Top-Down Parsers | Neso Academy

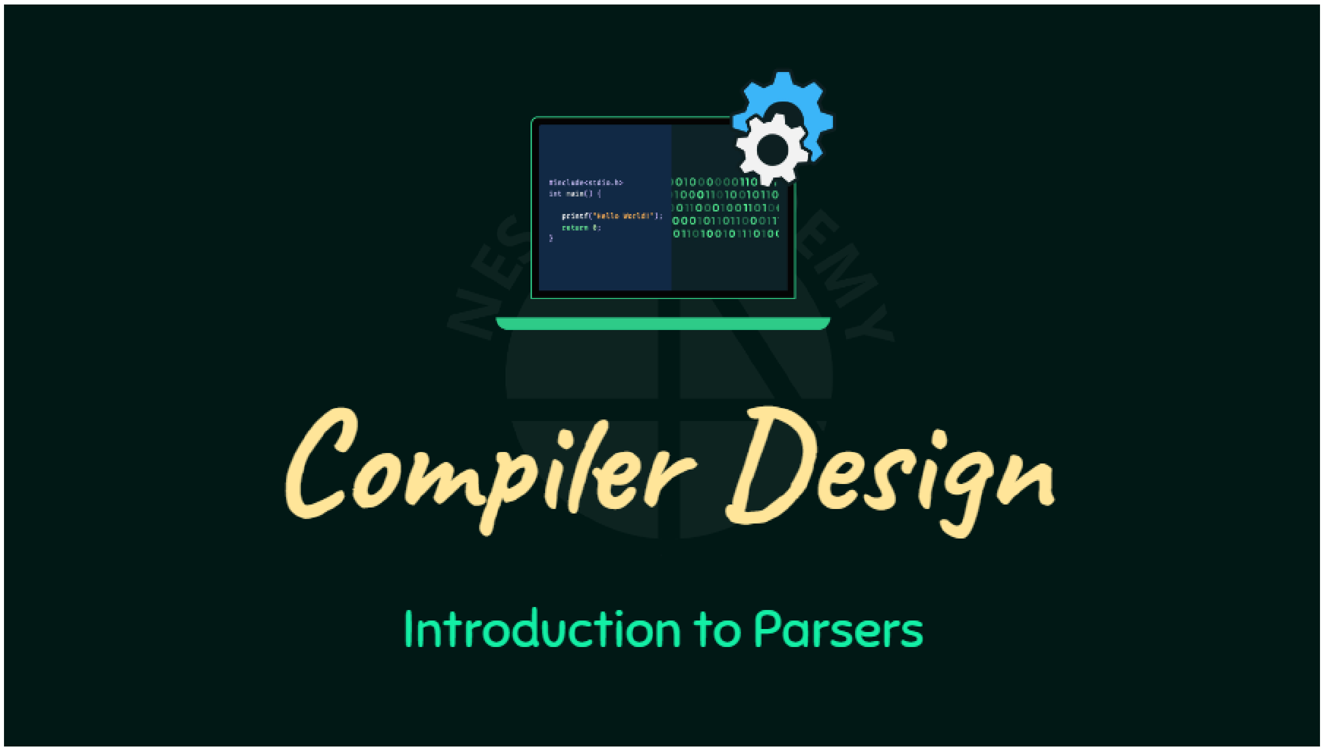
nesoacademy.org/cs/12-compiler-design/ppts/03-top-downparsers

CHAPTER - 3

Top-down Parsers

Neso Academy

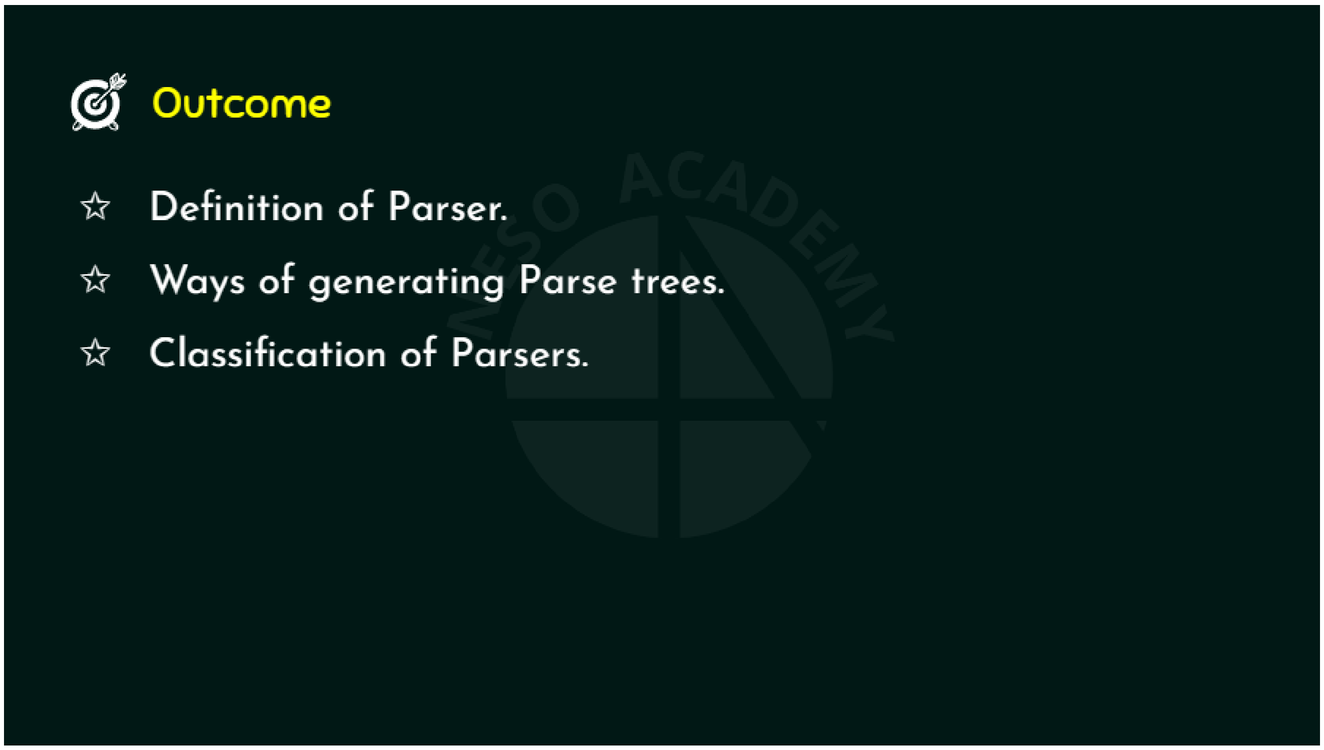
Top-down ParsersNeso AcademyCHAPTER-3



Compiler Design

Introduction to Parsers

Compiler DesignIntroduction to Parsers



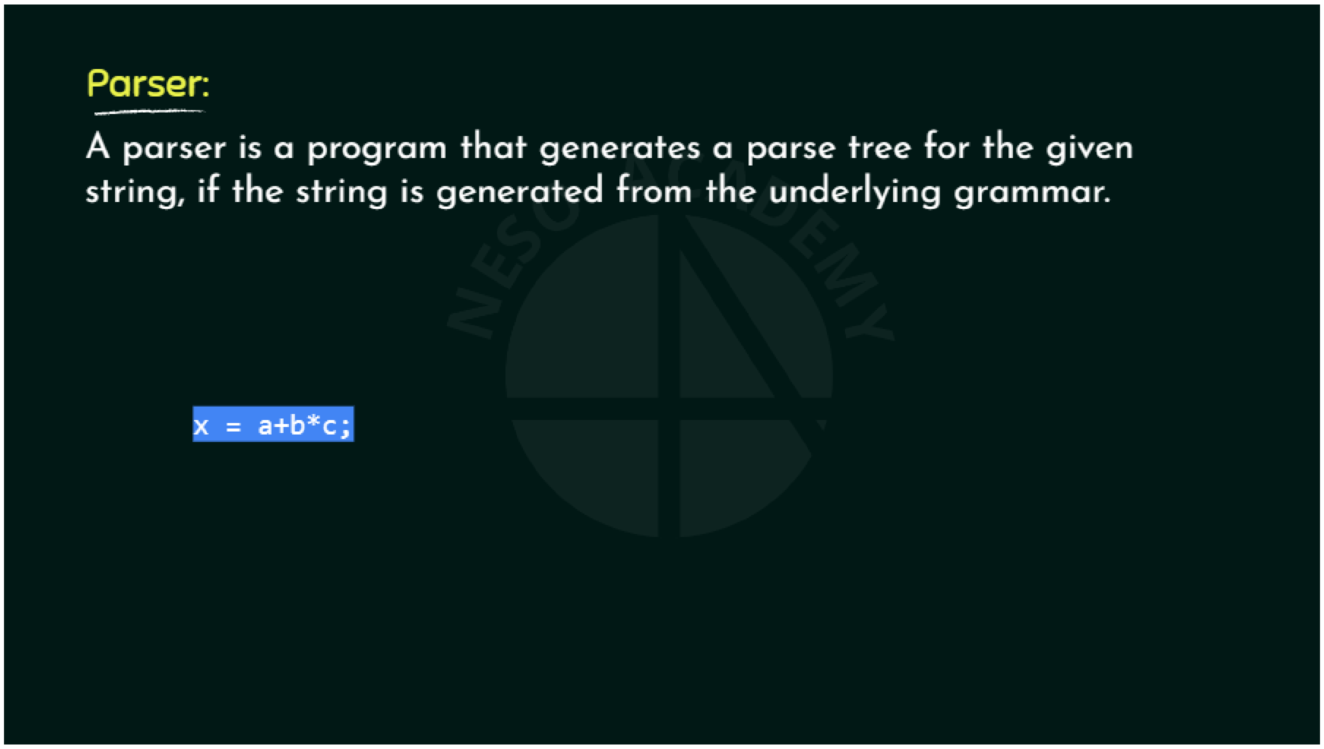
Outcome

☆ Definition of Parser.

☆ Ways of generating Parse trees.

☆ Classification of Parsers.

Outcome☆Definition of Parser. ☆Ways of generating Parse trees. ☆Classification of Parsers.

