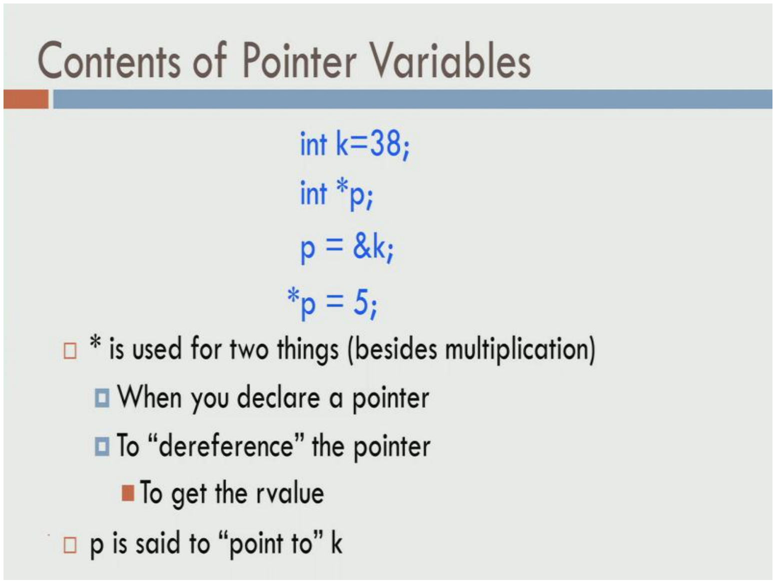
Programming, Data Structures and Algorithms
Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module - 9C
Lecture - 15
Contents of a pointer variable
Dereferencing operator, Null pointer
Pointer types, Typecasting and Examples
Pointers and arrays, Pointer arithmetic

In this module, we will see a little bit more of details on pointers itself.

(Refer Slide Time: 00:20)



Contents of Pointer Variables

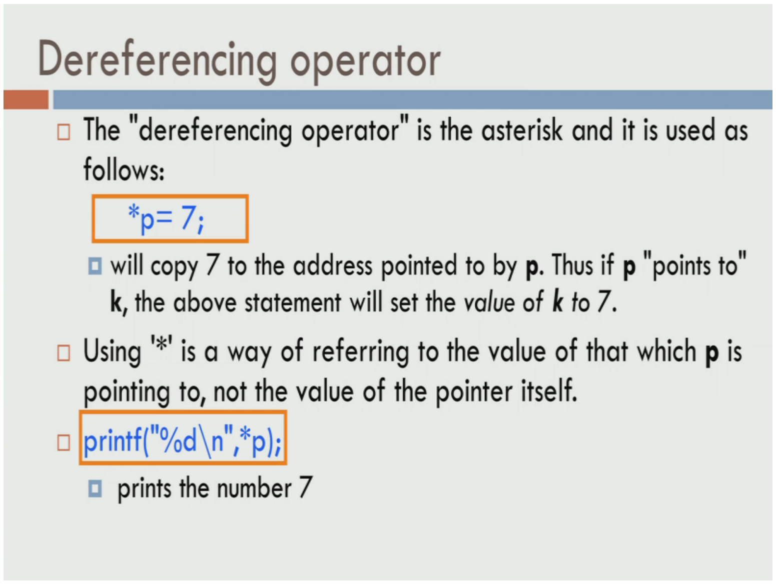
```
int k=38;
int *p;
p = &k;
*p = 5;
```

- * is used for two things (besides multiplication)
  - When you declare a pointer
  - To "dereference" the pointer
    - To get the rvalue
- p is said to "point to" k

So, let us go back to the code that we saw earlier. So, we have int k equals to 38, int star p and p is ampersand of k, star p equals 5. So, one thing that I did not emphasize earlier is this notion of asterisk, what is it used for. So, you can see that it is used in two places, in one place as int star p and the other place, we have star p equals 5. So, they are related to pointers, this asterisk symbol is used in two connotations. First time when you declare a pointer, you say int star p and that point you are saying that p is a variable of type pointer to integer. And later when you say star p equals to 5, the meaning is slightly different. What you are doing is, you are doing what is called dereferencing the pointer. So, at this point this is not telling the compiler to allocate space for the variable p or anything like that. Here, actually we are dealing with the r value of p.

