Introduction to Problem Solving and Programming

Prof. Deepak Gupta

Department of Computer Science Engineering

Indian Institute of Technology, Kanpur

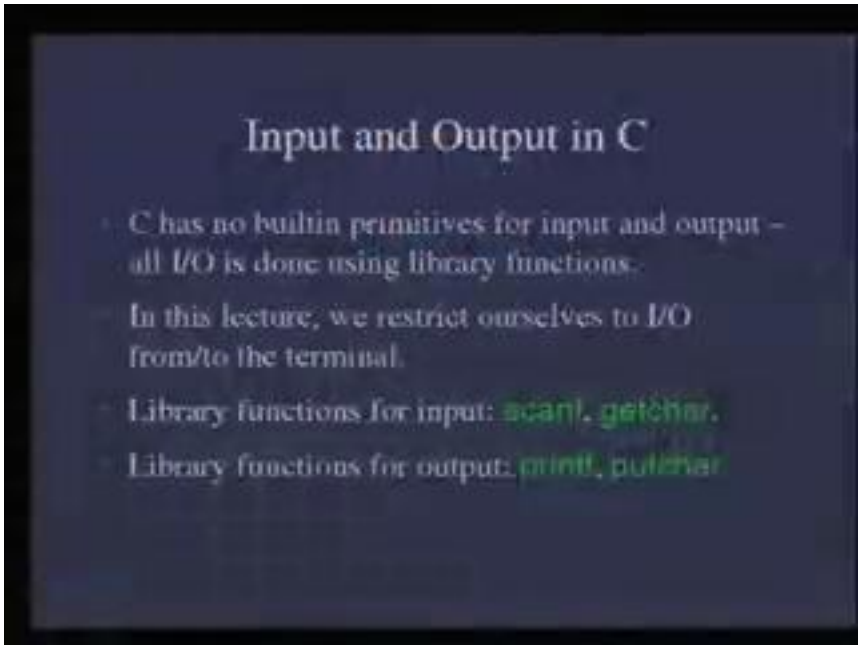
Lecture – 9

In the last lecture, we had talked about expressions and operators with side effect. In today's lecture, we will discuss a very important aspect of all programs that is input and output. We have already seen how basic input and output of data is done in the simple programs that we have written. So far today, we will go into some more detail of how input and output is done in C.

So, C has no built-in primitives or statement for input and output. For all input and output that is the IO, we have to use library functions. We have already seen several of the library functions in an action. For example, we have seen scanf for reading values from the terminal and the printf for printing some output on the terminal. In this lecture, we will restrict ourselves to input and output only from the terminal. Although in general program needs to read data from form and write data to the files as well.

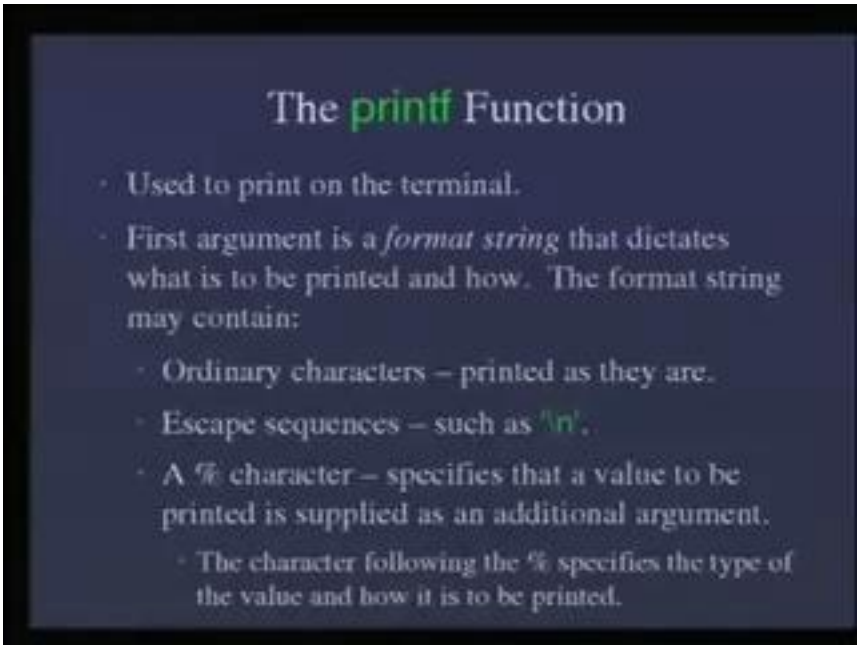
We will discuss that in later lectures. Today we will restrict ourselves to input and output from the terminal only, but as you will see using something called IO redirections as unique. We can actually use the same techniques to do simple input and output. Some files are there. So, the two library functions for input that we are going to discuss today are scanf and getchar.

