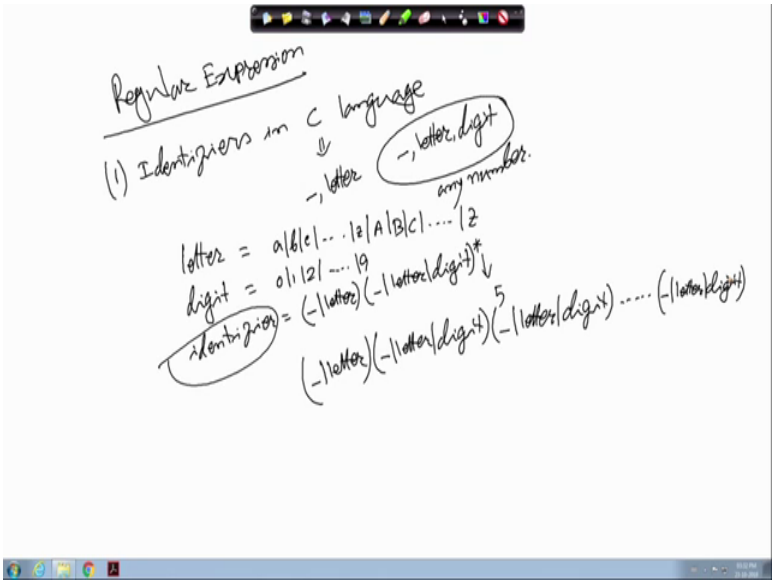
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 14
Lexical Analysis (Contd)

So, next we will be looking into a few examples of this regular expression and what do they mean; how to write the regular expression; how to convert them to NFA DFA etcetera. Because that should give us a good practice and I would suggest that you practice on your own taking many other examples available in different sources ok. So, to start with we will be looking into we will try to write a few regular expressions. We will try to write a few regular expressions.

(Refer Slide Time: 00:45)



The first one that we will try to write is for identifiers in C language; identifiers in C language. So, this is very common. So, almost all programming language it will have some identifier. So, if you are designing something some compiler for some programming language. So, this is the first step that you have to solve. So, you have to write down some regular expression for identifier. So, first you need to consult the language manual and if you consult the C language manual, it says that it has to be the first character can be underscore or a letter and it can be followed by underscore letter or digit.

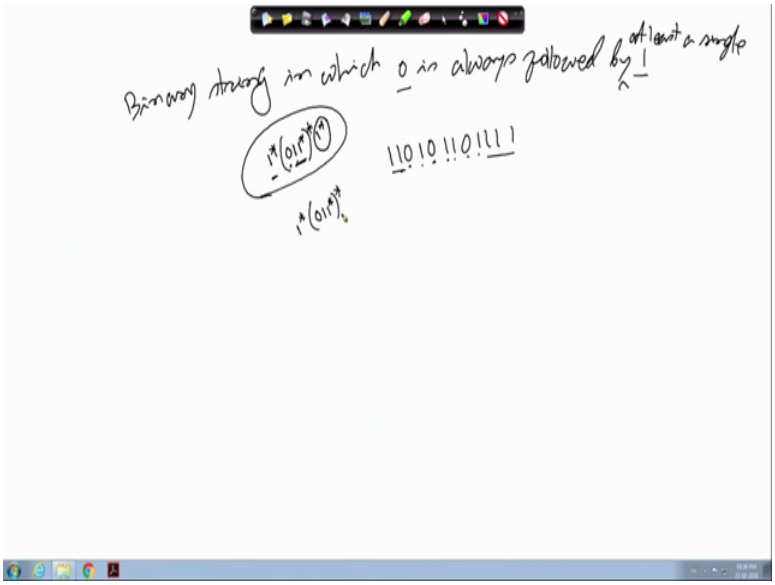
So, this can be any number of digits. So, this can be any number. So, that if you want to design. Then, what you have to do is that first we have to put the put some regular definitions; like the letter is defined as the a, b, c this characters till z; then, capital A, capital B, capital C till capital Z that is letter. Then, you can say digit is 0, 1, 2 going up to 9 ok; 0 to 9. After that you can define identifier; you can define identifier.

