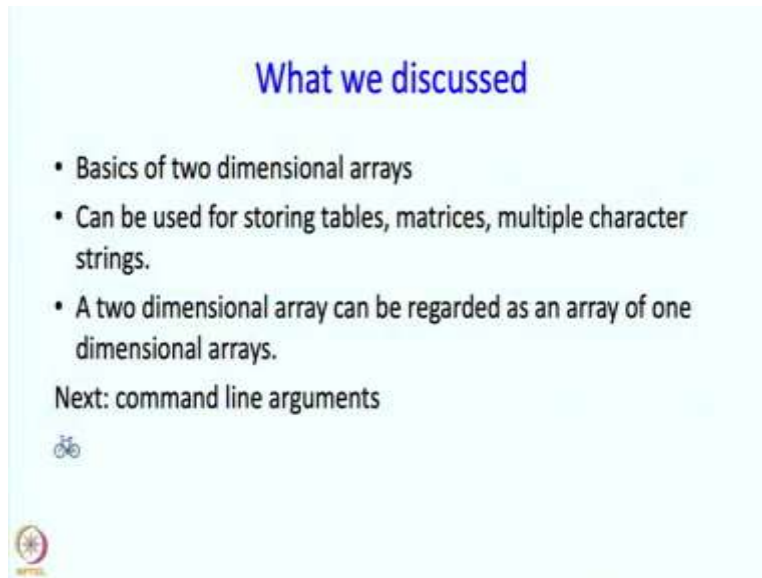


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of computer science and Engineering,
Indian Information Technology Bombay
Lecture 17 Part – 4
More on Arrays
How to accept command line arguments


(Refer Slide Time: 0:19)




What we discussed

- Basics of two dimensional arrays
- Can be used for storing tables, matrices, multiple character strings.
- A two dimensional array can be regarded as an array of one dimensional arrays.

Next: command line arguments






Welcome back in the last segment we discussed two dimensional arrays. Now, we are going to talk about command line arguments. This also uses arrays, and arrays, but arrays are not really two dimensional. So, what are command line arguments?

(Refer Slide Time: 0:37)

Command line arguments

- So far, we have executed our programs from the command line by typing `./a.out` at the shell prompt.
- It is possible to supply arguments also from the command line.
For example, we can type
`./a.out Math Biology`
- The program can get access to the arguments we type. **Next.**
- Aside: `a.out` is just the default name given by the compiler, we can ask it to give a different name or rename the file if we wish.



So far, we have executed our programs from the command line by typing `./a.out` at the shell prompt, or of course, we have executed them from by pressing the run button, but you can specify arguments from the command line, and even if you are using the IDE, you can get into a shell prompt and then there you can run the run the program and at that time you can give, you can arguments when you run the programs.

So, for example, you can say you can type something like `./a.out Math` followed by `Biology`. So, two words you are typing on the command line in addition to the name of the program that you want to execute. So, the program can get access to what you type and how to do that is the topic of this segment. By the way `a.out` is just the default name given to our program after compilation by the compiler.

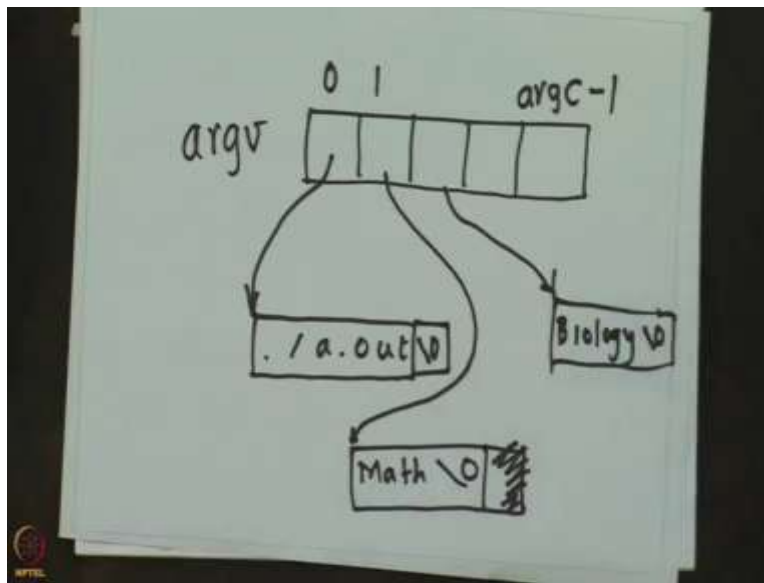
We can ask the program to give it a different name if you wish or we can rename the file ourself explicitly, but this is just to keep things simple that we are assuming that we are just using the name that the compiler gives to the executable program that it creates on compiling our programs.