(Refer Slide Time: 01:43)



Scanf we have discussed already. We have seen examples of usage of scanf already and the output library functions that we will discuss today are printf and putchar. Again printf we have seen some examples. So, let us start with the printf function. As we know this is used to print some data on the terminal. We have seen several examples of usage of this function.

(Refer Slide Time: 02:07)



The first argument is what is known as a format string which specify essentially what is going to be printed on the screen, and how it is going to be formatted. These format strings may contain ordinary characters like alphabet, numerals or else other special character which are printed as they are on the screen as they appear in the format string. Thus, we can have escape sequences such as backslash n for the new line character and so on. When we are going to print some value whose actual value is not part of the screen, but it is specified some expression which I additional to printf, we use what is known as format conversion specifier which start with the percent character in the format string. The percent character specifies the value to be printed is applied as an additional argument to the printf function and the character following the percent character in format string specifies what the type of the value is and how it is to be printed.

You have already seen examples of the percent d. When we percent d in the format string to printf argument which follows is assumed to be of type and it is printed like an integer. If you want to print an unsigned int, we should use percent u and similarly, ld for long percent, ul for unsigned long percent, hd for short percent, hu for unsigned short percent, f for double percent, lf for long double and so on. The percent c specifier says that the corresponding argument is of type integer, but it is not to be printed as an integer, but the character whose ASCII code is this integer that character should be printed. We will see examples of this and the percent have special meaning in the format

string. If you want to print percent character itself, we can use percent. Percent indicates that you want to print one percent character.

So, here is this simple example which demonstrates some of the specifier that we have seen. So, in this example we have several variables. The variable c is of type char and it is initialized to the character A. Remember that the variable pf, the type char is actually an integer and the value assigned to the character c in this initialization is the ASCII code of the character capital A, I is the variable of the type integer with the value 5, x is double with value 5.0, y is the float of value with 10.0 because that literal constant of type have by default type double and we have put an f of it to indicate that you want this value to be of type float. Now, here is the printf statement and this first argument here is the format string. Now, this has percent d, percent c, percent f, and since there are four format conversions satisfied percent c, percent d, percent f, there have to be four arguments.

Now, the first argument corresponds to the first argument falling the format string corresponds to the converting specifier percent d. So, therefore, it should be of type int. We have given the expression here as simply I which is the value of I will be printed as an integer, that is the value 5 will be printed and then the next specifier is percent c which means you want a character to be printed and the corresponding argument that is the second argument following the format string has to be of type int, and it should be the ASCII code of the character that you want to print. Now, c happens to be of type char actually, but what exactly is an int. So, therefore, we have to explicitly say that we want this c, this value to be converted to a value of type integer. So, that is why this int within the bracket indicate that we want the explicit type converted into beta is also called type cohesion. We have already seen that an expression sometimes automatic type conversion happens if the operand of the operator happens to be of different type and so on.

Sometimes you also need to explicitly perform type conversion and in the cases called type (( )). The third argument is x which corresponds to the percent f here and percent f requires the corresponding argument to be a size double with x is (()), but the next one is again percent f which means that we have to supply a double. What you want to print is y which is of type float. So, therefore, we have again explicitly converted this value to double by doing it type quotient here. So, these two are examples of type quotient. So, what will be the output look like evaluate percent d in the format string will get replaced

by the actual value of these integer I which is 5 and then after that the character corresponding to the ASCII code, character whose ASCII code is applied as the next argument will print that happens to be A. So, A is printed note that there is no blank between 5 and A because there is no blank between percent d and percent c in the format string and after that the two double values will be printed which is 5.0 and 10.0

So, it is to understand at this point in time what exactly is going on, where we are using printf. Now, you come of that when we do input and output? That input and output can always only be in terms of character. Only characters can be printed on the terminal, right. So, that means that to print anything on the terminal, the terminal must send a sequence of character that is ASCII codes of the character. Corresponding characters are what are going to be printed on that terminal. Therefore, if you want to print the integer 50, then we have to send two bytes to the terminal that is the ASCII characters for the character 5 followed by the ASCII code of the character 0 and the ASCII code of the character 5 found to be 53 and the ASCII code of the character 0 happens to be 48. Therefore, to print the integer 50, actually two bytes which are 53 followed by 48 have to be sent to the terminal, but in general the integer I is represented as the binary equivalent of the decimal number 50, right.