So, it can start with the underscore character or a letter. And after that the next character, next all of them can come underscore, letter or digit and they can come any number of time. So, there should be a star on that. So, this defines the identifier for the programming language. Some programming language is they have got a limitation on the number of characters that you can have. For example, some of them has got restrictions say.

So, it is very easy if the size is not restricted; if the identifier size is not restricted. But if I tell you that the in some language; so the instead of star, so this has to be 5 only. So, that is totally 6 characters. So, first character is underscore or letter and next 5 characters can be underscore letter or digit. But total is only 6 character. Then of course, it is very difficult because you have to write it explicitly. So, you have to write like first character, their first underscore or letter and then, you have to write explicitly underscore, letter, digit; then again, underscore, letter, digit. So, you have to go on doing like this you have to go on doing like this for 5 times.

So that way, you see that it is a blazing that this language is very flexible in terms of the length of the identifier and it helps the compiler designer, rather than putting restriction on the complier. Apparently it seems that I have to support a very large number of characters in the identifier, but it is not really so. And of course, with the help of the tool lex if we do, then it is the minimum amount effort that we have to put. So, next we will be writing another regular expression. So, that will be binary strings such as a 0 is always followed by 1.

(Refer Slide Time: 05:11)



So, Binary string in which 0 is always followed by 1. So, for writing this, so one possibility is that there is no 0 ok so, it is and even the strings are of 0 length that is null string. So, they are also valid because they do not contain neither 0 nor 1. So, one possibility is that I have got only 1's. So, any number of 1's can be there.

But if there is a 0, if there is a 0; then, it has to be followed by a 1 and after that I can have any number of 1's not only a single 1. So, whenever there is a 0 it is followed by so, at least a single 1 you can say. So, 0 1 1's at least 1 1's 1 symbol is there and then any number of them and then, this can go on any number of times and at the end you can have any number of 1's.

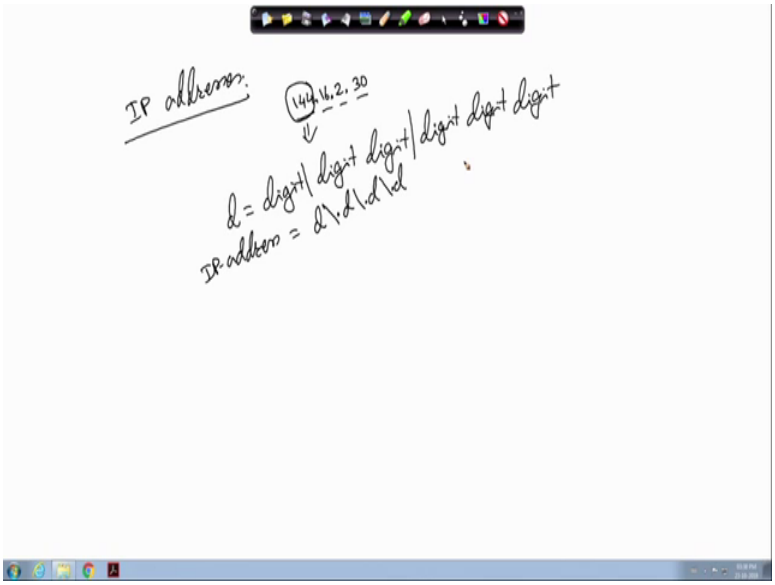
So, this regular expression; so, this will be capturing all those say for example, if I have got to say 1 1 0 1 0 1 1 0 1 1 something like this, then you see that it is a first two 1's will be matching with this 1 start then this 0 will be match with this 0. Then, this 1 will match with this 1. Then there is 0. So, this 1 star will be taken as 0 and then, for matching this 0. So, we will take another occurrence of this particular sub expression. So, this 0 will be matching with this 0, then these two 1's will match with these two 1's; then again, there is a 0.

So, another round of this symbol will be another this regular expression will be taken sorry. So, that will match with 0, 1 and then 1 star. So, 1 star will match with this. So, after that we have got 1 star and all. So, that way can take this 1 ok. So in fact, whether

this 1 star is necessary or not? So, that is also a question like if there are 1's at the end. So, that is taken care of by this part ok. So, I think. So, this was a this 1 star can be neglected also.