So, one small thing that you will see in the course as well as elsewhere is that, we usually say p is pointing to k. So, in this case p is ampersand of k. So, instead of using this technical term, the p equals ampersand of k, we usually say that p points to k or p is pointing to k and so, on.

(Refer Slide Time: 01:48)



Dereferencing operator

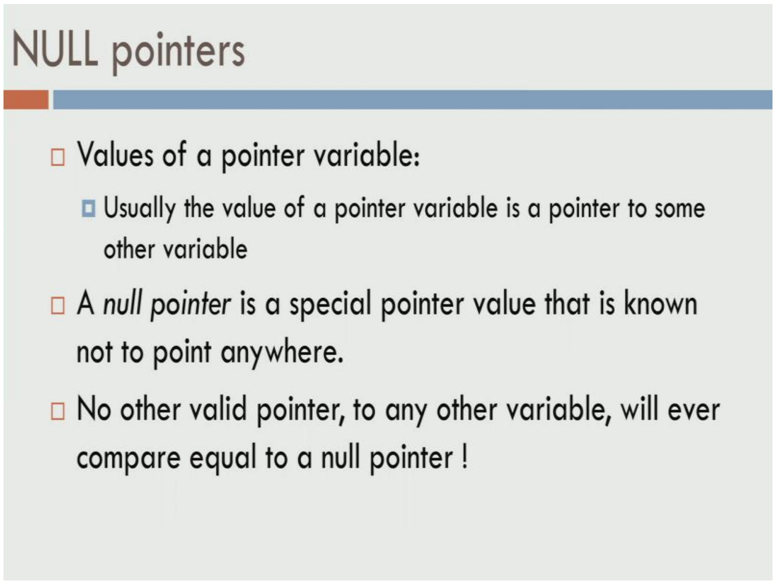
- The "dereferencing operator" is the asterisk and it is used as follows:

```
*p= 7;
```

  - will copy 7 to the address pointed to by **p**. Thus if **p** "points to" **k**, the above statement will set the *value* of ***k*** *to* 7.
- Using '*' is a way of referring to the value of that which **p** is pointing to, not the value of the pointer itself.
- `printf("%d\n",*p);`
  - prints the number 7

So, let us look at a dereferencing operator. So, it is the asterisk symbol and when you say star p equals to 7, it takes the value 7 and copies it to the address pointed to by p. So, we saw this, if p points to k, then the above statement will actually change the contents of the k also to 7. And this dereferencing is something that you will see as a technical term, that is used later in the lecture also.

(Refer Slide Time: 02:18)

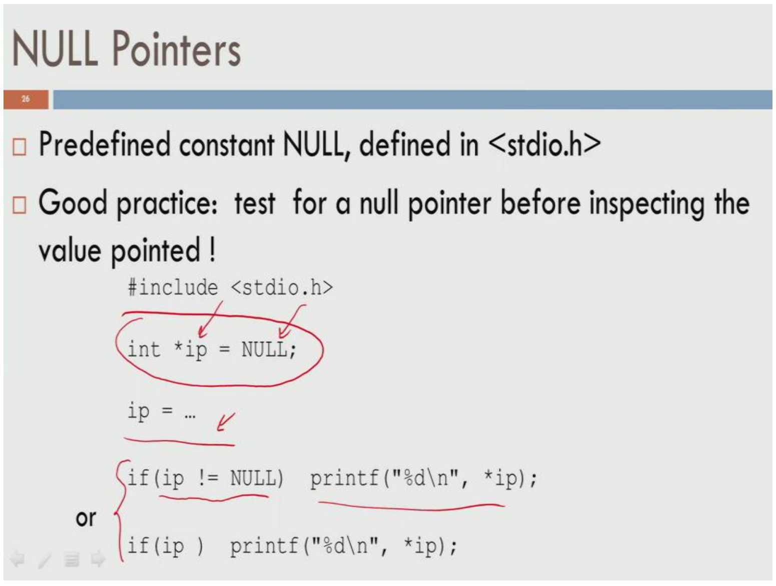


NULL pointers

- Values of a pointer variable:
  - Usually the value of a pointer variable is a pointer to some other variable
- A *null pointer* is a special pointer value that is known not to point anywhere.
- No other valid pointer, to any other variable, will ever compare equal to a null pointer !

