

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

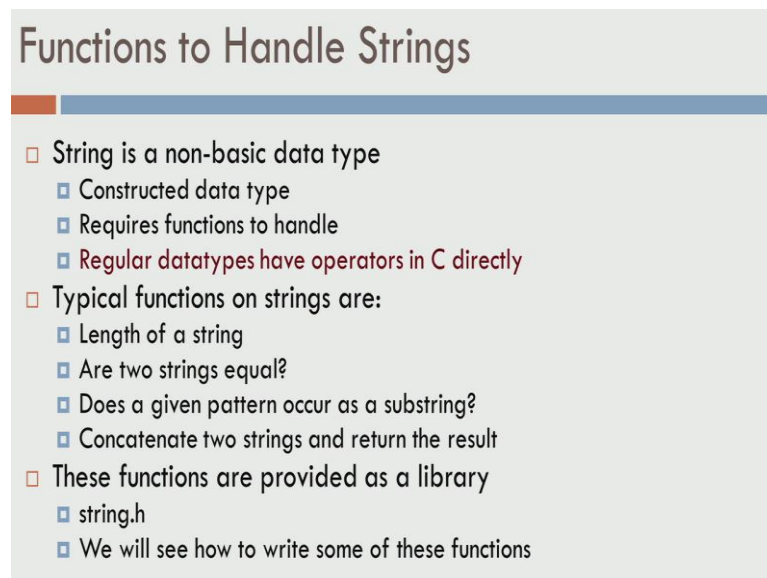
Module - 13b

Lecture - 19

Functions to handle strings; Length, copy, concatenate
Lexicographic ordering
String comparison
String function in library C

There are functions to handle strings, so we will see the notion of functions itself in little a detail later.

(Refer Slide Time: 00:12)



Functions to Handle Strings

- String is a non-basic data type
 - Constructed data type
 - Requires functions to handle
 - Regular datatypes have operators in C directly
- Typical functions on strings are:
 - Length of a string
 - Are two strings equal?
 - Does a given pattern occur as a substring?
 - Concatenate two strings and return the result
- These functions are provided as a library
 - string.h
 - We will see how to write some of these functions

So, the primary thing is string is not a basic data type. So, you have characters, integers, floating points and so on, they are all valid data types and valid basic data types, whereas the notion of a string is not a valid data type. So, there are other languages like C plus plus, java and so on, where string is actually part of the basic data type. Unfortunately in C, it is not. So, what it requires is you have to do something called a derived data type.

So, when I said a string is an array of characters, so it is actually an aggregate data type of characters, you have several elements which are all of the same type. So, it is an aggregate type where you have more than one element of the same type. And to

manipulate various things with functions, we will look at what are called string functions later.

So, the notion of functions itself is not too big, so the idea of functions is if you do something repeatedly, instead of you doing it every time, you typing the program every time, you type the program into what is called a function and you call the function once in a while. So, just like `printf` and `scanf` and so on, these are functions that you are calling, you did not a write `printf` and `scanf`. So, we will see the notion of how to write functions, how to specify them and so on later.

But, as of now various operations is related to strings, we will see what functions are out there that C provides. So, the typical functions on strings are finding out the length of the string, let us say I give my name, I want to find out the length of the string. So, I need there are functions which can go and count the characters in my name and tell me the number of bytes required. Given two strings, I can ask whether the two strings are equal. What I mean by that is, so two strings are said to be equal, if it has the same sequence of characters.

And we can ask a question like is something a sub string of the other or I have two strings, I want to attach one string to the other and so on. So, there are functions for each one of these, these functions are available and given to you in a library called string library to invoke functions or these methods from the string library, you need to include what is called `string dot h`. So, just like for `printf` and `scanf` you included `stdio dot h` for doing various string operations, we will need `string dot h`. So, one thing we are going to do in this class initially is that, we will write these functions ourselves or we will write these programs ourselves and then, I will finally show you how to use the C library calls.