So, this conversion has to happen and this conversion is what is being done by the useful library function printf, and the way perform conversion is guided by the conversion specifers. For example, percent d says that an integer is specified and it has to be converted to a sequence of byte corresponding to the digits in that integer. Let us now come to this simpler output function that putchar function. So, this function can be used instead of print f when we want to print just one single character on the terminal, and we would usually prefer to use putchar instead of printf because putchar is usually fastest. So, what you want to print is just a single character. You can use putchar instead of printf. The argument of print to putchar is an integer of type int which must be the ASCII code of the character that you want to print. So, for example, if you want to print x on the terminal, then we can use this kind of a statement putchar within bracket int followed by within quotes x.

Now, this within quotes x denotes the ASCII code of the character x, but the type is char since putchar expects an integer as an argument. We have explicitly converted it to an integer before supplying it to putchar. So, this statement will print x on the terminal. Let

us now look at the scanf function for reading input from the terminal. Scanf is the analogue of the printf for input. Again the input is in transfer characters only. Now, what we type on the key board when we supplying input to the user or character. For example, when I type 5 and 0, what actually being sensed to the program or it is a sequence of 2 byte which is the ASCII one of the character 5 followed by the ASCII code of the character 0. If you want this value 52 to be stored in an integer variable, we must convert this sequence of ASCII code into the binary equivalent of the decimal number 50. So, this reverse conversion is what is being done by the scanf function. The uses of the scanf we have seen earlier in examples. It is fairly similar to printf.

The first argument again is a format string which specifies how the input is to be converted whether we expect an integer or real number or whatever it is, and the remaining argument specifies there that is in which variable this input data has been stored. So, the conversion specifier for scanf is fairly similar to those for printf, where some minor differences. For example, percent d is an integer. Now, what is to remember that ASCII code reads character and expect characters which constitute integer. For example, an integer really means exactly should have optionally plus or minus sign followed by a sequence of digits and scanf for it has t reads character skip initial, white space character. White space characters are like the blank character, a new line character, a tab character and it starts converting once it finds the character which could be part of an integer. So, a plus or a minus sign are numeral and then it keeps reading till it reaches the end of the number and then converts the sequence of the character read. So far into a single integer and sign state or store state in the variable that is applied as the corresponding argument for the conversion specifier.

In this case, the argument must be a pointer to a variable of type int. We have not even discussed with what pointers we will discuss that in the subsequent lectures later on. Put it pointed to a variable usually is obtained by prefixing the ampersand sign before the variable name. So, in usage of scanf for we have already seen that the arguments after the format string are ampersand followed by some variable name. So, this ampersand is required because the argument must be a pointer to a variable of that appropriate type. So, in this case I must be an integer and we have to use ampersand I as the argument for scanf corresponding to percent d. There are minor differences from the printf specifier. For example, percent f is for float instead of double and percent lf is for double.

Similarly, for percent c corresponding argument must be a variable must be pointed to variable of type char, not int and in this case, what happens is that scanf read the character and assign its value with this ASCII code of the character to the variable which has been specified.

Note that no characters are skipped in this case as supposed to. When reading an integer initially white space characters are skipped, but when you are reading a character, no characters are escaped because even white space characters are our character. So, if the first character type was a blank space, say for example the variable will be assigned as the code of the blank character; the getchar function is similar to the putchar function. It can be used when we want to just read a single character from the input. There are no argument return value which means the value of the getchar expression is of type int and is simple the ASCII code of the next character in the input.

So, for example, if you want to read just one character from the input, we declare a variable x of type int and use an assignment statement x assigned getchar. This will read the next character from the input, assign its ASCII code to x. Note that when a character has been read from the input, the next time we call getchar again it will return the next character in the input that is the character which is read here has been in some sense consume the getchar. So, when we call getchar again are indeed scanf or any other function for reading from the input, then this character since it has already being consumed is no longer available. So, for example, if you use the type x y z and we call get getchar three times, then we successively get the code for the characters x, y and z, right.

