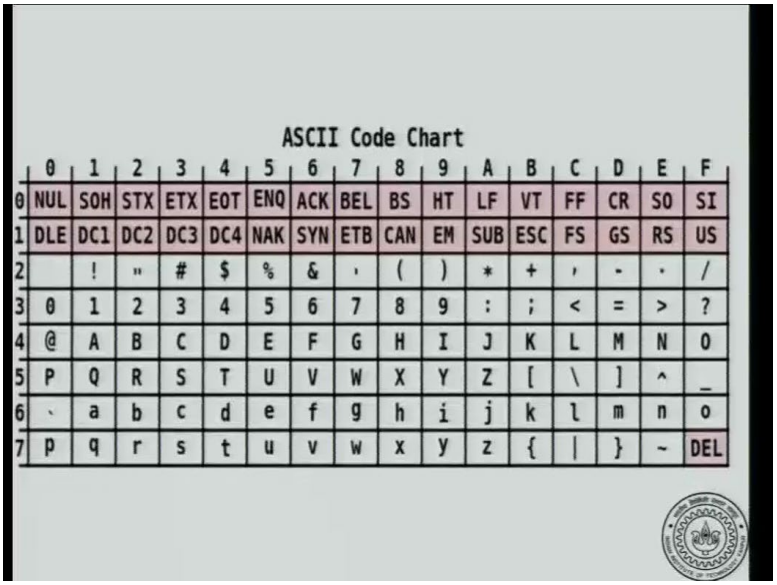
Introduction to Programming in C
Department of Computer Science and Engineering

In the previous session, we were talking about ASCII character set. And I said that, we do not need to remember the ASCII table. But, we need to remember some general properties of the ASCII table.

(Refer Slide Time: 00:15)



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

So, what are those general properties? First, we know that the initial 32 characters of the ASCII table are non printable characters. Then, the remaining or rather from ASCII value 32 to ASCII value 126 are printable values. Among them, you know that the integers, the digits are occurring together. Similarly, the capital letters occur consecutively, one after the other. And the small letters occur consecutively, one after the other.

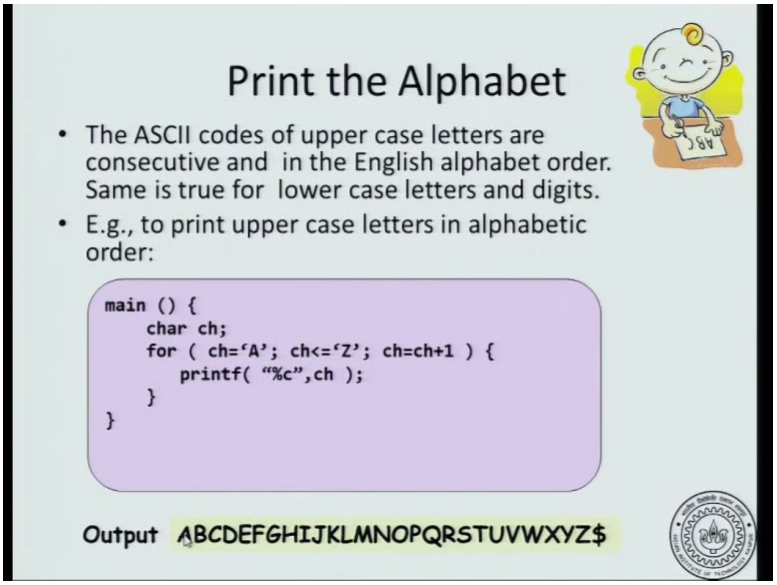
So, this is an abstract property of the ASCII code chart that, helps us in writing some useful code. We will see in a minute, what kind of use we can obtain using these general properties. Rather, than knowing the specific things like, the character value of A is hex value 41 or decimal value 65. This kind of information, we need not remember this.

For example, let us write a small program that prints… In our previous session, we had said that, we do not need to remember the exact ASCII code of certain characters. We just need to remember some abstract properties of the ASCII table. For example, abstract properties like all the digits occur together from 0 to 9. All the capital letters from A to Z

occur together in the table, in the alphabetical order.

Similarly, all the small letters occur together in consecutive locations in the ASCII table. Also, another property that you can observe is that, the small letters occur after all the capital letters. Let us see, how we can write some interesting code using these properties. And not by remembering the exact ASCII code of certain letters.