(Refer Slide Time: 03:01)

String Length

Find length of string A

A

```
void stringLength(char *A) {  
    int i = 0;  
    while (A[i] != '\0') i++;  
    return i;  
}
```

So, let us say I give you a string and I want you to find out the length of the string. So, in this example we have string A and let us say string A has Hello space world followed by a back slash 0 and I want to find out the length of the string. So, what is the length of the string? I will start from location which contains H and have to keep checking till I find a back slash 0. So, when I find a back slash 0, I know that it is the end of the string, because that is how C understands strings.

So, anything which is a valid sequence of characters ending with a back slash 0 is a string. So, you go and concentrate on this segment of the program, it starts with i equals to 0, we have i which is 0 and A of i is h to begin with. So, it starts looking at that and it keeps adding, so you can see that in the loop, we are incrementing i by 1, every time I see a non null character. So, the null character is back slash 0, every time I see a non null character, I will increment i by 1.

So, i starts with 0, the moment I see H, i is changed to 1, now I look at e, then e is a non null character, i is changed to 2, I look at l and so on. So, if we keep looking at it, I will see that the length of Hello world is 11, so you have... So, when i equals 11 you have back slash 0, so remember this is A of 0 and this location is actually A of 11. When i equals 11, this is the 12th time we are running the loop. So, i equals 11, it is a back slash 0 character.

So, what you actually have is length of this valid letters, so the array containing all these valid letters excluding back slash 0, so that is the length of the string. So, Hello world has 11 characters in it, so remember there is a space in between. So, what we have is we have i equals to 0 and we have a loop that does it, so do not worry about this notion of return and string length and so on, we will see that in a little while.

So, this is actually a function which takes A which is a pointer to the array and it finds out the length and it returns the length. So, we will see this notion of passing A and returning the values and so on in a lecture on functions later. So, the basic idea is what I want you to grasp.

(Refer Slide Time: 05:34)

String Copy

Copy array A to B

A: H e l l o \0

B: []

```
void stringCopy (char *A, char *B) {  
    int N1 = strlen(A);  
    for (int k=0; k<N1; k++)  
        B[k] = A[k];  
    B[k] = '\0';  
}
```

Let us look at another function in which I want to copy the contents of array A to B. Let us say, array A is of some length and array B is of something else and I want to copy array A to array B and the way to do that is, so I want to go and look at size of this array. So, what I want to do is I do not care about what is present in B, B could be actually of a size which is different from A. I also do not care about what are the contents of B already in place. I want to copy byte by byte or character by character, the contents of A into contents of B.

So, the H in location 0 of A should be copied to location 0 of B. For example, this l should be copied to this location, this back slash 0 should be copied here. Because, I am supposed to copy the whole string and the notion of a string is only if it is terminated by

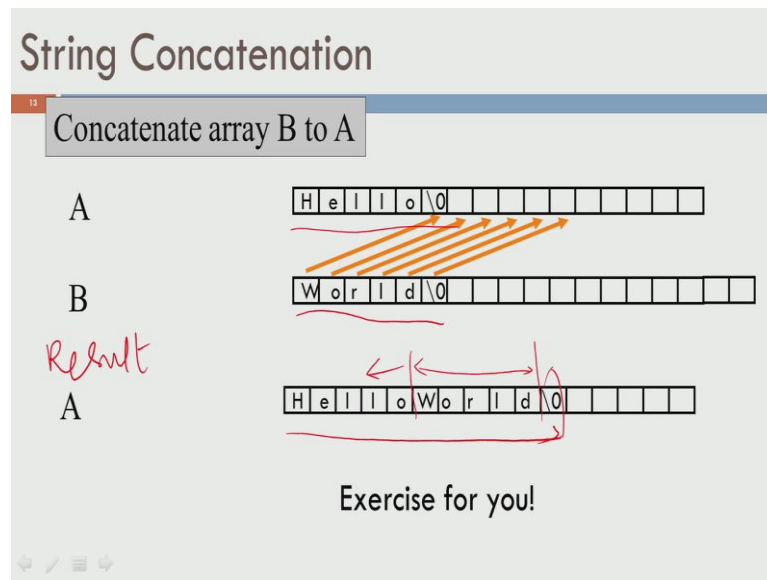
the null character. So, that is what this program segment does, so `int N1` is string length of `N1`. So, remember I wrote this thing called string length, if I pass `Hello` back slash 0 to it, it will tell me that the length of the string is 5 and we have `k` equals to 0, `k` less than `N1`, `k` plus plus. So, it will go from 0, 1, 2, 3, 4 and you will keep copying contents of location A to B.

