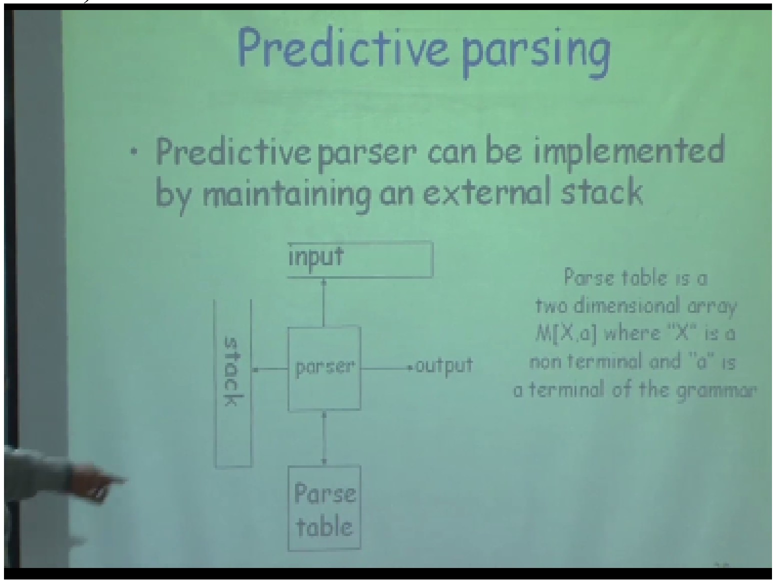
Indian institute of Technology
Kanpur
NP – TEL
National Programme
On
Technology Enhanced Learning
<u>Course Title</u>
Compiler Design
Lecture – 08

By…
Prof. S. K. Aggarwal.
Dept. of Computer Science and Engineering.

(Refer Slide Time: 00:14)



Predictive parsing

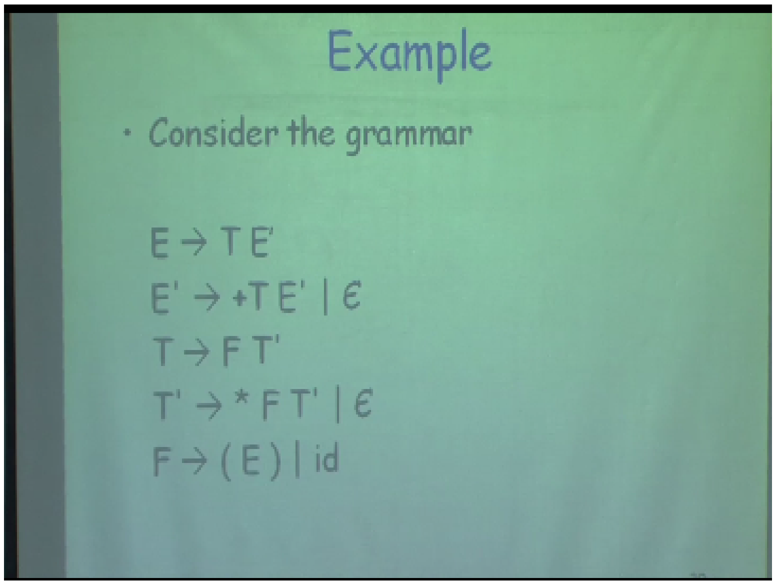
- Predictive parser can be implemented by maintaining an external stack

Parse table is a two dimensional array M[X,a] where "X" is a non terminal and "a" is a terminal of the grammar

input
stack
parser
output
Parse table

Instead of having a repressive reason for the Parser, we would to all use an external stack for stipulation and then we have parsing external and parse with parse table and grammar is encoated with the Tabular form and then what we did was we started looking at how this parser works we accurate put an example and we took an example of parser example before that we give parseculation of the grammar and the parse table actually is in XY non terminal and terminal of the language and for each of the formulation of non terminals and terminals we determine what is action taken in that action that stables itself and this is.

(Refer Slide Time: 01:00)



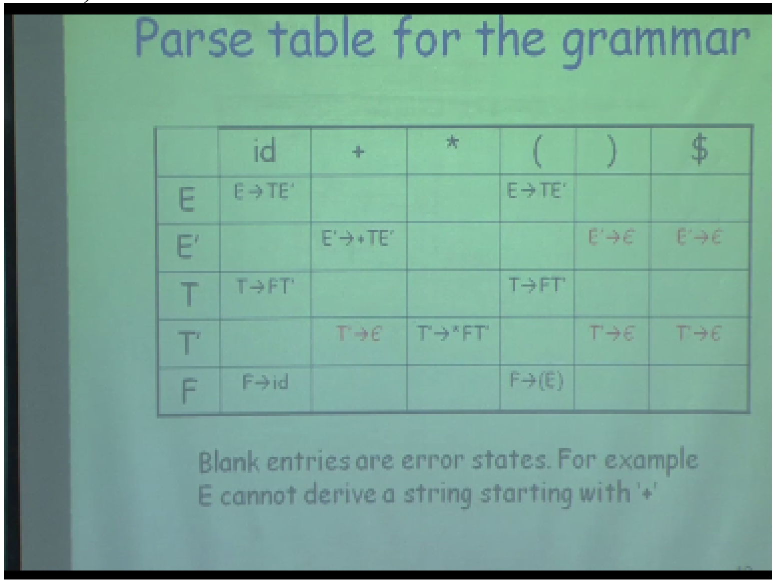
Example

- Consider the grammar

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \epsilon$
$F \rightarrow (E) \mid id$

The grammar to begin with this is the expression grammar from which can remove the level depression.

(Refer Slide Time: 01:08)



Parse table for the grammar

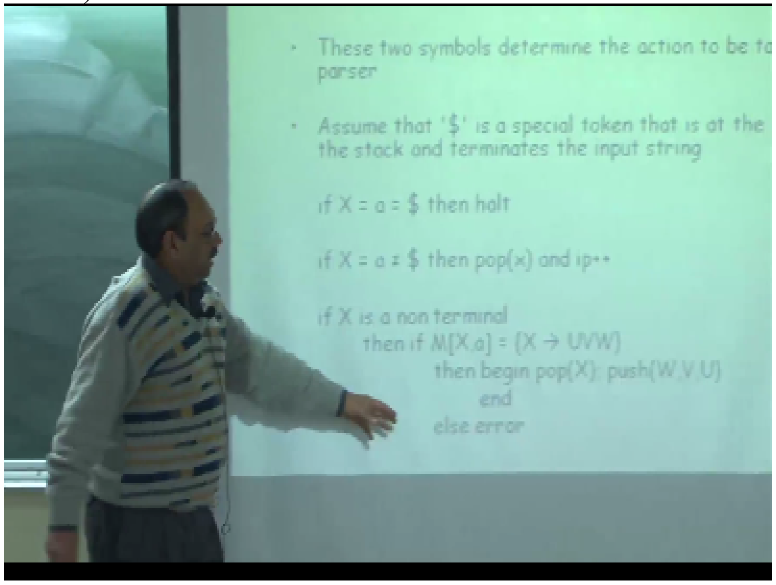
	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Blank entries are error states. For example E cannot derive a string starting with '+'

And then this is how the table looks for so what is table is saying is that I have this induction on non terminals and indexing on terminals dollar is special symbol it is a kind of the input string and at the bottom of the stack and what we want to say that if this is the symbol on top of the stack and this is my new head symbol then explain using this one right all the entries corresponding to these moves they are corresponding to expansion and all the entries corresponding to the plans.

They have error states which say that if this is my top of stack symbol and this is my loop ahead then and that we looked at the parsing algorithm so parsing algorithm was that whenever we

have a situation where we can do expansion that means top of the stack is a non terminal and then I explained this rule and when I say I expect my this move what that means is that I am going to pop this symbol from the sky and whenever top of stack is going to be a terminal symbol and it matches the input and if both bottom of the stack and the look at symbol they have dollar and then we stop saying that this thing has been accepted otherwise .
(Refer Slide Time: 02:46)



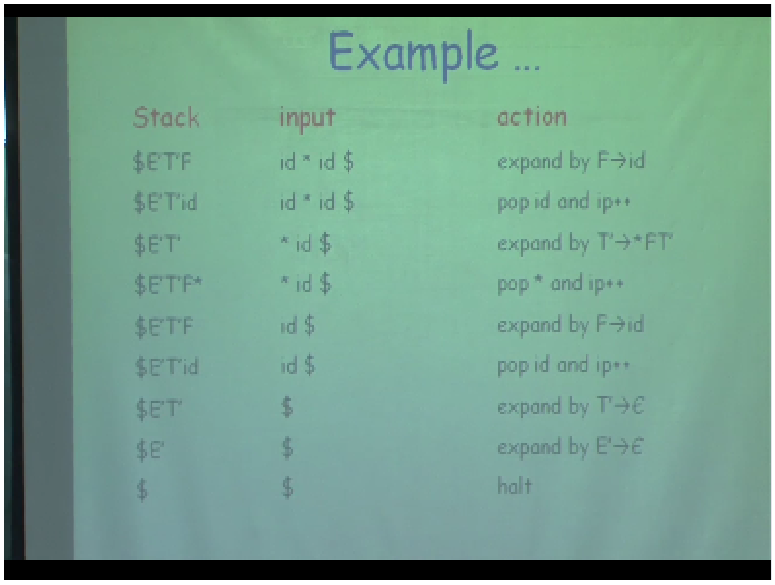
- These two symbols determine the action to be ta
  parser
- Assume that '$' is a special token that is at the
  the stack and terminates the input string

```
if $X = a = \$$ then halt

if $X = a \neq \$$ then pop(x) and ip++

if X is a non terminal
    then if $M[X,a] = \{X \rightarrow UVW\}$
            then begin pop(X); push(W,V,U)
                end
            else error
```

