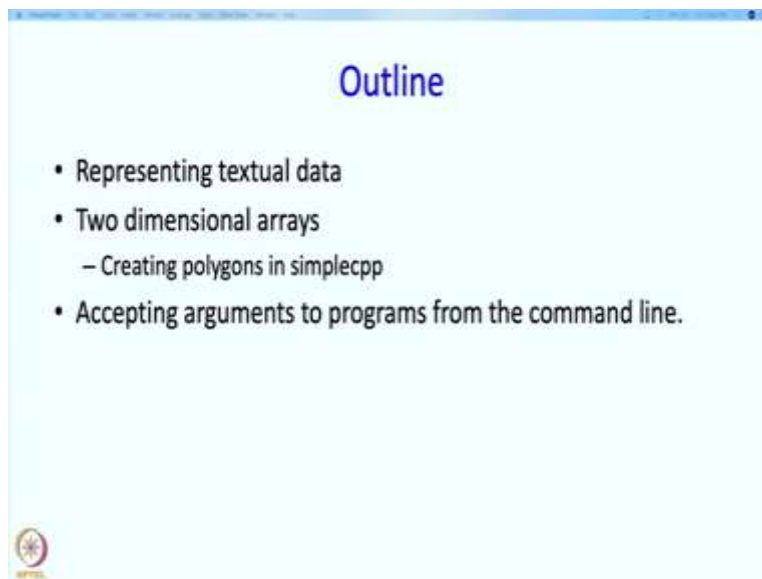


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 17 Part - 1
More on Arrays
Textual data

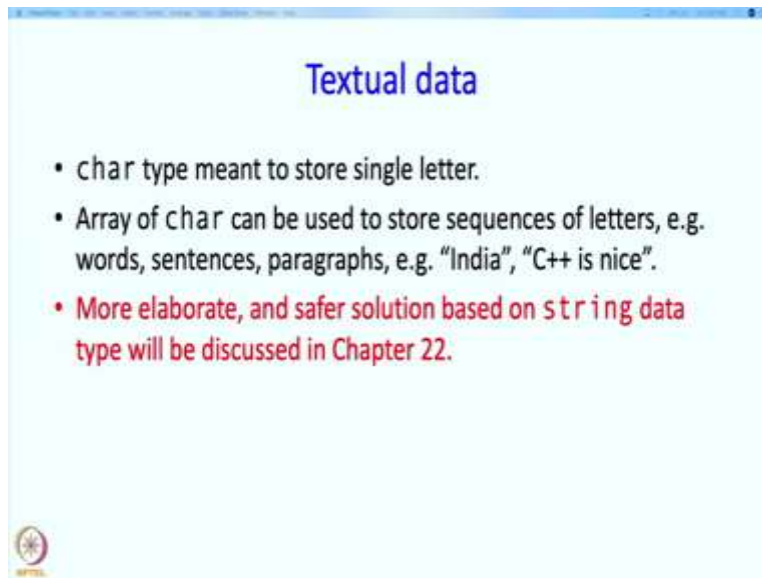
Hello and welcome to the NPTEL course on An Introduction to Programming through C++. I am Abhiram Ranade, and today's lecture is also about arrays and the reading for it is from chapter 15 of the text.

(Refer Slide Time: 0:35)



So, here is an outline, so the first topic that we are going to look at is how to represent textual data, then we will talk about two-dimensional arrays and in this, one of the examples will be how to create polygons in simplecpp. And then we will talk about accepting command, accepting command line arguments.