So, at the end of this you actually have to make B of `k` equals back slash 0 or you could actually change this to equal to. So, this only copies all the valid characters, we still need to do the last one, so we need to write B of `k` equals back slash 0. This is one way to do it or you change this condition to equal to, either one of them will take care of copying the string including back slash 0. So, one thing that this segment does is it ignores whatever was stored in B earlier.

So, there is a certain problem here, let us say A had only 4 bytes, so let us say A was all these bytes whatever number of bytes, but B does not have enough space to copy the contents of A. This will be a problem, because you are going to start looking at B and when I start looking at B, I am going to look for back slash 0 as a possibility. And for whatever reason, let us say I did not have any of this space, I would have copied H e l and l, but I did not have enough space for o and back slash 0.

This should be a disaster, because my array has lesser space than what it really needs. This is not something that you have to handle it explicitly, if B has lesser space than A, what is it mean to say copy the contents of A to B, you have to be careful about that.

(Refer Slide Time: 08:28)



Then, let us look at this function called string concatenation or the idea of concatenation. Let us say I have this string called world and I have a string called Hello and I want to make a single string Hello World. So, that is called concatenation, what I really want is I want to copy W to this back slash 0, o to the next location or to the location after that and so on, including back slash 0 of B to be copied here. The results I expect is Hello which were the contents of A followed by the contents of B, which is World followed by back slash 0 which says that this is the termination of A.

So, I am trying to concatenate B to A and as a result I want this, so you can see how this is supposed to be done. So, clearly A must have space for copying B to the end of it, so if A had lesser space than the size of B plus A included, there is a problem. So, in this example it is not a problem, but if B were something much larger, then concatenating B with A, you will run out of space in A. So, in this example however, it is not a problem.

Once we did the concatenation, so up till here is what you got from A, from here to here is what you got from B and this is the extra character that you have to add to the end to ensure that when you now treat A as a string from left to right, it terminates with B. So, I am going to leave this as an exercise for you to go and write a program on your own.

(Refer Slide Time: 10:12)

The slide is titled "Lexicographic Ordering". It contains a list of comparisons with handwritten annotations:

- Badri < Devendra (with a red circle around 'D' in Devendra)
- Janak < Janaki
- Shiva < Shivendra
- Seeta < Sita
- Badri < badri
- Bad < Badri
- Bad < badri ✓ (handwritten in red)
- Based on the ordering of characters

Below the list is the character ordering sequence: A < B ... < Y < Z < a < b < c < ... < y < z

A callout box on the right says "upper case before lower case".

Finally, let us look at the notion of ordering strings and so this is something that we do in our daily life. So, if I go and pickup dictionary I may want to go and look at what all the order in which the words are occurring. So, in this slide I am looking at names of people or words, so let us look at the word Badri and word Devendra. So, clearly if I go and arrange them in dictionary, the word Badri will come before the word Devendra on the dictionary. So, I will say that word Badri coming before word Devendra, I will use the notation less than for it.

So, Badri comes before Devendra is expressed as Badri less than Devendra. Let us look at this example, Janak is less than Janaki, so Janak has only five characters whereas Janaki has six characters, so clearly if you go and look at dictionary, all the smaller words will appear before all the bigger words. So, in this case Janak is a smaller word than Janaki, so even though the first five characters of both these are same, Janak has one less character than Janaki. So, therefore, we will say that Janak is less than Janaki.

Then, if you look at this example, Shiva is less than Shivendra, because if you go and look at the fifth character, it is A here, the fifth character here is E. So, clearly A will come before E in the dictionary, so Shiva is less than Shivendra. Then, Seeta is less than Sita, so Seeta is less than Sita, so we are not looking at the size of the string alone, we are also looking at dictionary ordering. So, clearly Seeta will come in the dictionary before

Sita. Even though the length of Sita is only four, Seeta is of length five, in English dictionary you would see Seeta before Sita.