So this is what my parsing does that we match top of the stack if it is dollar than we halt if it is terminal then we pop and increment input pointer by one and if it is non terminal then I must expand by a rule and when I expand by this rule what that means is I am popping X from the stack and I am pushing UVW in reverse order on this side otherwise I am going to go into and that we took an example of saying that if this is my initial condition.
(Refer Slide Time: 03:19)



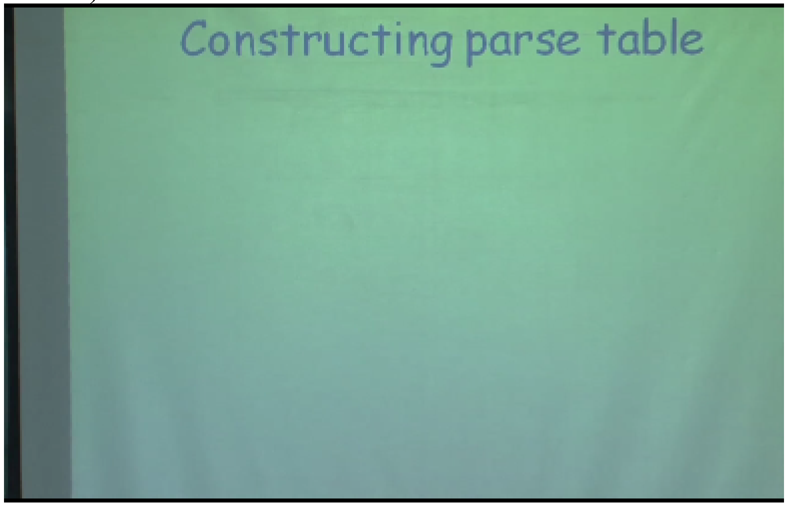
Example ...

Stack	input	action
$E'T'F	id * id $	expand by F→id
$E'T'id	id * id $	pop id and ip++
$E'T'	* id $	expand by T'→*FT'
$E'T'F*	* id $	pop * and ip++
$E'T'F	id $	expand by F→id
$E'T'id	id $	pop id and ip++
$E'T'	$	expand by T'→ϵ
$E'	$	expand by E'→ϵ
$	$	halt

And this is the string I am trying to pass then how do I go about doing it and we go about doing it by saying that my initial condition is dollar is at the bottom of the stack then this is the start symbol and this is string to be passed which is ended and when I say that top of stack is E and my look ahead is ID then I go and look at.

So take this fast table and then you look at the ID which says poppy from the stack and push T Prime to move this part please go back and study this part and what we are going to do now is that once we know what the algorithm is then we want to construct so we went through this example completely and we finally reached a state where we came to hot state where top of stack was dollar input is dollar.

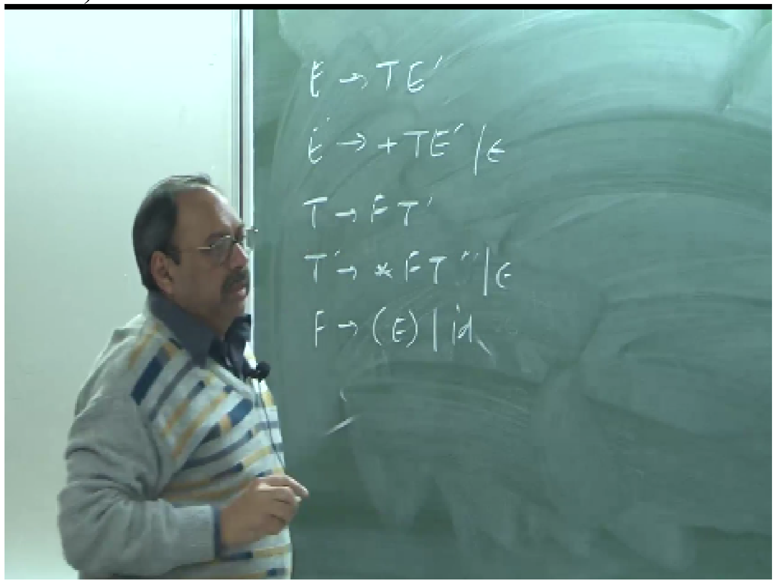
(Refer Slide Time: 04:20)



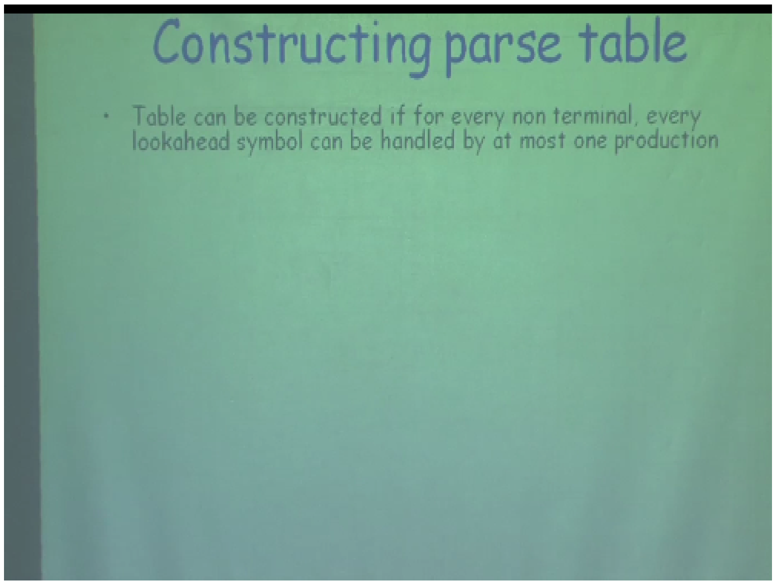
Constructing parse table

So next task in front of us is that now we want to construct a positive so there we go back to the parts at once again and here we are said that this is the parts we have discussed this algorithm

and this parse table which is including of this grammar into a tabular form like this and what we understand now and how you doing take this grammar convert.
This into a text this is making sense or you want me to take another extra class repeat everything what we did in the previous class okay we can move on from this class okay so what wrong to do now is let us take this grammar in which is this grammar And then see how do I convert this grammar into tabular form this is my grammar which I will keep using it so let me just write it as I go through this just slides.
(Refer Slide Time: 05:23)



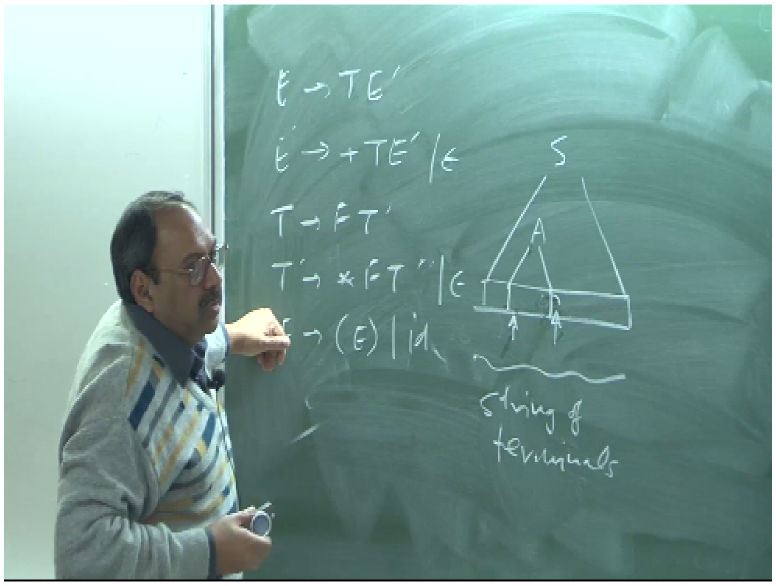
We are going to plus E prime TE is going to prime T prime is going to or FT prime and silent and f is going to direct expression of id this what I got and then what we will start doing this we will start analyzing this grammar and then we will come to somehow see that this grammar can be actually written into a form like this so how that will the past which is immediately in front of this okay how do we start doing it so first thing we started looking at was we said we are going to define what we know as a set of four symbols now I am also going to define another set of symbols which I falls fallow symbols okay.
(Refer Slide Time:06:10)



Constructing parse table

- Table can be constructed if for every non terminal, every lookahead symbol can be handled by at most one production

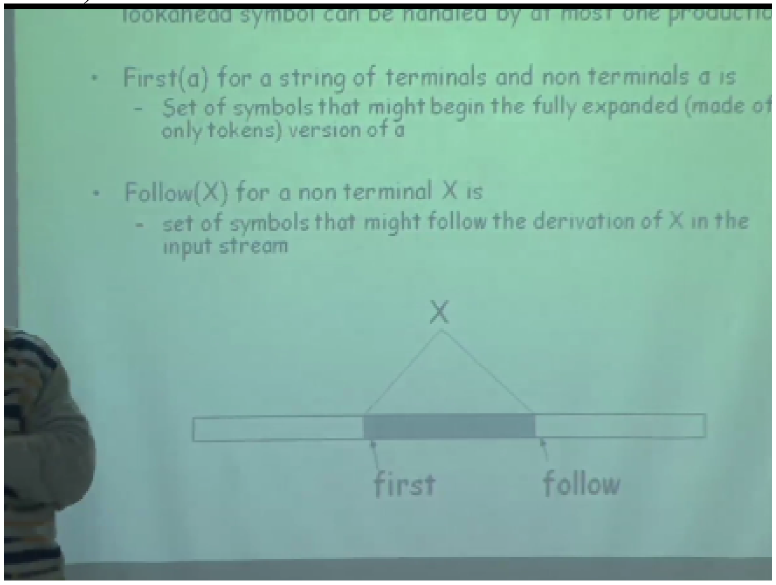
And then I will be fine how well come to fallow us but basically what this means is that.
(Refer Slide Time:06:20)



If I say that the recourse start symbols which is S what this means is that if I say that there is some stock symbols which is S which is deriving W which is the string in my language and then some other symbols let us say a which is a non terminal which denies this part of the string now I say that since a denies this symbol and obviously this is a string of terminals.
Let me say that the first symbol that can occur to any derivation starting from a that and we call as the first symbol and whatever is the symbol which can come in a string which is or which come immediately after this string which is derived from A it that I am going to call as policy and what we need to do is we need to compute for all the non terminals here the first and the follow sets and once I compute for all the long terminals.

