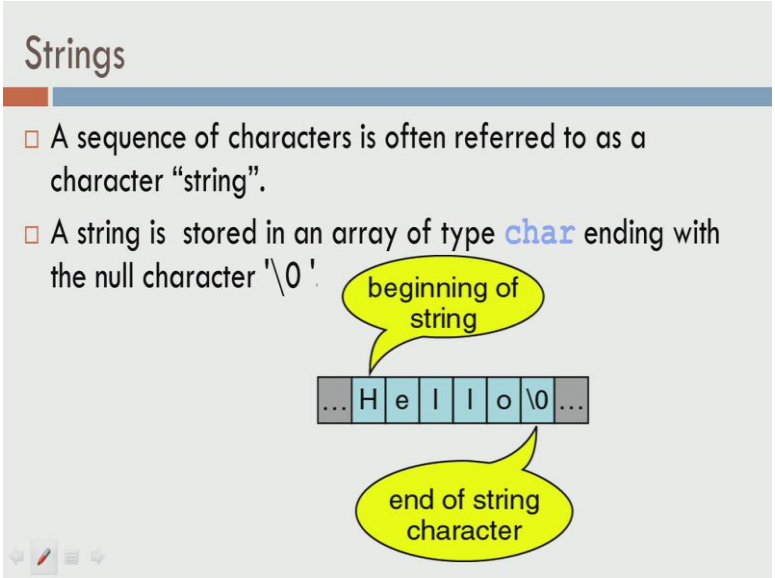
Programming, Data Structures and Algorithms
Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module -13a
Lecture - 18
What is s string?
How C stores a string? Character vs string
String initialization, Printing strings, Examples
Reading strings

Welcome to this module. In this module, we are going to learn something called Strings. This is something that happens in the real world very often, name, your name and my name, colleges name, things like that are all strings. C does not actually give you direct support for strings and this is why I wanted to have one small lecture on what strings are and how to manipulate strings, how to read, how to print and so, on.

(Refer Slide Time: 00:42).

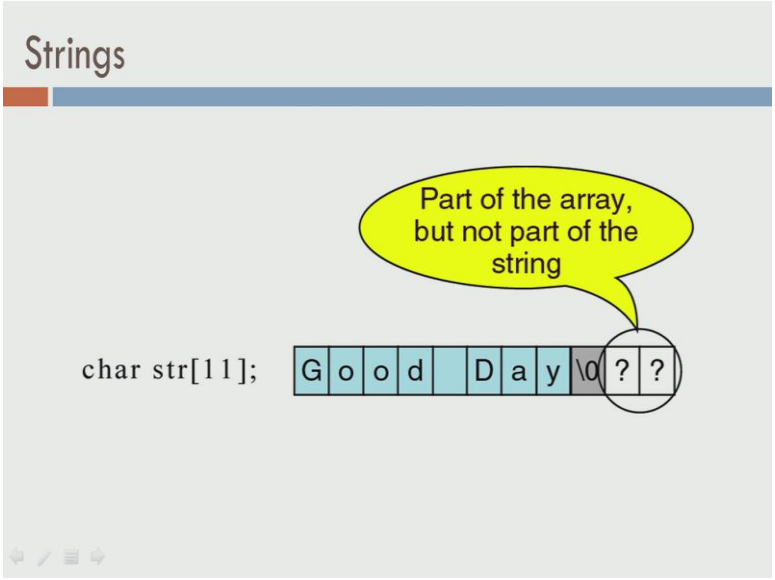


So, a string is essentially a sequence of characters and the way C handles this is, it handles it as a sequence of characters stored in an array and it is not just the characters that you want, it also has a special symbol back slash 0 or the null character. So, if you see this picture here, we have these 5 characters h e l l and o. So, you can see that this h e l l and o are the 5 letters that I need. But, there is an extra character called back slash 0 and the whole 6 characters is called a string.