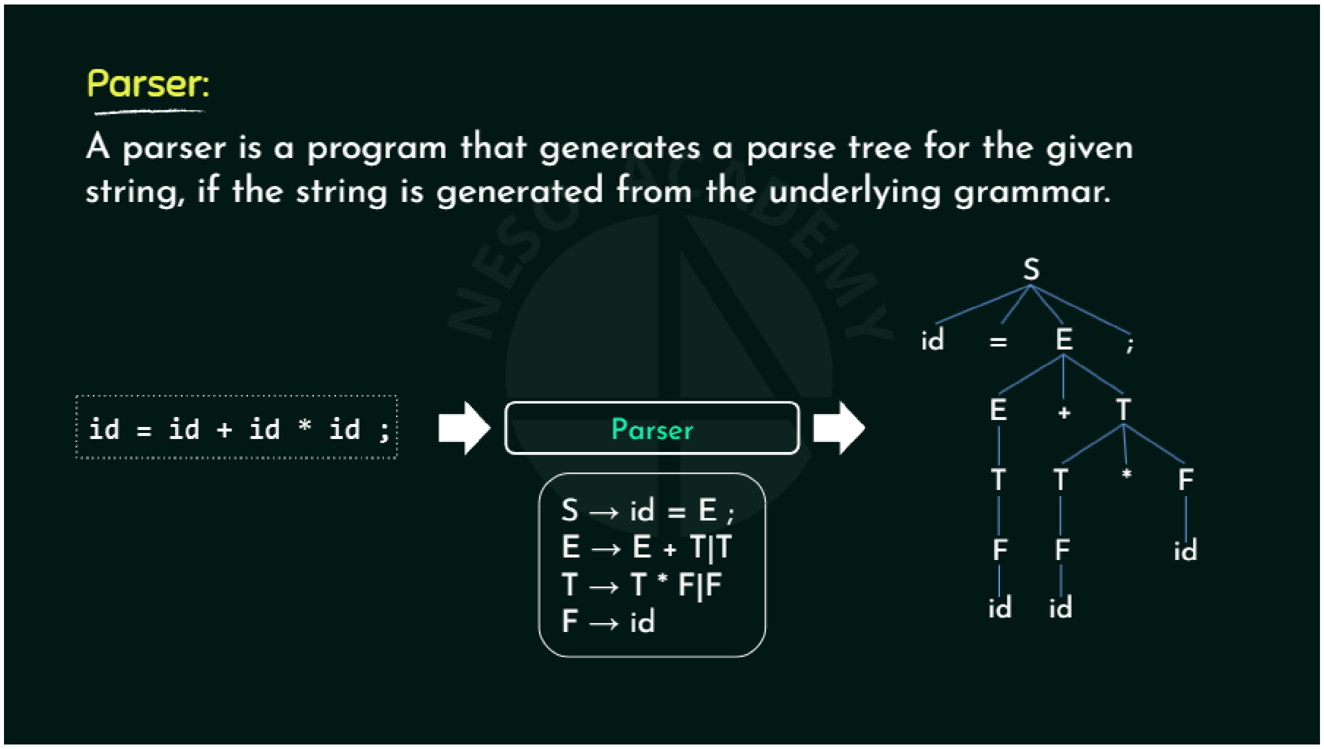


Parser:

A parser is a program that generates a parse tree for the given string, if the string is generated from the underlying grammar.

```
x = a+b*c;
```

Parser:A parser is a program that generates a parse tree for the given string, if the string is generated from the underlying grammar.x = a+b*c;



Parser:

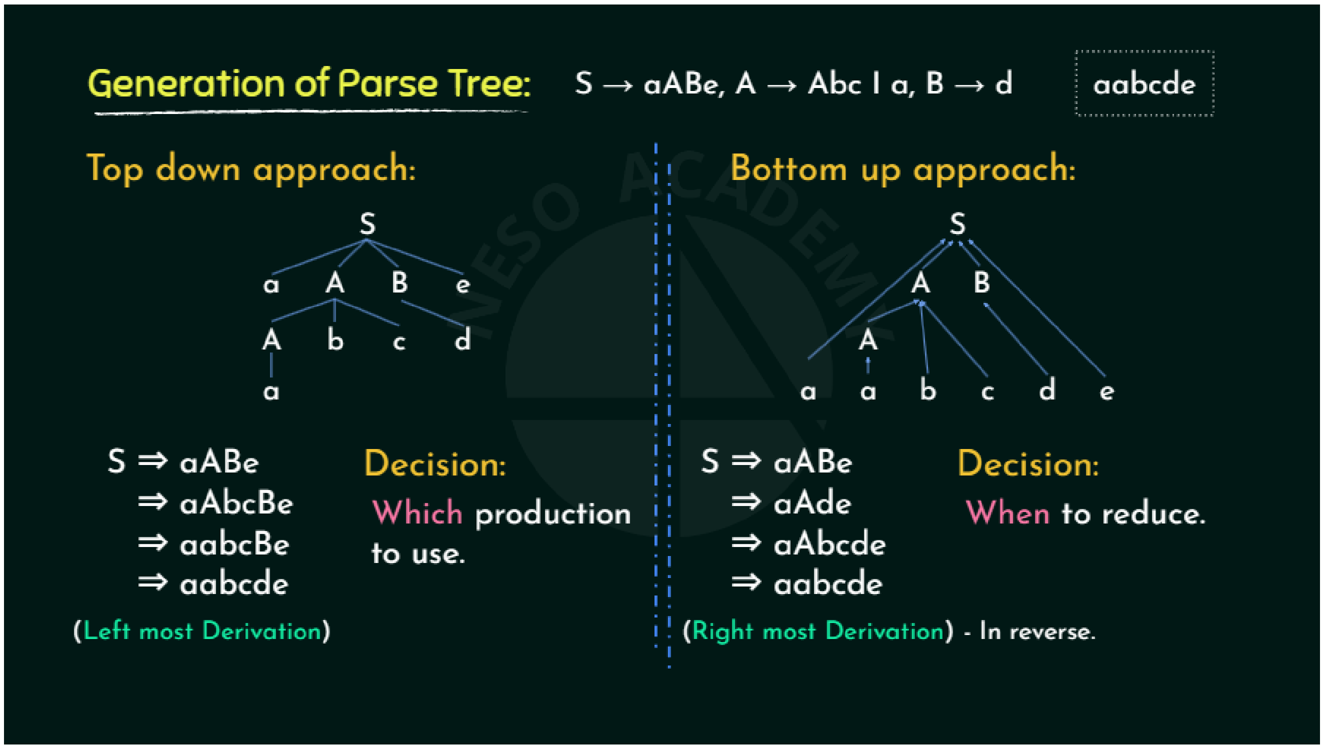
A parser is a program that generates a parse tree for the given string, if the string is generated from the underlying grammar.

```
id = id + id * id ;
```

Parser

S → id = E ;
E → E + T|T
T → T * F|F
F → id

id = id + id * id ;Parser:A parser is a program that generates a parse tree for the given string, if the string is generated from the underlying grammar.ParserSid=E;E+TT*FTFFidididS → id = E ;E → E + T|TT → T * F|FF → id



Generation of Parse Tree:

S → aABe, A → Abc I a, B → d

aabcde

Top down approach:

S ⇒ aABe
⇒ aAbcBe
⇒ aabcBe
⇒ aabcde

(Left most Derivation)

Decision: Which production to use.

Bottom up approach:

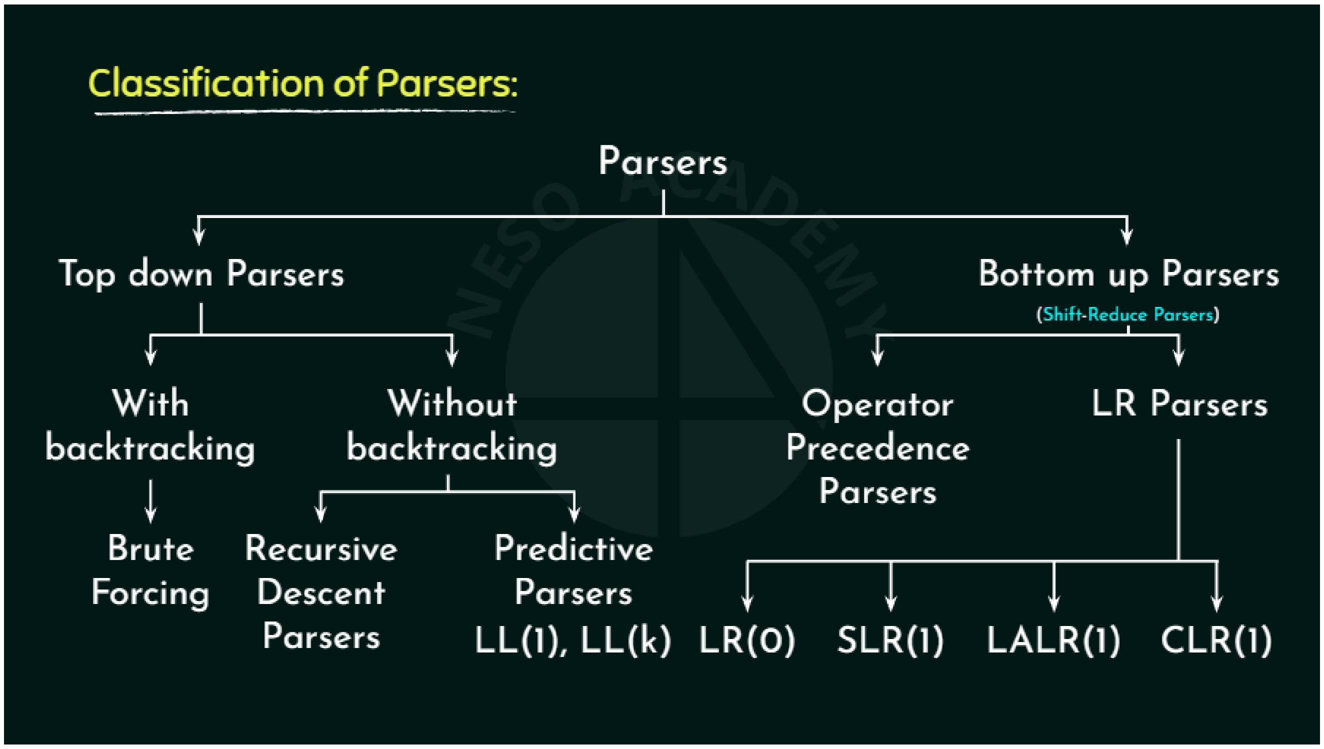
S ⇒ aABe
⇒ aAde
⇒ aAbcde
⇒ aabcde

(Right most Derivation) - In reverse.

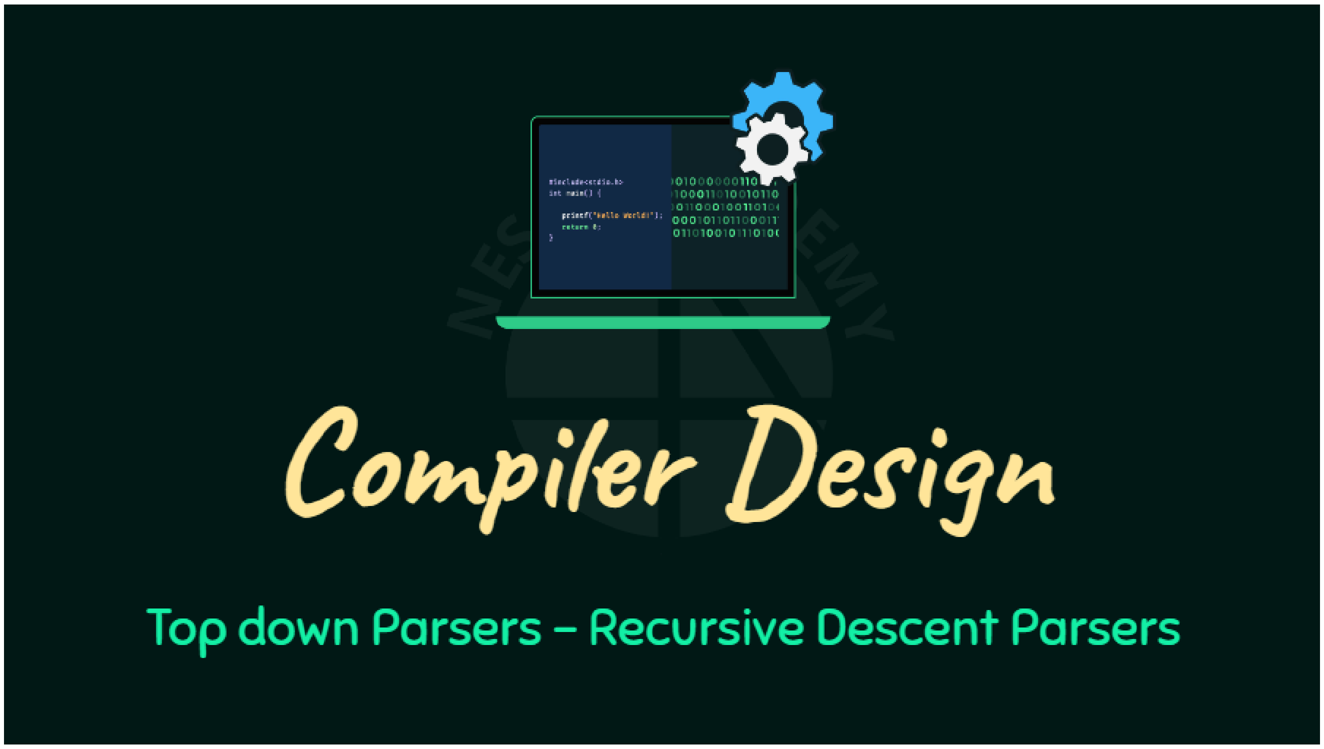
Decision: When to reduce.