So, you can write it as 0 1 1 start whole star. So, that can also be written. Of course, putting the 1 star at the end does not make it more number of strings to be accepted. So, that way there is no problem, but that is redundant. Next we will be looking into another regular expression for IP addresses.

(Refer Slide Time: 08:23)



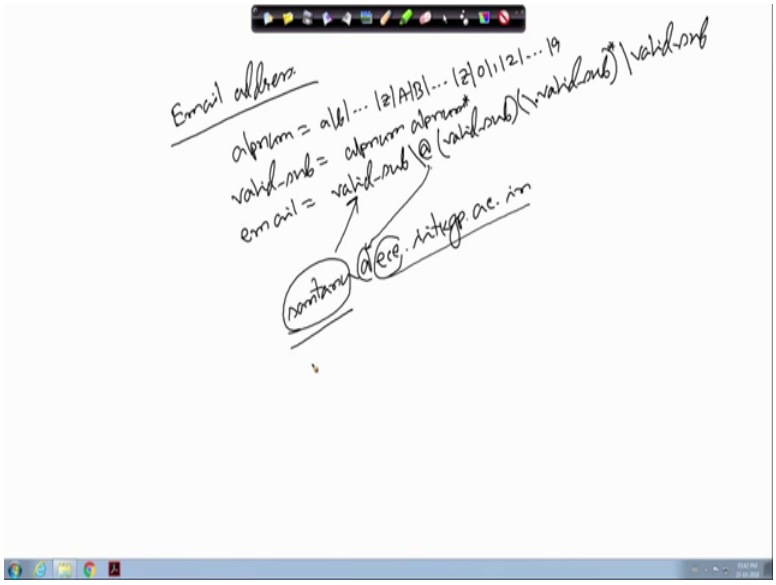
So, an IP address is so, in any IP address, so it has got 4 parts like if an IP address is say 144 16 2 30. Then, what happens is that so individually they are some numbers and there are some dots in between ok. So, there should be 4 such portions, 4 such parts and each part will be separated by a dot and each of them is a number.

So, what you can do? So, and this numbers; so there is a limit because this is a each of them is only 8 bit. So, you cannot have more than 3 digits in it ok. So, you cannot have more than 3 digits in it. So, we can say my d is equal to digit or digit 2 digits basically or 3 digits; digit digit digit and once it is done, then I can write that IP address is equal to d and then, this dot character has to be there and following the lex convention see we put this escape character d, then d, then d.

So, this is the regular expression for IP address. Of course, I did not check whether this values that are coming are valid or not; that is not the purpose of lexical analysis tool that will be done by parser or some high level program which will be having the values. And at what the lexical analysis tool can do? It can identify the corresponding number the matching number and it can return that part. So, that as an as attributes and then the program high level program can check; whether those values are correct or not.

So, that way we can have some regular expression for this IP addresses. Next, we will be looking into say another example say email address. So, email address.

(Refer Slide Time: 10:59)



So, for email address so, we have got alphabetic character and numbers. So, we say that we define alpha num as this lowercase characters a, b going up to z and the uppercase characters A, B going up to capital Z and then the digit 0, 1, 2 going up to 9 ok.

