

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

Module - 9D

Lecture - 17

Accessing arrays using pointers

Illustrations on IDE

Pre and post increment, sizeof, pointer arithmetic
Comparing pointers, Pointer subtraction in a string

Let us see, what I meant by pointers being a replacement for arrays.

(Refer Slide Time: 00:12)

```
Accessing Arrays with Pointers

#include <stdio.h>
int main(void)
{
    int myArray[] = {1,23,17,4,-5,100}, *ptr, i;
    ptr = &myArray[0]; /* point our pointer to the first element */
    printf("\n\n");
    for (i = 0; i < 6; i++)
    {
        printf("myArray[%d] = %d ", i, myArray[i]); /*<-- A */
        printf("Contents in address ptr + %d = %d\n", i, *(ptr + i)); /*<-- B */
    }
    return 0;
}
```

So, let us look at this small piece of code here. So, we have `int myArray` equals this. So, let us think about, what it really is doing? So, `int myArray` there is a left and right square bracket equals a list of values. So, what this does is, it takes the set of values from the right side. So, we have 1, 23, 17, 4, minus 5 and 100, there are six values. So, `int myArray` is actually an array, which can take six values. So, the size of `myArray` is six values, then we have `star ptr` and `i`.

So, instead of explicitly saying `myArray` should be an array of size 6, we let the compiler, figure out how many values are there on the right side? Based on that appropriately take the size for `myArray`. So, in this case the size of `myArray` is 6. So, following that we have `star ptr` and `i`. So, what we have in this line is, we have two integers. So, one integer called `i`, one integer array called `myArray` and one integer

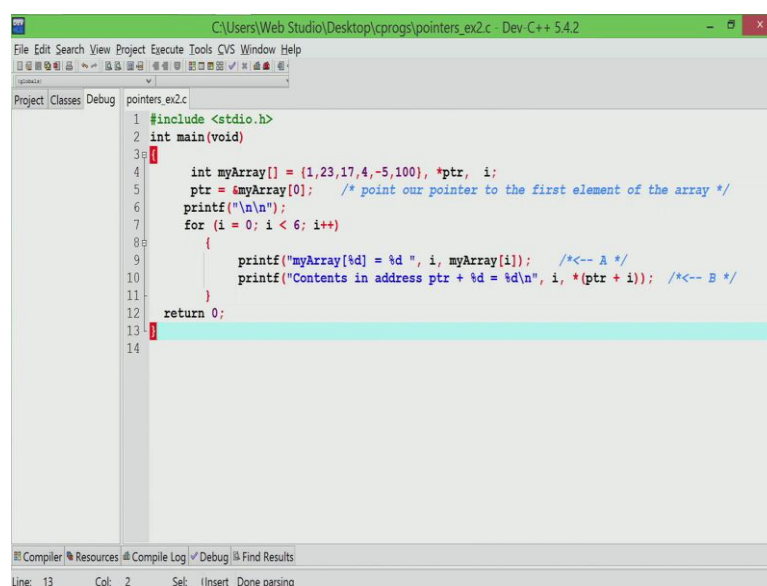
pointer called ptr.

So, that is what you have in line 1 and in the next line we have ptr equals ampersand of myArray of 0. So, ptr is now pointing at 0th location of myArray. You can also replace this with ptr equals myArray, that is also valid and it does the same thing. So, let us look at this loop here. So, printf myArray of percentage d equals percentage d. So, it is saying that print myArray of some integer equals some integer i comma myArray of i.

So, I am going to call this line a and in the next line, we have contents in address ptr plus percentage d equals percentage d and for the first percentage d, we give i and for the second percentage d, we gives star of ptr plus i as the variable to be printed. So, let us see what this means. So, in the first line, it is straight forward, you are printing the index of the array as well as the value contained in the index for i equals 0 through 5.