(Refer Slide Time: 0:55)



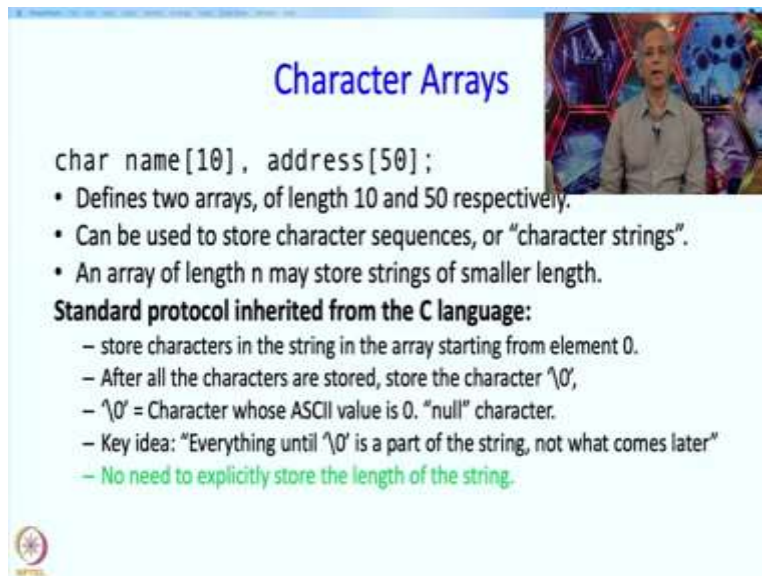
The slide is titled "Textual data" in blue text. It contains three bullet points:

- char type meant to store single letter.
- Array of char can be used to store sequences of letters, e.g. words, sentences, paragraphs, e.g. "India", "C++ is nice".
- More elaborate, and safer solution based on string data type will be discussed in Chapter 22.

A small circular logo is visible in the bottom left corner of the slide.

Textual data, we have already talked about a little bit. So we said that the char type was specially meant for storing single letters. Now, obviously the natural step would be to use an array of char to store sequences of letters. So, for example, you could use arrays, an array of char to store words, sentences, paragraphs, maybe a character string like "India", maybe a character string like "C++ is nice" with spaces inside it. And the textual data representation that I am going to talk about today is derived from the C language. C++ has nicer representations and we will learn about them a little bit later, but whatever we learnt today will also be useful even in that nicer representation.

(Refer Slide Time: 1:58)



Character Arrays

```
char name[10], address[50];
```

- Defines two arrays, of length 10 and 50 respectively.
- Can be used to store character sequences, or “character strings”.
- An array of length n may store strings of smaller length.

Standard protocol inherited from the C language:

- store characters in the string in the array starting from element 0.
- After all the characters are stored, store the character `'\0'`,
- `'\0'` = Character whose ASCII value is 0. “null” character.
- Key idea: “Everything until `'\0'` is a part of the string, not what comes later”
- No need to explicitly store the length of the string.

So, character arrays we already know, well we know arrays and therefore, character arrays are not really different. So as always we define an array by giving the type name, the name of the array and then in square brackets the size. So, here I am defining two character arrays, one array called name, of length 10, and another array called address, of length 50 and both have the type char. So such arrays can be used to store character sequences or character strings as they are more often called.

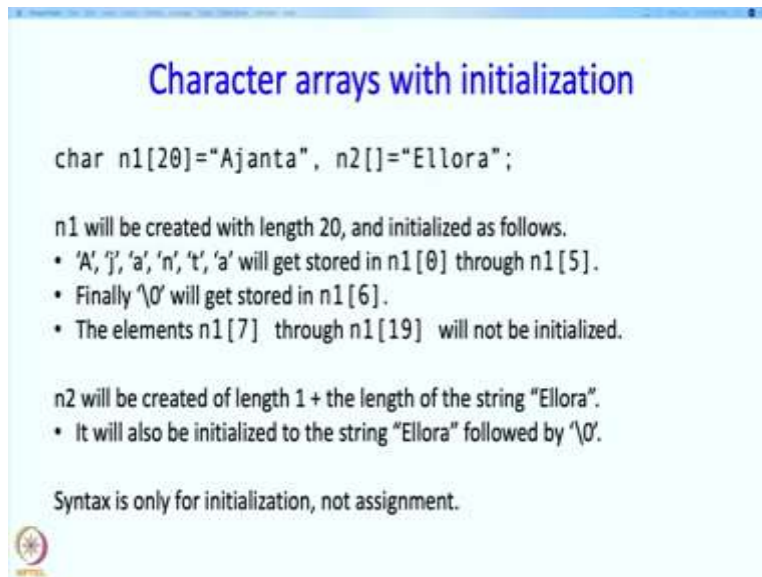
And you may, you often use an array of a larger length to store strings of smaller length that is because often you do not know how long the string is that is you want to store, say in the name in the in the array name because you may not, you may not always calculate what is the exact name of this person before storing the name of the person into the array, you may just allocate a large enough array for storing say the name of a person.