There is a special pointer called the null pointer and generally we use pointers to point to some specific variable. So, if I have int star p, then I did p equals ampersand k in that case, p started pointing to k. But, there are several cases where you explicitly want the pointer, not to point at anything at all. And C provides you a special way of doing that and that is called the null pointer. So, one key thing about the null pointer is that no other valid pointer. So, by that I mean, no other pointer that is actually storing some value or which is going to get point, which is an address of a variable and so, on, will ever compare equal to a null pointer. So, let us see what it means.

(Refer Slide Time: 03:11)



NULL Pointers

- Predefined constant NULL, defined in <stdio.h>
- Good practice: test for a null pointer before inspecting the value pointed !

```
#include <stdio.h>

int *ip = NULL;

ip = …

if(ip != NULL)  printf("%d\n", *ip);
or
if(ip )  printf("%d\n", *ip);
```

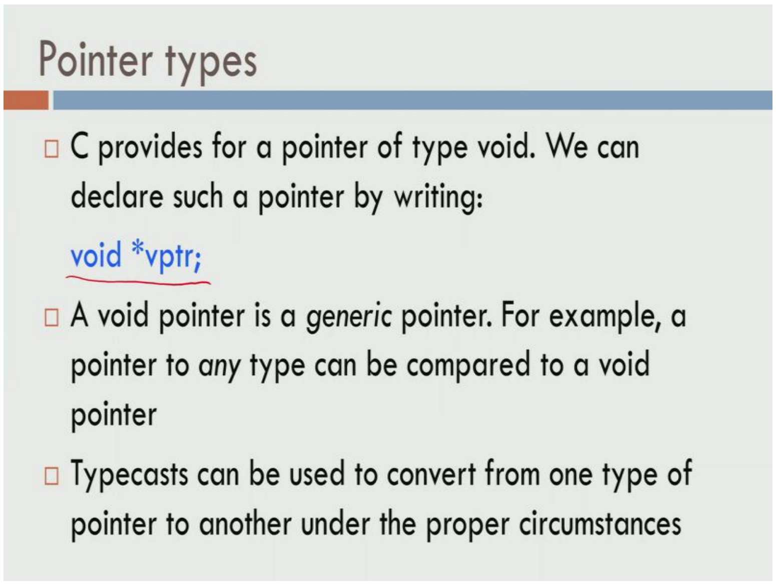
So, first of all C gives something called a predefined constant called null and this is defined in stdio dot h. So, just like printf and scanf, this constant null is also defined in stdio dot h. So, generally it is a good practice to check whether a pointer is null or not, before you use it. And we will see this in lot more detail, when you do what is called dynamic memory allocation. But, let us look at this code for right now.

So, what we have is, we have hash include stdio dot h and we have int star ip equals null. So, this is generally considered a very good programming practice, you not only said, you want a pointer variable of type integer pointer. The name is ip and the pointer is of an integer type, you also said here, that it is not pointing to any valid location. So, if you do this, then we never run into this problem of ip pointing at an unknown location and what happens in this unknown location and so, on.

So, the moment you have null, it means that it is not valid location and you will probably come back and change it later. So, you may do ip equals address of something later and this is what I was talking about earlier. So, this tip says, if ip is not equal to null printf star ip or if ip printf star ip and so, on. So, what this does is, it actually checks whether ip is null or not. If I know that it was initialized to null and at this point if ip is not equal to null, then there was a valid assignment ip that happen here.

If there is no valid assignment, ip will still remain at null and you will not print the value. So, this is a check that is done usually in practice. Again, we will see how this is useful when we do, what is called dynamic memory allocation. So, we have not done that yet,, but when we do malloc, we will use this template later.