Generation of Parse Tree:Top down approach:Bottom up approach:aabcdeSaABeAbcadaabcdeAABSDecision:Which production to use.Decision:S ⇒ aABe⇒ aAde⇒ aAbcde⇒ aabcdeWhen to reduce.S ⇒ aABe⇒ aAbcBe⇒ aabcBe⇒

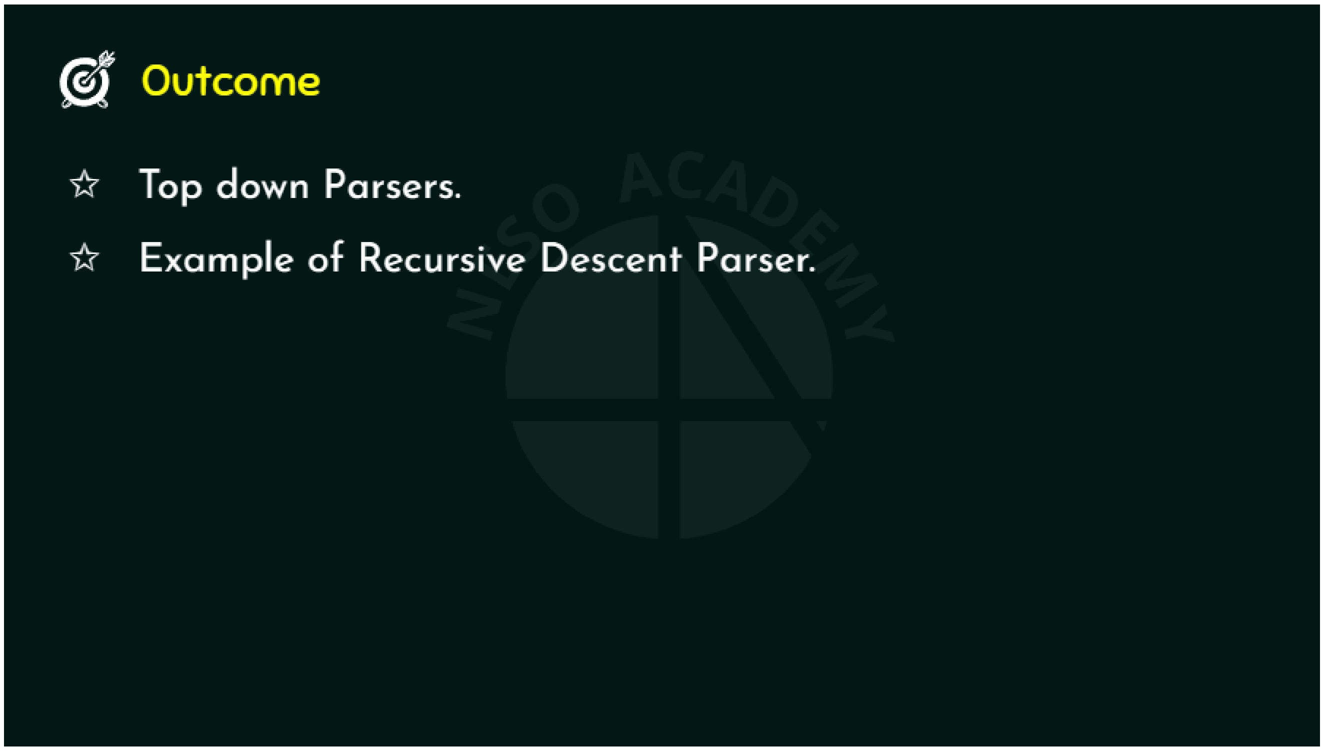
aabcde(Left most Derivation)(Right most Derivation)- In reverse.S → aABe, A → Abc I a, B → d



Classification of Parsers:ParsersTop down ParsersBottom up Parsers(Shift-Reduce Parsers)With backtrackingWithout backtrackingBrute ForcingLR ParsersOperator Precedence ParsersLR(0)CLR(1)LALR(1)SLR(1)Recursive Descent ParsersPredictive ParsersLL(1), LL(k)



Compiler DesignTop down Parsers - Recursive Descent Parsers

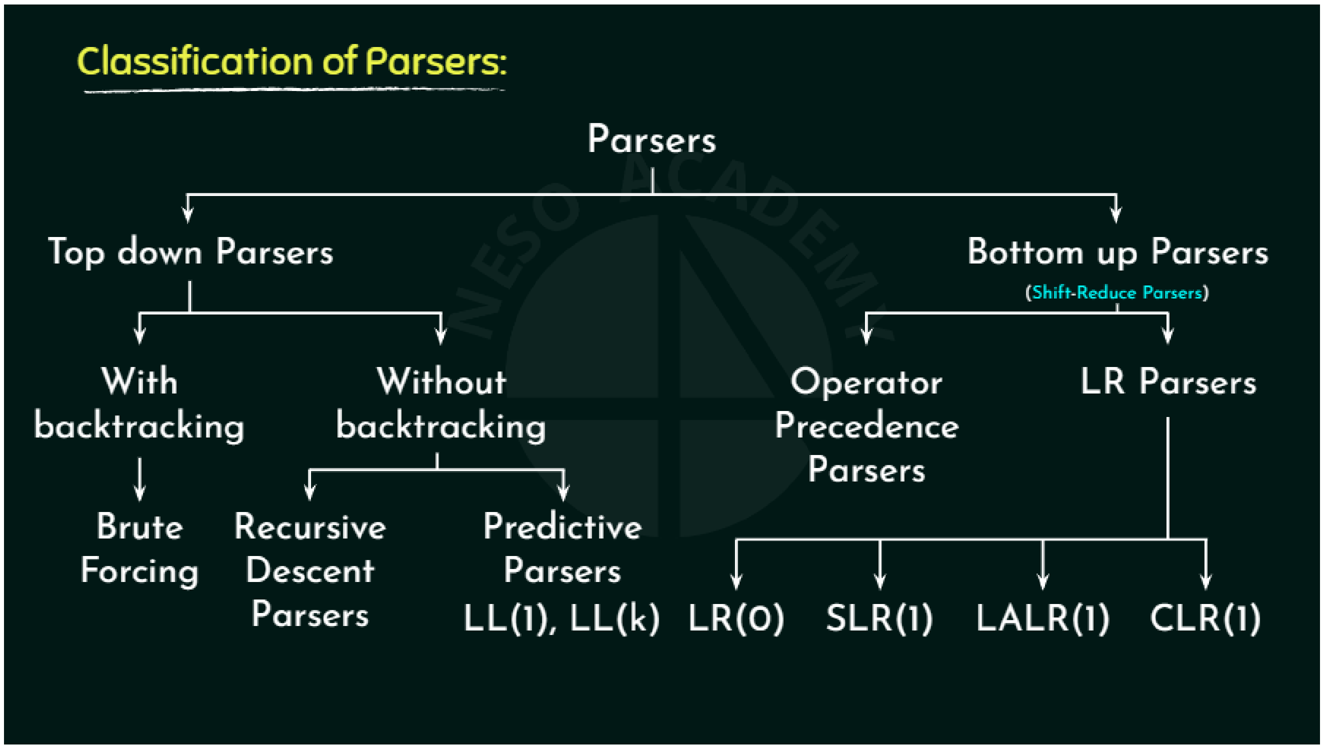


Outcome

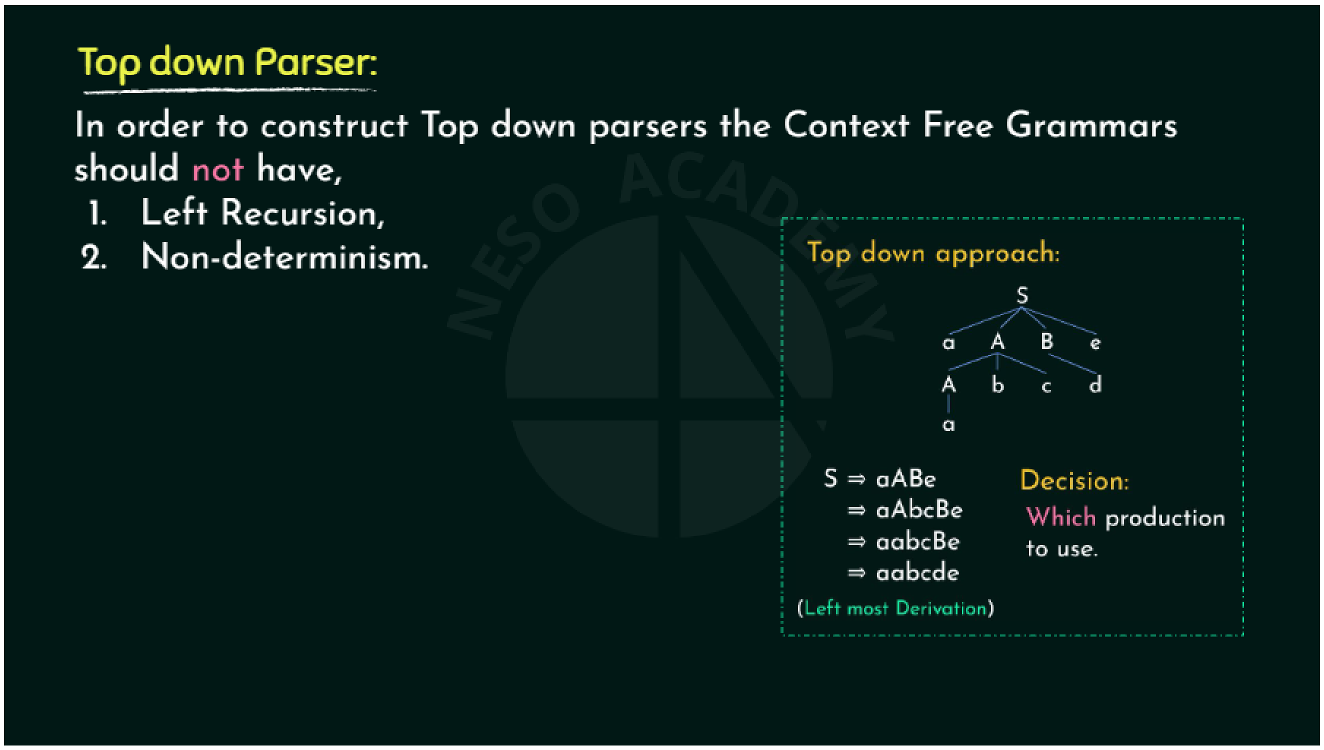
☆ Top down Parsers.