Now the arrays are character arrays are really not any different from other arrays, but there is a difference in the way they get used. Specially there is a protocol for storing character strings that is inherited from the C language and here is what the protocol is. So, if you want to store a character string into an array you store the string in the array starting from element 0. After all the characters in your character string are stored in the array, you also store the character whose ASCII representation is 0, often written as “`\0`”, this character is often also called the null character.

So what you store actually is the string that you want to store followed by this null character. So the null character sort of signals that the portion of interest has ended. And indeed, the way everything happens that character arrays is not using the defined length of the array, but rather the idea that only everything until the null is a part of the actual string and what comes later is supposed to be ignored.

So, in fact you will see that when you are processing character strings stored in character arrays, you do not really talk about the length because the length is never specified explicitly. So you just process until you find the null character.

(Refer Slide Time: 5:09)



Character arrays with initialization

```
char n1[20]="Ajanta", n2[]="Ellora";
```


n1 will be created with length 20, and initialized as follows.

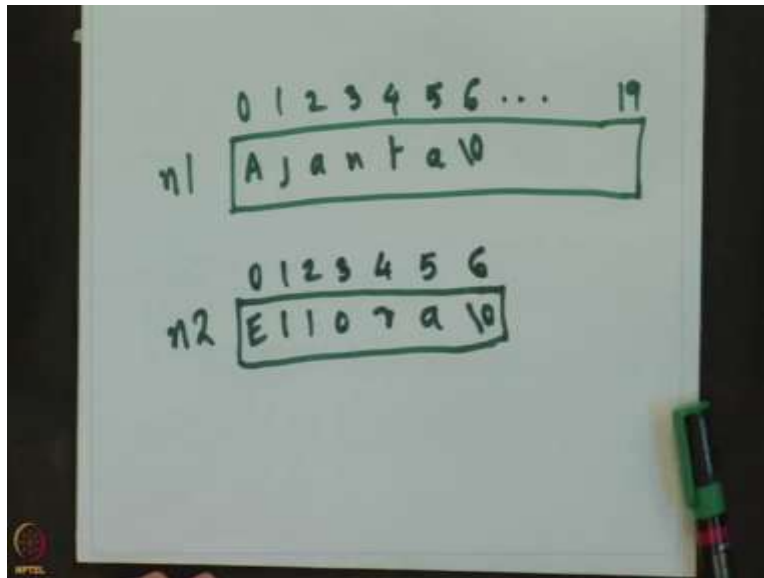
- 'A', 'j', 'a', 'n', 't', 'a' will get stored in n1[0] through n1[5].
- Finally '\0' will get stored in n1[6].
- The elements n1[7] through n1[19] will not be initialized.

n2 will be created of length 1 + the length of the string "Ellora".

- It will also be initialized to the string "Ellora" followed by '\0'.

Syntax is only for initialization, not assignment.





Now you can create character arrays with initialization. So for example, you might write `char n1[20]`, so you are allocating an array of length 20 and in which you are storing “Ajanta” and you are allocating an array `n2` without giving the size in which you are storing “Ellora”. These definitions are analogous to the definitions that you have seen say for defining arrays of integers. The initialization form for character strings is slightly different you do not need to, you do not enclose individual characters in braces, but you can just put up character strings.