So, myArray has 6 values and it goes from 0 to 5 and the next line contains an address ptr plus. So, if you see the loop, it will take I will take 0, 1, 2, 3 and so, on. You will see contents of ptr plus 0, contents of ptr plus 1, contents of ptr plus 2 and so, on, all the way up to ptr plus 5. So, that is what you are seeing on this. And star of ptr plus i means, take ptr plus i, which means move it by appropriate number of locations from the base and dereference it using the star operator and I am going to call this, line B. So, I will tell you quickly, why I want the lines to be named A and B. But, let us take this program and run it.

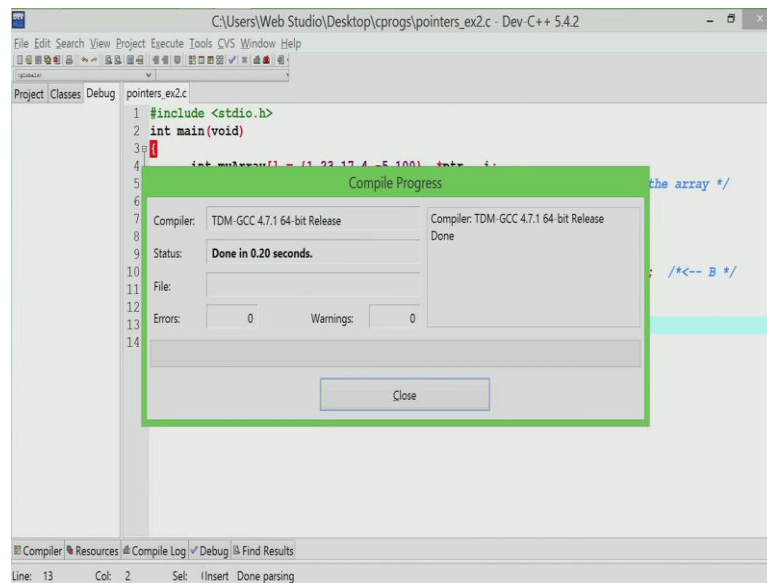
(Refer Slide Time: 03:37)



```
1 #include <stdio.h>
2 int main(void)
3 {
4     int myArray[] = {1,23,17,4,-5,100}, *ptr, i;
5     ptr = &myArray[0]; /* point our pointer to the first element of the array */
6     printf("\n\n");
7     for (i = 0; i < 6; i++)
8     {
9         printf("myArray[%d] = %d ", i, myArray[i]); /*<-- A */
10        printf("Contents in address ptr + %d = %d\n", i, *(ptr + i)); /*<-- B */
11    }
12    return 0;
13 }
14
```

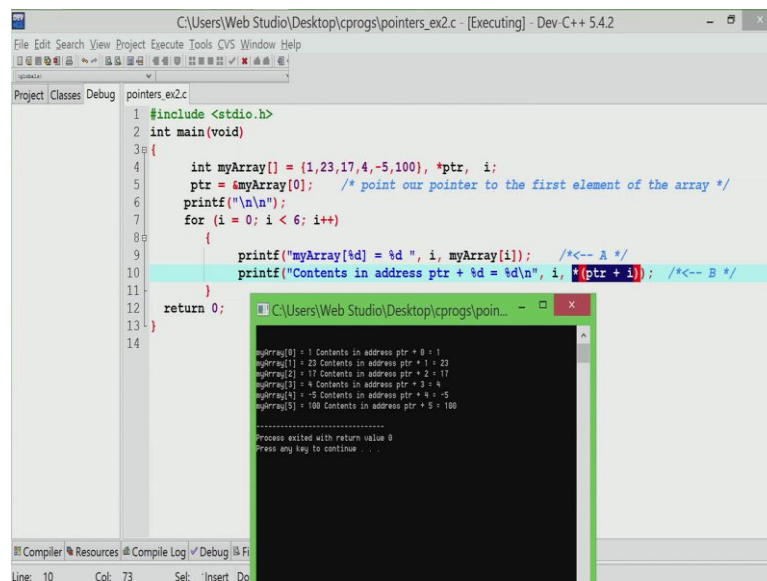
So, I have already taken this program and copied it into my editor.