So, we have a string which actually contains 5 letters and the beginning of the string is h,

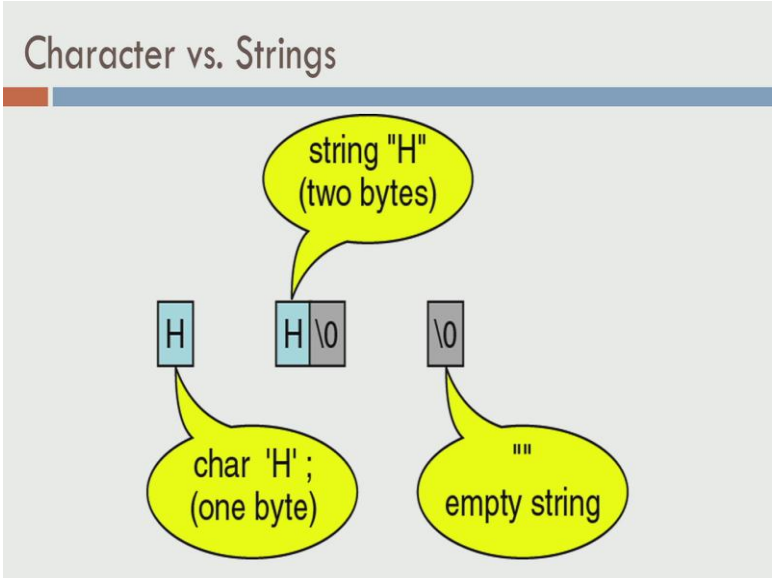
the end of the string is actually back slash 0 which is a special character that C uses. In a little while, we will see why that special character is needed.

(Refer Slide Time: 01:43)



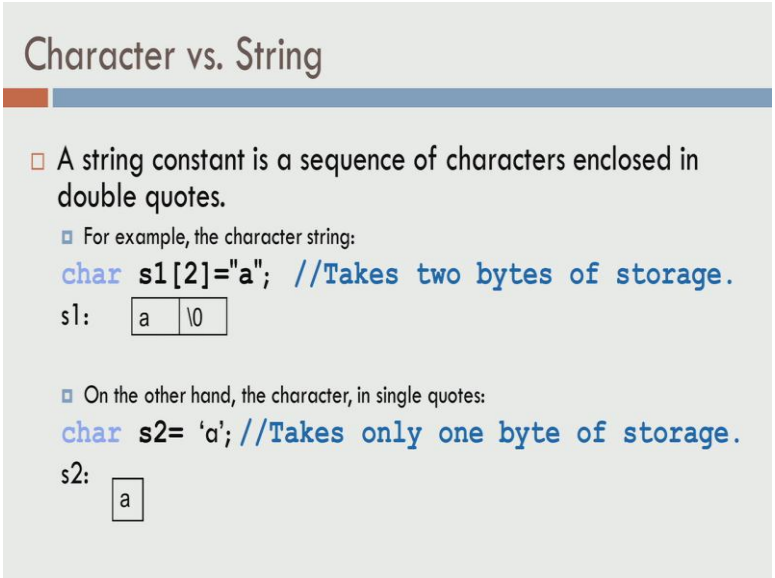
Let us say, I have an array str of length 11, you can see that in this example. The string even though it has 11 bytes of storage allocated to it. It only has a contents good space day, followed by this special character back slash 0. So, remember that is going to be part of every string. There are 2 characters or 2 bytes that are not used by the program. So, this character str of 11, it has 4 characters g o o d, a space that is the 5th character. Then, 6th, 7th, 8th are characters of day, then 9th character back slash 0. So, there are 2 characters that this string could take. So, this array can take two more characters, but the string ends at back slash 0. The other things, the last 2 bytes are not part of the string.

(Refer Slide Time: 02:47)



So, let us see the basic difference between what a character is and what a string is. So, let us I say that there is a character H. So, let us say there is a character called H, it is a single byte representation. So, it stores H and nothing else, whereas, if you store string H, it actually requires 2 bytes. So, it stores the letter H followed by back slash 0. So, this is why I said, every string has at least this character back slash 0. So, even an empty string, a string that has no valid characters at all, will still need, back slash 0 or 1 byte to store it.