Here the first and the fallow sets okay must I compute this I showed that using information on into coefficient information regular form to the titan then you apart from that is on this that is the critical purpose how will we compute first step and how do we compute fallowing sense so let us start by computing the first steps.

(Refer Slide Time:07:39)



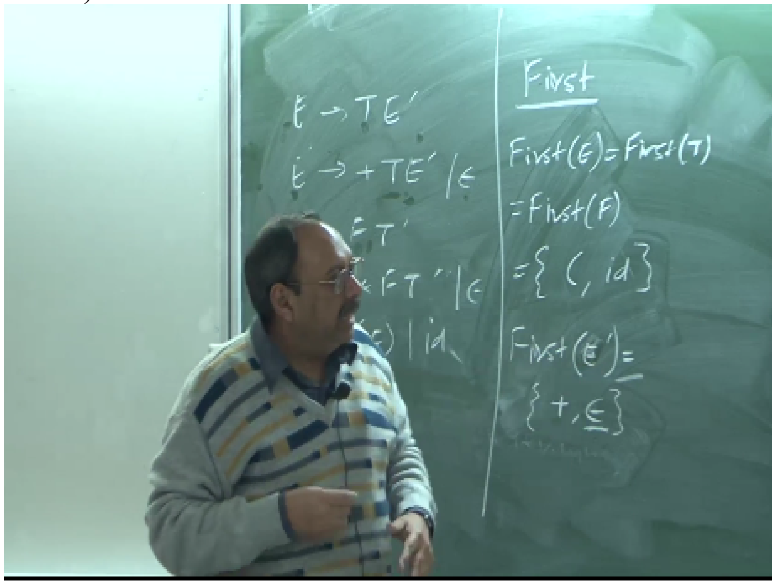
lookahead symbol can be handled by at most one productio

- First(a) for a string of terminals and non terminals a is
  - Set of symbols that might begin the fully expanded (made of only tokens) version of a
- Follow(X) for a non terminal X is
  - set of symbols that might follow the derivation of X in the input stream

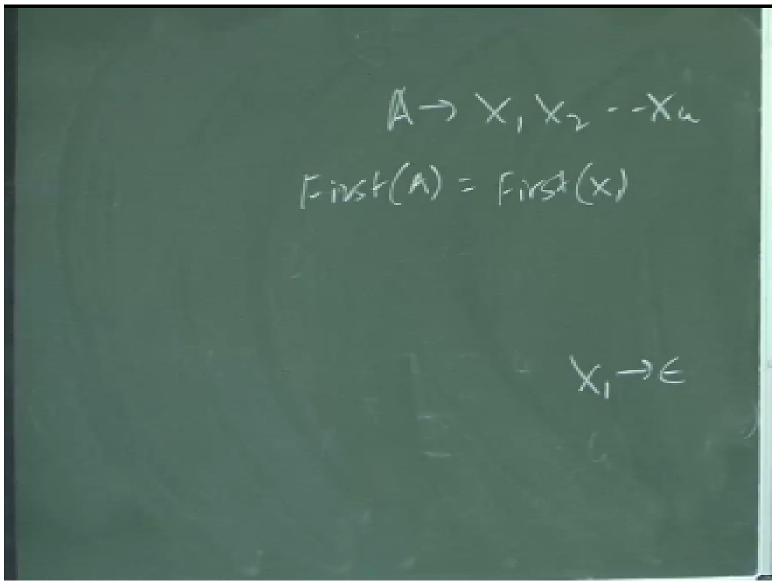
X
first
follow

So I keep this definition in front of you and that is slowly I will write to develop the terminals now this slide if I start any derivation from an non terminal then let me just find out what are the possible sets of what is the possible sets of terminal symbols you can solve so let us look at this if I start with derivation from E then what are the possible terminal symbols which can come into a string which can be derived from E okay.

(Refer Slide Time:08:20)



But how many roots do I have curry only one which says he goes to TE that means if I am trying to compute first of E then since T comes in the beginning it must be same as first of T cannot be anything else okay and therefore we say that this is first off T and if I now look at productions of T there is only one production corresponding to T which says T goes to F T prime which means that any string.
Which can be derived from T must start from something which is in first of F cannot be anything else and therefore we say that this is going to be same as first process and then if I look at first of F what are the terminal symbols which can come in the beginning of any derivation starting from F left parenthesis and ID these are the only two symbols which can come in twenty string which can be derived.
From so therefore I say that this is nothing but left parenthesis all right so first of each first of T in first off F they are the same time this is right okay now what about cost of each time so I am doing this computation for all the non permanent sigma so if I say first T prime sorry so let me first compute first off he planted let me go in that order.
So if I look at first of T prime how many productions do I have for E prime two productions so if this says a prime goes to plus T prime or this is E prime goes to X fine okay so what are the symbols which can come into any string which can be derived from E prime plus so basically this is nothing but plus and exile okay now this I will discuss in little more detail okay because how do I get excited here if this is slightly tricky okay.
What this means is that E prime T prime can derived yourself so the way to interpret this is that E prime when I say that E prime plus is in the first E prime that means is that if I start deriving from strings which start with E prime then string must begin with plus but E prime can also derive at side okay now when I say it can derive at side and I have to look now slightly more general.
(Refer Slide Time: 10:42)


$A \to X_1 X_2 -- X_n$

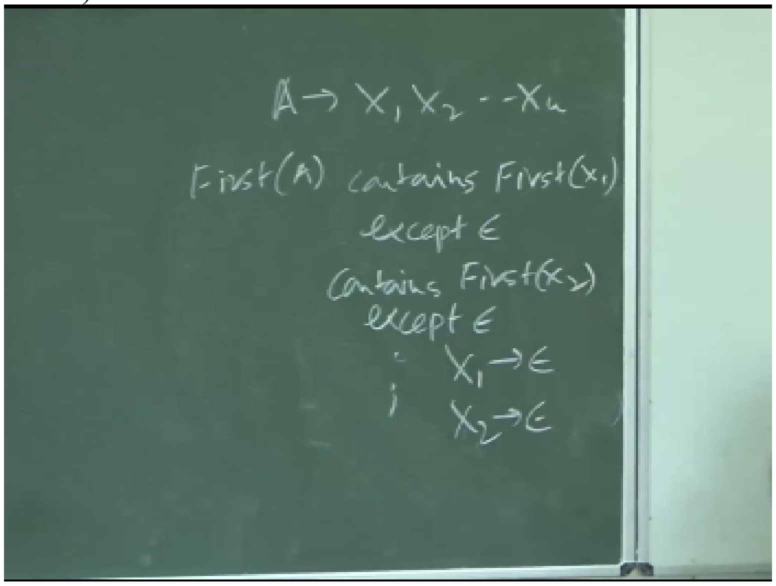
$First(A) = First(x)$

$X_1 \to \epsilon$

Suppose I say I have a situation like this which says a1 goes from X 1 X 2 to XN and I am trying to compute first of A now what is first of A first of x1 so now you have already got it right so it is first of X1 but if x1 derives a silence suppose I have a production of this form it says x1 derive epsilon then it is possible that in some derivation.

I can say a derives X 1 X 2 this and then I say x1 denies silent and then anything which is beginning with X 2 can also be part of this okay so therefore we do not say that first of A is equal to first of x1 but what we say is first off x1 it contains.