(Refer Slide Time: 03:44)



And let me compile it and run it.

(Refer Slide Time: 03:47)



So, let us see what is happening in the outputs. So, let us take some time to see, what is printed in the output. So, the first line is myArray of 0 equals 1, contents in address ptr plus 0 equals 1 that is what it says. So, let us see myArray in the program. So, myArray is 1, 23, 17, 4, minus 5 and 100. So, myArray of 0 is indeed 1 and ptr plus 0 which means the contents of ptr plus 0 is star of ptr plus 0. So, ptr plus 0 is ptr itself and star of ptr is pointing to 1. So, it is printing 1.

Then, myArray of 1 is 23 and ptr plus 1 is supposed to point at the location one step

away from ptr. So, remember ptr is only pointing at myArray of 0. So, one element away from that is a of 1. So, ptr plus 1 is the address of a of 1 and when you do star of ptr plus 1, you are asking for the value contained in 1th location of a, which means you are asking for the value of a of 1 and this goes on. So, ptr plus 5 is asking for the value. So, star of ptr plus 5 will get the value, which is at a of 5 and it happens to be 100 and that is what you see here.

(Refer Slide Time: 05:27)

```

1 #include <stdio.h>
2 int main(void)
3 {
4     int myArray[] = {1,23,17,4,-5,100}, *ptr, i;
5     ptr = &myArray[0]; /* point our pointer to the first element of the array */
6     printf("\n\n");
7     for (i = 0; i < 6; i++)
8     {
9         printf("myArray[%d] = %d ", i, myArray[i]); /*<-- A */
10        printf("Contents in address ptr + %d = %d\n", i, *(ptr + i)); /*<-- B */
11    }
12
13    ptr = myArray;
14    return 0;
15 }
16

```

Compiler (1) Resources Compile Log Debug Find Results Close

Line	Col	File	Message
		C:\Users\Web Studio\Desktop\cprogs\pointers_ex2.c	In function 'main':

Line: 14 Col: 9 Sel: Insert Done parsing

So, let us do a small change to the program and I want to show that myArray is something that cannot change. So, let us say I do this, myArray plus plus. So, what I want myArray to do is, point to 23. So, I want myArray of 0 to be 23, myArray of 1 to be 17 and so, on. Therefore, I am trying to increment myArray by 1. So, I do not, it will not point to 1 anymore, it is supposed to be pointing at 23. So, let us try and compile this.

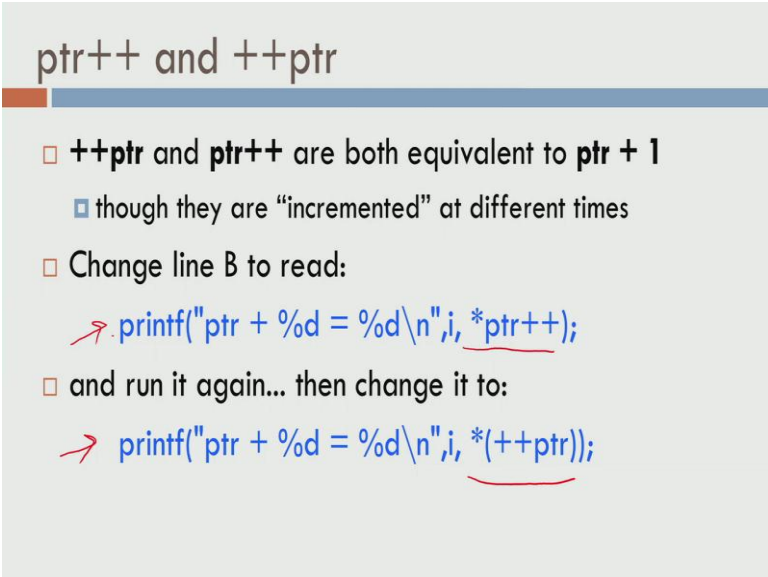
Let us try and compile that and let us look at this, error that happens. In fact, the compiler starts and points error at line number 13. It says, 1 value is required as increment operand. So, it is not a very useful error to C, but many compilers for C actually give you errors, that are slightly cryptic. In this case, what it is saying is. So, let us change this a little bit, like I said, this is equivalent to saying myArray is myArray plus 1.