(Refer Slide Time: 03:29)

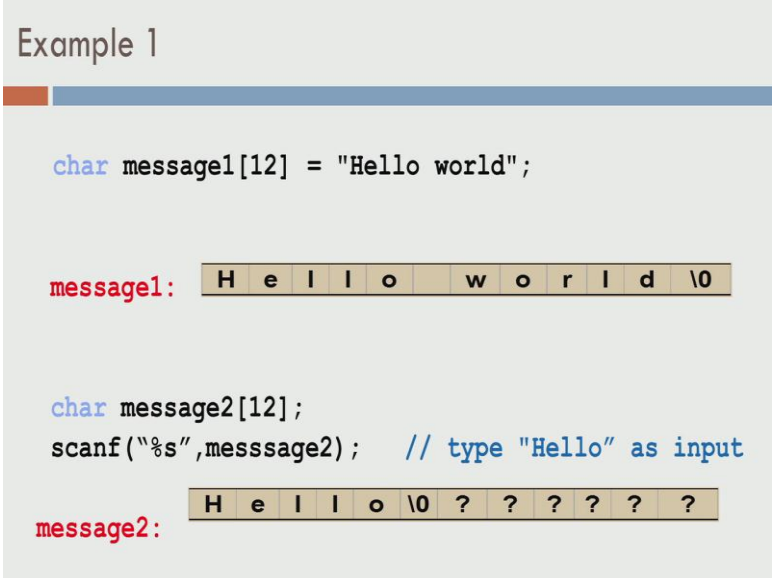


So, to reiterate these points, a string is a sequence of characters enclosed in double quotes. So, in this example in the top, s1 of 2 is declared to be a character array. So, s1 is

a character array of size 2 and equals within double quotes a means, the right side is a string expression. And in the string expression, remember a is a character and because it is a string, back slash 0 is always part of it. So, this will require 2 bytes of storage.

On the other hand if you do it, character s2 equals within single quote a, a is a single character and they require only 1 byte. So, s2 will allocate only 1 byte and you will store the letter a in it. So, this character s2 is a single character, it is not an array and s1 however, is an array of size 2 and it can store a as well as back slash 0. So, there is a difference between the storage that is given to both of these.

(Refer Slide Time: 04:40)



So, let us take a few other examples. Let say, there is this message1 and message2 I have in this example here. So, message1 is declared as an array of size 12 and we have initialized this to the string called hello world. So, hello space world, if you look at the length of it, hello requires 5 characters and world requires 5 characters. So, that is 10, plus space requires 1 character. So, that is 11 bytes.

On the left side, we have allocated 12, the reason for message1 of size 12, the message1 being size 12 is that. Hello and world requires 5 bytes, each plus space is 1 byte, 11 bytes and remembered there is always an implicit back slash 0. You do not have to put it explicitly back slash 0 is something that if you have as a string, back slash 0 is assumed to be added automatically. So, the internal representation of message1 would look like, what you see here.

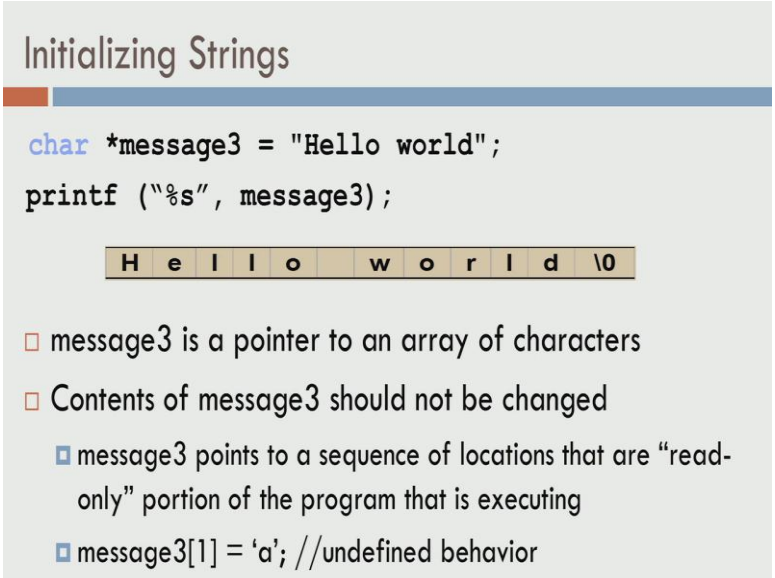
So, you see hello space w o r l d followed by back slash 0. So, it is actually using all the