(Refer Slide Time: 11:35)


$A \to X_1 X_2 -- X_n$

$First(A)$ contains $First(x_1)$ except ϵ

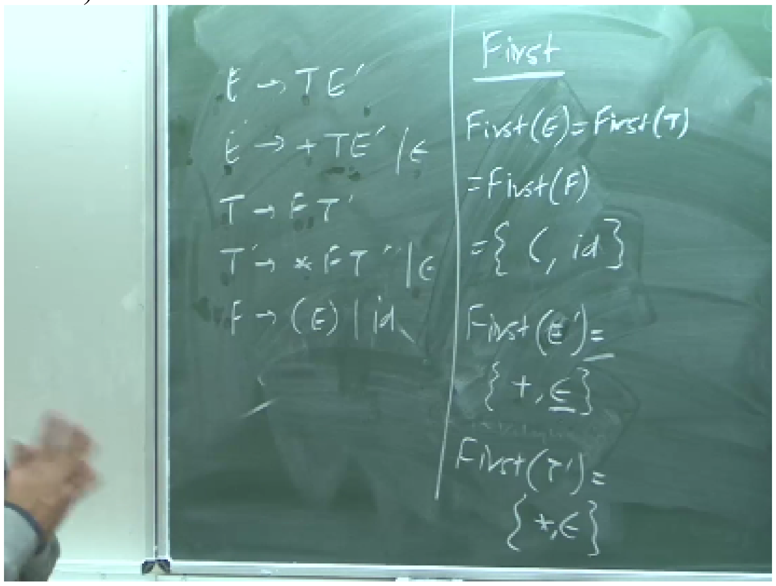
contains $First(x_2)$ except ϵ

$X_1 \to \epsilon$

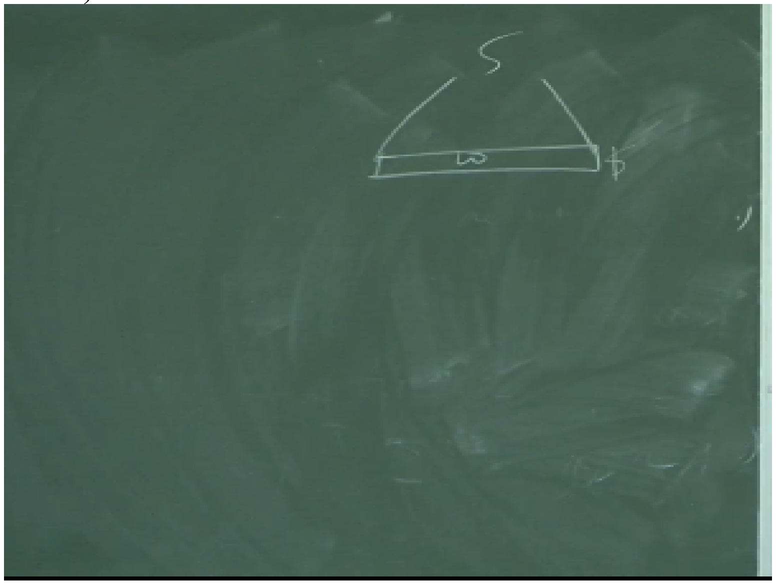
$X_2 \to \epsilon$

But a first of x1 except contain silent which means have this production that it also contains try not take a union it contains first of x2 and now it is possible that I have a production of this form it says X2 also goes to epsilon that is possible so then I will say this contains first of x2 accepted

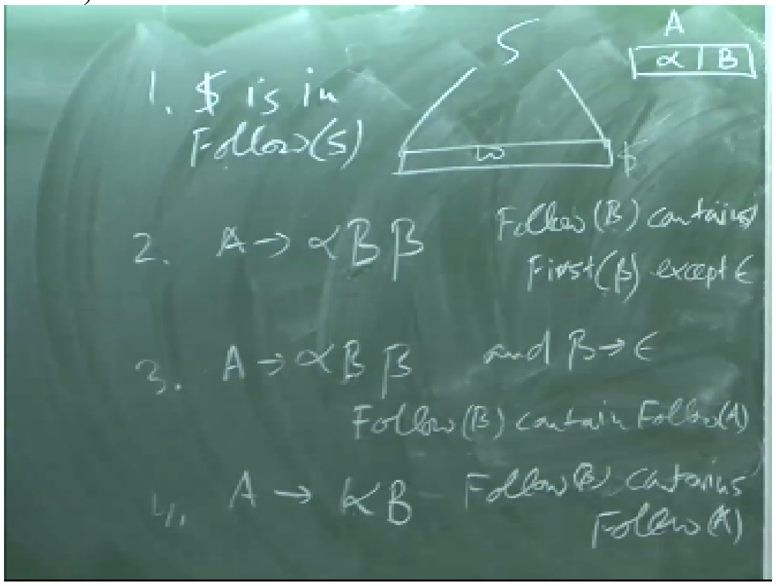
silently and I keep on doing this and if all of first of x1 first of X2 who and first of XN and all of them contain silent symbol then I say that A can therefore derive epsilon and therefore A epsilon is also contained in yeah .
(Refer Slide Time:12:00)



So this is the logic I will using here that first of each line contains plus + Epsilon and if I now look at what is first of T Prime what is first of T prime start at that side right I have only two productions T prime going to start at T Prime and T prime going to excite so now it will be so this is what my first example for all the non prime language right now let us get on to computation of the follow sets so let me keep this figure because that is which will give you some hints of along with to computation of the fallow sets so suppose s is my partial okay.
(Refer Slide Time: 13:38)



And I say that is deriving any space is any language that we had okay W is any string language which is because from us as is the partial number of prime and then what is containing fallow us look at this figure okay so what are the symbols which can fallow W because that means always I introduced the spacious symbol remember.
So that fallow us that why we used and we do not worry about kind of thing and so on here and we are saying is that always we are saying that putting this symbol here dollar is in fallows that was first definition first initiation or first initialization okay so always take a start symbol grammar and sat that the dollar is follow us parts okay anyway we remember that look compound keep on inputting and all these things now.
(Refer Slide Time: 15:00)


1. \$ is in Follow(S)

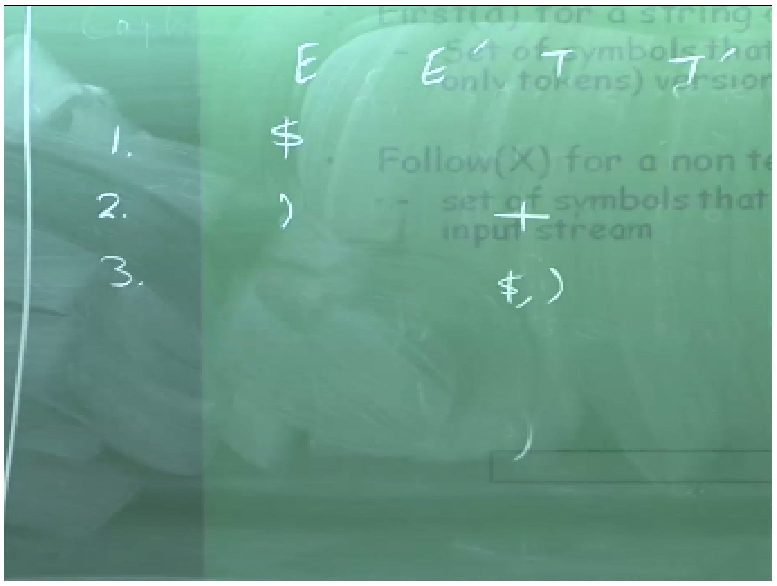
2. $A \to \alpha B \beta$ Follow(B) contains First(β) except ϵ

3. $A \to \alpha B \beta$ and $\beta \Rightarrow \epsilon$ Follow(B) contain Follow(A)

4. $A \to \alpha B$ Follow(B) contains Follow(A)

Let me look at the first rule is in follow S contains first of all pattern like this A goes to some of pattern like this okay and I to compute to the fallow B and the what is fallow B the contains beta is a string of string contains sets these people so fallow of B contains this is correct first thing as I saying is that fallow b contains first of beta expectance see some of that I can individually put any number of subtitles in therefore you will notice that follows actually.
So if I have a pattern like this then I will say fallow of beta contains first of beta except epsilon write everyone agree to this everyone comfortable with this okay next to next pattern it was just brought out if I had a pattern like this and beta derived epsilon okay right then what is happen I lost the following contains whatever is in fallow like we have only one more pattern and the pattern is if I say a goes to alpha B where B then the same rule will apply them.
And say that follow he convinced everything is fallow so it is a which is deriving and popping right like this a derives alpha b and this is therefore the this was my initialization by started to the

start symbol and then I am going down the with it from the context okay fallow b contains anything we need a pattern actually I have taken care of all the pattern where non-terminal occurs at the end non terminals occurs in the middle.
Where epsilon is not followed by the non terminal or non terminal occurs in the middle and epsilon is followed by this is what a pattern so using this and exactly what I am writing here is derived from just this figure if you can conceptually understand what this figure is say then these rules are nothing but that the convection of what we seeing this thing so now the trick the trick is you see rule number 3 and rule number 4 and the final form comes.
The form and therefore what will happens is that I can compute one fallow set and one four percent and then we look at another rule which again is going to change the follow set so this is going to finish so what may happen is that if I keep on computing this I must be compute it again because it is possible that depending upon the order of the rule sign picked up okay these follow sets been changed and I must recomputed fallow sets.
Can be change to compute fallow has change for using rule number three and four so I am cannot really I can give you what can I go into infinite loop because of this routine and I never saw is that possible yes no so look at this context every time what I am what is it that I am doing I am adding something right I am never removing anything from the contexts.
I am only adding and then if I add something and how many non how many terminal symbols having my grammar is that no there is not a bound on that right so is it monotonic function.
If it is monotonic then you know that it will can do that much so I do not have to worry about saying that I can keep on computing this and I may never terminate so I am adding something or I am keeping it constant but since I know if I keep on adding I can only reach the maximum terminals and therefore I have to stop it okay so let us compute now follow sets of safe grammar and let us compute the follow sets.
(Refer Slide Time: 21:41)



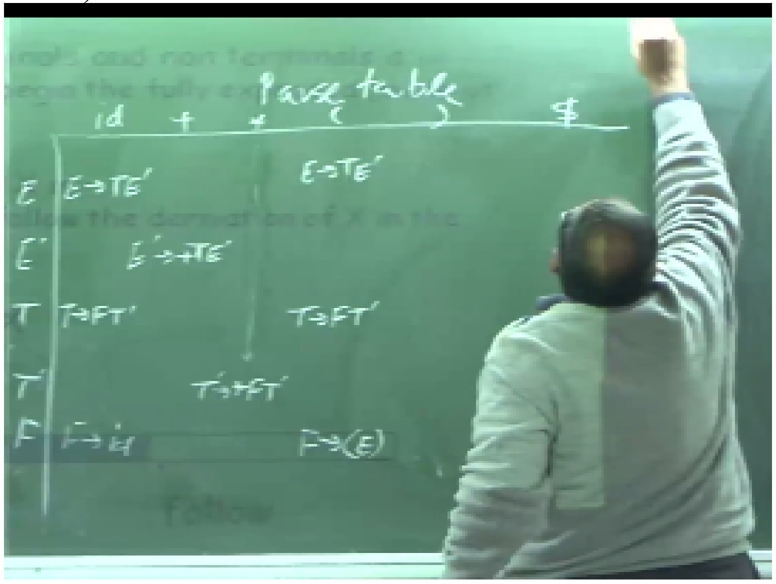
Let me write all the symbols goes here E prime T, T prime F that wise rule number 1 once I am going to do it I will going to do this like keep on applying the rule in that so apply the first rule says E is my past symbol fallows okay second rule saying on look at patterns we are non terminal occurs somewhere nil so now I am looking at this pattern we are saying that T occurs somewhere it has a symbol which is from the right hand side.