(Refer Slide Time: 2:11)

A program that prints the command line

```
int main(int argc, char *argv[]){
    for (int i=0; i<argc; i++)
        cout << argv[i] << endl;
}
```

- We call main with a more elaborate signature as shown.
- argc: gives the number of "words" typed on the command line
- Interpretation of char *argv[]:
 - char x[] says x is an array of char
 - char* argv[] says argv is an array of char*.
- argv has length argc.
- Each element of argv is a pointer to each word typed on the command line.
- How this executes: next



So, I am going to show you a program that is the simplest possible thing. So, it does not do anything with whatever is typed, it just prints and shows you what is typed. The main function earlier had a signature `int main()`. Now, here to get the command line arguments, you are going to have a different signature.

So essentially your computer when it executes `a.out`, is going to execute `a.out` or it is going to call the main function but it is going to tell the main function that, "Oh! Here are these additional, here are these things that the user typed when the user started off the program." So, therefore,

there are these additional things. So, what are these additional things? Well before getting into that let me just show you the program and then we will explain it.

So this is the main function and that is the only function over here but its signature is different and now we will tell you how these arguments are to be interpreted. So argc is going to give the number of words typed on the command line by the user. So, as you can, as I said when your computer starts off the main program, it is asking the main function to execute and at that time, it is calling the main function and it supplies these two arguments, the first argument says, "How many words are there on the command line?"

So this is just function overloading if you will, so you remember that I can have the same name with different signatures and this is just a different signature. So, both names are both, both ways of calling main are acceptable and if you want the program to get access to the command line arguments, then this is what supposed to be used. So, I told you what argc is, and then, let me tell you what this complicated looking thing is. So, to understand that, let us just go over what it could possibly mean or some, let us do it step by step.

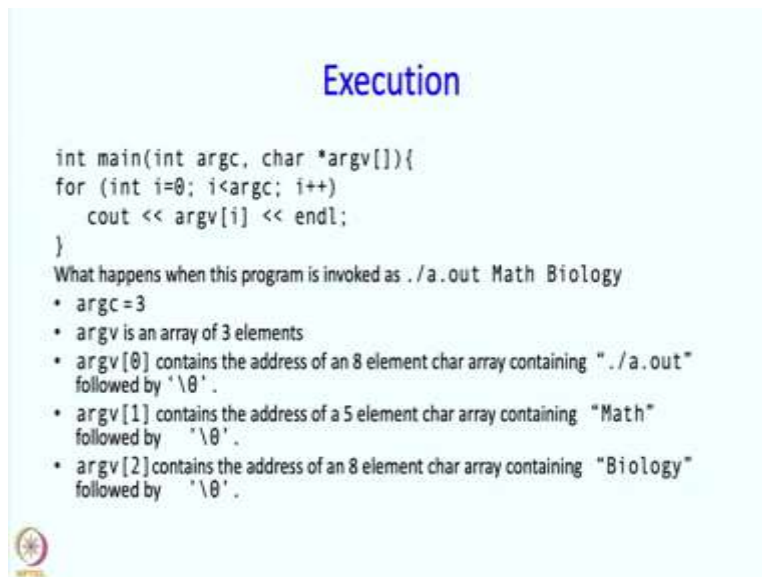
So, suppose I typed this as the declaration. What does it say? It says that x is an array of char. Now, what would this say? So, this would say argv is an array of char *. In other words, each element of argv is a char star or each element of argv is a pointer to a char array. And argv has length argc. So, this argc that is given is has length, is the length of this array, so how many elements are there, are argc elements in this array. So, argv is the name of an array it has argc elements and each element is a char star. So, it points to another char array.

