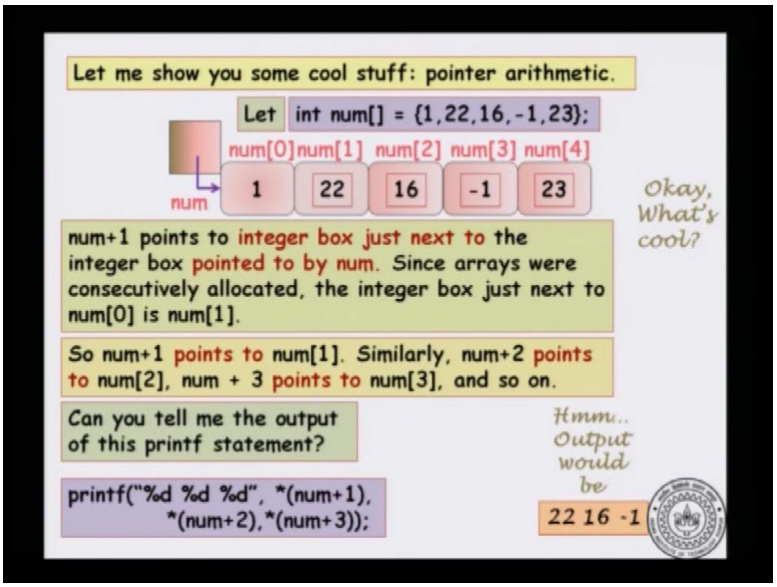
Introduction to Programming in C
Department of Computer Science and Engineering

In this video let me, so some cool stuff which is pointer arithmetic which helps you to understand the relationship between pointers and arrays in C.

(Refer Slide Time: 00:13)



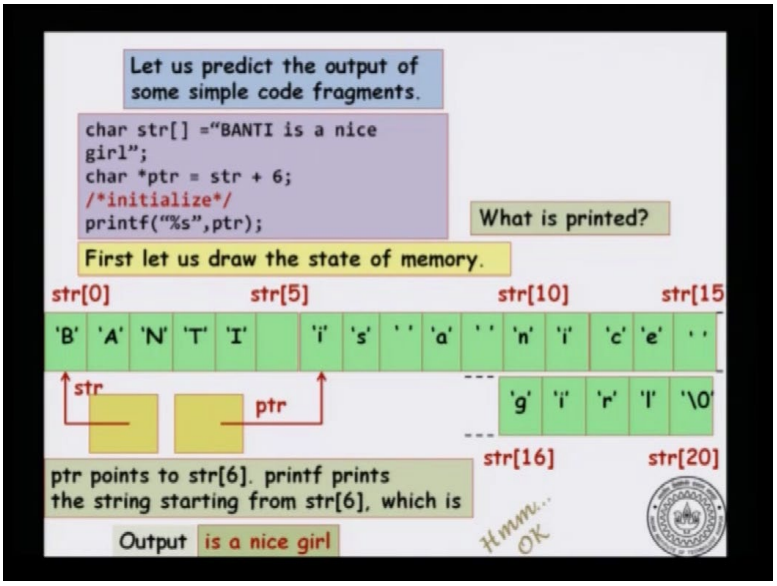
So, let us consider in array declared as follows int num, and then it has 5 numbers in the initialization list. So, the array will be initialized as follows; there are 5 consecutive integer locations in memory with the given elements + there is a 6th cell which points to the first location in the array. So, num points to the first location in the array. If num points to the first location, then you can do the following operator num + 1. So, num + 1 with point to the integer box write mix to the integer box pointed 2 by num.

And we also know that arrays are consecutively located. So, the integer box next to num is exactly num 1. So, num + 1 points to num[1]. Similarly num of num+2 points to num[2], and so on. Until num+4 = num[4]. So, this particular box, for example, num[4] can be accessed in two ways; you can write num[4] or you can write *(num+4). Can you tell me the output of the following printf statement. So, think about this for a minute, you have 3 integers to print using %d %d %d, and what are to be printed are *(num+1) *(num+2) and *(num+3). So, think about it for a minute…

num+1 is the address, which is the second integer box in the array, num + num points to

the first location therefore num 1 points to the second location, star is the dereference operator on a pointer. So, * of this pointer means go to that location which is this location, and get the value in that location, which is 22. Similarly num+2 is the box 2 boxes away from the first box in the array. So, 2 boxes away from num, that happens to be num+2 which is the… And then get the value there which is 60. Similarly *(num+3) will give you -1. so the output would be 22 16 -1. So, in this printf statement we have used two concepts. One is getting to a different pointer from a given pointer using pointer arithmetic operator +, so we have used + here. The second operator that we have used is * on a given on a given pointer. So, + will tell you go to the next integer location, and * will tell you for a given integer pointer give me the value in that location.

(Refer Slide Time: 03:31)



Now, let us look at the slightly different array. What happens if you have a character array. So, I have char str array which is initialize to let say given string BANTI is a nice girl, and then I have a character pointer. So, char *ptr and it is assigned str + 6, it is initialize to str + 6, what will happen here?