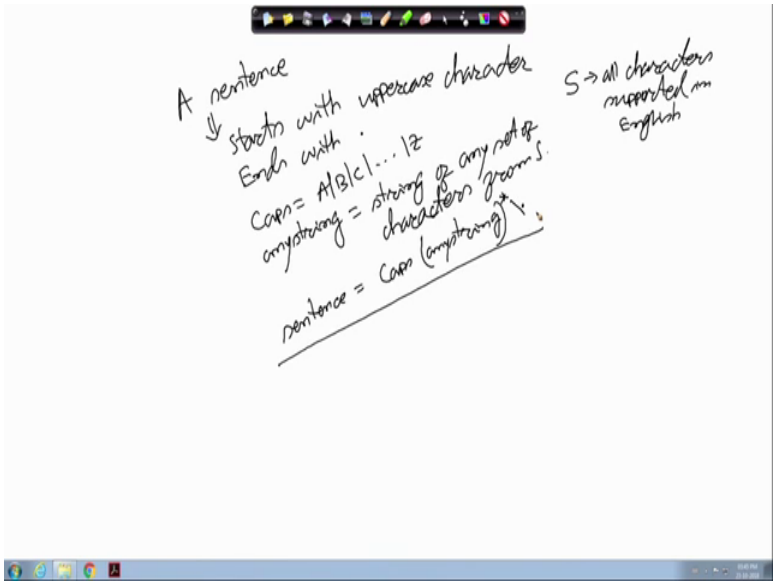
Then, we can say that a valid part valid sub part valid sub is equal to alpha num 1 of them followed by alpha num star. So, 1 or more occurrences of this alphabetic alpha alpha alphanumeric characters and then, the email can be defined as valid sub; then, at the rate symbol we have put purposefully this escape character to make it clear.

Then, I can have valid sub, then after this we can have valid sub. Basically, what I want mean is that suppose my email id is a santhanu at the rate ece dot iitkgp dot ac dot in. So, first before this at the rate character, so this part will be matching with this valid sub.

Now, this at the rate part matches with this at the rate part and after at the rate I can have any number of such sub parts. So, in this particular case, we have got 4 sub parts. So, you we have any number of sub parts. So, we can have this thing, this star. Sometimes, we are in email id, so we may not be mentioning this all this symbols. So, we can be it can be that it is just 1 1 valid sub part after that. So, we can put a or with this with valid sub.

Telling that there is no at the rate I just write santhanu; I do not write anything else. So, that also valid because by default it will attach some other parts to it. So, that will also become a valid part. So, this way we can we can write the regular expression for email address. Next we will be looking into another example. Say we are trying to design a system that can check whether it is a sentence or not.

(Refer Slide Time: 14:43)



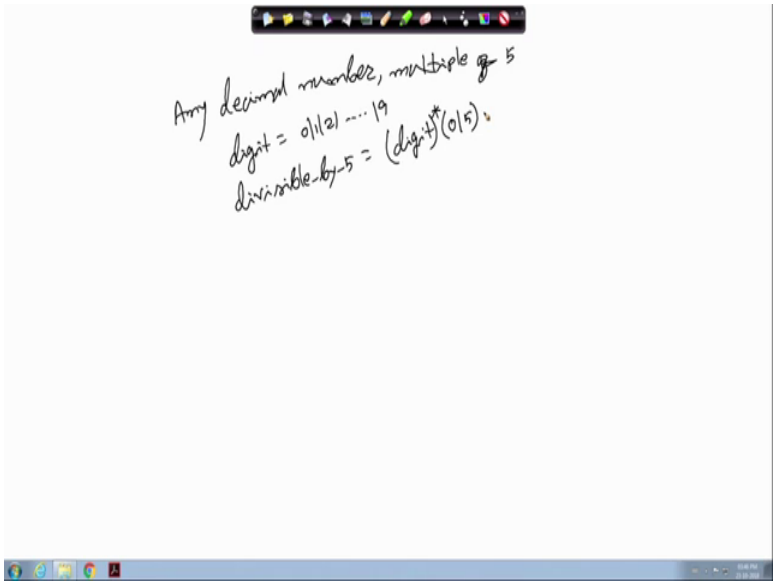
So, a sentence so, in English language a sentence means it starts with the uppercase character starts with uppercase character and it ends with a full stop. So, these are the bare minimum things that we need to have that we need to have in a English language statement. So, that you can do so, you can.

So, you can define like caps equal to all the upper case symbol A B C going till Z and then, any string you can define another definition any string; of course, I it is very difficult to tell what are the possible strings in the English language. So, it is so whatever the set of characters that English language supports. So, if I start writing it will be enormous.

So, I do not do that I assume that there is a set S which contains all characters supported by all characters supported in English language supported in English ok. Then, in that case the any string is string of any symbols, any set of characters, any set of characters from S; then, we can write that sentence is equal to caps followed by this any string and that can come any number of times. So, any number of strings, any number of words can come and then it should end with a full stop ok.