So, I think a picture is necessary, so let me draw a picture, so this is array argv, so it looks something like this and it has elements 0, 1 all the way till argc minus 1, because there are total of argc elements. And what is each of these elements? So, this is of type char star, so it means it is a pointer and it points to the 0th element of an array and this is the first word typed by the user on the command line. So typically this is going to be ./a.out, but of course, it is going to have a null at the end as well. So, this is, so this 0th element is going to type, is going to point to this. The first element is going to point to an array, which stores the second word typed by the user with a null terminator, so let us say the second word typed was Math and then there was a null terminator, so let me get rid of this.

So, this is the second thing, and then maybe there was a Biology also, so this points to another array, so Biology followed by the null character. So, this is this complicated looking structure that is given to you. So, what is this program doing with this structure? So, it is printing out argv[i], so let us see what argv[i] is, so this is the array argv, argv[i] is this, but what is this, this is char *, so suppose I write cout, after which I give char * value, what happens well that simply prints out this value, prints out whatever it points to.

So, this first thing would just point, print out a.out, and the rule to be used over here is that whatever gets printed out will be printed out only until the null character. So, cout<< argv[0] will end up printing ./a.out. Then in the next iteration cout<<argv[1] will end up printing Math and after that Biology will get printed. So in this case argc, if I had just typed these three things, argc would have been 3.

(Refer Slide Time: 8:55)



Execution

```
int main(int argc, char *argv[]){
    for (int i=0; i<argc; i++)
        cout << argv[i] << endl;
}
```

What happens when this program is invoked as ./a.out Math Biology

- argc=3
- argv is an array of 3 elements
- argv[0] contains the address of an 8 element char array containing "./a.out" followed by '\0'.
- argv[1] contains the address of a 5 element char array containing "Math" followed by '\0'.
- argv[2] contains the address of an 8 element char array containing "Biology" followed by '\0'.

So, I have basically told you how this executes, but let us just do it again very clearly over here. So, argv is an array of 3 elements, argv 0 contains address of an 8 element array containing ./a.out followed by null, so why 8 elements “./a.out” and null. So, notice that exactly C++ will allocate exactly as much space as we want. So argv[1] contains Math followed by null and argv[2] contains Biology followed by null. So, basically in in a when this things are executed dot slash a dot out Math and Biology will get printed.

(Refer Slide Time: 9:44)

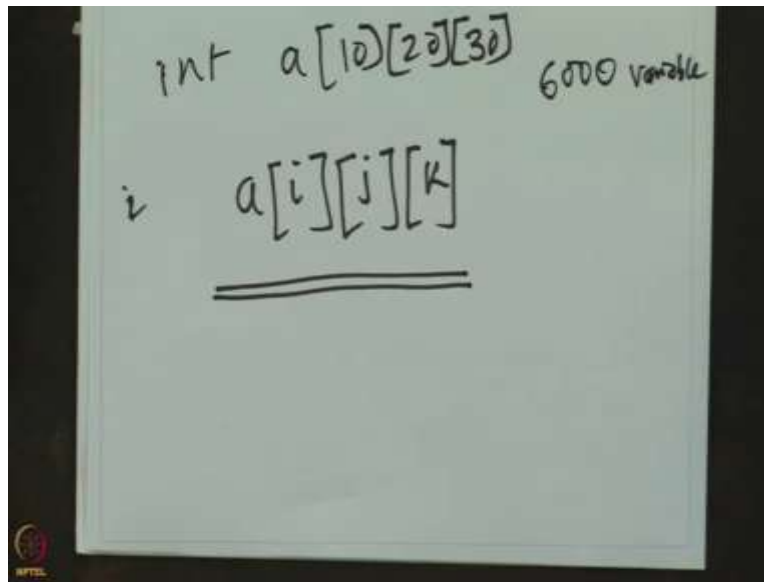
Concluding Remarks

- We studied a way of representing character strings.
 - Later we will study the `string` class which is nicer and safer.
 - However you must know `char` arrays because basic ideas in `string` are similar.
 - Also because `char` arrays are used a lot in C language code, which you may encounter.
- We studied two dimensional arrays
 - More dimensions: add indices: `int a[10][20][30];`