One important point to note is that the variable which is used to store the return value of the getchar must be of type int always we will come to the reason why this is so in a little while, but never use the variable of type char or something else to store the result of the getchar. Let us know what we have learned today. To write a couple of simple program in the first example is particularly simple. All we have to do is to write a program which copies the input to output, that is the input and the output is exactly the same. So, the way we are going to write this program is clearly simple. We are going to use getchar to read character again and again. We will put this in a while loop till the end of the input is reached and for each character read, we will use putchar to output it. Now, how do we know that the end of input has been reached? We know that when getchar returns a

special value EOF. EOF stands for end of file. So, this indicates the end of the input on the terminal as well and on the terminal.

How does the user indicate that he has finished giving the input, but I think control d on line by itself. Now the reason that I would said earlier the return value of the getchar before the variable of the type integer is that the value EOF is the minus 1. It is the negative value. So, that is why it must be stored in an integer. So, let us now see this program and try to execute it. There is the program, very simple program, but there are some instructive points about these programs. So, we declare a variable of type int. All the variable characters that we read from the input one by one and this loop says read a character. This is not end of the file. Use putchar to print the same character. Note that we have used an expression with side effect in this particular loop condition. So, you look at this particular expression c assigned, get char. What it does is that it calls getchar to get the ASCII code of the next character that is assigned to the variable c and the value of the expression itself is the value that has just now assigned to the variable c EOF. That value is going to be compared now against EOF.

If that value is not EOF that is the end of file has not been reached, then we use putchar to print the character c. So, this is the very common idiom and this often simplifies the loop condition to use assignment to a variable. Then, compare the value that has just been seen with something else. So, let us now run this program. I have already compiled this. So, let us give some input. You can see that we have typed several characters, but no output has appeared so far. Output will appear as soon as we press the return character. As soon as we press the return character, you see that all the character set we typed so far, they appear on the output exactly the same as they were in the input.

Let us give some more import and when we want to immigrate the end of file, we can simply press control d the line by itself that will cause the getchar to return the EOF value which will terminate the loop and the program. Program will exit now. We have written this program to read input from the terminal, but Unix has the facility called input output redirection. Using this we can make the program believe that this input is actually coming from the file. So, for example, if we use this notion to execute the program, we need some input file. So, let us say copy dot c itself and then the contents of the file copy dot c. This is program we just wrote have been printed on the screen. So, what is happening in this case is that the import is coming from this file. The program

does not need to change the file in order to go that the operating system ensures that when the program reads from what it believes as the terminal, the input actually start from the file and of course, the file does not contain, does not need to contain the control d character because the file has finite length and when it finishes the end of the file indicated to the program by the operating system automatically.

So, actually any program we can write that is expected to take input on the terminal and we can supply the input from the file instead and similarly, we can actually redirect the output of the program to the file. So, the terminal for example we do something like this. So, in this case what we saying is this is greater than sign says that the output of the program is being redirected from the terminal to a file called xxx that is whatever the program prints, this output will not appear on the terminal, but will go in the files instead. So, let us give some input. You see that it does not appear from the terminal at all and if you see now we have already called a file called xxx. We can see what the contents of the file are. This is exactly what we typed as the input and instead we can combine both input and output reader. Let me first delete the file xxx. We can actually combine on the input and output redirection, both, so that the input comes from the file and the output goes to different file. If we do that we can actually use the program. We just hope in a very useful manner to copy files. So, for example we want to make a copy of the file copy dot c, we can do this like this. So, what we are telling the operating system is that when the program copy is done, imports should actually come from the copy dot c and if the output must go to the file copy 2 dot c, of course you must not use the same file name for both the input and output redirection. It will just corrupt both the files.

So, if you just do this, you see that there is nothing that appears from the terminal and we did not have any input either, but the file copy 2 dot c now contains the copy of the file copy dot c. So, you can use as we have seen copy of the program, very simple program that we wrote to see the contents of the file by doing input redirection. We can use it to come into file output redirection and we can use it to copy, make a copy of the file into another file by doing both input and output redirection. So, as we saw one of the observation that we made by executing this program is said from input from the terminal. The program receives the character only after the user ends the line because as soon as we type a character, it is again on the output because if the program receives all the

character that type only after we ended the line by pressing the enter key and similarly, although it is not really appearing from the execution, the output also appears only after a new line which is printing by the program. So, this is called buffering by the operating system.

We have seen that programs that read input from the terminal and output to the terminal can use the files easily using I redirections facility of the UNIX. In later lectures, we will be discussing in more detail all the program can directly work with file because in many cases the simple IO redirection is not sufficient. The program may want to open and read several files and write different output to different files and so on which is simple IO redirection facility. It is not sufficient, but we will come to that in the later lecture. Here is the example 2. We want to write a program with counts, the number of characters, the number of words, and the number of lines in the input.