And loosing this rule at the rule number 2 I will say everything in front of T prime except the silent in this so right hat was he the rule 6 let us look at the E prime which have already compute here which is plus and silent so fallow of the we will contain plus remember everything except the contains okay what about this here look again the non terminal okay.

But this factor is same which is not going to adding now what about this fallow aspects have been everything in first of T prime everything what is T prime is direct silence so fallow of F will be star and this pattern there are giving new this is same pattern and what about this fallow of E contain fallow E will also contain like epsilons yeah okay.

Now it is a go to third rule the third rule is so just mechanically hope that rules the third rule is a group of patterns where B occurs are non terminal occurs in the middle but right hand side of the non terminal is epsilon so if I apply the third rule this says this one that occurs says this T prime is goes to epsilon first of fallow of the T will contain everything in follow of E which is $,) okay I have already computed follow T and that is where you see the problem starts coming that I know and if I look at this is follow of T because T prime derives of silent therefore follow of T will contain everything in follow feedback so follows T will contain everything in follow fee prime which is what about this T because T prime is dividing silent's.

So what is follow of T +$ right parenthesis right so follow of F will contain +,$,), wakefulness and what about this one this one same because T prime derive epsilon therefore follow of F will contain everything follow of T prime which does not add anything okay what about this can I apply this rule on this pattern no right because this is not what about this non dominant so I stop now I will number four and he rule number four I am looking at patterns when I say that so here is a non-terminal which is in black post position.
So fall of E - time we attain everything follow feet so this will be my parenthesis and here says although each E - time contains everything is follow of T. So this will contain of + $,) okay and this one says follow of contains T – Time is any non terminal on the potion. Now this rule number three and four have to plait again and again thuritham okay still I am know that I have noted anything more to the set. The applications of the rules clear what we doing here, now let me symatareously spot making a pause table.
(Refer Slide Time: 26:25)

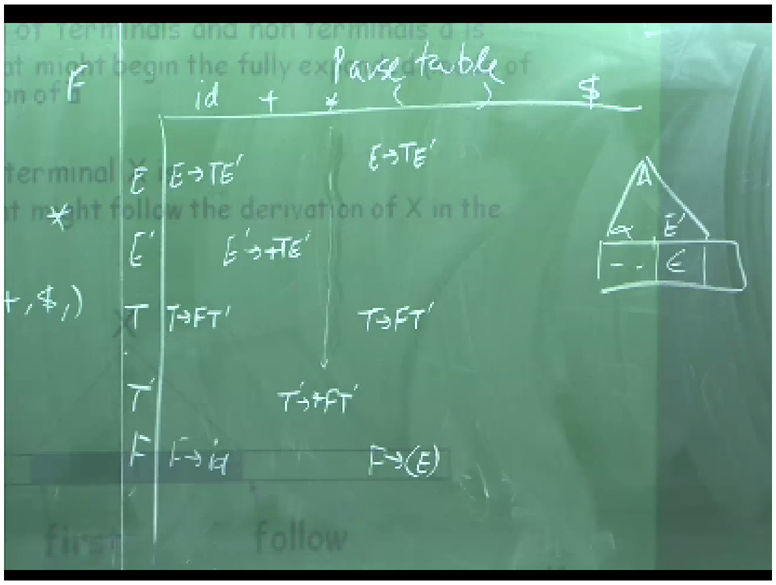


Pause table

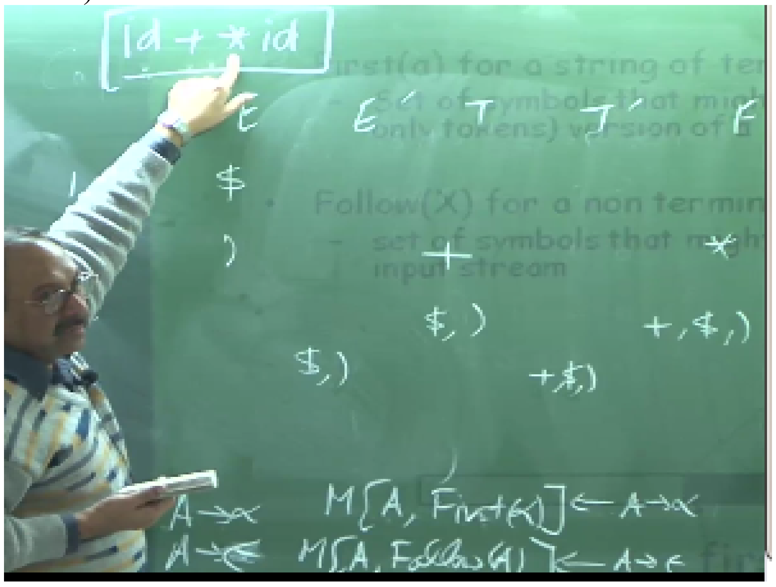
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$				
T	$T \to FT'$			$T \to FT'$		
T'			$T' \to *FT'$			
F	$F \to id$			$F \to (E)$		

The pause table remember that for index by the symbols which was E, E', T, T', F non terminal symbols. And then if I also index by the entire terminal symbol okay, so what are the terminal symbols in this cause are Id + * ( ) and $ which something has introduce right. So the + * ( ) and this is ID then at all okay. So now what we say is that because we are saying that key bows ID What that mean Id that look at all possible patterns look at all possible rules and if I have a rule like this which says it goes well yeah then what this means is that if I now have a table and I look at A and first of α.
So I am looking at so corresponding to this rule I am looking at an entry in the table which corresponds to this non-terminal and anything well what will that be what should the entry in the

table corresponding to this entry all I am saying here is that what are the century corresponding to this thing that if I this is top of my stand and this is the look at symbol then what should I do denies using this particular rule right so corresponding to this entering the parse table corresponding to this non-terminal.
And for all first of α except that silence use this tool for renovations okay. So when I start using this rule what it says is that if I look at first of a first off it contains left consists for ID right so E goes to T prime is the rule here because all I am saying is that corresponding to this is that all valid strings in this language must start either from ID or from that parenthesis they cannot start from either plus or multiplication Oh - what about E, E prime ? The E prime says that all valid strings which can be derived from a prime must start with plus okay.
So forget about epsilon productions for the time. So this says that if I now look at first of this that is plus so E’ and + it is going to give me a rule which says if I am goes to plus what about this rule this says t goes to a E-time and what is first effects first off this left parenthesis and ideals so first one into T I will say that T I D is going to T goes to F T Prime and that parenthesis in here so this is T both okay now what about this rule this is T Pangos to start at T time that means.
Whenever I see is thought and this is the launch of Allah then I can use this rule so this says T Prime and Star is here so this is T prime goes to star F prime so just and if I look at this rule which is opposed to bracketed expression then what do i do what do i use and what is the entry I am making here so corresponding to x and let phenol phases I will say F goes to bracketed expression and corresponding to F ID I say F goes to I so just by looking at the first end if I can make these entries in my pasta this year now let us come to website reproductions Howard says that somewhere eat prime course website.
I do not have any entry because there are a rule kick possibly they all be arranged sequels only to eat I'm going to exile now what are the situations and has aged so conceptually not visualize that so I'm not giving you algorithm I mean despite much later on sexually if you can understand – you will be much clearer right so what are the situations under which each prime goes to an exile and so you look at a situation like this again from the derivation point of view that if I say somewhere it goes through select me this is actually the draw this together a little more space.
(Refer Slide Time: 32:13)