☆ Example of Recursive Descent Parser.

Outcome☆Top down Parsers. ☆Example of Recursive Descent Parser.



Classification of Parsers:ParsersTop down ParsersBottom up Parsers(Shift-Reduce Parsers)With backtrackingWithout backtrackingBrute ForcingLR ParsersOperator Precedence ParsersLR(0)CLR(1)LALR(1)SLR(1)Recursive Descent ParsersPredictive ParsersLL(1), LL(k)



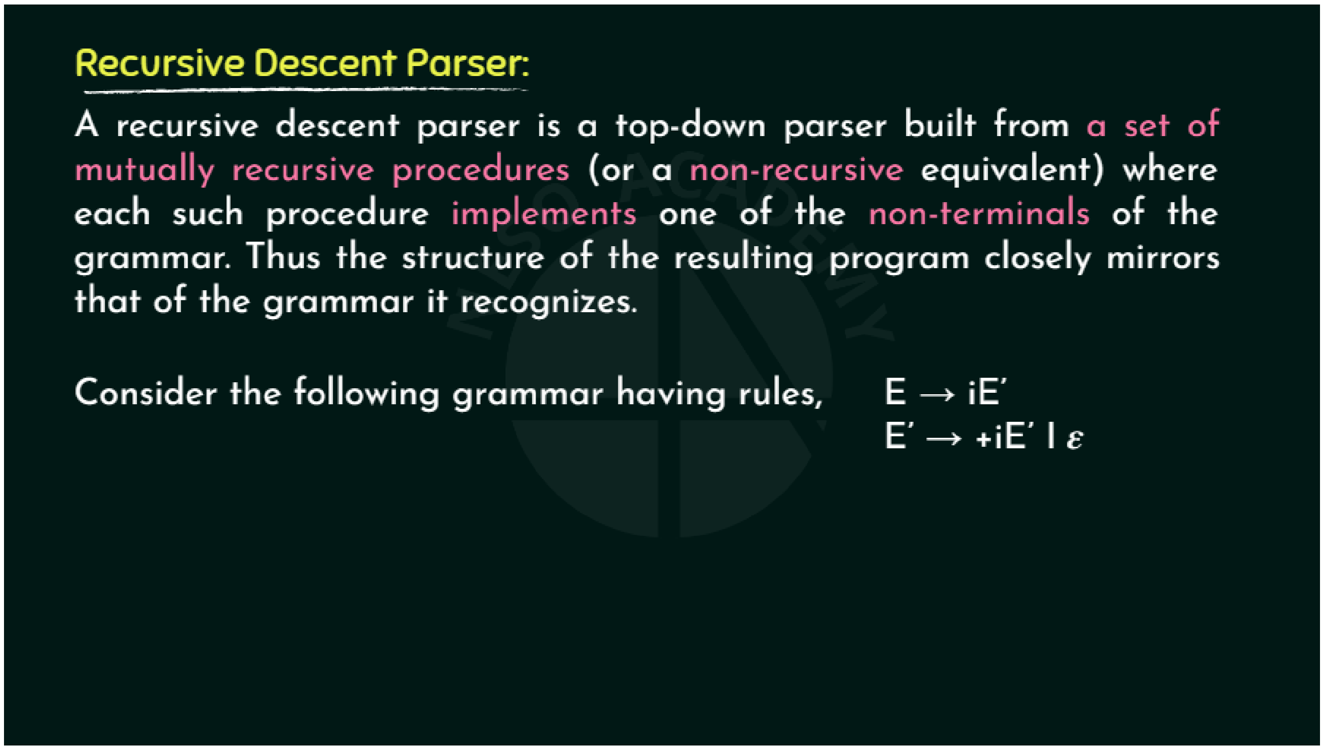
Top down Parser:

In order to construct Top down parsers the Context Free Grammars should not have,

1. Left Recursion,
2. Non-determinism.

Top down approach:
S
a A B e
A b c d
a
S ⇒ aABe
⇒ aAbcBe
⇒ aabcBe
⇒ aabcde
(Left most Derivation)
Decision:
Which production to use.

Top down Parser:In order to construct Top down parsers the Context Free Grammars should not have,1.Left Recursion,2.Non-determinism.



Recursive Descent Parser:

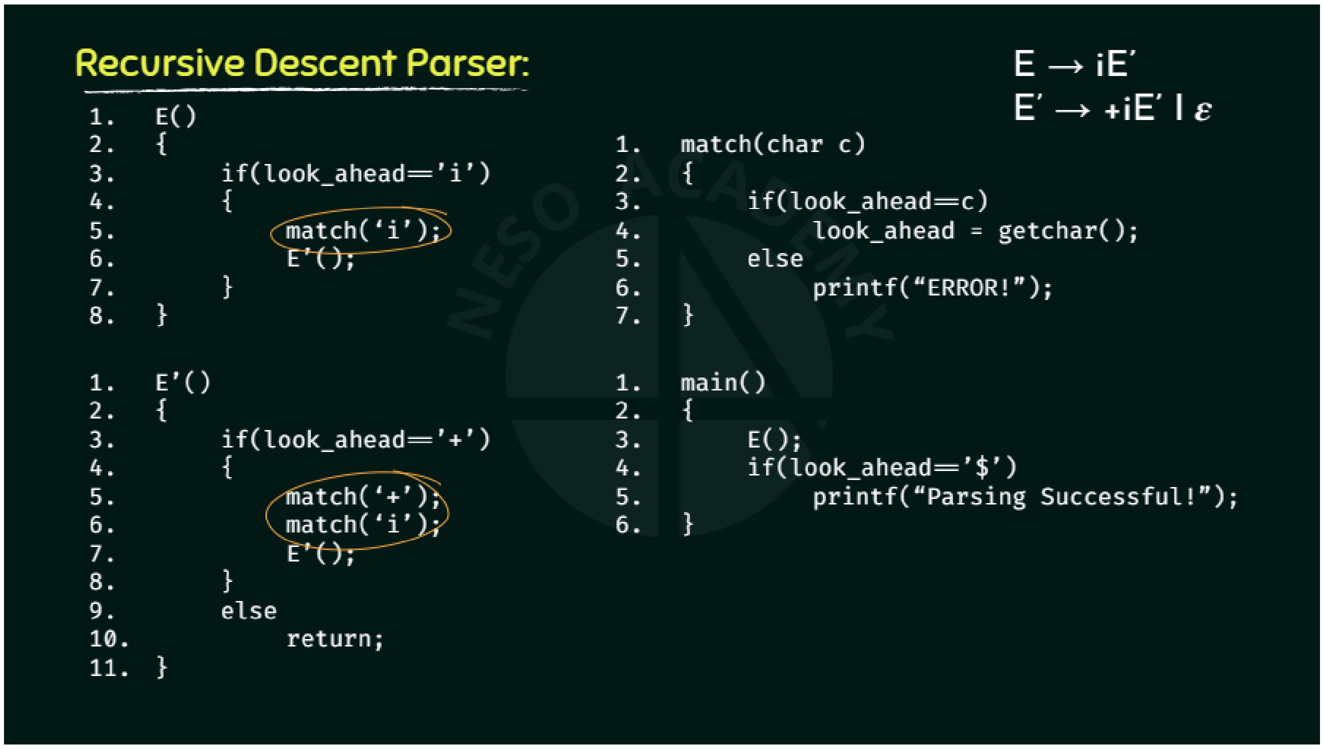
A recursive descent parser is a top-down parser built from a set of mutually recursive procedures (or a non-recursive equivalent) where each such procedure implements one of the non-terminals of the grammar. Thus the structure of the resulting program closely mirrors that of the grammar it recognizes.

Consider the following grammar having rules,

$E \rightarrow iE'$

$E' \rightarrow +iE' \mid \varepsilon$

A recursive descent parser is a top-down parser built from a set of mutually recursive procedures (or a non-recursive equivalent) where each such procedure implements one of the non-terminals of the grammar. Thus the structure of the resulting program closely mirrors that of the grammar it recognizes.Recursive Descent Parser:Consider the following grammar having rules,E → iE’E’ → +iE’ I ε



Recursive Descent Parser:

$E \rightarrow iE'$
$E' \rightarrow +iE' \mid \varepsilon$

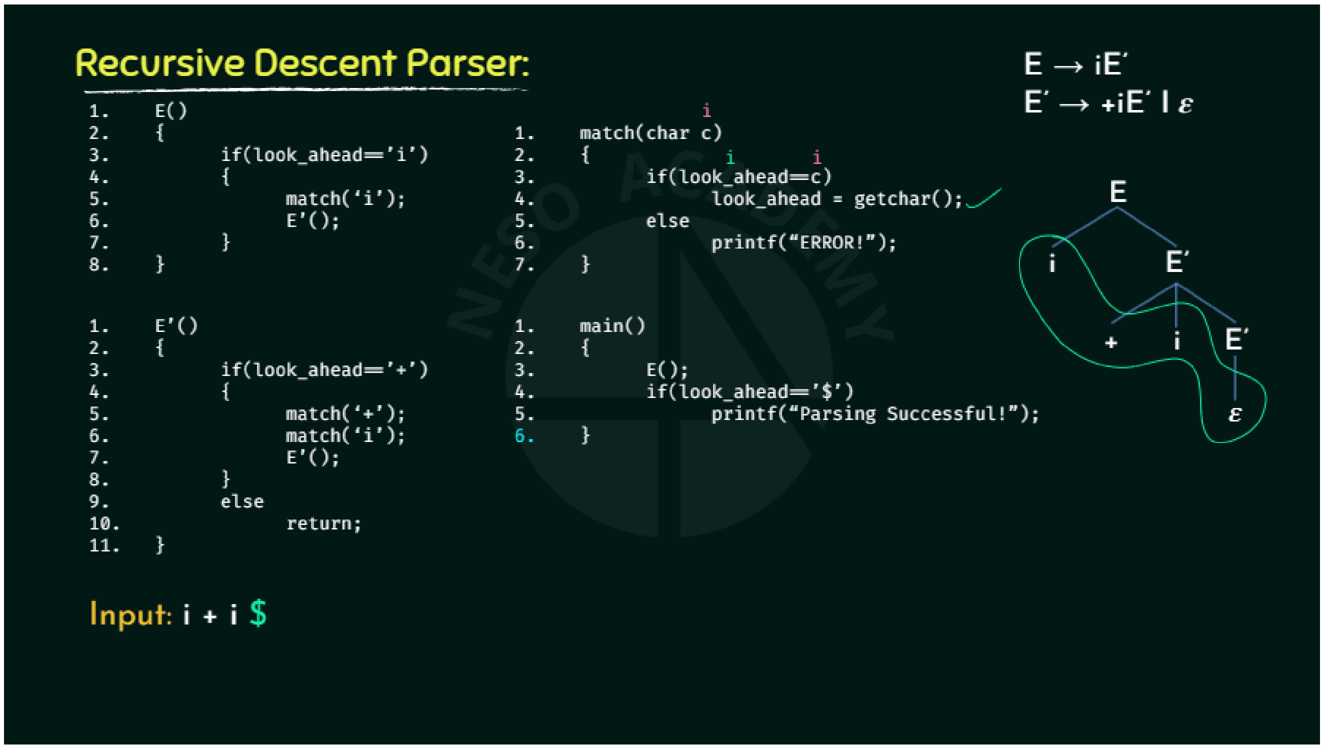
```
E()
{
      if(look_ahead=='i')
      {
            match('i');
            E'();
      }
}
```

```
E'()
{
      if(look_ahead=='+')
      {
            match('+');
            match('i');
            E'();
      }
      else
            return;
}
```

```
match(char c)
{
      if(look_ahead==c)
            look_ahead = getchar();
      else
            printf("ERROR!");
}
```

```
main()
{
      E();
      if(look_ahead=='$')
            printf("Parsing Successful!");
}
```