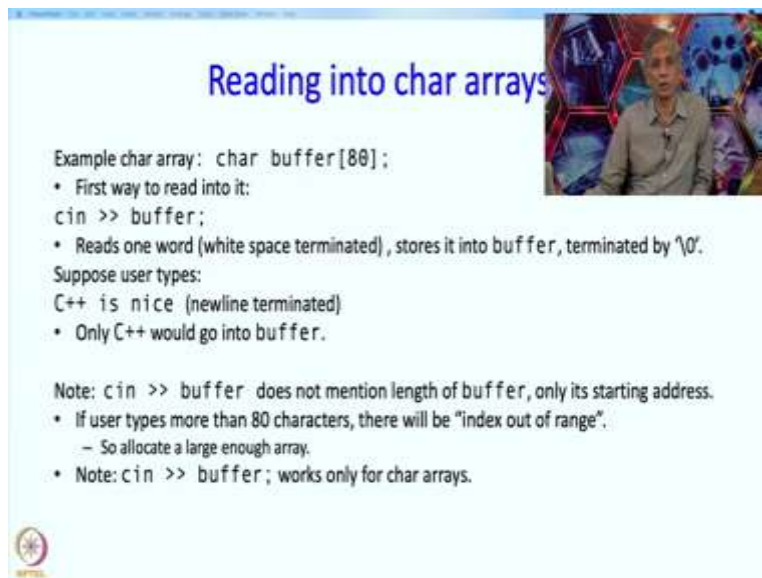
So, how does this work? So `n1` will be created with length 20 and it will be initialized as follows. So `A` will be put, capital ‘`A`’ will be put in `n1[0]`, ‘`j`’ will be put in `n1[1]`, the next ‘`a`’ will be put in `n1[2]` and so on. So ‘`A`’, ‘`j`’, ‘`a`’, ‘`n`’, ‘`t`’, ‘`a`’ will be stored in array elements `n1[0]` through `n1[5]`, but that is not all, in `n1[6]` the null character will be stored. So this is this as I said this is just according to the protocol that we talked about earlier. And the elements `n1[7]` through `n1[19]` the remaining elements of this array “Ajanta” will not be initialized and it is expected in fact that if you are going to process this array these elements will not be looked at, so only the elements until the null character will be looked at.

`n2` will be created of length 1 plus the length of the string “Ellora”, the length of the string “Ellora” is `E-l-l-o-r-a`, 6 plus 1, so 7 the size of `n2` will be 7. And it will also be initialized to the string “Ellora” followed by the null, null character. So just to draw a picture, `n1` will look like this, so this will be index 0, this will be index 19, and then in `n1`, we will store `A-j-a-n-t-a` and then this null character, so this is 0 1 2 3 4 5 and 6 the null character is stored.

n2, because the length of n2 is not given, n2 will be created so that it uses the minimum length. And so what will it look like? It will look like E-l-l-o-r-a and then the null character, so this requires 7 characters and in fact exactly 7 elements will be allocated. So element 0 through element 6. And I should point out that the syntax n1[20]="Ajanta" looks like an assignment, but that is not true you cannot, you cannot use this as an assignment, you can only use it as an initialization.

You can assign individual elements of the array, so you could write n1[0]='a', n1[1]='j', and so on. But assignments will be one element at a time and this feature of assigning a lot of elements simultaneously is reserved only for initialization.

(Refer Slide Time: 9:22)



Reading into char arrays

Example char array: `char buffer[80];`

- First way to read into it:
`cin >> buffer;`
- Reads one word (white space terminated), stores it into `buffer`, terminated by `'\0'`.

Suppose user types:
`C++ is nice` (newline terminated)

- Only `C++` would go into `buffer`.

Note: `cin >> buffer` does not mention length of `buffer`, only its starting address.