So, let us compile this, you still see the same error. On the right side, it says error incompatible types, when assigning to type int of 6 from type int star. So, on the right side it is doing myArray plus 1, which means you are asking for one location away from

myArray and in the left side, you are assigning it to myArray. Even this, it gives a compilation error. So, based on what you write, so, it is changing the error.

But, the basic point is, it is not letting you change, what myArray should be pointing to. Whereas, if I had done this, let us say I did ptr equals myArray, if I did that... So, let us look at line number 13. So, if I compile it, it is technically not a problem. Because, even in this line, ptr equals myArray of 0 is actually equivalent to this line, ptr equals myArray.

(Refer Slide Time: 07:51)



The slide is titled "ptr++ and ++ptr" and contains the following content:

- **++ptr** and **ptr++** are both equivalent to **ptr + 1**
 - ▣ though they are "incremented" at different times
- Change line B to read:
→ `printf("ptr + %d = %d\n", i, *ptr++);`
- and run it again... then change it to:
→ `printf("ptr + %d = %d\n", i, *(++ptr));`

So, I said these are lines A and B, let us see why this is useful. So, I want to use it, to illustrate something now. So, both plus plus ptr and ptr plus plus are actually equivalent to saying ptr plus 1, you are incrementing ptr by 1 though at different times. So, there is a difference between saying plus plus ptr and ptr plus plus, plus plus ptr is called the pre increment operator. So, it means increment the value of ptr and then use it and ptr plus plus is post increment operator. Use the current value of ptr and then later increment the value of ptr.

But, I want you to do this little exercise. So, change the line B. So, if you look at line B, it saying contents in address ptr plus percentage d is i comma star of ptr plus i. So, change that to star ptr plus plus and run it again and once more change that line to star plus plus ptr and run it once more.

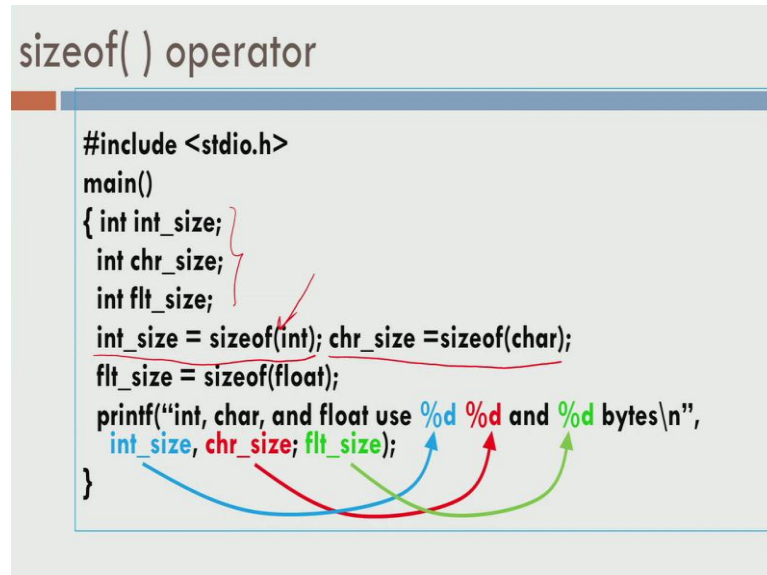
So, you take this piece of program which is in this slide, but make changes to line B, using either this line or this line, compile and run it and try and reason why, whatever is

happening, why it is happening? So, try a reason about that. So, let us move forward, there is another useful thing in C, which is called the sizeof operator.

