I would have normally said `num[1] = *ptr + 5;` but `num[1]` is the same as `*ptr`. So, I can say `*ptr = *ptr + 5;`. So, this will look up that location; add 5 to its contents; and store it in that location. So, `num[1]` will now become 10.

(Refer Slide Time: 14:46)



Similarly, you can consider other examples. For example, I can consider a statement like `num[2] = *num + *ptr;`. The novelty here is that...