- If user types more than 80 characters, there will be "index out of range".
 - So allocate a large enough array.
- Note: `cin >> buffer;` works only for char arrays.

So we have declared arrays and maybe we initialize them, but we surely want to read data into arrays, so this is the first way in which we are going to be able to read data. So for example, we have a character array 'buffer' of length 80. Suppose it has been defined as shown here, the first way to read into it is to you cin greater than greater than our standard mechanism. And this for the purposes of reading into character arrays works in the following manner. So it reads one word and what is the word? Well, it is anything which is terminated by white space, white space is a space character or a newline character, tab character so any of these appears then the word will be considered to have ended. So whatever that one word was typed in by the user will be taken and will be stored into this array buffer from the beginning and it will be terminated by a

null character. So suppose the user types “C++ is nice”, so there is C++-space-is-space-nice and after that nice say there is a new line.

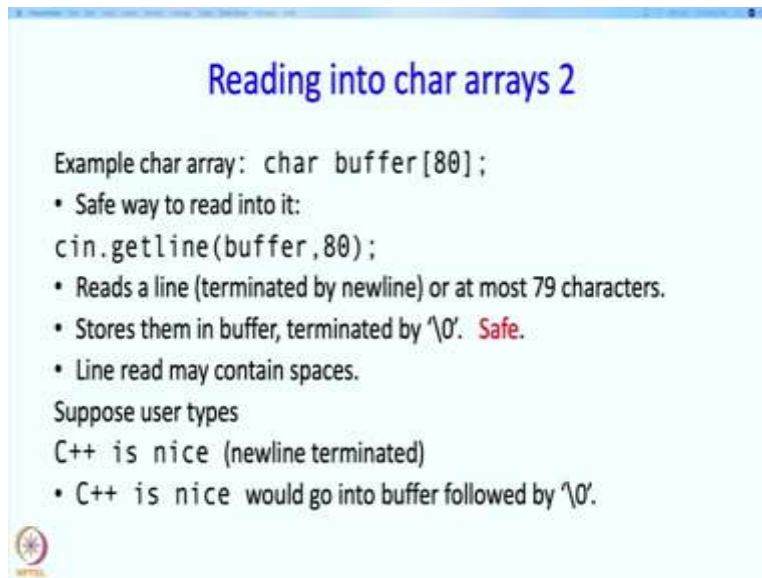
If you remember you have to hit a new line before the computer actually starts looking at what you typed. So say here you hit a new line, so what happens in this case? So in this case as we have said one word is read, so the first word is C++, so that C++ is read so three characters are read and they are stored into `buffer[0]`, `buffer[1]`, `buffer[2]` and following them the null character is stored.

Now `cin>>buffer` does not mention the length of the buffer because if you remember the name of the array is only the starting address, so this is needed of course you need to know where to start putting in the data that you are just reading. But there is also problem here, so if the user types more than 80 characters, then this will cause an index out of range error. C++ will attempt to store whatever you typed into say index 80, 81 how many over characters you need and that will cause an error.

So, what should you do? So, we will have a more safer way to read, we will show that all in a minute, but the standard idea is to just allocate a large enough array or the simple idea I should say. So quite possibly, if you want to write a really good program you should probably not be doing this, this is sort of a quick, this is a way to quick, write a quick trusting program. I should also mention that the command `cin>>buffer` works only for character arrays.

In fact, `buffer` is constant, so normally `cin>>` is supposed to have a variable after it. So normally if you type any symbol that symbol had better be a variable, but for character arrays there is a special case made by C++ so that it is interpreted as put whatever is read into this array starting from the 0th character. So if it is not a 0 character, if it is not a character array however, then C++ does not allow you to read things in this manner, so that will actually cause an error, say `buffer` is an integer array then you write `Cin buffer` would cause an error.