(Refer Slide Time: 09:21)

```
sizeof( ) operator

#include <stdio.h>
main()
{ int int_size;
  int chr_size;
  int flt_size;
  int_size = sizeof(int); chr_size = sizeof(char);
  flt_size = sizeof(float);
  printf("int, char, and float use %d %d and %d bytes\n",
        int_size, chr_size, flt_size);
}
```

A diagram showing a C program snippet. The code defines three integer variables: int_size, chr_size, and flt_size. It then assigns values to these variables using the sizeof operator: int_size = sizeof(int), chr_size = sizeof(char), and flt_size = sizeof(float). Finally, it prints these values using printf. Colored arrows connect the variables to their corresponding sizeof calls and printf arguments: a blue arrow from int_size to sizeof(int) and the first %d in printf; a red arrow from chr_size to sizeof(char) and the second %d; and a green arrow from flt_size to sizeof(float) and the third %d.

So, it is not directly relevant to pointers, but this is something that I want to talk about now and again use this sizeof operator to explain a few things later. So, in this program we have three integers called int size, chr size and float size, flt size and we are going to do this, int size is size of int, chr size is size of char and flt size is size of float. So, as the name might indicate to you, you are looking at the size of an integer or size of a character and size of a floating point and so, on.

What is it mean by asking for sizeof an integer? So, integer is actually a set of values and this can be. So, if you go and look at the real world, the set of integers is unbounded in size. So, you have all the way from minus infinity to plus infinity, it is not something that a C language supports to. So, you cannot store every possible integer that is out there. C instead restricts it to be at least 4 bytes in size.

So, if you say sizeof int, it is actually giving you the number of bytes that is use to represent an integer. Similarly, sizeof character will give you the number of bytes required to store a character and sizeof float is going to give you number of bytes for storing a floating pointing number and so, on. So, these things can vary across different machines and so, on. But, the basic point is whenever you do sizeof int, for the machine that you are running on, it will tell you how many bytes are required to store an integer.

So, in most modern machines the sizeof integer is either 4 bytes or it can be even 8 bytes.

So, let us look at a few things related to pointer arithmetic. So, I am going back to pointer arithmetic, you can do a lot of things with pointers and arithmetic related to pointers.

(Refer Slide Time: 11:30)

Pointer Arithmetic

- *Valid pointer operations:*
 - Assignment between pointers of the same type
 - Addition/ subtraction between a pointer and an integer
 - Comparison between two pointers that point to elements of the same array
 - Subtraction between two pointers that point to elements of the same array
 - Assignment or comparison with zero (NULL)

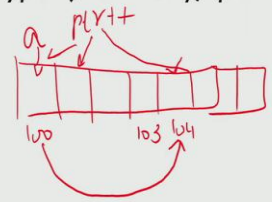
So, you can assign between pointers of the same type, you can do addition and subtraction between a pointer and an integer. We already saw this example, we did `ptr plus 1` and `ptr plus plus` and so, on. So, `ptr` was of type pointer and `plus 1` is actually adding an integer 1 to `ptr`. So,; however, C allows you to do that, you can compare two pointers that point to elements of the same array, you can even subtract two pointers. We will see a small example at the end of this module and you can assign and compare. So, these are several things that you can do with pointers.

(Refer Slide Time: 12:05)

Pointer Arithmetic – Increment/Decrement

□ **Increment/decrement:** if p is a pointer to type T , $p++$ increases the value of p by $\text{sizeof}(T)$ ($\text{sizeof}(T)$ is the amount of storage needed for an object of type T). Similarly, $p--$ decreases p by $\text{sizeof}(T)$;

```
T tab[N];  
T * p;  
int i=4;  
p=&tab[i];  
p++; // p contains the address of tab[i+1];
```



Let us see each one, one after the other, the first one is increment decrement. So, if p is a pointer to type T . So, there is actually no data type called T in C. So, I am using this as a template. Let us say, I have a data type called T and p is a pointer to T . So, T could be an integer, character, floating point, it could be anything p plus plus will increase the value of p by $\text{sizeof } T$ steps.