(Refer Slide Time: 05:22)



Pointer types

- C provides for a pointer of type void. We can declare such a pointer by writing:

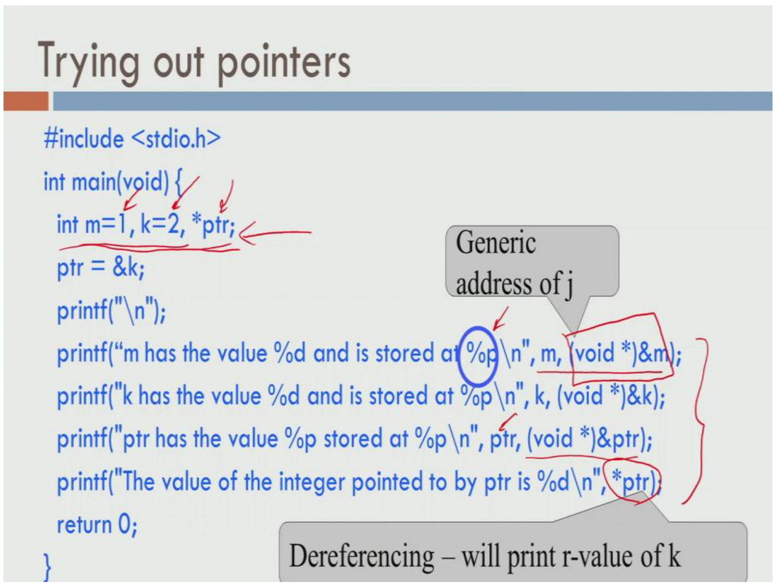
  void *vptr;

- A void pointer is a *generic* pointer. For example, a pointer to *any* type can be compared to a void pointer
- Typecasts can be used to convert from one type of pointer to another under the proper circumstances

So, another settle thing with C is that, it provides a pointer of type void. So, void is,. So, you declare it by saying void star v ptr and this void pointer is actually a generic pointer. What I mean by that is, it can be made to point at anything. So, far we saw integers, star pointer,. So, you could also have float star, ptr 1 and so, on. But, this void star v ptr is a generic pointer.

You can make it point to an integer, you can make it point to a floating point, you can make it to point a character or you can actually make it point to a pointer itself and so, on. So, this is another thing that comes very handy and this is just for you to notice now, we will see this in detail later.

(Refer Slide Time: 06:17)



Trying out pointers

```
#include <stdio.h>
int main(void) {
  int m=1, k=2, *ptr;
  ptr = &k;
  printf("\n");
  printf("m has the value %d and is stored at %p\n", m, (void *)&m);
  printf("k has the value %d and is stored at %p\n", k, (void *)&k);
  printf("ptr has the value %p stored at %p\n", ptr, (void *)&ptr);
  printf("The value of the integer pointed to by ptr is %d\n", *ptr);
  return 0;
}
```

So, let us look at a small code segment using pointers. So, here we have two integer variables m and k and we have a variable ptr, which is actually of the type, integer pointer. So, one thing you can notice is that in this single line, we have done a few things, we have declared and initialized two variables m and k, we have also declared a pointer variable of integer pointer type. So, you can combine declarations in one line and this is the way to declare both integer variables and pointers to integers in the same line.

So, in this case ptr is made to point at k. So, ptr equals ampersand k is made to point,. So, you are making ptr to point at k. So, let us look at these four lines of code,. So, the first line says m has the value percentage d and is stored at percentage p and we give two parameters as inputs to printf. So, the first thing you notice is, this percentage p is a format specifier for pointers. So, just like we have percentage d for integers, percentage f for floating point and so, on.

For pointers, it is recommended that you use percentage p and. So, let us see what happens in this line. You have percentage d as a specifier that will attach it to m. So, here you want m printed as percentage d format, which means print m as an integer and the second argument is void star ampersand of m. So, this is the case of what is called typecasting. So, ampersand of m is of pointer type and it is an integer point type, you typecast that into what is called void star.

