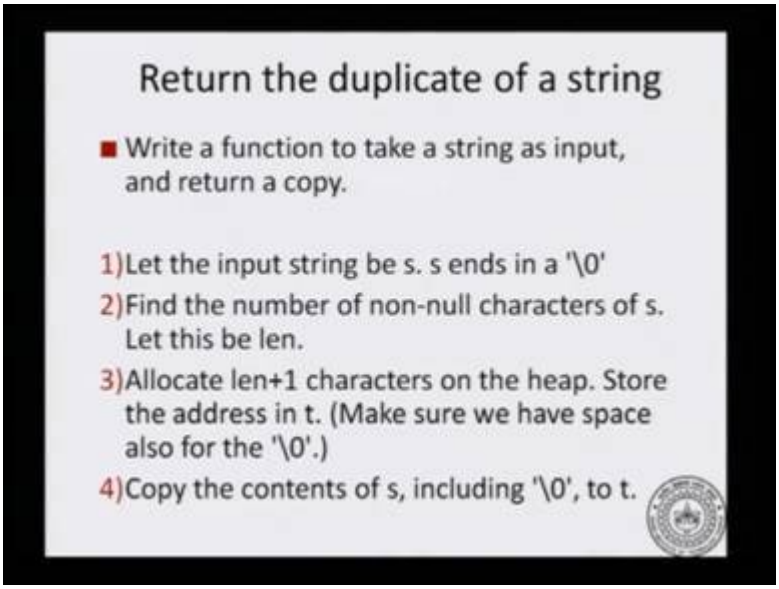
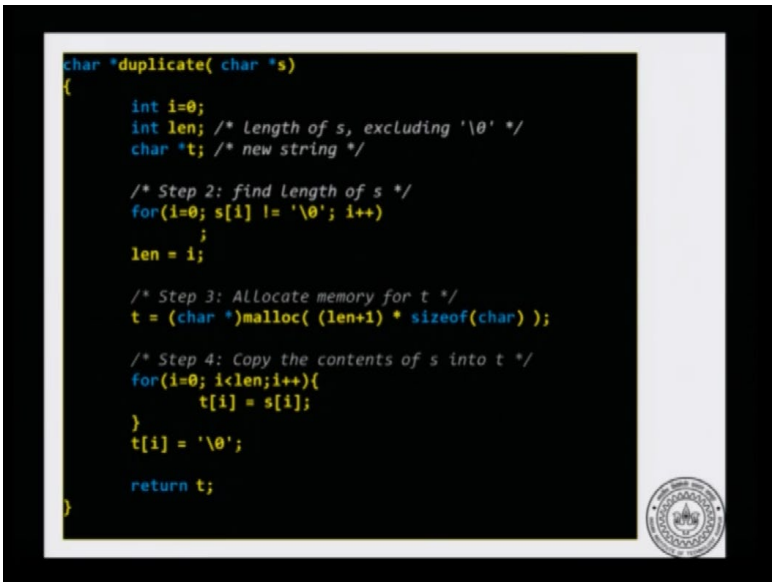
Introduction to Programming in C
Department of Computer Science and Engineering

(Refer Slide Time: 00:06)



In this lecture let us look at an application of malloc and free to solve some problem that we are interested in. So, the problem that I will define is to write a function to return the duplicate of a string; a string is given us the argument and you have to return the duplicate of that string. So, we have to write a function to take a string as input and return the copy. Now let us assume that the input string is s, and it ends in a null character. Assume that we can find the number of null non null characters in the string. So, this is will be refer to us the length of the string. What we will do is allocate length + 1 characters. So, there are length non null characters, and then one more for storing the null characters. So, we will allocate len + 1 characters on the, heap using malloc, and we will copy the contents of s to that space on the heap. And finally, return the address of that location. So, that will be the t th the new array. So, notice that the original array may be on the stack, and the new array the duplicate array will be on the heap. Let us write this function.