Recursive Descent Parser:E → iE’E’ → +iE’ I ε1.E()2.{3.if(look_ahead==’i’)4.{5.match(‘i’);6.E’();7.}8.}1.E’()2.{3.if(look_ahead==’+’)4.{5.match(‘+’);6.match(‘i’);7.E’();8.}9.else10.return;11.}1.match(char c)2.{3.if(look_ahead==c)4.look_ahead = getchar();5.else6.printf(“ERROR!”);7.}1.main()2.{3.E();4.if(look_ahead==’$’)5.printf(“Parsing Successful!”);6.}



Recursive Descent Parser:

```
E()
{
      if(look_ahead=='i')
      {
            match('i');
            E'();
      }
}

E'()
{
      if(look_ahead=='+')
      {
            match('+');
            match('i');
            E'();
      }
      else
            return;
}
```

```
match(char c)
{
      if(look_ahead==c)
            look_ahead = getchar();
      else
            printf("ERROR!");
}

main()
{
      E();
      if(look_ahead=='$')
            printf("Parsing Successful!");
}
```

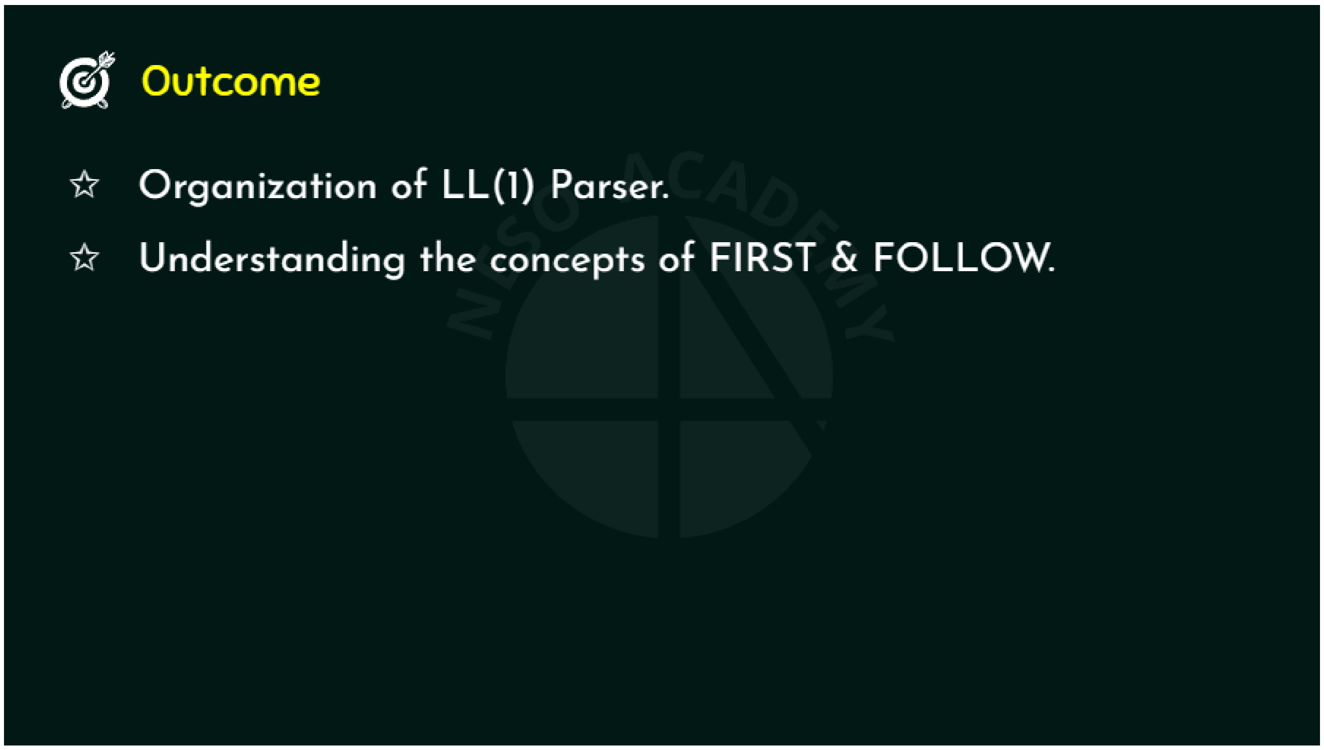
$E \rightarrow iE'$

$E' \rightarrow +iE' \mid \varepsilon$

Input: i + i $

Recursive Descent Parser:1.E()2.{3.if(look_ahead=='i')4.{5.match('i');6.E'();7.}8.}1.E'()2.{3.if(look_ahead=='+')4.{5.match('+');6.match('i');7.E'();8.}9.else10.return;11.}1.match(char c)2.{3.if(look_ahead==c)4.look_ahead = getchar();5.else6.printf("ERROR!");7.}1.main()2.{3.E();4.if(look_ahead=='$')5.printf("Parsing Successful!");6.}E → iE'E' → +iE' I εInput: i + i $EiE'+iiiE'ε

Compiler DesignTop down Parsers - LL(1) Parsers

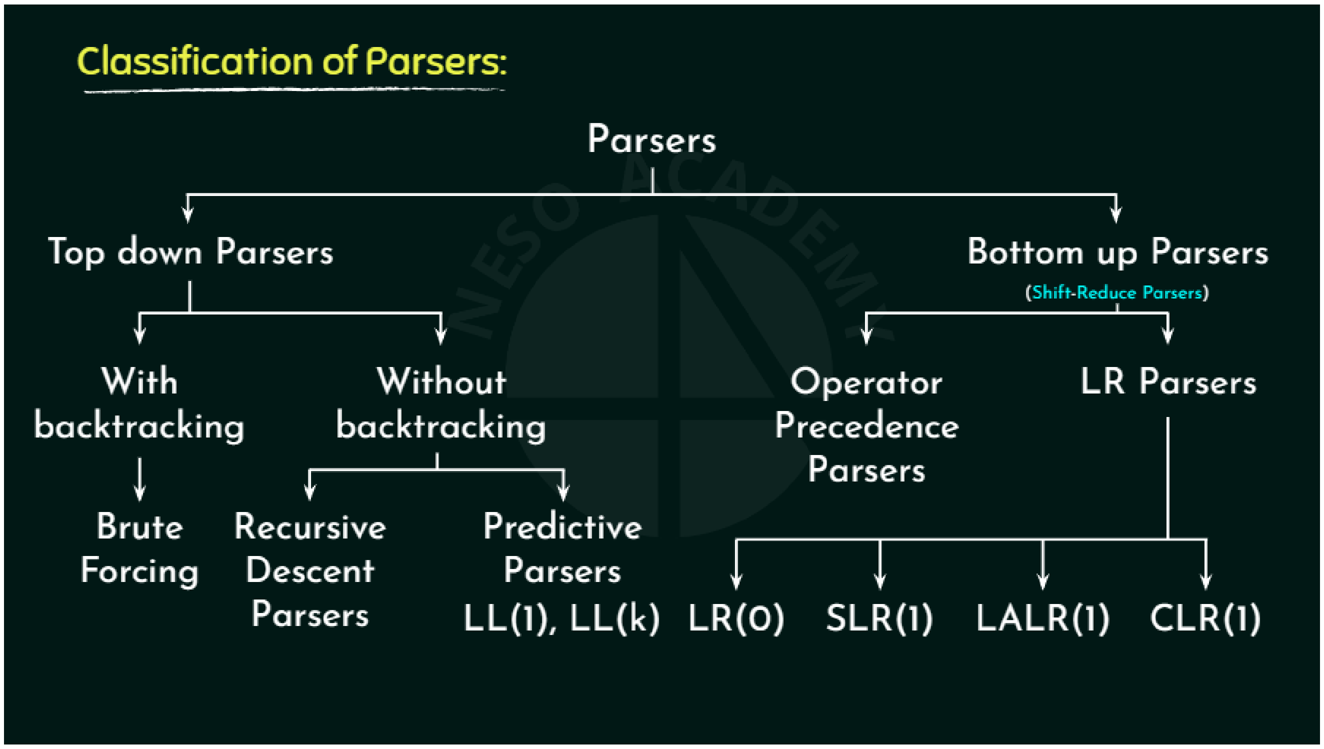


Outcome

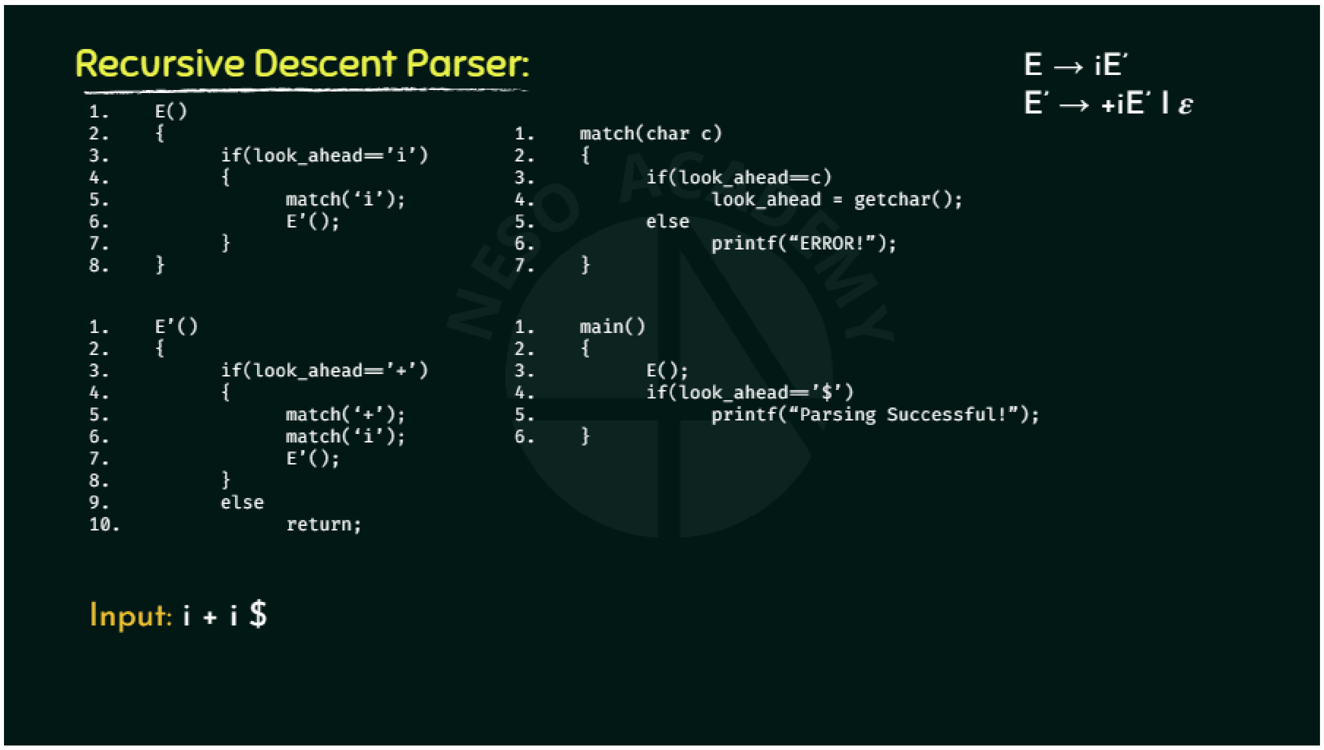
☆ Organization of LL(1) Parser.