12 bytes that is allocated to message1. So, it is not that, you always want to initialize it and. So, declare an array and initialize it. Sometimes, you want to read things from the user. For instants, let us say I want to read the message from the user. So, I have message2 which is of size 12 and I scan the message. So, the specifier for scanning a string is percentage s.

So, we have seen how to scan integers and characters and so, on, before. Percentage s is a specifier for scanning a string. So, s stands for string. So, message2 and let us say, I typed hello as input. Then, H e l l o will go as the first 5 characters in message2 and I said C automatically puts in a back slash 0 at the end of the strings, if you scan them from the user. So, back slash 0 is the 6 character. So, that happens in the 5th location.

So, 0th location has H, 1th location has e, 2th location has l and so, on, the 5th location would have back slash 0. And in this case, there are only 6 bytes that are required, the bytes following that even though message2 has space for 12 bytes, it has only 6 valid bytes that are useful for the string, the other 6 are unknown. We have already seen this, when we talked about the notion of arrays and what if, you have not initialized them.

(Refer Slide Time: 07:22)



So, let us look at another example, where we have initialization on the left side, we see character star message3. So, message3 is a pointer to a character or it could be a pointer to a character array, on the right side, we have hello world. So, what happens here is, this hello world again as I said before requires 12 bytes and if you print message3, it will print hello world with a space in between. But, message3 here is actually a pointer to an

array of characters.

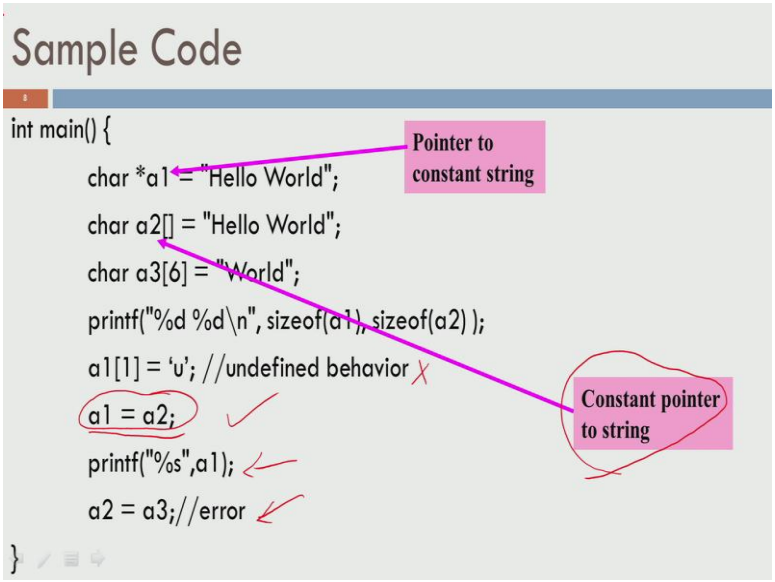
So, one thing that happens is, if you notice the left side, there is no explicit storage allocated for message3. So, unlike message2 or message1 where you see that, there are 12 bytes allocated, message3 does not have any storage allocated to it. So, the way C handles this is, the right side what you see as hello world is treated as a constant string. What; that means, is, this is the string that cannot be change and this string is put into what is called the read only memory of the process or the program, before the program starts running.