How do we do that? The idea is very simple and similar to what we have just seen. Again read the input character by character. How do we count the character? That is quiet easy. For each character that we read, we increment the character count by one which of course initialize to 0. For the new line character we see that is the back slash and the within the quote. How will we use it in our program? We increment the line count by 1. We are maintaining a word count and line count in a predefined variable. All these initialize to 0 and for each character seen we increment the character count by 1. For each new line character that we see, we increment the line count by 1, but what about the word. Well, we assume that words are separated by white space character. Between two words, there must be some white space character. Remember white space characters are blank character, tab character and the new line character. So, how do we count the word?

Well, what we are going to do in this program is that we will increment the word count when a new word starts. When does a new word start? It is when we see a non-white space character followed by a white space character. That means that a new word has started where white space character is of course blank or a new line or a tab or we will also start if the first character in non-white space character. For example, if we give the input star to characterize, then the word has started. So, we must count that as one which means that in addition to storing the current character that we are processing right now, we must whether or not the previous character was a white space character because s what we have just seen is a non-white space character. Then, we must increment the

count only if the previous character was a white space character and not otherwise because if we have the blank followed by x, followed by y, then we see x you should increment the word count, but when you see the y, you must not increment the word count because it is not part of the new word.

So, how do we keep track over whether the previous character that we saw was a white space character or not? That is very easy. We can use the Boolean variable to keep track of that. So, we will use this Boolean slash to keep track of whether the previous character was white space character or not and we will have to you know initialize this properly. Initially it will be true because if the first character itself is the non-white space character, we must assume that it is starting a new word for count one. So, we initialize it to true and subsequently it will be true or false depending on whether the previous character that we saw was a white space character or not, ok.

So, now let us look at this program. Now, here is the program. This is slightly more complicated program. So, note that I have put a lot of comment which describes what the program does. We are going to use of course the integer values 1 and 0 as true or false. So, we have as for our usual practice, put define for true and false and here the main program starts. There are three variables that is lines, chars and counts. These are the counts of the lines and the characters and the words that we have seen so far, they are all initialized to 0. Now, the in space is the Boolean variable that we are talking about and this is meant by true initially, and when the previous character that was seen was a white space character. Remember that a Boolean variable can be implemented by using a integer type. We have chosen to use the type char. We could have used the int or short instead that would be perfectly find to. So, in spaces will be true if the last character seen was the white space character and it will be true initially. Otherwise, it will be called and this variable c of type int of course is a variable that will hold a ASCII code of the character c.

The loop condition looks very similar to what we had for the copy program. We read character and compared it with end of file. If we get the end of file as a result of calling getchar, then the loop terminates otherwise c is assigned the character that we have just seen and to execute the loop body. So, now, the first thing we have to do is to increment the character count. We have seen one more character. So, immediately increment the number of characters that we have seen so far and is what we have seen a new line

character, then we increment the number of lines seen. So far that part is very easy. The name is the figure out whether this is the new word is starting or not, and appropriate update the in space boolean variable. So, covering that we need to first figure out whether the character we have just seen is a white space character or not.

So, if c is equal to the blank character or the tab character or the new line character, then when we just see a white space character, in this case we need to set in space to true note that it might have already been true because the previous character we saw might also have been a white space character, but that does not really matter. Whenever we see a white space character must be true in the next iteration. So, that is why we are making it true otherwise, if we have seen a character which is not a white space character. Now, we have to decide whether this is the starting of the new word or whether it is the continuation of the new word and the distinction between two. A new word is starting. The previous character was the white space character. In the other words, the in space is true earlier. Then, this is the starting f, the new character of a new word. Note that in this case we have not chase in space to true, we have to change the in space to true in this situation only if this character was a white space character and of course, a character cannot be both the white space character and a non-white space character.

So, if in space is true, that means that work we have seen the first character in the input or the previous character was a white space character and now of course, in space should become false because certainly this character is not a white space character. So, in the next iteration of the loop, in space must be false, so that we encounter another non-white space. In the next iteration, we do not found that as a beginning of another word and since this has being a new word; you must increment the count of the number of words. Note that we have used the increment operators plus plus to increment the values of lines, chars and words by one. This we have already seen what the operator does in the previous lecture. It increment the variable and this is the post increment. So, what it returns is the old value of the variable, but in many cases we are using this as the statement. So, therefore, we are discarding the value of the expression itself and depending upon the expression, we could have used the pre-increment or post-increment operator of the results could have exactly the same because side effect of both is to increment the variable by one and the return value is the only thing which is different.