(Refer Slide Time: 01:29)



So, I will call it duplicate it takes one array which is the same as a pointer. So, I can declare it has char *s or char s with square brackets, it does not matter. So, I will just declare it has a character *s, and what will it return? It will return another array, and array is the same as a pointer. So, I will return character star. So, the input argument is an array, and the output is also an array. I will declare 3 variables; i which is for the loop, len which will store the number of non null characters in s. So, let us be very specific, I do not want to store the number of characters in s, because I want to say that I do not want to count null.

Now if you want to count null as well in the length then you will modified the code, but typical convention is that when you mention the length of a string, you do not count the null character. I will also declare a char *t. Now the code proceeds and stages; first I have to write a loop to find the length of the string, I can write a very simple loop to do that I can say for i = 0 as long as s of i is not null, do increment i. So, as soon as I see the first null in s, I will stop. When I exit out of the loop, I will be the number of non null characters in s. So, I can say len = i. So, in the first step of the function, we just find the length of the string excluding the null character at the end.

Now comes the important thing, we have to copy that array to somewhere. If we copy

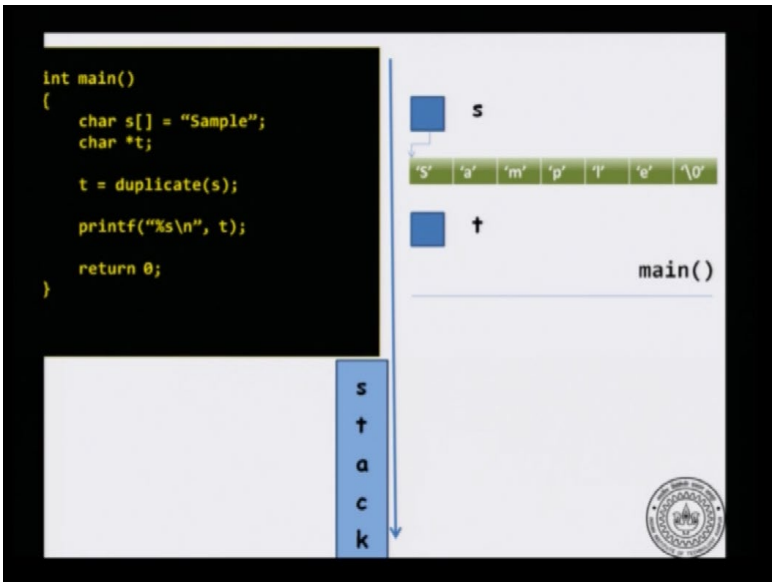
that array to the stack itself that is if I copy that array to some space within the duplicate functions stack, it will be erased when I return. So, I should allocate the space on the heap. I can allocate space on the heap using the malloc function. So, let us look at the malloc function. I want to allocate a bunch of space on the heap, how much do I have to allocate, I have to allocate len + 1 number of characters.

In other words, I have to allocate len + 1 times sizeof a single character; these many bytes on the heap. Notice that it is not len times sizeof(char), because if I allocate only that much then I will not have to space to copy the last null character. So, I should the input is a null terminated character, it is duplicate should also the null terminated. So, I should makes space for all characters including the null character on the heap. So, I will allocate (len+1) * sizeof(char) many bytes on the heap, it will return you the address of the first byte and that address I will convert to a char star. So, malloc returns a kind of an unsorted. So, here are these many bytes. Now it will return you the address of the first byte that was located, now I want to treat that as a character pointer. So, I will can do that using the casting operator.

Why do I have to do that think about it for a minute, because you want pointer arithmetic to work. When I say t[i], I should correctly execute star of t + i. So, go back to that lecture and understand why it is important that you know the it is not just a byte address, it is a character pointer. Once you do the allocation, you can copy s array into t array. We do not really care about the fact that t is not on the stack, t is on the heap, because copying is done exactly the same way. So, I can say i = 0, i < len, i ++ t[i] = s[i]. And then finally, this will copy all the non null characters, and finally I will say t[i] = null, the last character will be the null character.