(Refer Slide Time: 13:40)




Reading into char arrays 2

Example char array: `char buffer[80];`

- Safe way to read into it:
`cin.getline(buffer, 80);`
- Reads a line (terminated by newline) or at most 79 characters.
- Stores them in buffer, terminated by `'\0'`. **Safe.**
- Line read may contain spaces.

Suppose user types
`C++ is nice` (newline terminated)

- `C++ is nice` would go into buffer followed by `'\0'`.



Now here is a different way to read into character arrays, so again we have our same definition `char buffer 80`, the safe way to read into it is this we say `cin.getline(buffer, 80)`, here you can notice that we are specifying the length, so C++ will actually make sure that the reading does not happen beyond the array boundary. So this will read a line and this time it is not terminated by white space, but it is terminated by a new line, and if the line is very long then C++ will only pick up the first 79 characters.

So whatever is picked up will be placed in buffer and it will be followed by the null character. So this is safe because you do not attempt to write beyond the end of your allocated region and another difference between this and what we had earlier is that the line that you read may contain spaces. So, if you want to get in spaces into what you are reading this is the way to do it. As an example, if the user types “C++ is nice” as before and again after nice there is a new line so the computer starts processing it, this time that entire text would go into the buffer followed by the null character.

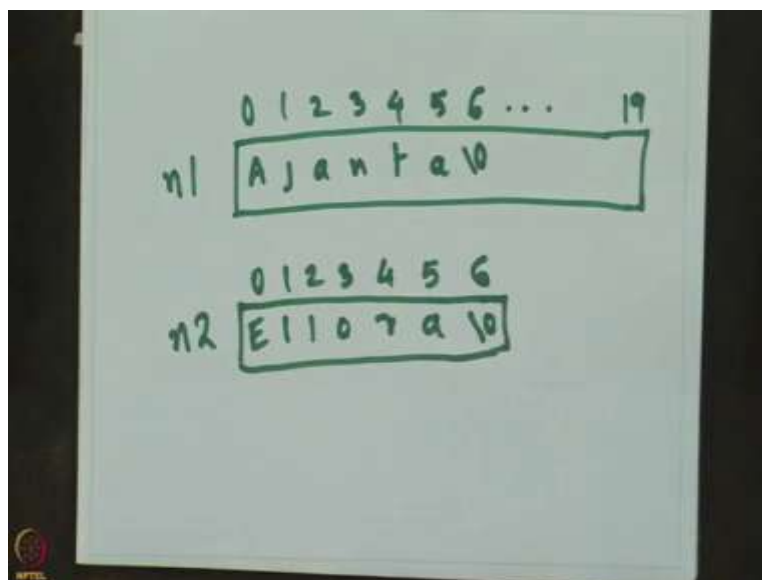

(Refer Slide Time: 15:20)

Printing char arrays

Example char array: `char buffer[80];`
... Code to assign value to buffer ...

- How to print:
`cout << buffer;`
- Print buffer content till `'\0'`.
- buffer assumed to contain a `'\0'`.
- Length of array is not important.

Note: For other types of arrays above would print address.



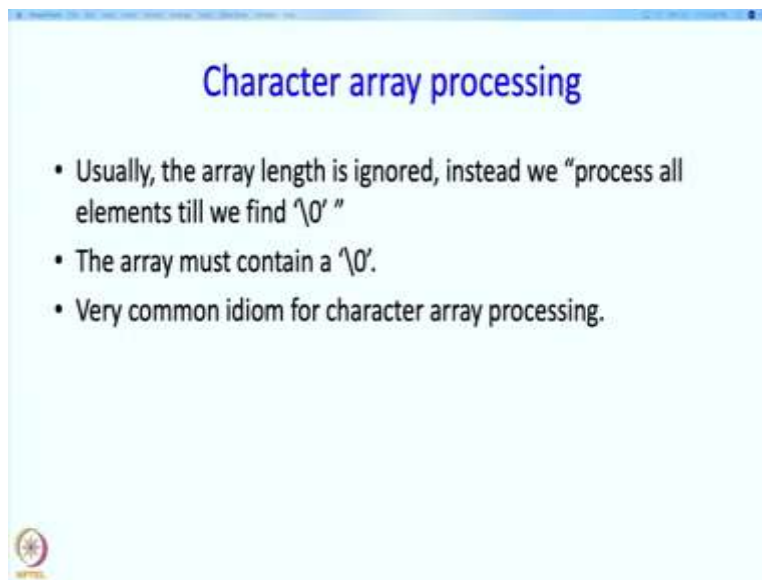
Printing char arrays is simple, so if you have a char array again the same char array, then say you somehow put data into it, we have seen one way of the putting data which is to read into it, but there could be other ways which we will see soon. And I should just point out, that the way you put data into the elements of any array is the same, so I can put data, I can make assignments to `char buffer[0]`, so to `buffer[0]`, `buffer[1]` just like I make assignments to the elements of an any array.