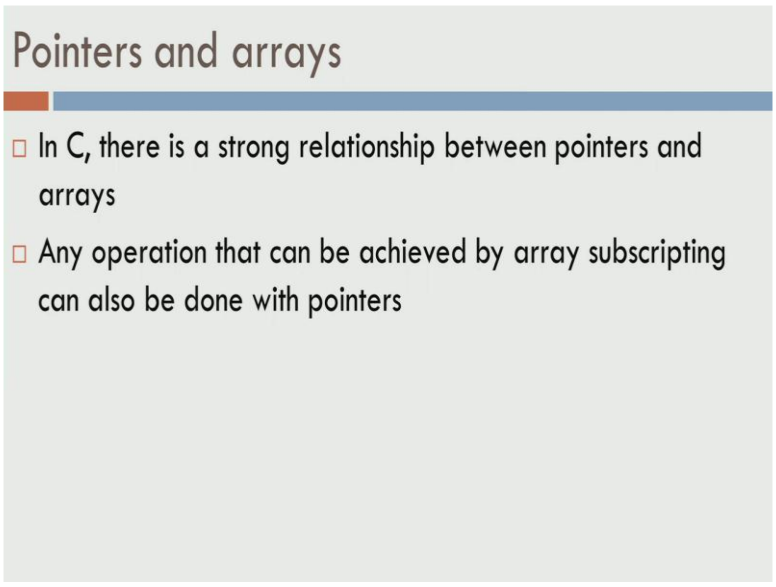
So, you can take a pointer which is of integer type and make it a generic pointer. So, if you look at the data type of void star ampersand m, it is actually a void pointer now, it is not an integer pointer anymore and you are printing that using a format specifier for you called percentage p. So, in the second line k has the value percentage d and is stored at location, whatever is the address of k. So, I already showed something very similar.

So, ptr itself is a variable which means ptr is given space or it is given some memory. So, it is given a memory location. So, ptr has the value percentage p and stored at percentage p. So, in this case this one, the first argument itself is a pointer and the second argument is taking the address of ptr and casting that to void star. And finally, the last line says the value of the integer pointed to by ptr is percentage d.

So, star ptr as I said is dereferencing ptr. So, see here, you are not declaring ptr. So, that has already happened in this line, here you are doing dereferencing of ptr. So, this is what I was pointing to earlier. So, there are two ways, in which you can use the star associated with the pointer. One is for declaration, the other one we are seeing here, which useful for dereferencing. So, it is getting the r value of k and it will print the value of k.

So, the key thing to notice in this slide is, there is percentage p specifier for pointers and we did what is called typecasting. We took integer pointer ampersand m and we converted that to void pointer using this bracket void star. So, I suggest that you actually take this piece of program and tried it out to understand, what is happening there.

(Refer Slide Time: 10:28)

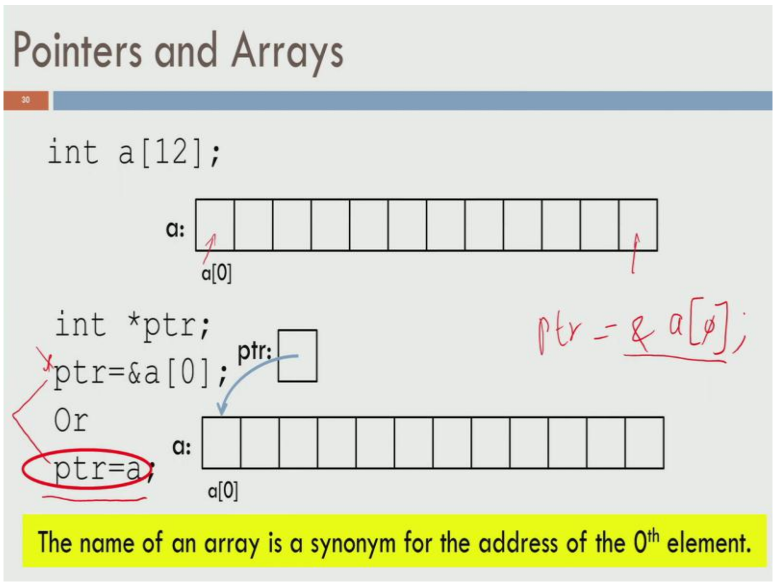


Pointers and arrays

- In C, there is a strong relationship between pointers and arrays
- Any operation that can be achieved by array subscripting can also be done with pointers

So, one key use of pointers is that it is something that is used for manipulating arrays. So, in C there is a very, very strong relationship between pointers and arrays and without pointers you cannot do a few things in C, when you actually want to use arrays. So, we will see this again in more detail, when we look at functions. So, you take it for now that there is a strong relationship between pointers and arrays and anything that you do using array subscripts can actually be done using pointers. So, pointers in that sense, is more powerful and more generic way of dealing with either contiguous set of locations or something that is not contiguous.

(Refer Slide Time: 11:16)



Pointers and Arrays

```
int a[12];
```

```
int *ptr;
ptr=&a[0];
Or
ptr=a;
```

The name of an array is a synonym for the address of the 0th element.