(Refer Slide Time: 02:28)

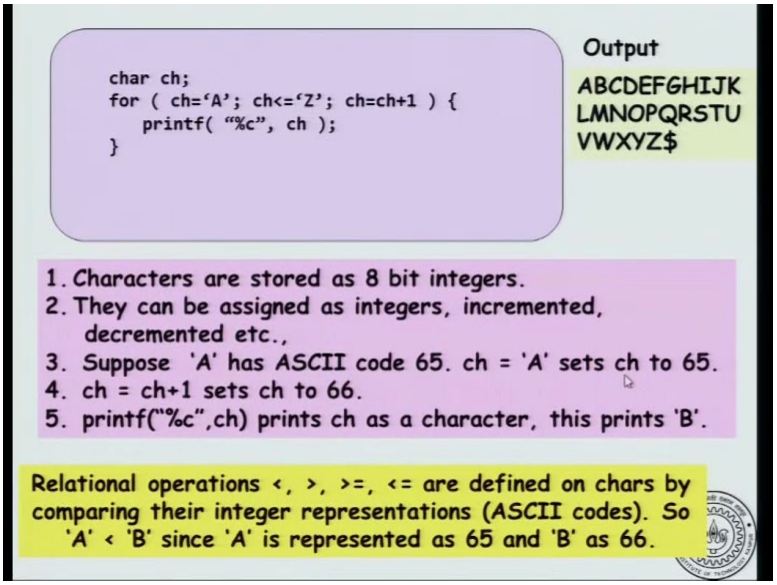


So, let us write a simple program, to print the alphabet. The ASCII codes of the upper case letters are consecutive and the ASCII codes of the lower case letters are consecutive. This is the property that, we will exploit in order to print the alphabet. So, for example, let us say that, we are going to print the letters of the alphabet in capital letters. So, for that we can use the following program using a for loop. So, what you have to do is, to initialize a particular character variable to capital letter A so, the ASCII character A. So, note that A within single codes stands for the character constant A.

If you look at the integer value, then it is the ASCII code for A. We are not particularly interested to know, what exactly the number is. Now, we can write the for loop in an interesting way. We can say that, start from capital A and then, print the characters until you hit capital Z. And the update statement is, after printing go to the next ASCII letter. So, what this is doing is, starting from A and then, it will go to A + 1, which is the ASCII code for B.

Then, it will go to B + 1, which is the ASCII code for C, so on up till Z. So, once you reach Z, it will print that character. It will update once more, where it is the ASCII character one more than, the ASCII character next to Z in the ASCII table. We do not really need to know, what it is. But, certainly it will be greater than the ASCII value of Z and at that point, we will exit the code. So, the output of it will be consecutively A to Z.

(Refer Slide Time: 04:38)



Let us look at, what is happening here in greater detail. All the characters are stored as 8 bit integers. Now, they can be assigned as integers, incremented, decremented, etcetera because, essentially they behave like integers. So, suppose A has ASCII code 65, but we are not concerned about that. Now, so ch equal to character constant A, sets c h equal to 65. Now, ch + 1 is the number 66, which corresponds to the ASCII code of B.

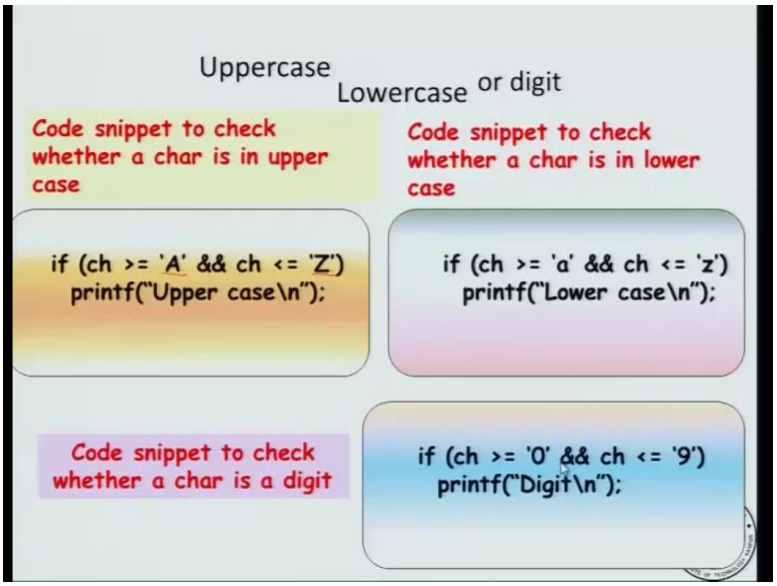
So, addition, subtraction all these can be performed on character values because, internally they are represented as 8 bit integers. Similarly, relational operations like less than, greater than, <=, >=, all of these also make sense. So, for example, if we use the relational expression capital letter A, ASCII constant A less than character constant B. Then, notice that A is the ASCII value 65 and B is the ASCII value 66.So, A less than B is correct.

Now, for realizing that A less than B is correct, we do not need to know that, A is 65 and B is 66. All we know is that, the abstractly in the ASCII table, the character code for A is less than the character the code for 6 because, B occurs after A. So, if it is 65 and 66 or it

is 0 and 1, the answer is still the same.