☆ Understanding the concepts of FIRST & FOLLOW.

Outcome☆Organization of LL(1) Parser. ☆Understanding the concepts of FIRST & FOLLOW.



Classification of Parsers:ParsersTop down ParsersBottom up Parsers(Shift-Reduce Parsers)With backtrackingWithout backtrackingBrute ForcingLR ParsersOperator Precedence ParsersLR(0)CLR(1)LALR(1)SLR(1)Recursive Descent ParsersPredictive ParsersLL(1), LL(k)



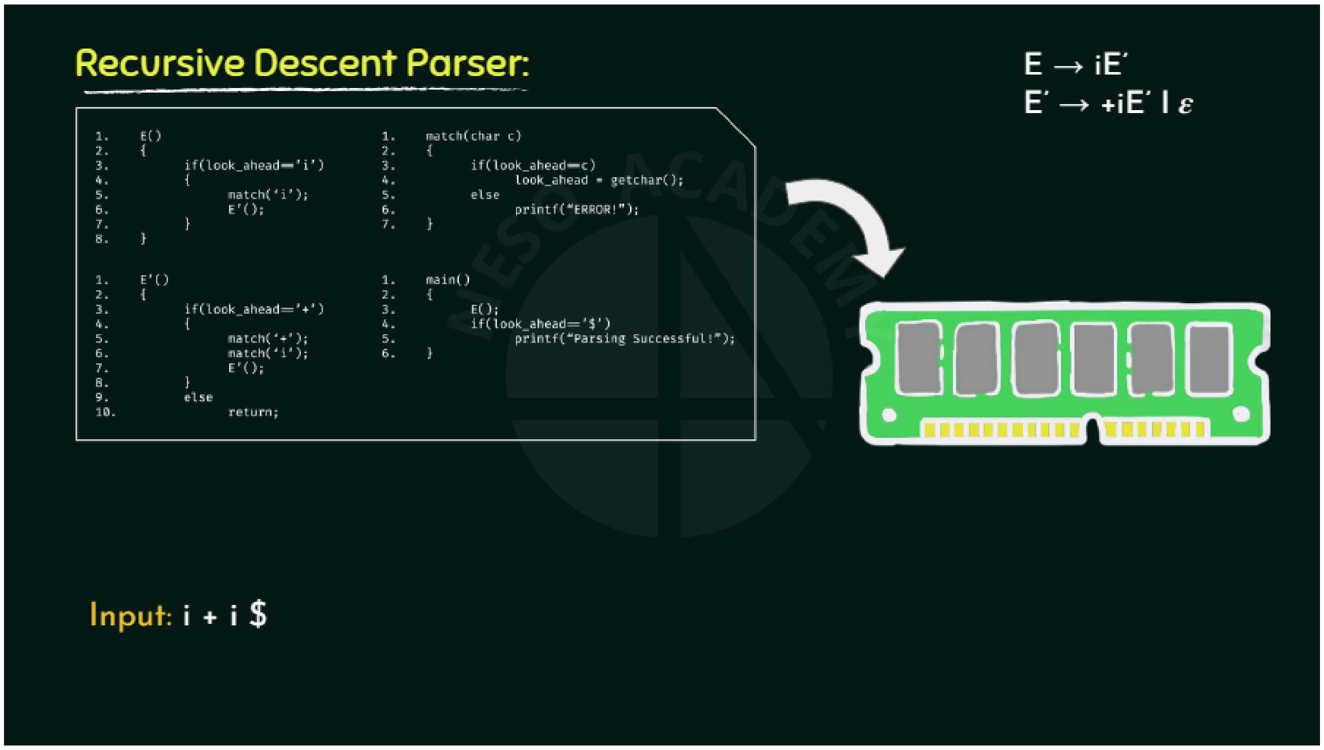
Recursive Descent Parser:

$E \rightarrow iE'$
$E' \rightarrow +iE' \mid \varepsilon$

```
E()
{
        if(look_ahead=='i')
        {
                match('i');
                E'();
        }
}

E'()
{
        if(look_ahead=='+')
        {
                match('+');
                match('i');
                E'();
        }
        else
                return;

match(char c)
{
        if(look_ahead==c)
                look_ahead = getchar();
        else
                printf("ERROR!");
}

main()
{
        E();
        if(look_ahead=='$')
                printf("Parsing Successful!");
}
```

Input: i + i $

Recursive Descent Parser:1.E()2.{3.if(look_ahead==’i’)4.{5.match(‘i’);6.E’();7.}8.}1.E’()2.{3.if(look_ahead==’+’)4.{5.match(‘+’);6.match(‘i’);7.E’();8.}9.else10.return;1.match(char c)2.{3.if(look_ahead==c)4.look_ahead = getchar();5.else6.printf(“ERROR!”);7.}1.main()2.{3.E();4.if(look_ahead==’$’)5.printf(“Parsing Successful!”);6.}E → iE’E’ → +iE’ I 𝜀Input: i + i $

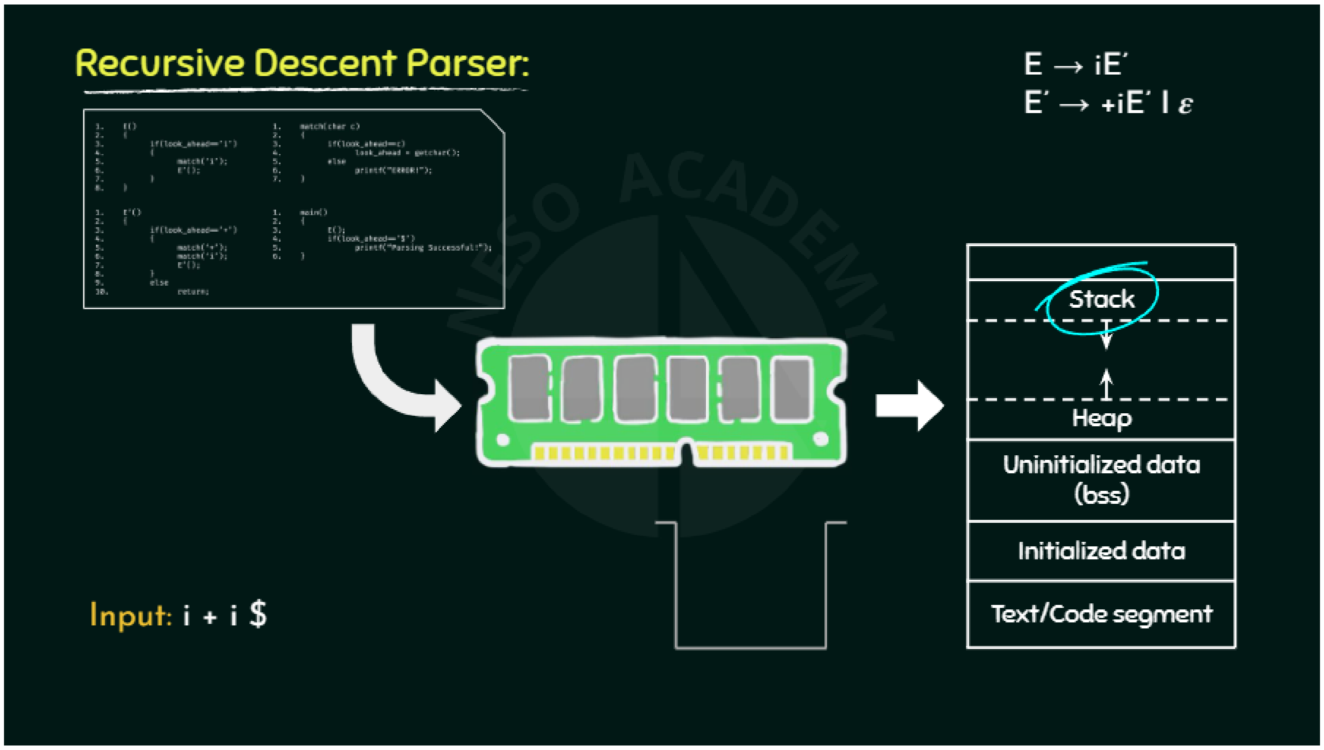


Recursive Descent Parser:

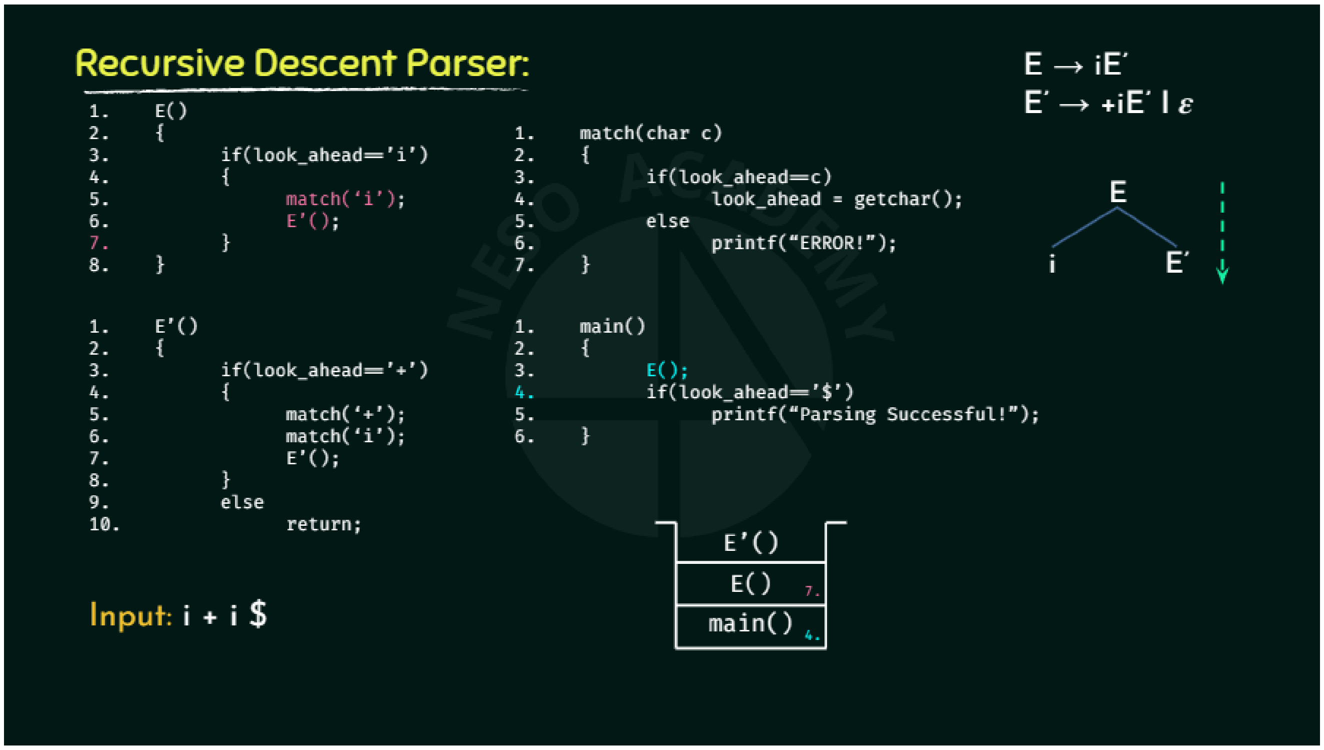
$E \rightarrow iE'$
$E' \rightarrow +iE' \mid \varepsilon$