So, that way it can. So, this may be the regular expression for sentence. This is not very rigorous because of this definition of any string is not very clear, but anyway so we can proceed in that direction. Next, we will be taking another example, where it is a decimal number multiple of 5.

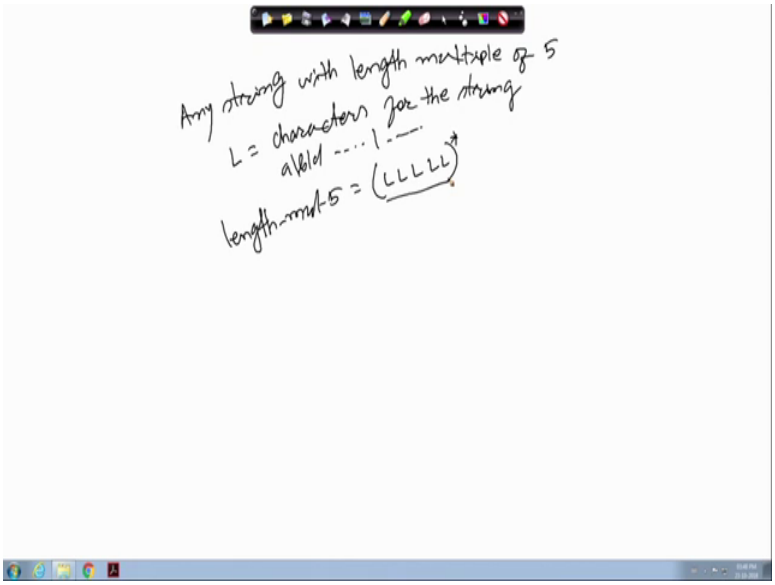
(Refer Slide Time: 17:27)



Any decimal number multiple of 5; so, if I need to do that, then the number decimal numbers which are multiples of 5, they should end with a 0 or a 5 ok. So, I can say I can first define the digit as I did previously for numbers 0 or 1 or 2 or 9.

And then, I can I can define the regular expression divisible by 5. I can define this regular definition divisible by 5 to be digit star and then, it should end with a 0 or 5. So, any number of digits can appear, but after that the last digit should be 0 or 5. So, in that case the number resulting number is definitely divisible by 5. So, we can have a regular definition like that.

(Refer Slide Time: 18:53)

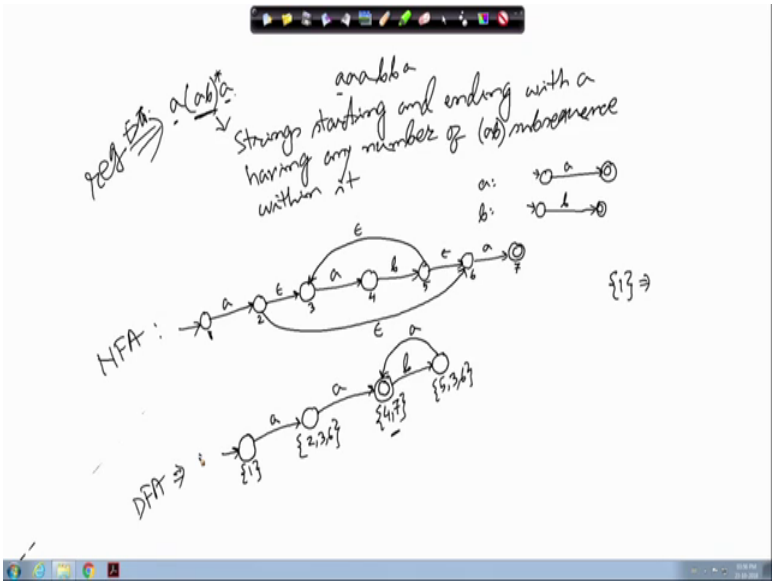


So, then any number who any then we will write another one any string, any string whose length is multiple of 5 whose length is multiple of 5; so if you want to do that then we can set the first I define L to be the set of characters that can come.