So, there is a portion of the memory for every program, which is marked as read only and this hello world as a string is put into the read only portion of the program and message3 is just a pointer to this read only portion of the program. So, we already saw the notion of pointers. So, message3 is just a pointer to the beginning of this array, which has hello followed by space, followed by world, followed by back slash 0.

So, the reason why this is critical is, message3 is not allocated space, you do not have space allocated for it, it is done in the beginning of the program and you do not even have control over it explicitly. So, if you do message3 of 1 equals a and since hello world here is saved in a read only memory location, this changing any contents of message3. Since, you did not allocate space explicitly for it, would result in undefined behavior.

So, this is the common mistake that programmers make. So, if you allocate space like what you see in message1 or message2, it is to go and change contents of it, you can read the characters as characters that you allocated, we saw integer arrays and so, on earlier. So, this is just character arrays. But, if you did not explicitly allocate a space like in message3 now, C allocates space for the strings and you cannot touch it, you cannot change the contents of it.

(Refer Slide Time: 10:06)



So, let us look at a small example. In this case, we have character star a1 which is hello World. So, what we have is, we have a pointer to a constant string. So, what I mean by that is, the string itself is constant, it cannot change and a1 is a pointer to it. Character a2 of square brackets is hello world. What this does is it allocates space for a2 as much as it is required. So, let us look at the right side, the right side requires 11 characters for hello space world plus remember 1 character for back slash 0. So, a2 size will be 12 bytes.

And character a3 of 6, we have explicitly said a3 is supposed to be allocated with 6 bytes. On the right side as long as we have 5 or less characters, everything will be. So, if I said world it is, but if I put worlds for instants, worlds itself will require 6 characters w o r l d s and there will be no more space for back slash 0. So, this example a2, this character a2 is called a constant pointer to strings. So, I want to make this small distinction.

So, a1 is a pointer to constant string and a2 is a constant pointer to string. So, let us see what this difference is. So, a1 is a pointer and it is pointing to a constant string. What; that means, is, you cannot change the string, whereas, character a2 is a constant pointer. What; that means, is, you cannot change it, you cannot make it point to something else, but wherever it is pointing to the contents of it can be changed. So, I will explain this in a little while, but before that let us look at line 4.

So, I want to show the distinction between, how a1 is handle verses how a2 is handled by C. So, if I go and use this sizeof operator, size of is supposed to tell me, the size in the

number of bytes of the data type that is passed to it. So, if you look at size of a1, a1 is a pointer to character and a pointer to character will have the same size as pointer to an integer and so, on. So, size of all the data types could be different, but size of the pointers in a specific machine will all be the same.

So, size of a1 will be the size of the pointer. So, in my machine the size of pointers is all 8 bytes. So, size of a1 will print 8, whereas, size of a2. So, a2 being a constant pointer to a string is actually going to look at, not the size of the pointer itself, but the size of what is pointer to. In this case a2 is pointing to hello world that requires 12 bytes. So, size of a2 would be 12. So, if I took this program and run it on my machine, I would get 8 for the size of a1 and 12 for the size of a2.

So, this distinction happens because, even though pointers and arrays are treated as something which can be interchanged, it is not always true. So, this is a classical example of a place, where pointers and arrays do not really mean the same thing. Now, I said a1 is a pointer to a constant string, therefore, if you make a1 of 1 equals u, you are trying to change a constant string. The string is supposed to be not changing through the program, but you are making a1 of 1 is u, that will result in undefined behavior.

So, you may have a1 even changing to h e l l o, but you cannot always expect that this will work. Sometimes, when you run the program, this can even result in an error. Let us look at the next line, a1 equals a2. So, a1 is a pointer, you can make it point to anything else. So, here a1. So, if you look at a1, a1 is a pointer, it is pointing to something which is constant, but it itself can point to something else. So, in this place we are making a1 point to a2.