Input: i + i $

Recursive Descent Parser:E → iE’E’ → +iE’ I 𝜀Input: i + i $



Recursive Descent Parser:E → iE'E' → +iE' I εInput: i + i $StackHeapUninitialized data(bss)Initialized dataText/Code segment



Recursive Descent Parser:

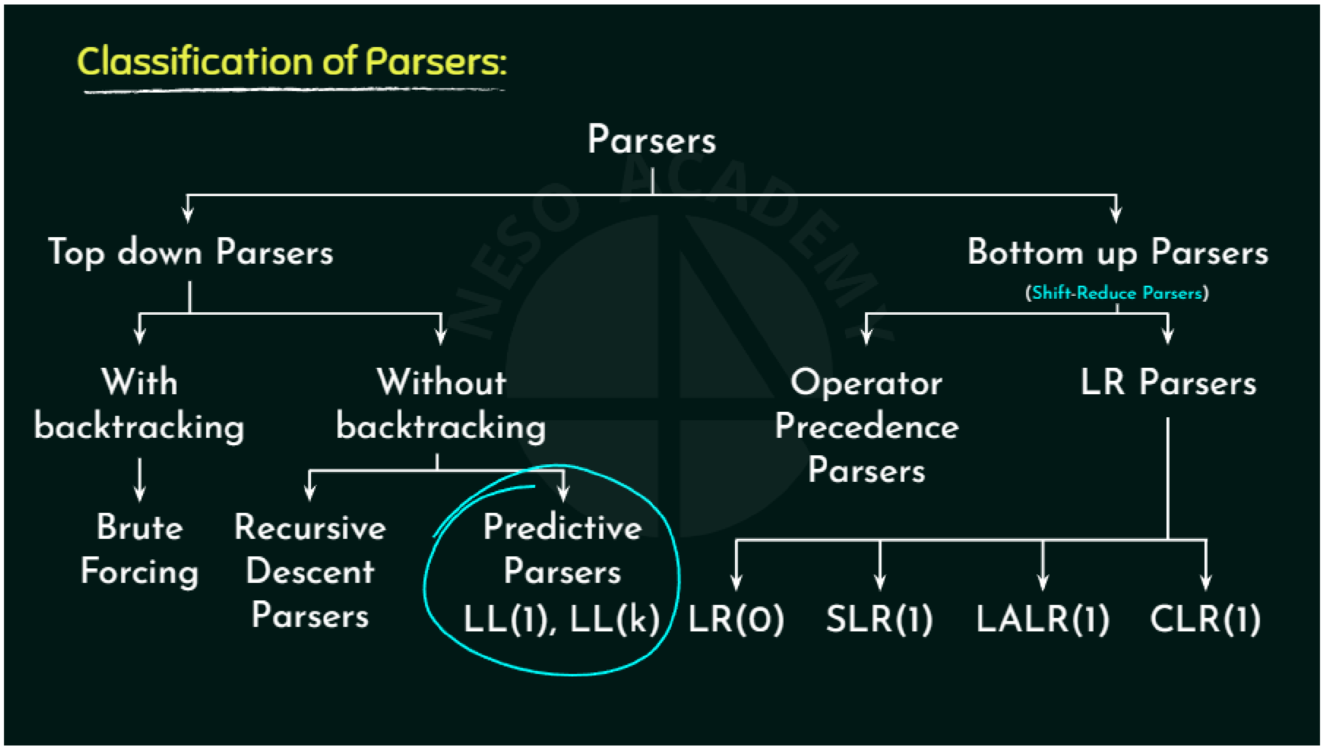
$E \rightarrow iE'$
$E' \rightarrow +iE' \mid \varepsilon$

```
1.    E()
2.    {
3.          if(look_ahead=='i')
4.          {
5.                match('i');
6.                E'();
7.          }
8.    }

1.    E'()
2.    {
3.          if(look_ahead=='+')
4.          {
5.                match('+');
6.                match('i');
7.                E'();
8.          }
9.          else
10.               return;

1.    match(char c)
2.    {
3.          if(look_ahead==c)
4.                look_ahead = getchar();
5.          else
6.                printf("ERROR!");
7.    }

1.    main()
2.    {
3.          E();
4.          if(look_ahead=='$')
5.                printf("Parsing Successful!");
6.    }
```

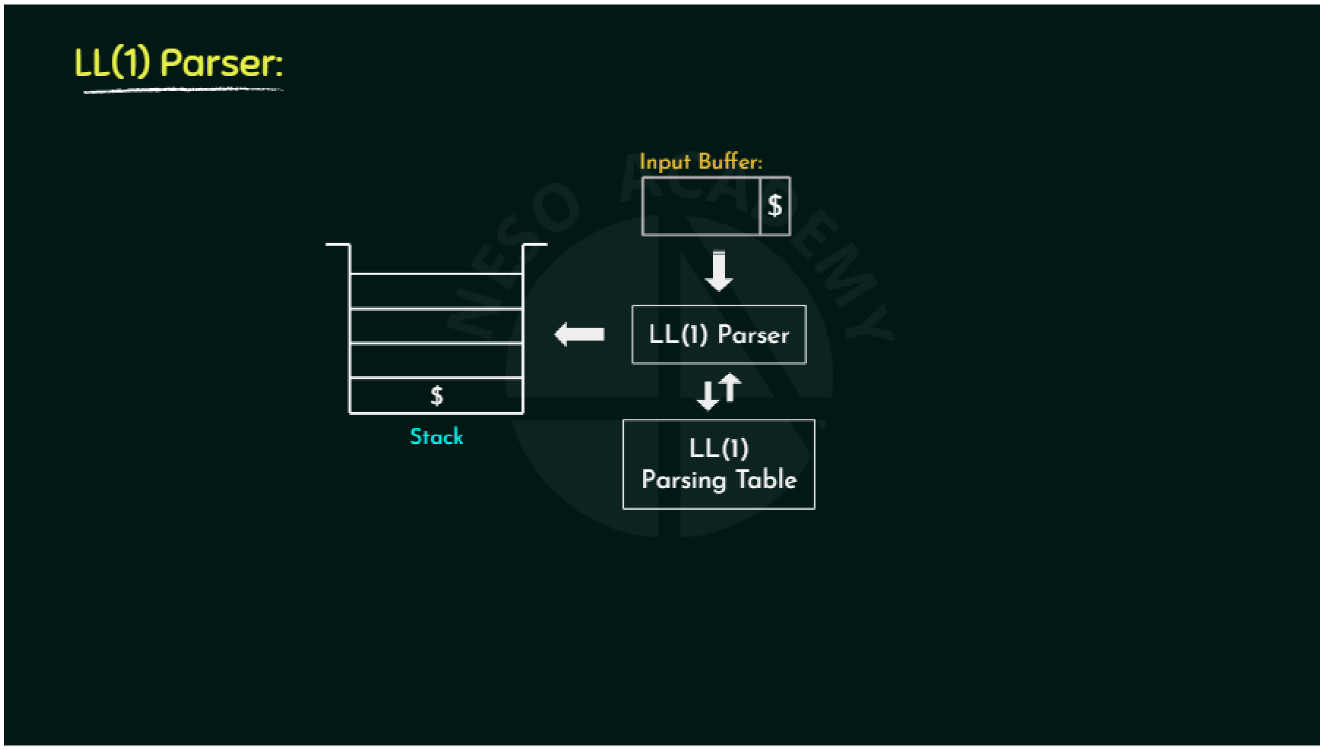
Input: i + i $

Recursive Descent Parser:1.E()2.{3.if(look_ahead=='i')4.{5.match('i');6.E'();7.}8.}1.E'()2.{3.if(look_ahead=='+')4.{5.match('+');6.match('i');7.E'();8.}9.else10.return;1.match(char c)2.{3.if(look_ahead==c)4.look_ahead = getchar();5.else6.printf("ERROR!");7.}1.main()2.

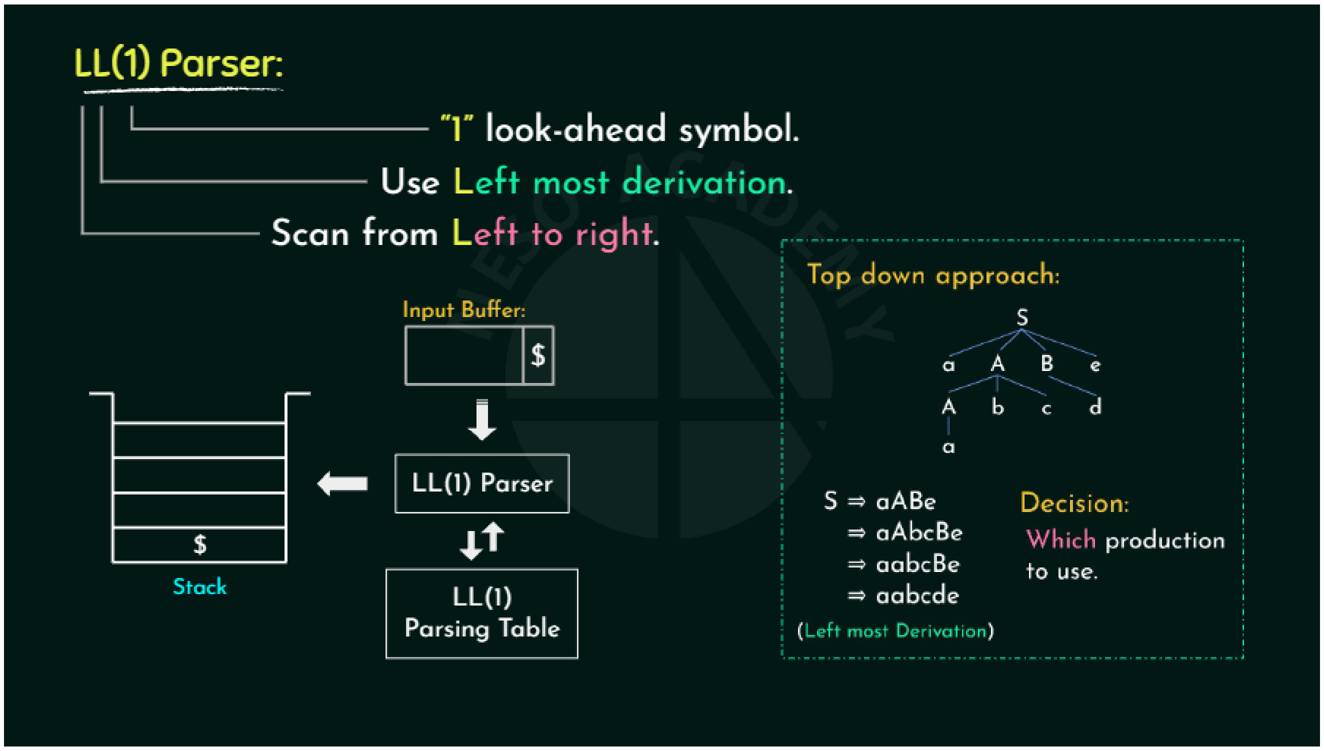
{3.E();4.if(look_ahead=='$')5.printf("Parsing Successful!");6.}E → iE'E' → +iE' I εInput: i + i $EiE'main()E()4.7.E'()



Classification of Parsers:ParsersTop down ParsersBottom up Parsers(Shift-Reduce Parsers)With backtrackingWithout backtrackingBrute ForcingLR ParsersOperator Precedence ParsersLR(0)CLR(1)LALR(1)SLR(1)Recursive Descent ParsersPredictive ParsersLL(1), LL(k)



LL(1) Parser:$LL(1) ParserLL(1) Parsing TableStack$Input Buffer:



LL(1) Parser:

- "1" look-ahead symbol.
- Use Left most derivation.
- Scan from Left to right.