Then, there is something peculiar about characters and languages. So, one thing that is true in Machines is that all the upper case letters in an alphabetical order appear actually before all the lower case letters. So, this is a property of how the machine treats various characters. All the upper case letters are supposed to be lesser than all the lower case letters and within the upper case letters A is less than B and so on up to Z and within the lower case letters, a is lesser than b and so on up to z.

But, capital A is actually less than lower case a, capital Z is less than lower case a and lower case z and so on. Because of that if I go and look at this example, capital B Badri is actually less than small case b badri. So, in dictionaries you do not really have to make a distinction between upper case and lower case, but in programming, strings can be different. So, lower case could be different from upper case, so we have to be careful about that.

As a final example, Bad is actually less than Badri with capital B, because Bad has three characters and Badri has five characters. So, this is similar to this Janaki and Janak example. So, in fact Bad is also less than Badri, so that is also true, because Bad is actually three characters in length and so Bad starts with the capital B and whereas badri starts with lower case b, so this is also true, so this is called lexicographic ordering.

So, just remember that it is just like dictionary ordering, with the one extra addition that lower case letters are considered to be coming later than upper case letters. So, we will also have to deal with blanks and other things for example. So, let us look at this example Bill Clinton and Bill Gates.

(Refer Slide Time: 14:16)

Lexicographic ordering

- What about blanks?
 - "Bill Clinton" < "Bill Gates"
 - "Ram Subramanian" < "Ram Subramanium"
 - "Ram Subramanian" < "Rama Awasthi"
- In ASCII the blank (code = 32) comes before all other characters. The above cases are taken care of automatically.

So, if you go and arrange them in order, Bill Clinton should actually come before Bill Gates, so the space itself should not matter, so C comes before G. Therefore, if I go and look at character by character from the left side of this and character by character from the left side of this, so B and B match, i and i match, l and l match, l and l match, space and space matches, C is actually less than G, therefore Bill Clinton is less than Bill Gates.

So, I am not making any comment over their personalities, so in terms of a string Bill Clinton is less than Bill Gates. Then, let us say we have this example Ram Subramanian and Ram Subramanium. So, here up till i the characters are the same, whereas when we look at this character a and this u, a is less than u therefore, the string Ram Subramanian is less than the string Ram Subramanium. And finally, if we go and look at this example, Ram space Subramanian verses Rama space Awasthi.

So, Ram space Subramanian is supposed to be lexicographically earlier than Rama space Awasthi. The reason is that if you go and look at the first three, they are matching Ram and Ram, the fourth character is a space whereas the fourth character is a and in storage space gets a code 32 and that comes before the code for a, which is actually 97. So, the code for lower case a is 97, so code for space is 32, so since space comes before any valid letter in the sequence of characters, Ram space Subramaian is less than Rama space Awasthi.

(Refer Slide Time: 16:05)

String Comparison

```
int strcmp(char *A, char *B, int N1, int N2) {  
    int k=0;  
    while ((A[k] == B[k] && k < N1 && k < N2)  
           k++);  
    if (N1 == N2 && k == N1) printf("A = B");  
    else if (A[k] == '\0') printf("A < B");  
    else if (B[k] == '\0') printf("A > B");  
    else if (A[k] < B[k]) printf("A < B");  
    else printf("A > B");  
}
```

Handwritten notes on the slide:

- Red box around the while loop condition: `while ((A[k] == B[k] && k < N1 && k < N2) k++);`
- Red checkmarks next to the if-else statements.
- Handwritten examples on the right side:
 - A: Hello, B: Hello (A=B)
 - A: Hell, B: Hello (A<B)
 - A: Hello, B: Hell (A>B)
 - A: Bell, B: Bull (A<B)
 - A: Hull, B: Hello (A>B)

So, there is a small program that I have written here which goes and looks at comparing two strings `strcmp`. I am assuming that there are two strings A and B, so this is supposed to be a comma here and let us assume that there are two integers N1 and N2. N1 is supposed to be the length of A and N2 is supposed to be the length of B. So, I want to find out, if when I compare A and B, whether A is less than B or A equal to B or whether A is greater than B.