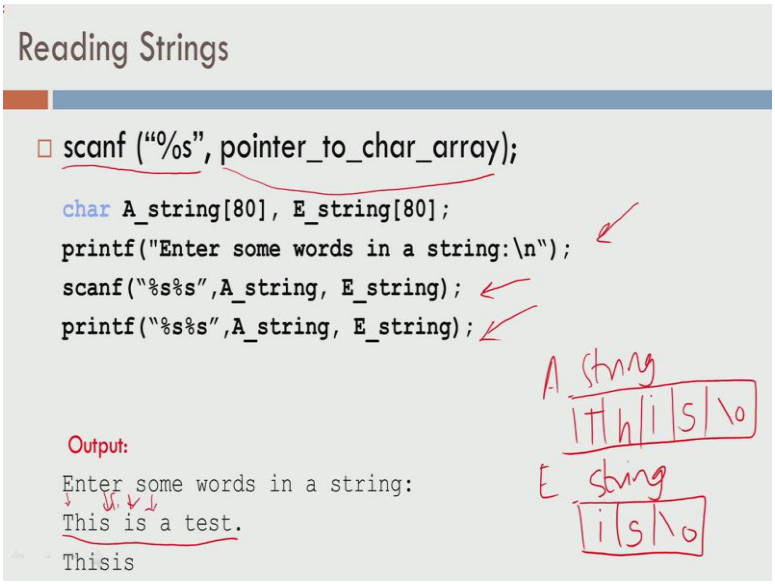
So, this is acceptable, this is not a problem. However, changing the contents of a1 is not acceptable; a1 can start pointing to something else. So, if you see what happens after this printf statement, you will see that a2. So, a1 is pointing to a2. So, it will still print hello world, because a2 is also hello world. Let us look at the last line, a2 equals a3. So, this is where this notion of constant pointer comes through. So, you can change the contents of a2.

So, I can make a2 of 1 equals u, that is acceptable. However, a2 is a constant pointer, what; that means, is, you cannot make it point to something else and therefore, a2 equal to a 3 will result in an error and this error will be got by the compiler itself. So, if you take this program and type it up, your compiler will come and tell you that there is an

error in this line.

So, again I will read this. So, a1 is a pointer to a constant string, you can make a1 point to something else, but you cannot change the contents of a1, a2 is a constant pointer to a string, you can change the contents of a2, but you cannot make a2 point to anything else.

(Refer Slide Time: 15:30)



So, let us go and look at, how strings can be read? We saw how strings can be printed. Let us see, how strings can be read. So, for example, scanf takes this percentage s as a specifier and you can give a pointer to the character array as a parameter to it. So, let us assume that there are two strings, A underscore string and E underscore string and let us assume that both of them have 80 characters in them, which means they can take 79 valid characters and you have to reserve at least 1 byte for the back slash 0.

So, let us say I end up typing something which is 79 characters. The 79 characters will go into location 0 to 78 and automatically back slash 0 will be put in as the 79th character. So, let us say I did printf, enter some words in a string, I prompt you and I scanf, A string and E string and if I print it. So, that is what the program is doing. So, this line is prompting the user to type something. This line is expecting the user to type something in the keyboard. And finally, it is printing whatever is read in the keyboard.

And let us say when this program run, I type this line, this is a test. Let us say, I type that, this space, is space, a space, test, followed by a dot. So, what happens is that this scanf, percentage s starts from a particular location and keeps reading letters, till it finds a white space. So, in this case A string, right when scanf works, it starts with t and then it reads h,

reads i, reads s and it reads space. The moment it is reads space, it says that.

So, scanf the way it works is, space is not scanned into the A string, it stops here, it puts t h i s followed by back slash 0 into A string. And subsequently, when you are reading E string what happens is, it ignores the space, goes to the next location which is not a space, which is i here and it starts reading from there, till it finds another space. In this case, this space in after i s. So, it keeps reading till it finds it is space. So, it found the characters i and s.

So, A string in the memory, it would have t h i s and back slash 0. This will be the 5 letters in A string and if you look at E underscore string, the variable name E underscore string, it would have i s followed by back slash 0. So, it will have 3 characters, even though, you have space for 80 characters, it has only 3 characters of which only two are really valid, these how we read strings.