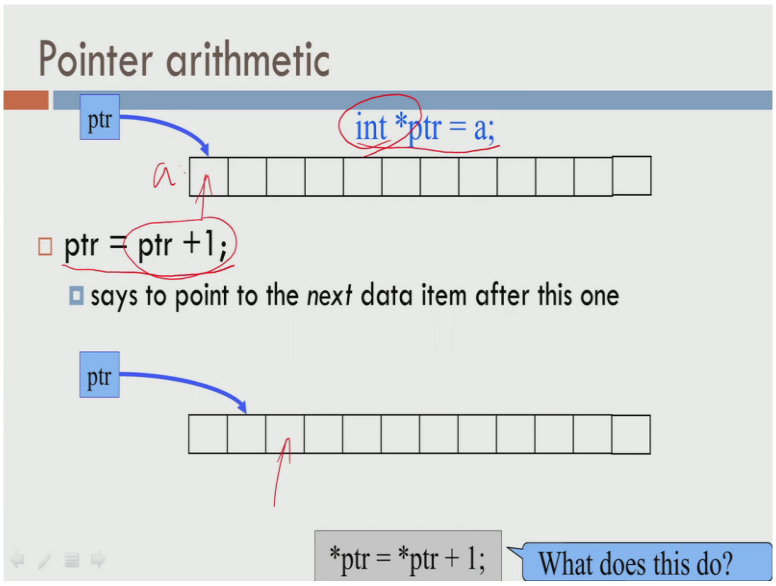
So, let us see the small example, we have int a of 12 as I said earlier, int a of 12 will declares space for a and let us say that is the top set of boxes that you are seeing. And we know that a of 0 is the very first location, a of 1 is the second location and so, on and a of 11 would be the last location. So, indexes go from 0 to 11, 12 is not a valid index for a. So, if I have int star ptr, this will give me a memory location called ptr.

And if I do either of these two, if I say ptr equals ampersand of a of 0 or ptr of a, then ptr starts pointing here. So, let us see, what really happens here. So, let us look at this statement first. So, if I look at this statement, ptr is ampersand of, a of 0. So, let us see what this does. So, a of 0 is a value, it is not a pointer, a of 0 is actually a value,. So, we have seen this in array so, far. But, the moment you put ampersand in front of it, it means do not get a value that is stored in a of 0, instead give me the pointer to the 0th element.

So, the left side is asking for a pointer and the right side actually gives a pointer. So, this is valid. So, what is this really doing is, ptr begins to point at the 0th element of a. You can also achieve the same thing by a slighter shortcut, which says ptr equals a. So, this is also something that you will see, a good programmers doing. So, experienced programmers do not use this format, instead they use ptr equals a. So, what this is doing is, ptr is now going to contain the address of the 0th element of a.

So, in some sense the name of the array a is actually only a synonym to the address of the 0th element. So, when we say a, a is an integer array of size 12. But, the variable a is actually just a synonym for the address of the 0th element of this array. Let us see, what all these means.

(Refer Slide Time: 13:53)



Pointer arithmetic

- ptr = ptr +1;
  - says to point to the *next* data item after this one

*ptr = *ptr + 1;

So, if I do int star ptr equals a. So, in this piece of code, you are declaring a integer pointer called ptr. So, this says we are declaring an integer pointer and the variable name is ptr and right when we are declaring, we are also initializing it to a. So, if a is, this array that we have here, then ptr starts pointing at the 0th location. So, this is something that comes from the using a as a synonym for the address.