So suppose you have assigned some value into buffer and you have done it in the right way which is that the string that you have stored has a terminating null character, so if that happens

and then you can write `cout<<buffer`. So this would push the content of `buffer` onto the screen. And how much content? Well everything until the null character. So, for example, earlier be initialized `n1` to be "Ajanta" and `n2` to be "Ellora". So if you write `cout<<n1`, then "Ajanta" will be printed because as it is said over here everything until the null character is to be printed. And for this it is important that your buffer actually contain the null character, if it does not contain the null character then C++ will just go on printing whatever there is after that address, so again in a sense this is an unsafe, slightly unsafe mechanism, but it is not that unsafe because it is in your hands as a programmer to make sure that it contains the null character.

The length of the array is not important because only everything until the null character is going to be printed. And I should also point out that if `buffer` is an array of a different kind, then what would get printed, would be the address of that array because as you may remember in general the name of an array stands for the address, the starting address of that array. So here when I say `cout`, so this is something special intended for char arrays.

(Refer Slide Time: 18:08)



Now, how do you process character arrays? So we are going to do a few examples of this and the general principle however, is that the array length is ignored and instead we process all the elements till we find the null. And it is expected that the array will contain a null if you are going to do this for processing. And this is the most common idiom that we do not really keep track of the array length, but we process everything until the null character.

(Refer Slide Time: 18:46)

A simple program

- Write a program to read in a name and count the number of letters in the name.
- For simplicity we use the unsafe method of reading.
- Standard linear search.
 - Terminate if find '\0'.

```
int main(){
    char name[80];
    cout <<"Name: ";
    cin >> name;
    cout <<"You typed:"<<name<<endl;

    int L=0;
    while(name[L] !='\0')
        L++;

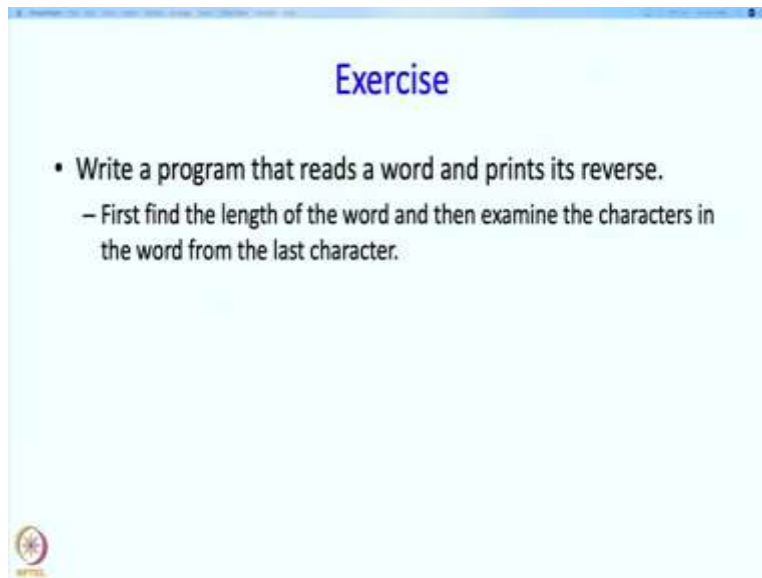
    cout <<"Your name has "
        <<L<<" characters"
        <<endl;
}
```

So, let us write a very simple program. So this program is going to read in a name from the keyboard and we are going to count the number of letters in the name. So here is the program, so what does it do? Well first it defines this array name of length 80 to store whatever you are going to read, then it prints out a message saying “please type your name”, well for short it is just saying name.