So, let us look at the examples on the right side, if A is Hello and B is Hello, then A is actually equal to B. So, for this example this one, this condition A is actually equal to B. In this example, the second one A equals Hell and B equals Hello, so the set of characters first four characters are the same, but A does not have the fifth character, whereas B is a longer string. So, therefore we claim that A is less than B, like I said small words will come in the dictionary before bigger words.

And if A is Hello and B is Hell, A is actually greater than B, then in this example the both are of the same length, the length is not matter, but if you look at the second character there is e here and there is u here. Therefore, in the dictionary Bell will come before Bull, therefore string A is before B and in this example, Hull is and Hello, Hull will come in the dictionary after Hello, so we have A greater than B. So, these are some examples I want to be able to handle all such cases.

So, let us start with the first one A equals B. When will I have two strings to be equal? All the characters of A must be the same as the characters of B and the length of A should be equal to the length of B also. In fact every character including back slash 0 should be matched from A to B. If all the characters including back slash 0 are exactly the same, then in the order for A and B, then A is supposed to be equal to B.

Let us see, how to detect conditions A less than B. What are the two cases in which A is less than B? I have matching characters for A and B, but A ran out of characters, so in this case Hell has only four characters and Hello has five valid characters. I ran out of characters in A to compare with B, in that case A is less than B or I do not ran out of characters, but I am still in the middle of checking characters, but I see that some character in the middle is actually less than some character at the end. In this case also we have A is less than B.

So, these are the two conditions or examples for which A is less than B and for everything else, we already considered A equals B and I showed you, when A is less than B, in all the other cases A is supposed to be greater than B. So, this program that you see on the left side is supposed to take care of printing whether A equals B or A less than B or A greater than B based on these checks. So, let us see what this loop here is supposed to be doing.

So, k is initially initialized to 0, we are starting with the 0th character of both A and B and if both the characters of A and B match and if I have not exhausted A and I have not exhausted B, then I can go and look for one more character. So, if I keep doing this, k will keep incrementing. So, if I start with let us say Hello and Hell, k is supposed to point to the 0th character of both. So, let us say this is my A and this is my B, so k initially starts with pointing to H, these both are same I increment k.

So, k now, so A of k is e and B of k is e, these two are same. Then, I increment k, now A of 2 is l and B of 2 is l. So, as long as things are same I have to keep moving and I will stop, when either the condition that the characters are same is violated or I ran out of characters in either A or B, so that is what these three conditions are doing. So, now let us go and see there are three ways in which you could have come out of this loop.

Either, you violate this condition or you violate this condition or you violate this condition, you could have come out because of violation of any of these. If A of k equals

B of k for all the characters that you saw so far, then at some point you are N1, you would have hid N1 the end of A and you would have hid N2 which is the end of B. So, if N1 is actually equal to N2 and if k equals N1 which means you have looking for equality in A and B, then A and B are equal.

So, that is the example we have on the right side Hello and A equals Hello and B equals Hello are actually the same, else you ran out of characters in A, but the first set of characters N1 characters in A and B are the same. But, you ran out of characters in A, there are more characters in B to process which means you have a word which is shorter than the other word and so for example, Hell is shorter than Hello and Hell is a prefix of Hello. In this case A is less than B.

If you ran out of characters in B and if you still have characters in A and if the first N2 positions matched, then you print A is greater than B. So, we have taking care of three cases where we hid in some sense back slash 0 So, in this case we hid back slash 0 on both of them with the same time, in this case we hid only on A of 0, in this case we have hid only on B of 0, none of these need to be true. For example, in Bell and Bull, I do not have to go to the end of the array to see that Bell is less than Bull.

As soon as I see the 1th location, I see that e is less than u therefore, A is less than B, so that is what you are doing here. I did not run out of characters in either A or B, however so there are still more characters to process in A and B, but however I found something lexicographically different. So, A of k is less than B of k means A is less than B, in this case Bell is less than Bull or I started looking at H in both of these, I see that in the next location u and e, u is actually greater than e. So, Hull should come after Hello and therefore, A is greater than B.