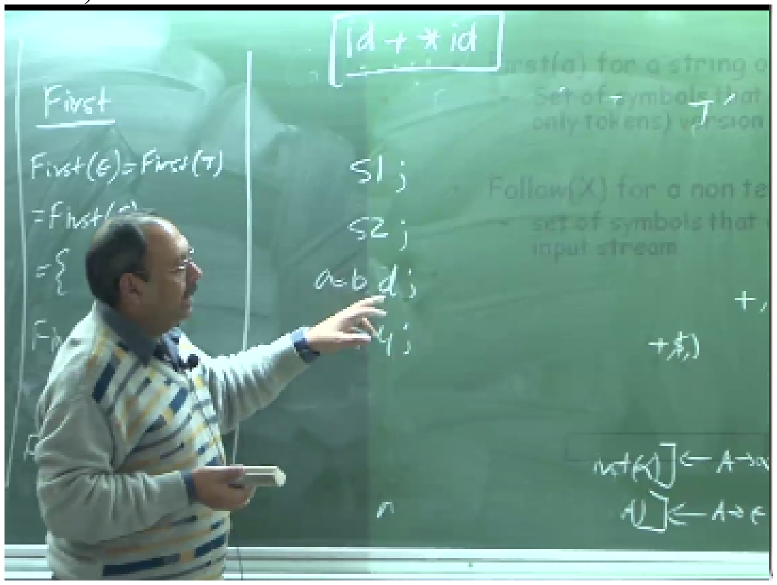


So somewhere I have a situation like this a goes to let us say some alpha and let us the heat right and then eat mangoes to outside and I will buy some okay what are the situations under which I would use this production which says that goes silent because you say no other production is working what that piece is that I can make those entries only when I am parked in this way but I made possible before I start to power all right then I just reduce it.
So that is the valley then remember that I talked about sentential forms I am saying that one of the causes of all possible elevations which can happen from this right so I must all the time have value sentential forms so therefore in any valid sentential form I am capturing this information either by first or follows X so now I am saying that in case each line goes to epsilon then look at photo set of a prime and in that entry so I say that if I have an entry like this.
If I am going to or equal to epsilon then look at entry corresponding to this and follow of A and that entry is going to be and you see this what is happening here so this is all I need that is how I computed follow sets okay so now if I see that I have a production which says if n goes to Epsilon let me look at what is the follow set of keep trying and if I say follow set of V prime is Donald by parentheses then I say corresponding to this e trying and similarly I will say I have a production it says E prime goes to Epsilon.
So I look at follow set of C prime right and I say that corresponding to plus so there is a prime corresponding to plus I will say keep language silent corresponding to right parenthesis deep anguish silent and corresponding to this is performing to $ value in website say I did not take you to next level of the application of three and four but basically and this is really what I have done I started with this level then I computed my first sets I compute my follow sets.

And I encode this first and follow information this tells you already sensitive forms which can come with language and let us top it can be just tag that analysis so anything else will say that this is not a validation function form and therefore should be discarded here questions deleting it if something is not clear ask me now so just by pre-processing this information computing husband Paulo sex and everything comes from just this figure really there is nothing else.
If you can just remember this that I derived in this particular string and part of it derives from game and here is first and here is followed then everything can be just visualized not let me come to a different situation now suppose you encounter a situation where somewhere so let us take this particular self are you encounter a situation where you say that top of my stash is he playing mind put a star pausing and say so input is incorrect.
What will you do something move right so we were saying that at some point of time it is not sufficient to say that I have an error and should do more parsing you find more and more errors now what does recovery need so I bought into a situation when I say that my input is let us say stock and my top of stack is prime okay this is my input pointer this is my stack pointer and parts of says I have reached an error state right. So how we continue from here only by if I reach one of these entries I must manipulate my stack and input to reach one of these entries. So first event game suppose I say I know that this input is incorrect so look at this,
(Refer Slide Time: 38:42)


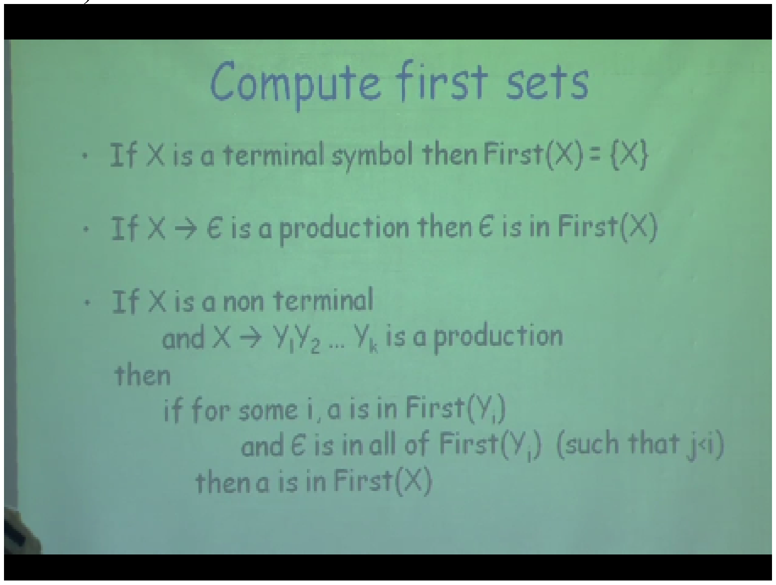
Suppose I am trying to parse a little more space suppose I was trying to pass I D + * not I am not trying to guess the intention of the user I am not trying to guess because I already know this is incorrect and by guessing I may create more mess but I want to fly more and more errors they do not come P that I will be able to give all right errors but suppose I say ID has been consumed

plus has been consumed then I see start is what I was expecting or something else I could either have got an ID here or I could have got a left parenthesis here nothing else was landed okay.
So if I discard this input and I reach this can I continue parsing I can write so what is the general principle therefore what was the single time at this time in fact so if I say that's my proper stash in the time and I have reached an error state I say start with sorry symbols then you hit a symbol which is in first off the non-terminal which is on top of stack and then I find an entry corresponding to that in the past able and continue parsing from this that is one standing so is that because we by deciding part of the input I think continued possible so another way so if I pick up an example from programming languages.
(Refer Slide Time: 40:14)



So Super Lab statement 1 followed by; S 2 followed by; S 3 statement forth okay. Now suppose or let me say there is a semicolon here but let us say this statement was something like this okay and I found that something is missing here okay then but look at this is let me just start this in boo okay and when I hit here okay then hopefully I will be able to continue parsing from that there are no hard and fast tool which you say that I will be able to decipher from this error because it is possible that in way of recovery I mean actually discard more errors and in some situation worse cases I may reach end of the program.
But this strategy is telling me that at least there is a pole that I will be able to recover from this and can be powerful so first strategy we use is that if top of stack symbol is a then discard input will I hit something which is in first okay and then try to continue parsing from it that is one okay somebody said is me so another strategy is continued discarding input till you get a symbol which is in follow of a and then pop a and try to continue parsing from them.

So these are the two strategies which are used and this is also known as panic mode as a recovery where we say that I will either discard some symbols now can I keep that is part of my power stable what are the symbols because I know the first is follow sense right so I can always compute that if I am using now follow sex okay. I can say I know form of E and what are the symbols returning follow of E follow fee contain high precision and collar.
So let me say I call that as a same symbol so I am saying if I get into error state then I can synch on this and thinking all it means that looking for symbol which is not the same set then pop this symbol and continue passing the other strategies people have worked on so there is a lot of literature which may be available if you see where you will find that people will try to guess in terms of the user and correct and therefore the common ever think of these nodes that is use this that first you try to find the symbol in first off top of stack .
They start everything up to that symbol from the input try to continue parsing and the second strategies is look for symbols which are in follow of what is on top of stack pop this symbol and try to continue parsing. So now we know how the parser works now we do not want to take it down would modify the number to do all kind of recursion from that then what that is so let me now quickly take you through this life-size sort of discarded and so in the slides you will find the same thing for me a bitch bust here and I will just take it. So if you do little more structure then what I was discussing all right so first I compute for sex.
(Refer Slide Time: 44:12)

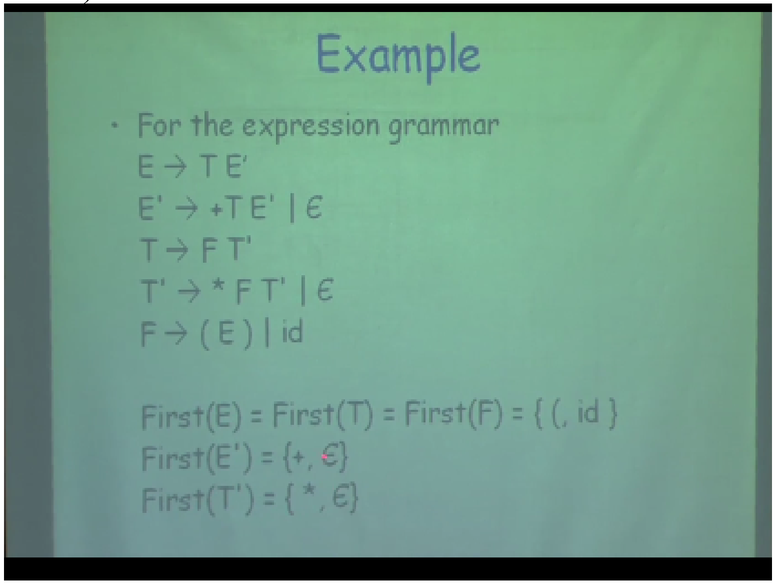


Compute first sets

- If X is a terminal symbol then First(X) = {X}
- If X → ε is a production then ε is in First(X)
- If X is a non terminal
  and X → $Y_1Y_2 \ldots Y_k$ is a production
  then
  if for some i, a is in First(Y_i)
  and ε is in all of First(Y_j) (such that j<i)
  then a is in First(X)

So we say that X is the terminal if it is a terminal symbol then you maniacs and if there are a production of the swamp then we say of silence in first of X and if I have a situation which says X goes to Y 1 2 YK and each of y1 is by I is included in first of X except silent and if all of them my eyes contain epsilon then you say that Roscoe cracks also contains that side okay so this is what we did in the consultation of first step.