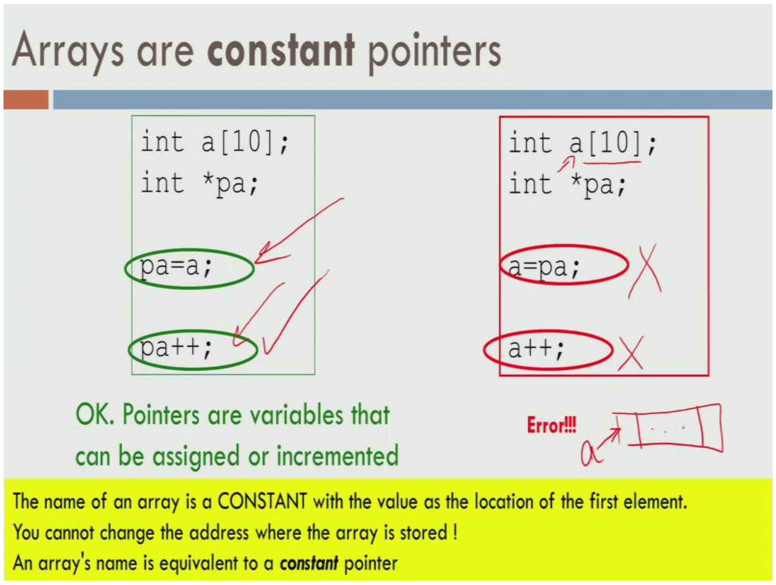
But, the nice thing you can do with ptr is that you can do something like this, ptr is ptr plus 1. So, let us see what this means, on the right side, we actually have a pointer variable and we are adding 1 to it. So, what; that means, is, instead of making ptr point at a of 0, now you actually pointed to the next element after the current one. So, at this point ptr was pointing at a of 0. So, you can see that here. But, the moment you do ptr is ptr plus 1, it is starts pointing to the next element.

So, clearly you can see that if you do ptr is, if you do another ptr equals ptr plus 1, it will start pointing to the 2th location and so, on,. So, it is starts with 0th. So, like as I said, I abuse this, abuse English language I say 0th, 1th and 2th and so, on. So, right now it is pointing at 0th location and if you do ptr is ptr plus 1, it will starts pointing at the 1th location. And if you do one more ptr equals ptr plus 1 and it is starts pointing at 2th location and so, on.

So, we can use like arithmetic, like you would do on integers and so, on. But, there are certain rules, we will see this in a little more detail in a little wide. So, one thing I would want you to think about is, what does star ptr equals star ptr plus 1 do? So, clearly there is a difference between saying ptr equals ptr plus 1 versus star ptr equals star ptr plus 1. So, take some time and think about it. So, it will be useful to think about, what this statement ptr equals ptr plus 1 does versus star ptr equals star ptr plus 1 does?

Take a while to think about it. So, let us go back to arrays,. So, in some sense arrays I said are less flexible than pointers. So, arrays are what are actually called constant pointers.

(Refer Slide Time: 16:39)



Arrays are **constant** pointers

```
int a[10];
int *pa;

pa=a;

pa++;
```

```
int a[10];
int *pa;

a=pa;

a++;
```

OK. Pointers are variables that can be assigned or incremented

Error!!!

The name of an array is a CONSTANT with the value as the location of the first element.
You cannot change the address where the array is stored !
An array's name is equivalent to a ***constant*** pointer

So, let us look at piece of code on the left side, we have int a of 10 and we have int star pa. So, pa is an integer pointer and when we said pa equals a, now pa starts pointing at the 0th location of a. And if you do pa plus plus, we are incrementing pa, what it does is, it makes pa point at a of 1 instead. So, this is actually ok to do because, pointers are variables and here you did one assignment to pa and you change the assignment to pa here, which is perfectly fine for pa, because pa is a variable. Let us do something which is slightly different here.

On the right side in the red box, we have int a of 10, we have int star pa and we say a equals pa. So, the first thing is that a equals pa is not permitted at all, so, this is not permitted. The reason is that when we do a equals pa, a is supposed to point at some 10 locations and we are now asking a to point at some other location, as indicated by pa. So, this is not allowed nor is a plus plus allowed. So, even though a is the synonym for the 0th location, it is not valid to take the variable name, that is used for the array and do any kind of arithmetic on it.

You cannot do a plus plus, you cannot do a equals pa or anything of that sort. So, to summarize a cannot appear in the left side of any expression. So, a plus plus is actually just a equals a plus 1. So, a cannot appear in the left side of any manipulation at all of any expressions at all, whereas, pa being a variable, can appear freely on the left side. So, the variable a is actually an array and it is called a constant pointer. So, what I mean by the constant pointer is that a of 10 is actually a set of 10 locations. So, we have 10 locations and a is actually pointing to a of 0, technically and it cannot point at anything other than a of 0. That is what I mean by constant pointer, a can point at a of 0 and nothing else, the moment using int a of 10.