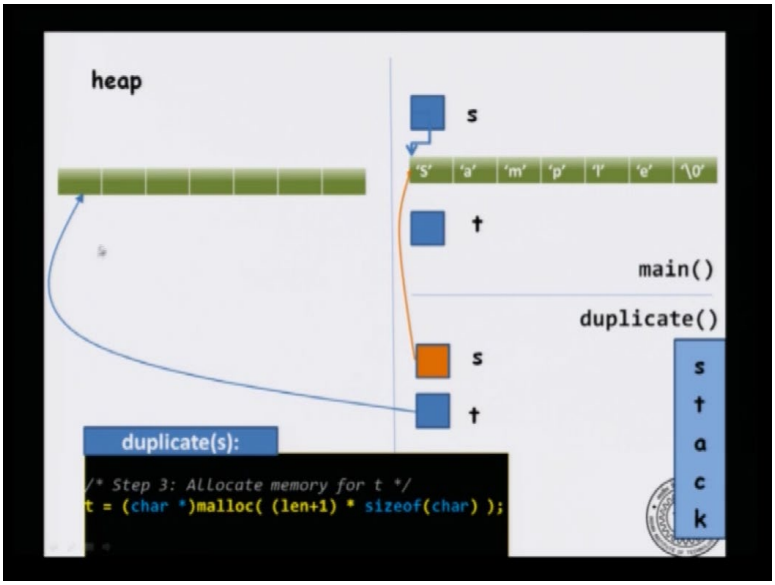
Now, if you want to understand it in slightly greater detail understand why the character star cast was required in order for t[i] to work properly. Once I have done copying the array, I can just return t, and I will not leap it will not lead to a dangling pointer because t is allocated on the heap.

(Refer Slide Time: 06:37)



So, let us pictorially understand what happens during the execution of this program. I have main function, and I allocate a char array. Now this is allocated on the stack. As soon as I declare a character array and initialized it with in main, it is allocated in the stack corresponding to main. So, s is a pointer to the first location in the array. And I declare another *t, and then I call t = duplicate(s), I should return a separate copy of s.

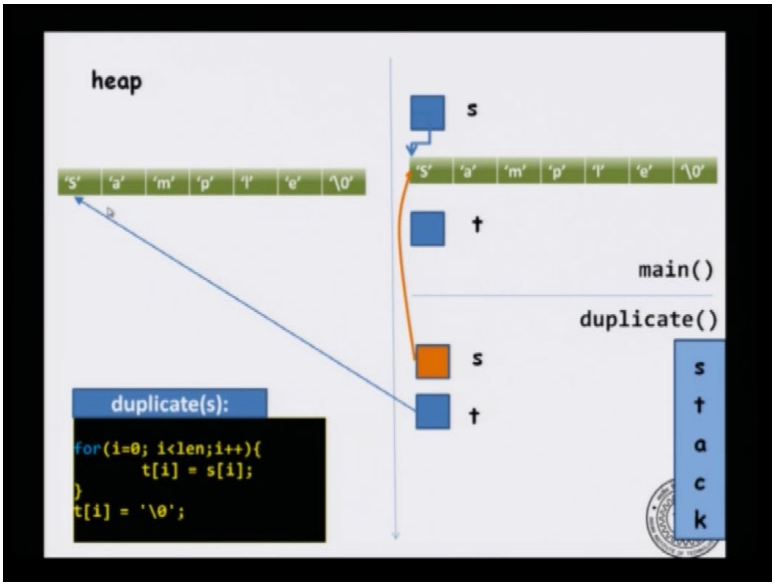
(Refer Slide Time: 07:19)



Let see what happens in the duplicate function? We do allocation for all the local