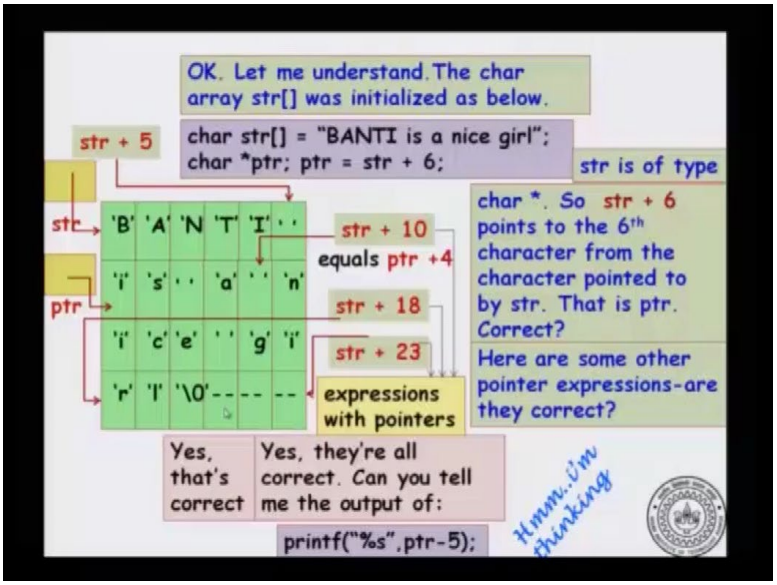
What is different about this example is that, earlier I said that in an integer array + 1 for example, would go to the next integer location in memory. So, wherever num was num+1 would go to the next integer location here, str is a character array. So, it has to go

to the next character location, and that is exactly what it does. So, what is printed? Lets first consider the state of the memory. So, you have an array which is a character array, it starts from str[0], and goes on up to str[20]. So, there are 19 characters followed by the null character. Why is the null character there, because I initialize the two a string constant; every string constant has a null character implicitly at the end.

So, this is the straight of the str array. Now I say that I declare a pointer, the pointer is pointing to a char. So, it is a char * pointer and what is the location it points to it points to str + 6. Str is a point out to the first location of the character array, and + 6 would jump 6 character locations away from str[0]. So, you would reach this character. The important difference between this example at the previous example is that, if you declared an integer array + 1 would jump 1 integer location + 6 would jump 6 integer locations. Here since such a character array str + 6 would jump 6 character locations. So, how the + operator is interpreted in the cases of pointer depends on what array am I pointing to right now? Now what will happen with the printf statement? So, if I say printf % as ptr what will happen?

So, ptr points to str 6. So, printf will print whatever string is starting from that location until the first null character. So, it will start printing from this i, and then go on tip printing till it reaches the null characters. So, the output will be just is a nice girl. So, when you want to printf it is not important that you start from the absolute beginning of the array. We can start from arbitrary location in the character array, and if you say printf %, it will start from there and go on and print until the first null character.

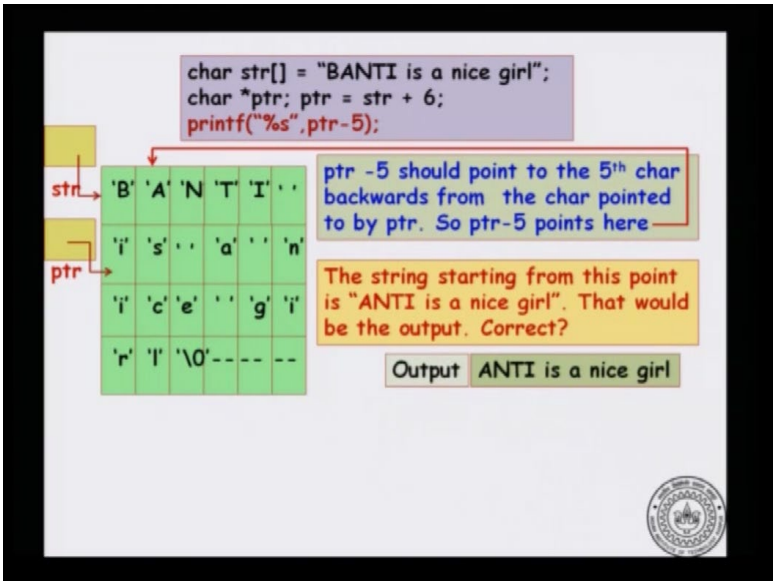
(Refer Slide Time: 06:42)



So, let us look at it once more. So, it was the code that you had, and let say that the one-dimensional array for this seek of convenience, I will just… So, it like this. It is actually in a row, but here is the first part, here is the second part, and so on. So, when I say str, str is a character array, and ptr + 6 would goes 6 locations away from the first location. So, str is pointing to then first location in the array, it will go to the 6th location in the array ptr, and ptr is pointing to the 6 location.