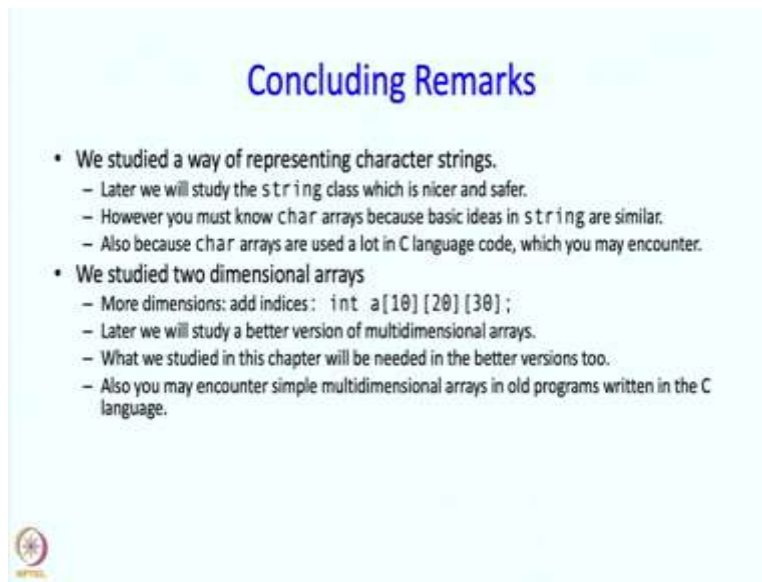
So, what are we studied in this entire lecture sequence. Well we studied array of representing characters string first, later on we studied the string class which is nicer and safer. But as I said many of the ideas that are presented over here will be applicable there as well. And these ideas in any case are used alot in the C language and you may encounter programs which are specially, which only use the c language facilities, and therefore, this is something that is useful for you to know. Then we studied two dimensional arrays and basically you can extend that idea, so if I write `int a` and give 3 numbers in 3 square brackets than that gets me a 3 dimensional array.

(Refer Slide Time: 10:43)




So, what is a 3 dimensional array? So, it is a collection of elements or collection of variables, say $a[i][j][k]$, so these, you can think of these as the different names and for this, for this declarations $a[10][20][30]$ i will range from 0 to 19, j will range from, sorry, i will range from 0 to 9, j will range from 0 to 19, and k will range from 0 to 29. So, in all if you write int, sorry if you write int, a 10 20 30 you will get, so 600 times 10, 6000 variables. And just as we thought of a two dimensional arrays as organised in rows and columns, you can think of two dimensional array as organised as a three dimensional objects, so rows and columns but stacked on top of each other as well. So, those are also needed in many cases and they can be had over here, as I said earlier the C++ equivalents will be a little bit nicer and you will see them later on.

(Refer Slide Time: 12:10)



Concluding Remarks

- We studied a way of representing character strings.
 - Later we will study the `string` class which is nicer and safer.
 - However you must know `char` arrays because basic ideas in `string` are similar.
 - Also because `char` arrays are used a lot in C language code, which you may encounter.
- We studied two dimensional arrays
 - More dimensions: add indices: `int a[10][20][30];`
 - Later we will study a better version of multidimensional arrays.
 - What we studied in this chapter will be needed in the better versions too.
 - Also you may encounter simple multidimensional arrays in old programs written in the C language.



And yeah, so the ideas of indexing and things like that are, of course, needed for C++ versions, the nicer versions that we are going to see later. Then we saw command line arguments and these are often useful, so you may want to say that look this is the file that I want to use or rather than, gets some data after running the program, so by `cin` you can directly get it as you type the name of the file, as you type the name of the executable, as you start up the executable program.

So, this is very convenient and you should definitely write a few programs of using command line arguments and master this idea. Alright, so that concluded this lecture, as always the text has several exercises and I will urge you to solve those, Thank you.