And then I could have used the safe variant but just for simplicity and compactness I am using the unsafe variant and this variant will read the first word. So whatever you type, delimited by a white space will go into name. And then the next, the next line is just going to print that out, so it is going to first print the message you typed whatever and then whatever you type the first word from it will get printed, the first word will be taken and only the first word will be taken into name and only that will be printed.

After that we are going to search through the name array, this search is sort of similar to what you have in other arrays, but it is different in the sense that you are not going to go, you are going to start at 0, but you are not going to go till the end. You are going to go only until you find a null and this is the body of the while and in the body you are only going to increment L. So as a result after this entire code executes, L will be the index of the null character or since our counting starts at 0, L will be precisely the length of the name. So yeah, so that is what got printed over here in the last, in the last bit.

(Refer Slide Time: 21:08)

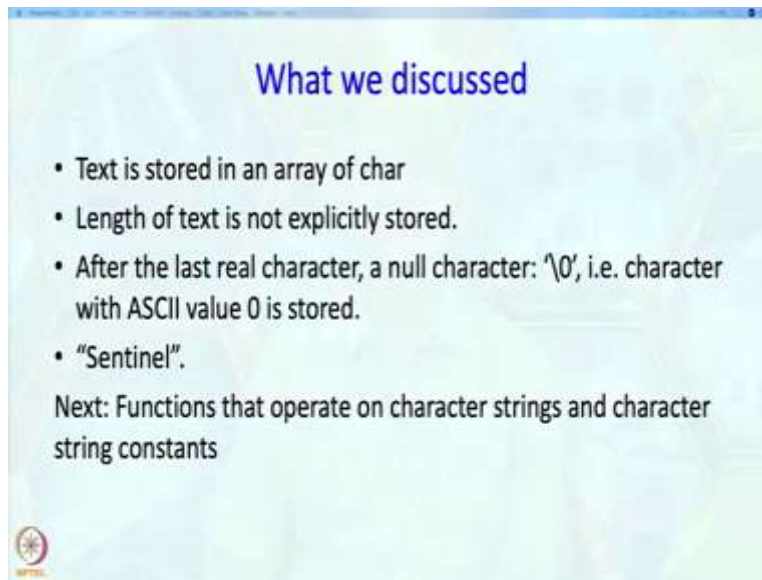


Exercise

- Write a program that reads a word and prints its reverse.
 - First find the length of the word and then examine the characters in the word from the last character.

So here is a quick exercise for you, write a program that reads a word and prints its reverse. So just to clarify what you are expected to do, you first find the length of the word using what we just saw, and then examine the characters in the word from the last character coming back. So again, now this is not, the second part is sort of standard array stuff, it does not use this convention about the null character or anything like that. But anyway, so the first part does and so do write this program.

(Refer Slide Time: 21:45)



So, what have we discussed so far? So he said that text stored in an array of char, the length of this text is not explicitly stored and this is, this is not a, this is not a rule as such, it is just a convention but it is a very-very stringently followed convention. After the last real character, a null character or the ASCII character with value 0 is stored. Again this is also a part of the convention.

And sometimes it is this last null character is called the "sentinel", "sentinel" means guard so it is sort of guarding the actual text that you have stored and it is standing at the end and guarding it is sort of. So next, we are going to look at functions that operate on character strings and also something called character string constants, so we will take a break.