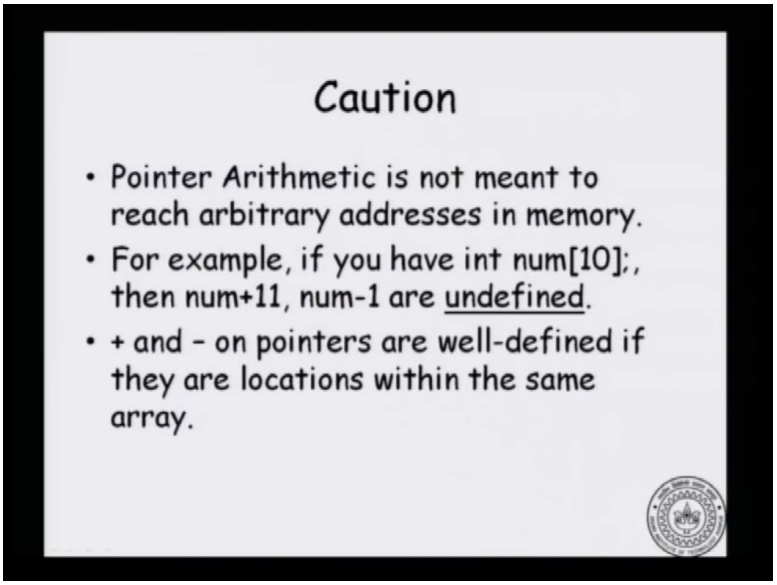
So, you can ask more expressions do the make sense, can I say str + 5 is the this location. Similarly can I say str + 10 is this location, and so on. So, these are all correct expressions. Now can you tell the output of printf % ptr -5, we have talked about + operator on pointers. So, it will whatever the nature of array that the pointer is pointing to it will jump n locations away from it. So, if I say ptr + n, it will jump n locations of that type away from it. So, by the same logic can I argue that if I do -5 ptr -5 can I say that it will go 5 locations previous to what ptr is pointing to right now. And the answer is yes.

(Refer Slide Time: 08:30)



So, it will behave exactly as you expect. So, ptr is pointing to this location, here is the previous location. So, it will jump to 5 locations before the location pointer 2 by ptr, I will happens to be A. So, the location which is str[1], that is = ptr-5. So, if you printf on that location, it will say BANTI is a nice girl, that is the output.

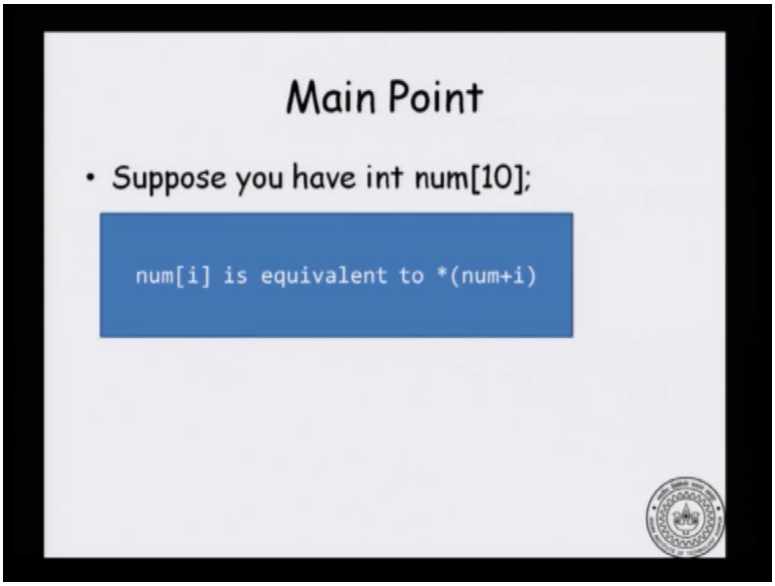
(Refer Slide Time: 09:10)



Before I proceed this one thing that I want to emphasize, and it is often not emphasized when you see online material on pointer arithmetic. C pointer arithmetic is not suppose to be meant for navigating the array, meant for navigating arbitrary locations in the memory. So, you cannot take a pointer. Let say character pointer and just say pointer + 1000. It will give you some location in the memory, but the behavior of the program will be undefined. So, the c pointers are well defined, pointer arithmetic using C pointers are well defined only when the pointers are pointing to locations within an array. So, within an array + n will take you n locations away from the given pointer, -n will give you -n away from the give behind the given pointer and so on. Whatever type the character of whatever type the given pointer is pointing 2.

For example, if you have int num 10, and then you have num+11, you know that num+11 is not a valid location in the array. Similarly num -1 the num arrays starts at num of 0, which is equivalent to num + 0. So, num -1 is also out of the given array, therefore these two locations are actually undefined, because c does not guarantee you, that if you try to deference these pointers, you will get any meaningful information. So, + or -and pointers are well defined, and their behavior is easy to predict exactly when you are navigating within the bounce of an array.

(Refer Slide Time: 11:14)



So, the main point of lecture was that if you have let say for example, and integer array int num 10, then num of 5 which is the array notation is exactly equivalent to *(num+i). And I am not saying this that you can think of num of 5 as *(num+i), it is not an analogy this is exactly what C actually does. So, num of 5 is translated to *(num+i). So, arrays and pointers in c are very intimately related.