Now, let us write a few more interesting programs, where the spirit is that, we do not need to understand what the exact ASCII code of a letter is. But, just we want to remember the layout of the ASCII table.

(Refer Slide Time: 06:45)

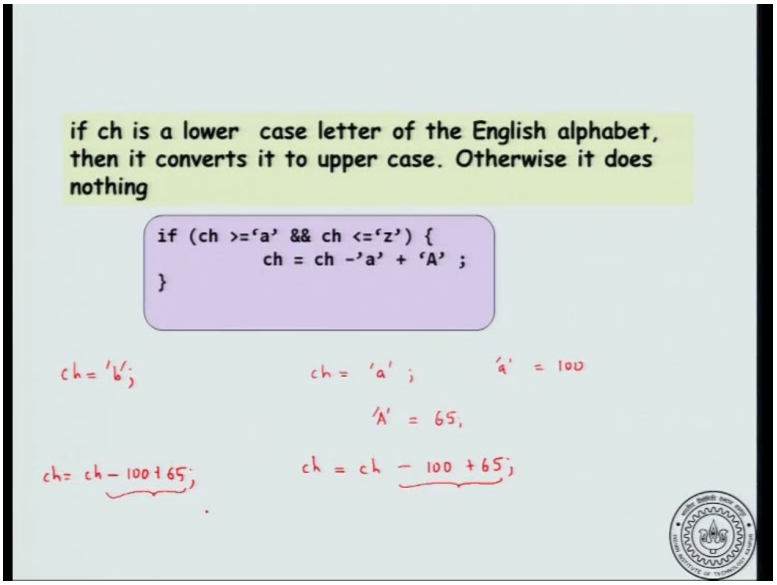


For example, suppose I want to write a conditional expression an if condition, which says that, if the given character is capital letter, then print that, it is in upper case. So, all I need to do is, if the character value is >= the character constant A and <= the character constant Z. Then, you print that, the given letter is in upper case. Again, please remember that we did not need to know that, this was 65 and this was, whatever it is 90.

It could as well have been 0 and 25. It would still have worked because, all we are need to remember in the ASCII table is that, A through Z occurs in consecutive locations in the standard alphabetical order. From that we can understand that, if I write this if expression, it will print up the message upper case, only if the given character ch is an upper case letter. Similarly, let us say that, if you want to check whether a character is in lower case. You can analogously write, character is >= ‘A’ , ‘a’. And it is <= little z, in single quotes. If that is true, then you print that, it is in a lower case. Now, if you want to check whether a given character is a digit, similarly you can say that, it is >= the character 0. And this is <= the character 9. Now, here is a suttle point which I hope, you notice.

The character 0 is the ASCII constant, ASCII character constant 0. So, it corresponds to some particular ASCII value. It is different from the number 0. So, this is something that... So, we are looking for the ASCII value corresponding to 0 it is >= that and <= the character value corresponding to that character 9. So, if that is true, then the given character is a digit.

(Refer Slide Time: 09:10)



Now, here is a snippet that, I would advise you to take a look at it. And tell me, what it actually does. So, take a moment yourself and try to figure it out. So, what it does is, the given character ch, if it is a lower case letter. Remember, this is the example that we just saw. This condition checks, whether the given character is a lower case letter, a small letter between a and z, little a and little z. If it is true, then what you do is, add capital A - a, to the character.

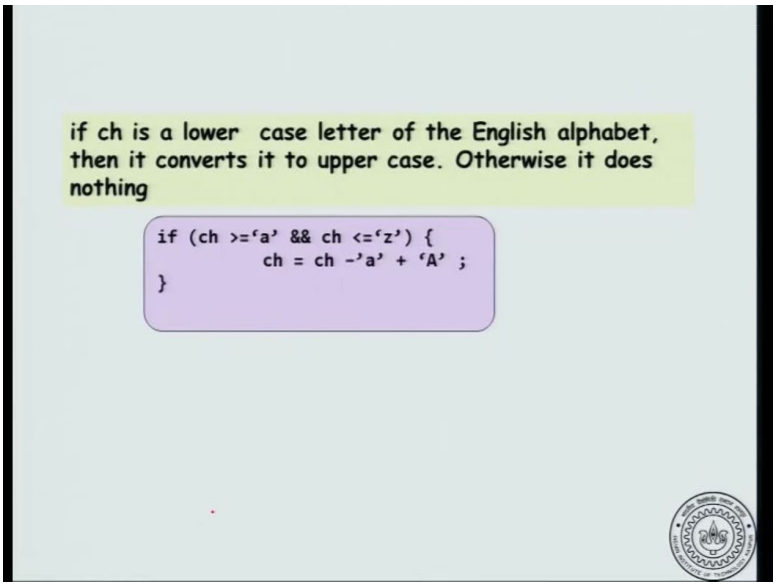
So, what does it accomplish? So, let us say that, we actually had c h equal to little a. Now, for the purpose of illustration let us say that little a, was ASCII value 100. I do not know, whether that is true. But, it is not important. That is, what I want to illustrate. Now, what does capital A represent. It represents some ASCII value let us say 65. So, if the given character was little a, what I would do is, I would say character = ch - 100 +65

Similarly, if ch was character constant b, I would still add ch = ch - 100 +65 So, it is adding a constant difference to the given character regardless of what, whether it was a or b. The additive constant that we are adding is still the same. And if you think about,

what is happening it is adding, exactly the difference between little a and capital A. Notice, that the difference between little b and capital B is the same as little a and capital A.