So, I already mentioned this, if you say sizeof data type, it tells you, what is the number of bytes required to store the data type. So, when you do p plus plus, it is not incrementing p to point to the next byte, it is going to make p point to the next valid element of the data type T . So, let us say my integer is actually 4 bytes. So, I have an integer called a and let us say, it is occupying 4 bytes, this is not an array of size 5.

So, I am drawing a memory footprint and let us say variable a is stored in 4 bytes. Let us say a location 100, 101, 102 and 103 and let us say location 104, 105, 106 and 107 are the next set of locations. If ptr is made to point a and if you do ptr plus plus, ptr will not point to 101, because the data type ptr is pointing to is integer, the next valid integer is four steps away from 100. So, ptr plus plus will make it point to the integer at location 104.

So, let us look go back to the slide. So, if we have data type T , tab of n and if you have a declaration $T * p$. So, you can substitute T with int , $char$, $float$ whatever you want. So, T itself is not a valid data type and let us say $int i = 4$, if I do $p = \&tab[i]$, so, i is 4. So, $\&tab[4]$ will make me point to the 4th entry in tab .

So, remember the entry start at 0. So, we are talking about 0th, 1th, 2th, 3th and 4th.

So, the i th entry is the 4th entry, starting from 0. So, p is going to point at the 4th entry in `tab` and when you do p plus plus, it actually starts pointing at the next entry in `tab`. So, this is an example of pointer arithmetic, where you mix pointer and integer operations on it.

(Refer Slide Time: 15:08)

Add/Subtract Integers from Pointers

- **Addition/subtraction with an integer:** if p is a pointer to type T and n an integer, $p+n$ increases the value of p by $n*\text{sizeof}(T)$.
- Similarly, $p-n$ decreases p by $n*\text{sizeof}(T)$;

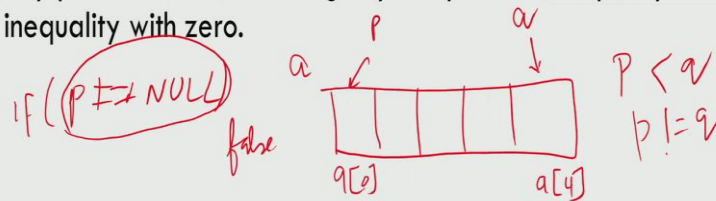
```
T tab[100];
T *p;
p=tab;
p=p+5; // p contains the address of tab[5].
```

You can also do add and subtract from pointers, you do not have to do just increment and decrement, you can also add. So, in this example what we have done is, we have made p point at the 0th location of `tab` and we are adding 5 to it, which means it will start pointing to `tab` of 5.

(Refer Slide Time: 15:30)

Comparing Pointers

- If p and q point to members of the same array, then relations like $=$, $!=$, $<$, $>$, etc., work properly.
 - For example, $p < q$ is true if p points to an earlier element of the array than q does.
- Any pointer can be meaningfully compared for equality or inequality with zero.



You can compare pointers, if p and q are pointers of the same, if they are pointing to members of the same array, then you can do things like equal to equal to, not equal to, less than and so, on. So, let us say I have an integer array and. So, in this case I am drawing an array, not memory, I am drawing an array. Let us say, I have an array a and let us say this is a of 0 and this is a of 4 . If I make p point at this location and if I make pointer q point at this location, then I can actually ask the question is p less than q .

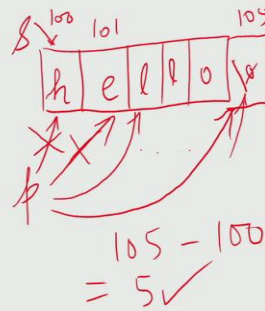
So, the meaning of that is p pointing to a location, that is earlier in the array than what q is pointing to, in this case p is less than q is actually true. So, you can also do is p not equal to q . So, in this case p is actually not equal to q , p is pointing to the 0 th location, q is pointing to a of 4 . So, p is actually not equal to q . So, you can meaningfully compare pointers which are pointing to the same array. You can ask is this is the same, is this earlier than the other pointer, is it later than the other pointer and so, on.