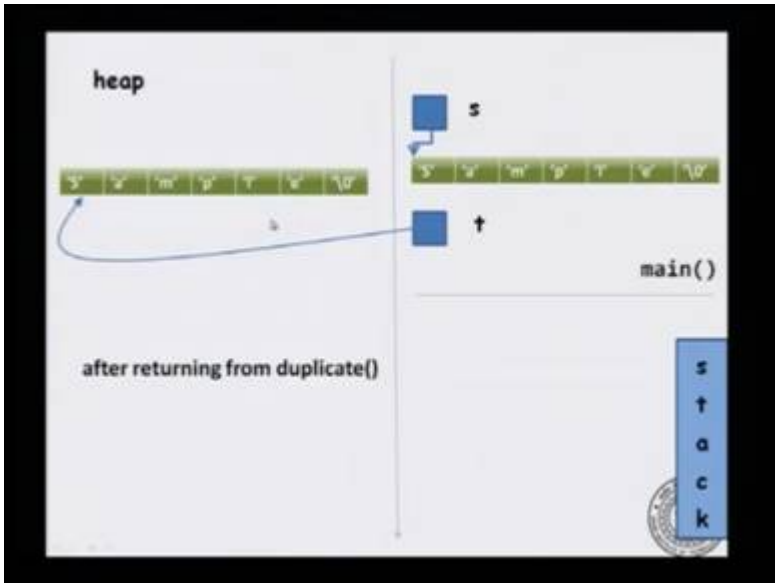
variables all that, but important thing is that we have s and t which are new pointers. Now s is the input argument to duplicate, and it will be pointing to the array in the main function, because I call duplicate(s). So, duplicates s will point to the same array as the s of mean. So, it is pointing to the array on the stack, Now as soon as I allocate memory for t on the heap, which was step 3 of duplicate, I would say t = (char *)malloc((len+1) * sizeof(char)). What is len here? len is 6; there are 6 non null characters len + 1 is 7. So, I allocate 7 characters on the heap, and its return address will be cost to a character pointer. So, t is now pointing to this space on the heap.

(Refer Slide Time: 08:28)



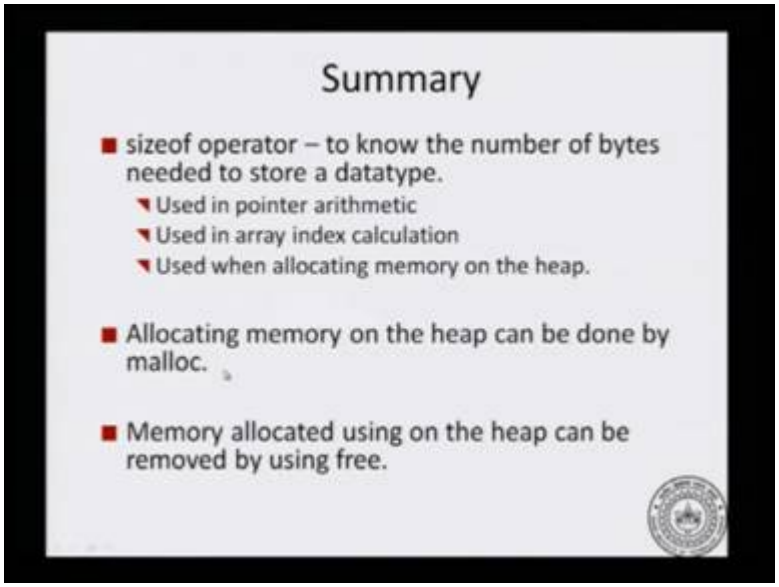
Now, once I am done ((Refer Time: 08:33)) creating the space on the heap, what I can do is, I can copy the location the s array into the t array on the heap. So, once that loop executes it will look like this, here is the s array inside main, here is the heap the array allocated by duplicate, and you will just copy t[i] = s[i] for. So, you will copy ‘s’ ‘a’ ‘m’ ‘p’ ‘l’ ‘e’, that is within the loop. And then finally, I will say that t[6] = ‘\0’. So, here is an array of size 7, it has a 6 non null characters and the last element is null.