(Refer Slide Time: 44:28)



Example

- For the expression grammar

  $E \rightarrow T E'$

  $E' \rightarrow +T E' \mid \epsilon$

  $T \rightarrow F T'$

  $T' \rightarrow * F T' \mid \epsilon$

  $F \rightarrow (E) \mid id$

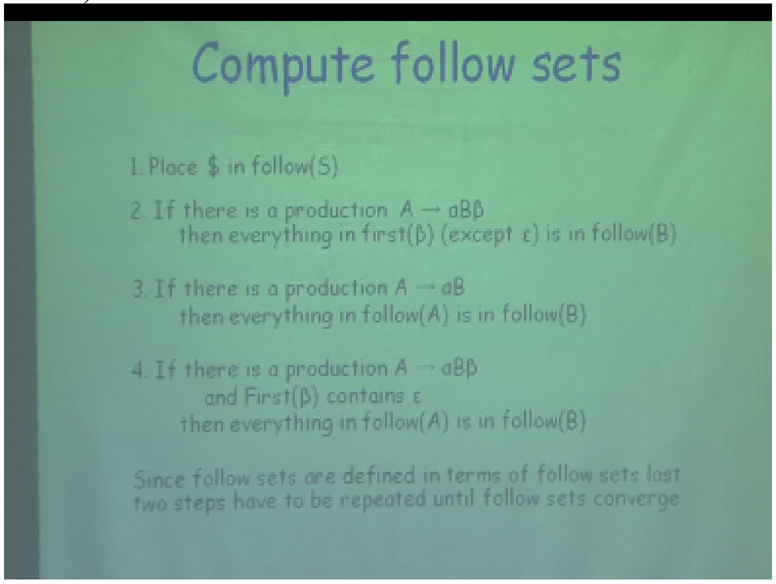
  First(E) = First(T) = First(F) = { (, id }

  First(E') = {+, ϵ}

  First(T') = { *, ϵ}

And this is what I just computed the first of each end is left parenthesis and ID and first of e prime and cost of T time are three sets and then we computed with follow sets.

(Refer Slide Time: 44:51)



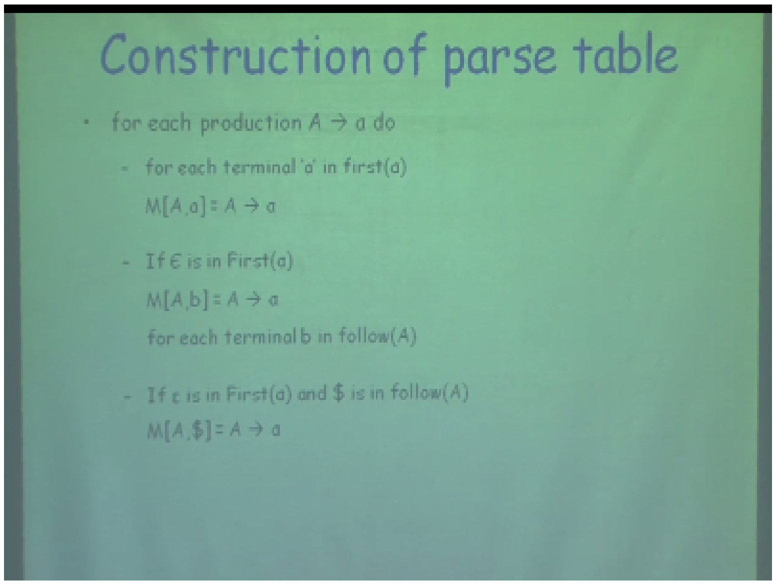
Compute follow sets

1. Place \$ in follow(S)
2. If there is a production $A \rightarrow \alpha B \beta$
   then everything in first(β) (except ϵ) is in follow(B)
3. If there is a production $A \rightarrow \alpha B$
   then everything in follow(A) is in follow(B)
4. If there is a production $A \rightarrow \alpha B \beta$
   and First(β) contains ϵ
   then everything in follow(A) is in follow(B)

Since follow sets are defined in terms of follow sets last two steps have to be repeated until follow sets converge

So we said dollar is in follow of s which is the start symbol and then I looked at various patterns which says a goes to alpha beta and it goes first offense Epsilon and then you can put it follows X and the same for the same grammar there is the follow set we come out with and once we have these follow sets.

(Refer Slide Time: 45:13)

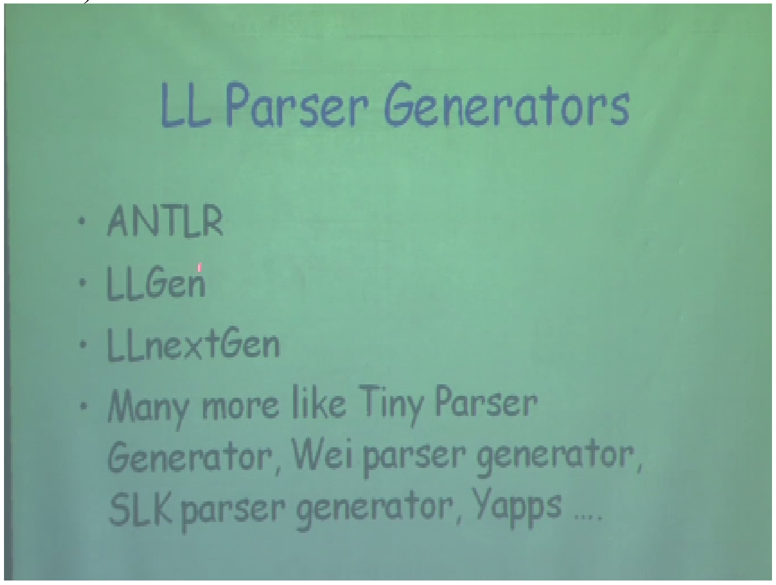


Construction of parse table

- for each production $A \rightarrow \alpha$ do
  - for each terminal 'a' in first(α)
    $M[A,a] = A \rightarrow \alpha$
  - If ϵ is in First(α)
    $M[A,b] = A \rightarrow \alpha$
    for each terminal b in follow(A)
  - If ϵ is in First(α) and \$ is in follow(A)
    $M[A,\$] = A \rightarrow \alpha$

Because that are stable then what do we say that for all rules of this form it goes well far we say that if a is in first of α then I am going to include it here and for all symbols if epsilon is in first of a then I am going to use B here which is in form of its ID yes and any grammar which has a so in this case might look at is 1 here so this is another test which is used that you actually construct the cross table and see whether for some grammars will not be able to construct a particle with only single and within each cell is possible as normal people that is also a little one and these are of the various parts of generators.

(Refer Slide Time: 45:58)

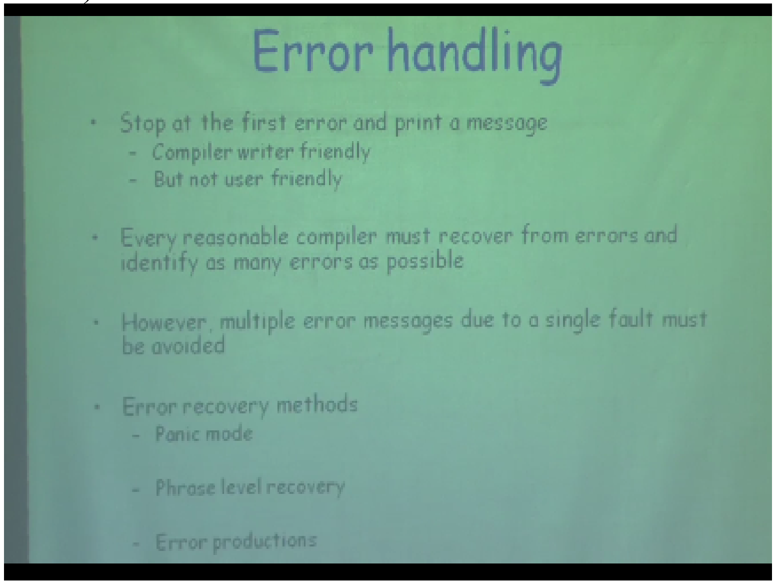


LL Parser Generators

- ANTLR
- LLGen
- LLnextGen
- Many more like Tiny Parser Generator, Wei parser generator, SLK parser generator, Yapps ....

Which are available to us which are compound parsers enter is one very popular parser but then there is Jen Alde next-gen and then you can search on that so there are something called assignee parser generator by parts again because I said gates which are available to us.

(Refer Slide Time: 46:19)

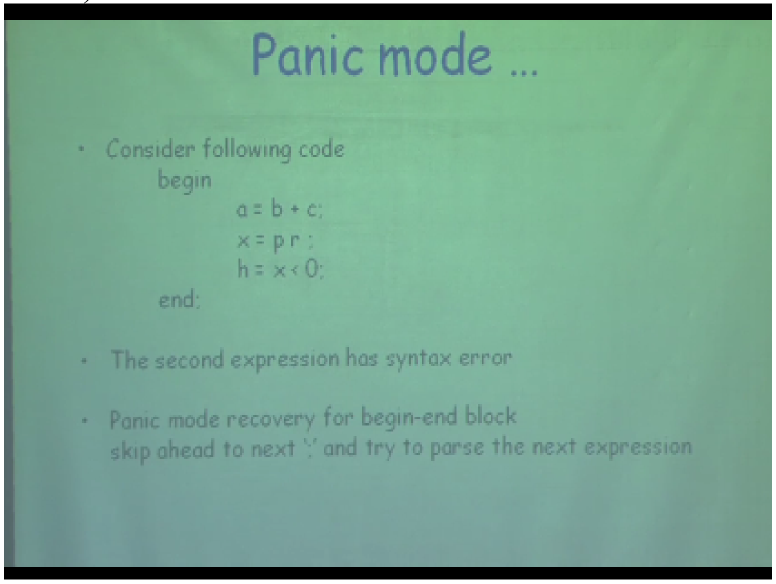


Error handling

- Stop at the first error and print a message
  - Compiler writer friendly
  - But not user friendly
- Every reasonable compiler must recover from errors and identify as many errors as possible
- However, multiple error messages due to a single fault must be avoided
- Error recovery methods
  - Panic mode
  - Phrase level recovery
  - Error productions

Error handling okay if we stop at the first error and just even have a message that may be really friendly versus compiler right there but definitely is not something so we must do every recovery and every compiler actually recovers from errors so panic mode. So I straightaway take you to what I discussed.