Return value or the value of the expression is ignored in the particular case, in any case. That is all we see in this program. When the loop ends, we have the count of the character towards, so we just printf. So, we printf using this format, string in this format, string note that the characters chars, colon, blank, we will get as they are the characters percent d will be actually replaced by the value of the variable character here and again, these characters get printed as they are this percent d is replaced by the value of the variable words. These characters get printed as they are and this percent d gets replaced by the value of the variable line and this new line character is printed as it is which means that because they compound to the next line and the program ends.

Let us now try out this program. I have already compiled it again. So, let us just execute and let us give some input. We already know how to terminate the input, so that no output will appear till actually end the output because we are not outputting every character in the input. So, let us terminate the input. Now, what they say? They say 100 and 400 characters have been typed. Note that this will include the blank spaces and the new line characters we typed. Number of words is 17 can count that easily enough to ensure that is correct. On the first line, we have the 5 words. On the second one, there is 11 and third one makes it 17 and there are 3 lines and again we can use IO redirection to count the number of words from some other files in some files. So, as before let us count the number of words in the file, word count dor c itself which continues the program which we just wrote and the program tells us that these files contain 985 characters, 178 words and 41 lines, and we can actually verify the output by using the standard word counting such facility in UNIX.

So, there is a program called WC does pretty much the same as our simple word count program. So, we use the standard WC utility to count the number of words, line, characters etcetera. In the file word count dot c, we find that we get the same output. The WC program prints the number of lines and words and then character. So, we see that it said 985 characters, 178 words and 41 lines which is precisely what we also do for the same file. So, at the end of the lecture, here is the simple exercise for you. Besides the copy program yourself, but this time we want make small changes to it that the case of all alphabets characters are changed that is all lower case characters in the input or converted to the corresponding upper case letter, and the upper case letters in the input

should be converted to the corresponding lower case letter, and the other characters are like special characters for blank for new lines etcetera should remain as they are.

Now, here are couple of means are to help you with writing in this program. We need to do two things in this program really. When we read a character, we need to check whether or not it is lower case letter. If it is a lower case letter, then we need to output the corresponding uppercase character and similarly, if the character happens to be an uppercase letter, we need to output the corresponding lower case letter and if the character is neither the lower case nor the upper case, then we output the character as it is. So, how do we check whether for example the character c is the lower case letter of the alphabet? We need to compare the value of all lower case alphabets letter from A to Z because remember that s in the ASCII codes of the various characters. The characters, the codes for the characters A to Z are one after the other that is b, the ASCII code that is b is one more than a, then the ASCII code of the character c is one more than the ASCII code of c of character b and so on so forth.

So, if given the property of the ASCII codes, it is enough to check that if c is greater than equal to the character a and it is less than or equal to the character z, then it must be some lower case letter. Similarly, we can check whether the character is an upper case letter or not. So, having check whether the character is let us say lower case character, we need to find the corresponding upper case letter. Now, how do you compute that? Now, again we use the same properties that the ASCII code for small a to small z are contagious. Similarly, ASCII codes for capital A to capital Z are contagious now which means that if the lower case letter, then if you subtract the ASCII code of a format, then what we will get will be the place of the letter c of the character c in the alphabet starting 0.

So, if c happens to be the character a itself, this will give a 0. C happens to be character b. This will give the value 1. If it happens to be character c, it will give a value 2 and so on. Remember that we can do this on arithmetic on character because the characters are nothing, but all integers only, the ASCII codes of these characters. So, once you know these places of the particular lower case character that we have seen in the alphabet. Now, to that place the value the ASCII code of the capital A will give us the corresponding upper case character. For example, suppose c happens to be the lower case character d, now d minus a will give us the value 3 because b minus a will give us 1, c minus a will give us 2, d minus a will give us 3 and if you add c to the ASCII code of the

character a, that will give us the ASCII code for the character d. So, this way we can find out what the ASCII code for the upper cases corresponding to a certain lower case letter is.

Similarly we want to find the lower case letter corresponding to an uppercase letter. Procedure is similar. We subtract the ASCII code of the capital A and add the ASCII code of small a. With this, finishes today's lecture. In the next lecture, we will start looking at the control statements, if statement and while statement and so on in more detail. Then, we introduce some more useful control statements.