So, you can make pointers point at different locations in an array. You can ask the questions, whether one pointer is lesser than the other and so, on and we can also compare it for inequality with zero, you can ask the question, is p equal to null? So, let us say p is pointing to a of 0 and I can ask the question, if p equal to equal to null? If I ask this question, this will actually evaluate to false, because p is actually pointing to a of 0 . So, you can do comparisons of this kind.

(Refer Slide Time: 17:20)

Example: Pointer Subtraction

```
/* strlen: return length of string s */
int strlen(char *s)
{
    char *p = s;
    while (*p != '\0')
        p++;
    return p - s;
}
```



So, we looked at increment decrement and adding an integer or subtracting an integer from a pointer. One final beautiful thing about pointers is that, you can also do subtraction of pointers. So, does a small piece of code here which actually tries to show that. So, this one is slightly loaded, but let us see what it is doing. So, there are two pointers here, one called s and one called p and this is actually a function which is called string length strlen.

So, given a string it tries to find out, what is the length of the string? So, let us see what a string is in C. I will anyway do this in more detail later, but a string in C is actually just a sequence of characters followed by a special character called back slash 0. For instance, let us say I have a string called hello, it will be stored in this form, you have h e l l o. So, this is letter o and on top of that in C, there is a special character called back slash 0. So, this case is actually 0 or what is called the null character.

So, a string called hello, it supposed to be of length 5, but in reality it actually takes 6 bytes. So, let us I want to find out what is the length of this string? So, I can keep checking starting from the 0th location of the string, I can keep checking for one character after the other and I will look for this special character called back slash 0. The moment I see a back slash 0, it means that this is the end of the string and this is the convention used in C.

So, like I said we again see this in more detail later, but as of now let us look at this example. So, let us I want to find out what the length of the string is. So, I have written a

small function which does that. So, character star s and character star p. So, at this point p is initialized to s. So, s is a pointer to a character array that is what you see here. So, s is pointing to a character array and character star p equals s. So, p also points at the 0th location of s. So, s is the string for which we want to find out the length.

So, there is a while loop here that keeps checking, if the contents of the memory location pointed by p is set back slash 0 or not. So, there is a special character back slash 0, I am going to start from location h. So, where h is stored, all the way till I see a back slash 0. So, initially p is pointing at h and star p is not equal to back slash 0, we do a p plus plus. So, p will start pointing here.

So, p has already moved by one location, it has started pointing at s of 1, s of 1 is e and it is not back slash 0. So, the while loop will continue and start pointing at l and this will keep happening and it will stop, when p points at back slash 0. So, let us assume that this is at location 100, this is at 101 and so, on, this will be at location 105. So, when the while loop will stop, when star p is equal to 0. When the star p equal to 0? When p is 1 naught 5, star p is back slash 0 and which means when p equals 105, the loop will stop.

So, at that point, if you go and ask for p minus s. So, p is 105 and s is memory location 100, 105 minus 100 is 5, that actually gives us the correct length of hello. So, this is a very neat and beautiful trick to show with pointer arithmetic. So, let us not worry about, what this written statement is and what is this strange looking int strlen and so, on is later. But, as of now I just want you to pay attention to the while loop.

So, it iterated over the elements of s and it is stopped at back slash 0 at that point, p got value 105 and if you do p minus s, it actually gives you 105 minus 100 which is 5. So, when you do arithmetic over pointers, if you do subtraction of pointers, you actually get an integer back and this is something that is very handy. We will see use cases for this in a lot more detail, later. So, we have now come to the end of this module. In the subsequent modules, we will see how to use pointers for manipulating arrays and we will also see how pointers are useful in the context of order called functions.

So, we will start looking at functions and when we visit functions, we will revisit how pointers are handled, when we pass them on to functions.

Thank you, very much.