(Refer Slide Time: 09:23)



And then once I am done, I will return from duplicate. Again keep in mind what is erased is the stack. Everything that was allocated to duplicate on the stack is erased. Those s t and the local variables in duplicate no longer exist, but the work that was done by allocating on the heap that still remains. So, the return value t, that is return value which is the address of the array in heap will be assign to t. So, t now points to heap. Notice how it executed s was allocated on the stack, and the effect of the duplicate function will be that, the duplicate of the array will be created on the heap.

(Refer Slide Time: 10:09)

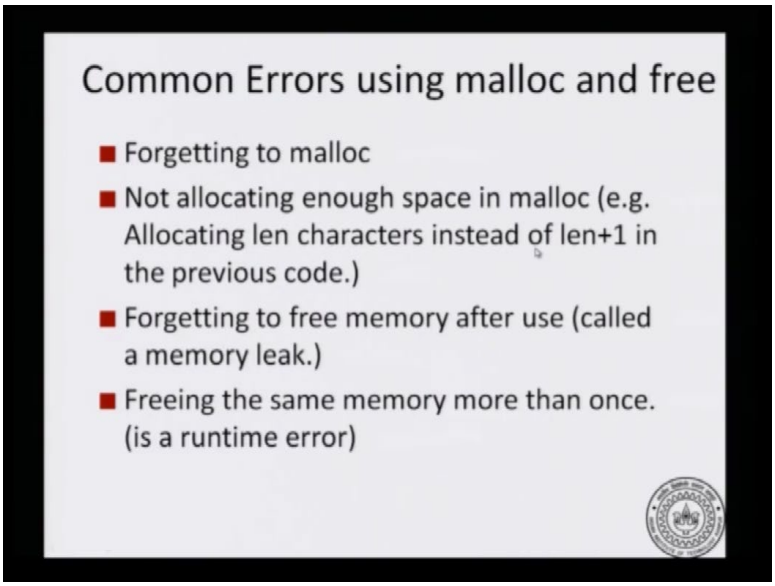


So, notice what we understood about the sizeof operator. Sizeof operator was use to know the number of bytes need at the stored data type. It is used in pointer arithmetic, it is used in array index calculation, and it is also used when allocating memory on the heap, because malloc needed to know how many bytes to allocate. And suppose I have wanted to allocate 10 integers, instead of me saying that on this machine I know that and integer is 4 bytes. So, you go ahead in allocate 40 bytes. The problem with doing that is you take your code to another machine, and that machine integer is 8 bytes; and your code will no longer allocate sufficient space. So, the real way to write portable code would be to say 10 times sizeof int, that code will work regardless of which machine you execute on.

So, here is the use of sizeof operator when you call malloc, it helps you to write portable code which will execute on any machine. So, we have seen that allocating memory on the heap can be done using malloc, we have understood what it means to allocate memory on the heap the difference between stack and heap; stack is erased as soon as a function returns, heap is not erased when a function returns you have to explicitly say that i ((Refer Time: 11:39)) now freeing that using free function. Again remember the asymmetry within malloc and free; malloc needed to know how many bytes you allocate, free just needed to know which pointer to de allocate, which pointer to free. Did not want

to know how many bytes to free, it does that automatically.

(Refer Slide Time: 12:03)



And to repeat common errors using malloc you could forget to malloc. Now you could not allocate enough space in heap. For example, in the code that we have just seen, suppose you are allocated just len number of characters instead of len + 1. Then you would not have enough space on the heap to copy the last null character. So, you will violet that t is an exact duplicate(s). Now you could forget to free memory after use, this is called a memory leak, and you could have the sub square error of freeing the same memory twice, that leaves to run time errors.