So, I can define all characters for on which we are going to form the string. So, it is the characters for the string. So, you can you can write it like say a or b or c etcetera. So, all the characters that are allowed for the language so that you can put in L and then, it says that length should be multiple of 5. So, that definition length multiple of 5 so, it can be L L L L and L and then, whole star. So, that means, so 0 occurrences. So, 0 is also multiple of 5.

So, that way it is a 0 length string. So, but I cannot have length to be equal to 1, 2, 3, 4 because then this part has to much and there are 5 such Ls. So, I will have length must be equal to 5 ok. So, this way we can define regular expressions for different patterns. Next, we will be looking into some regular expression and then, try to understand what is the corresponding pattern.

(Refer Slide Time: 20:45)



Say I have got this one a ab star a. So, what is this regular expression? What does it mean? So, you see that all the strings, they start with a and end with a and in between I can have a is this n number of ab's ok. So, if I have got a string like a a a b b ok. So, that say a. So, is it a valid string? So, this a the first a will match, then this ab has to match; but it will not match with a b. So, that is a problem.

So, it will be the string that is that it will recognize that strings starting and ending with a having any number of a b subsequence within it ok. So, this is the regular expression. Now, for this if we try to construct the NFA ok, then for a we know for a I have got the regular expression the NFA like this; for b I have got this one. This is for b. Now, I have to do it for a b ok.

For doing for a b, I can do it like this that this is the state from where it goes on a to this is the final state and from here on b I can go to the final state there. Now after that I have got a b star and for doing a b star, the construction is that I have to add 1 epsilon transition from here. Similarly, from here I have to add an epsilon transition to this ok, then I have to have another epsilon transition taking to the final state and then, from this state there should be an epsilon transition there.

So, this is the construction of star that we have done previously and then so, this construct the a b star. So, this portion and before that I have got this a and ending with another a. So, I will be having transition like this a and this is the start state and from

here, I will have another a going to this as the final state fine. So, this way I can have the regular expression converted into NFA. Now, this NFA.

So, if I number the states this is state number 1 2 3 4 5 6 7; then, I can construct the corresponding DFA. I can construct the corresponding DFA like this. So, I can take. So, initially I have to see that from state 1 what are; so, on state from state 1 on this on this state 1, from the state 1 how can I go to other states so on epsilon transition what are the other states on which to which I can go.

And here you see that on epsilon transition you cannot go anywhere. So, your state number 1 is the start state of the DFA. So, this is the start state of the DFA. Now from state from this state if you go on a, you will be reaching the state 2 and from state 2 on epsilon transition, you can go to the state 3 and 6. So, this from this one, I can come to a state which is 2, 3, 6 on a and from this 2, 3, 6 states, so if you go on a, say from 2 on a; you cannot go anywhere from 3 you can come to 4.

And from 6, you can come to 7. So, from this I can say that on a. So, on a I will be able to go to a state which is 4 and 7. From 3, you can come to 4 and from 6, you can come to 7 and from 4 and 7, there are no epsilon transitions defined. So, this is one of the final. This is this is the state where you can go. Similarly, from 2, 3, 6 if you try to go on b; so there is no transition define from 2, 3, 6 on b. So, on b there is no transition. Now from state 4 and 7, there are no transitions defined on a ok, but on b from 4 you can come to 5 so on.

So, this from 4, you can come to 5. So, the state is 5. Similarly, from 7, you cannot go anywhere on b. It is not defined. Now from 5 on epsilon transition you can go to 3 and 6. So, this 5, 3, 6 so, this 5, 3, 6 and here since 7 was a final state of the NFA. So, this also happens to be a final state of the DFA. Now this 5 3 6 from the 5 3 6, if you try so this is on b and from 5 3 6 if you get a ok. So, from 5 from 5 on a there is no transition; from 3 on a it can come to 4 and from 6 on a it can come to 7. So, that gives us this state only.

So, you see that this is a transition which is on a. So, this way we can construct the DFA ok. So, this is the corresponding DFA. So, this is the DFA. So, this is the regular expression. This is the regular expression. This is the corresponding NFA and you can have the corresponding DFA made here. So, this way we can have this regular

expressions for several strings and then, we can determine the corresponding NFA and DFA.