(Refer Slide Time: 46:44)



Panic mode ...

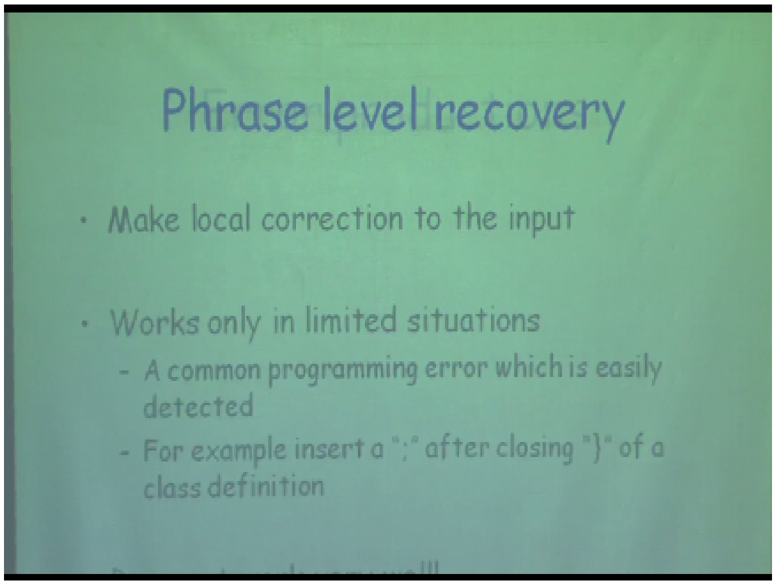
- Consider following code

```
begin
        a = b + c;
        x = p r ;
        h = x < 0;
end;
```

- The second expression has syntax error
- Panic mode recovery for begin-end block
  skip ahead to next ';' and try to parse the next expression

You can read,

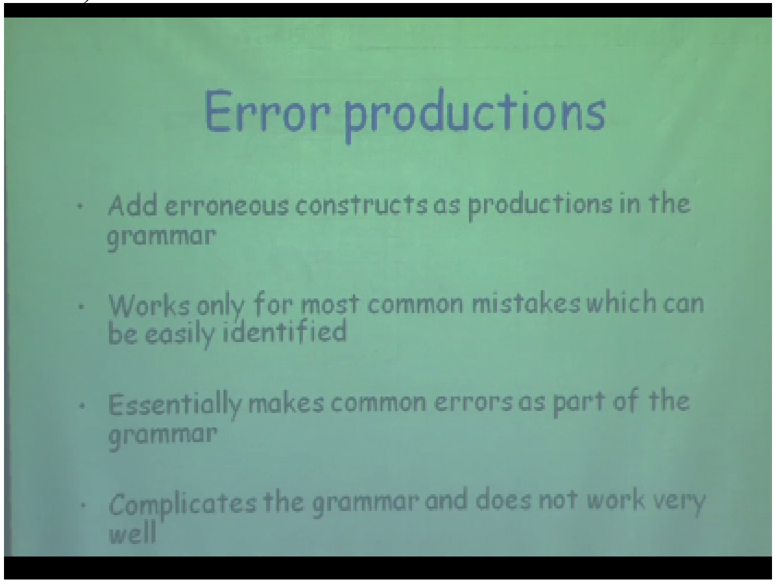
(Refer Slide Time: 46:46)


Phrase level recovery

- Make local correction to the input
- Works only in limited situations
  - A common programming error which is easily detected
  - For example insert a ";" after closing "}" of a class definition

Rests of it there are many methods of,

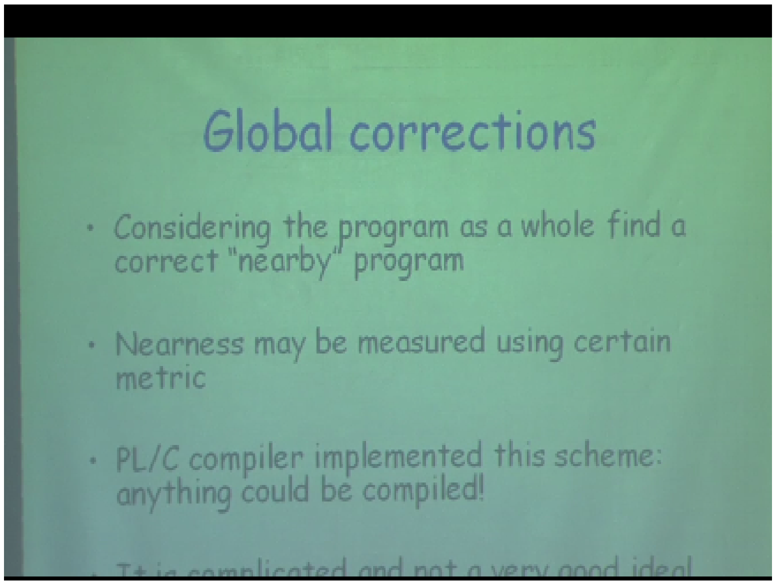
(Refer Slide Time: 46:48)


Error productions

- Add erroneous constructs as productions in the grammar
- Works only for most common mistakes which can be easily identified
- Essentially makes common errors as part of the grammar
- Complicates the grammar and does not work very well

So nobody actually will try to do correction except some compilers.

(Refer Slide Time: 46:55)

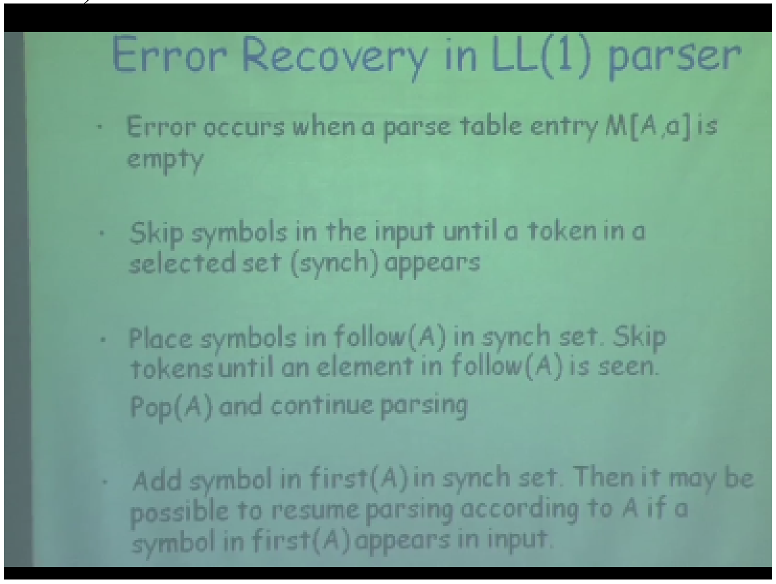


Global corrections

- Considering the program as a whole find a correct "nearby" program
- Nearness may be measured using certain metric
- PL/C compiler implemented this scheme: anything could be compiled!
- It is complicated and not a very good idea!

Where you say that anything can be compiled like PL/C become a pylor where they were saying does not matter what you write they will try to always come by it by guessing your intentions was not very successful.

(Refer Slide Time: 47:09)



Error Recovery in LL(1) parser

- Error occurs when a parse table entry M[A,a] is empty
- Skip symbols in the input until a token in a selected set (synch) appears
- Place symbols in follow(A) in synch set. Skip tokens until an element in follow(A) is seen. Pop(A) and continue parsing
- Add symbol in first(A) in synch set. Then it may be possible to resume parsing according to A if a symbol in first(A) appears in input.

So Arabic ugly the way it works in LS one part that we have just discussed ever such that when you have a particle entry which is empty and so skip symbols in the input until you see a selected set of coconuts which is in same set okay, and what are these tokens I can either place symbols which having follow up in the swing set then skip terminals until you see something in follow of A pop a and continue parsing from there or add symbols which are in first off a in the sing set and then it is possible to resume parsing according. To it if you can write so this is we are going to stop today and the next class we doing to start following task.

Acknowledgment
Ministry of Human Resources & Development
Prof. Phalguni Gupta
Co-ordinator, NPTEL IIT Kanpur
Satyaki Roy
Co Co-ordinator, NPTEL IIT Kanpur
Camera
Ram Chandra
Dilip Tripathi
Padam Shukla
Manoj Shrivastava
Sanjay Mishtra
Editing
Ashish Singh
Badal Pradhan
Tapobrata Das
Shuubham Rawat
Shikha Gupta
Pradeep Kumar
K.K Mishra
Jai Singh
Sweety Kanaujia
Aradhana Singh
Sweta
Preeti Sachan
Ashutosh Gairola
Dilip Katiyar
Ashutosh Kumar
Light& Sound
Sharwan
Hari Ram
Production Crew
Bhadra Rao
Puneet Kumar Bajpai
Priyanka Singh
Office
Lalty Dutta
Ajay Kanaujia
Shivendra Kumar Tiwari
Saurabh Shukla
Direction
Sanjay Pal
Production Manager
Bharat Lal
an IIT Kanpur Production