So, this is the way in which you can write the program for string comparison. So, we looked at three programs, one to find out the length of the string, one to concatenate and one to copy and finally, we now looked at comparing two strings. So, one thing that I did not show you is for concatenation, I did not show you how to write the program. So, we can keep writing all these things every time, but it is a laborious thing. So, for integers and floating point and so on, we had plus and minus and so on which are basic operations.

However for strings, it may be useful to have some of these things given by the programming language. Unfortunately C does not give that, but C gives it you as a library. So, these are all built in string functions. If you do hash include string dot h, it has several built in functions.

(Refer Slide Time: 24:17)

Built-in string functions

- `#include <string.h>`
- `int strlen(const char* s)` - strlen returns the length of a string, excluding the NUL character.
- `char* strcpy(char* dest, char* src)` - strcpy copies one string to another. The destination must be large enough to accept the contents of the source string.
- `char* strcat(char* dest, char* src)` - strcat combines two strings and returns a pointer to the destination string. In order for this function to work, you **must** have enough room in the destination to accommodate both strings.

So, for example you have a function called strlen. If you pass character array to it, it will find out the length of the string excluding the length of the null character and return an integer. So, strlen is a function which takes a character array as a parameter. So, it takes pointer to a character array and returns the length of the string. So, you will see the notation and meaning of this in a subsequent video on functions, so you can treat these as a inputs that is given and this is the output.

So, the input is of type string and the output is of type integer, strlen is the name of the function. It takes a character array or pointer to a character array as an input. So, it has a program like what we saw for the length inside which actually goes and looks at every single character and finds out back slash 0. It reports the length and returns an integer. There is a function called strcpy, so you take two parameters character star destination and character star source.

So, if you give strcpy of A comma B, it will copy contents of B in to A. So, clearly destination must have a size which is at least as large as the size of source. Finally, there is another function called strcat. So, again as before we have destination and source, so it

will take source and concatenate it to destination. So, destination must have space at least as big as the original string that you had plus the new string that you have concatenating.

So, these are things available from as built in functions from C itself and we can write it on our own, but we are prone to making mistakes, instead we use libraries given by C to do this. So, these are three basic things that you may want to use. So, there is another function called strcmp which can also come in handy.

(Refer Slide Time: 26:18)

Built-in string comparison

- `int strcmp(const char *s1, const char *s2);`
- `int strncmp(const char *s1, const char *s2, size_t n);`

The return values are

- 0 if both strings are equal.
- 1 if first string is lexicographically greater than second.
- -1 if second string is lexicographically greater than first.

Compares first n characters only

Strcmp takes two parameters as inputs which are both pointer to arrays and it returns an integer. So, we wrote a program which printed things on the screen as A less than B, A equal to B and A greater than B, instead it returns an integer. So, if both s1 and s2 are equal strings which means, if I gave Hello and Hello as s1 and s2, then both strings are equal, you will get 0 as the result. Instead of printing on the screen it gives you an integer, you can go and look at the integer value and decide, whether they are equal or not.

You get 1, if the first string is lexicographically greater than the second string which means, in the dictionary if the word pointed to by s1 will come later than s2, then you will get 1. You will get minus 1 as the return from strcmp, if s1 will be in the dictionary order before s2. So, you get three values 0, 1 and minus 1 depending on whether A equals B and whether A greater than B or whether A is less than B. There is also another function called strncmp.

What it does is, instead of comparing all the characters including back slash 0. It will compare only the first n characters of two strings. So, if I have string s1 let us say, this is of size 10 and let us say s2 is an array of size 15 and if I specify size as 4, strcmp will compare only the first four characters, it will not go till the end of s1 or s2, it will go only till the first four characters and it will give you a comparison of the first four characters of A and B. It will still returns 0, 1 and minus 1 by comparing only the first n characters.

So, this brings me to the conclusion of strings, this is something that can come in quit handy for you. So, later you will see programming exercises which are actually on strings. So, I suggest that instead of writing things on your own, you go and use the functions that are given in the string library. All you have to do is do hash include string dot h and just like printf, scanf and so on, you call functions by passing appropriate parameters. So, how to pass parameters and so on will probably become clear, once you go through the video for functions.

So, thank you very much and see you in the next lecture.