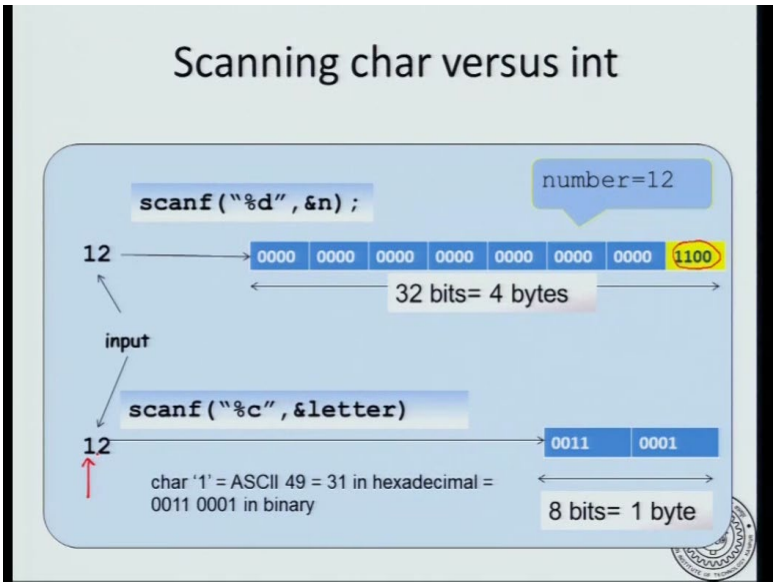
Why? Because, all the capital letters occur consecutively and all the small letters occur consecutively. So, suppose a minus z, little a minus capital A is… Let us say 35, then little b minus capital B will also be 35, because you advance one in each case. So, if you think for a minute, you will see that what this code does is… Take the ASCII code corresponding to the small letter. And add a constant difference. What is that difference? That difference is, what will take you to the capital letter, corresponding capital letter.

(Refer Slide Time: 12:30)



So, in short what this does is, to convert the given character in lower case letter to an upper case letter. So, if it is a lower case letter, it will convert it to an upper case letter. Otherwise, it does nothing. Now, let us think about, what we mean by scanning a character verses scanning an integer.

(Refer Slide Time: 12:49)



So, let us say that I have two variables, in number and character letter. So, let us say that I am scanning now n, which is a number and the input was 12. So, when I say scanf("%d", &n), I am trying to read 12 into an integer variable. Now, on a typical machine an integer variable may occupy 32 bits. or 4 bytes. So, it has 32 bits in it. And if you know the binary notation, you will see that 1100 in binary is the number 12.

So, this is 8 + 4, which is 12. So, when you see that, the input is 12. And then, I ask in the c program, I am doing scanf("%d", &n). What will happen is that, n is an integer with 32 bits wide. And it will have the following pattern encoded into it. This is what, it means by scanning an integer. And if you try to print it out, it will try to interpret n as a decimal number. And it will print and the digit 12, here the number 12.

On the other hand, for the same input, here is the difference I want to emphasize. If the code was saying, %c and letter, so scan the input 12 using the scanf statement, scanf %c and letter. What will happen is that, the c program is looking at the first character, which is the digit 1 and scanning it in. Now, character 1 is ASCII 49 it is not important, you remember that. But, it has some ASCII value, and that ASCII value 49 is 31 in hexadecimal because, it is 3 * 16 + 1, which is 48 + 1= 49. So, that character 1 is 31 in hexadecimal. And hence, what will be stored? Remember, that a character ASCII character is 8 bits wide. So, it will store 31 in hexadecimal. So, it will be 3. This is the number 3 and this is the number 1. So, when you scan the input into a character variable

called letter. What will happen is that, letter will have the number, hexadecimal 31 or ASCII value 49, which corresponds to the number, which corresponds to the character constant 1.

So, this is the difference between scanning a given input as a number and scanning a given input using a character. So, when you scan it using a number, this entire thing will be scanned. It will be converted into binary and you will store it in an integer variable. When you scan it as a letter, it will scan the first digit only because, that is the character and then store the ASCII value, inside the letter variable, inside the character variable. So, this corresponds to the letter variable 1 within a single quote, the character constant 1.