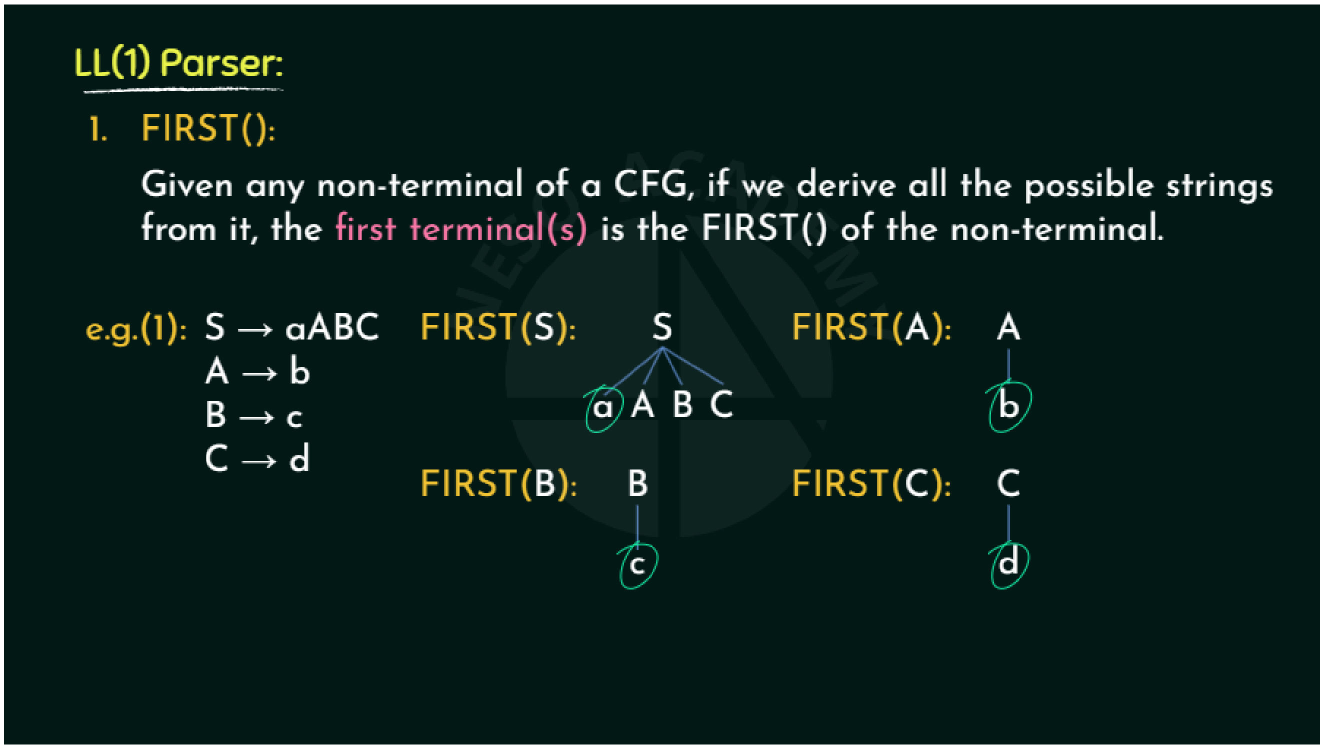
Top down approach:

$$\begin{aligned} S &\Rightarrow aABe \\ &\Rightarrow aAbcBe \\ &\Rightarrow aabcBe \\ &\Rightarrow aabcde \end{aligned}$$

(Left most Derivation)

Decision: Which production to use.

LL(1) Parser:$$LL(1) ParserLL(1) Parsing TableInput Buffer:StackScan from Left to right.Use Left most derivation.“1” look-ahead symbol.



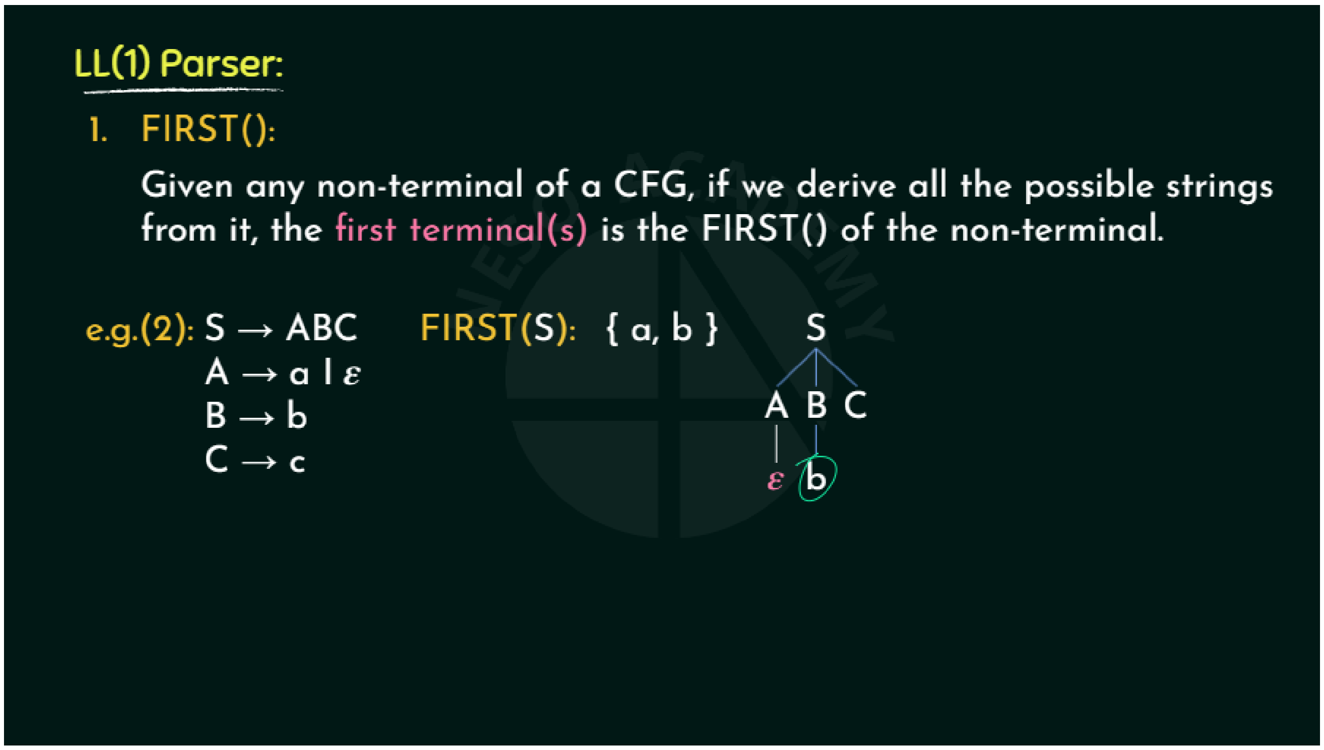
LL(1) Parser:

1. FIRST():

Given any non-terminal of a CFG, if we derive all the possible strings from it, the first terminal(s) is the FIRST() of the non-terminal.

e.g.(1): $S \rightarrow aABC$
$A \rightarrow b$
$B \rightarrow c$
$C \rightarrow d$

LL(1) Parser:1.FIRST():Given any non-terminal of a CFG, if we derive all the possible strings from it, the first terminal(s) is the FIRST() of the non-terminal.S → aABCA → bB → cC → de.g.(1):FIRST(S): SaABCFIRST(A): AbFIRST(B): BcFIRST(C): Cd



LL(1) Parser:

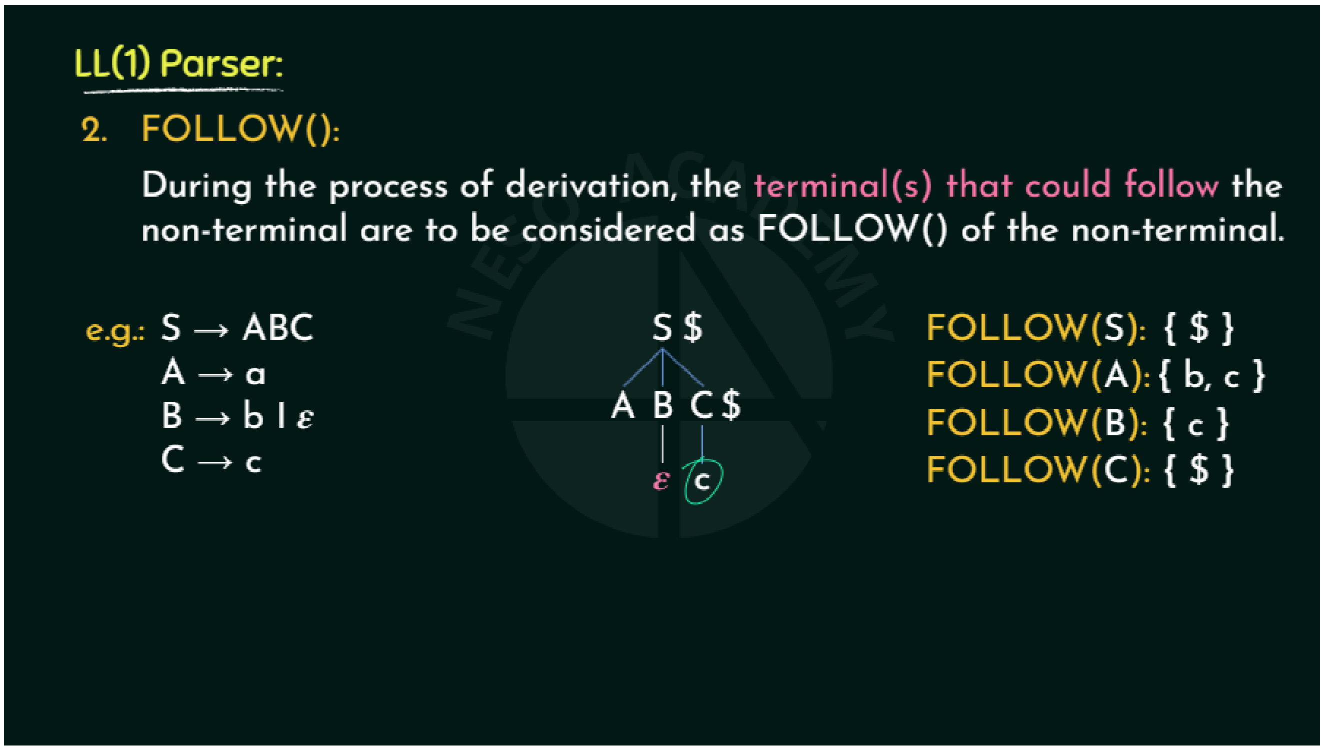
1. FIRST():

   Given any non-terminal of a CFG, if we derive all the possible strings from it, the first terminal(s) is the FIRST() of the non-terminal.

e.g.(2): S → ABC
A → a I ε
B → b
C → c

FIRST(S): { a, b }

LL(1) Parser:1.FIRST():Given any non-terminal of a CFG, if we derive all the possible strings from it, the first terminal(s) is the FIRST() of the non-terminal.S → ABCA → a I εB → bC → ce.g.(2):FIRST(S): SεABC{ a, b }b



LL(1) Parser:

2. FOLLOW():

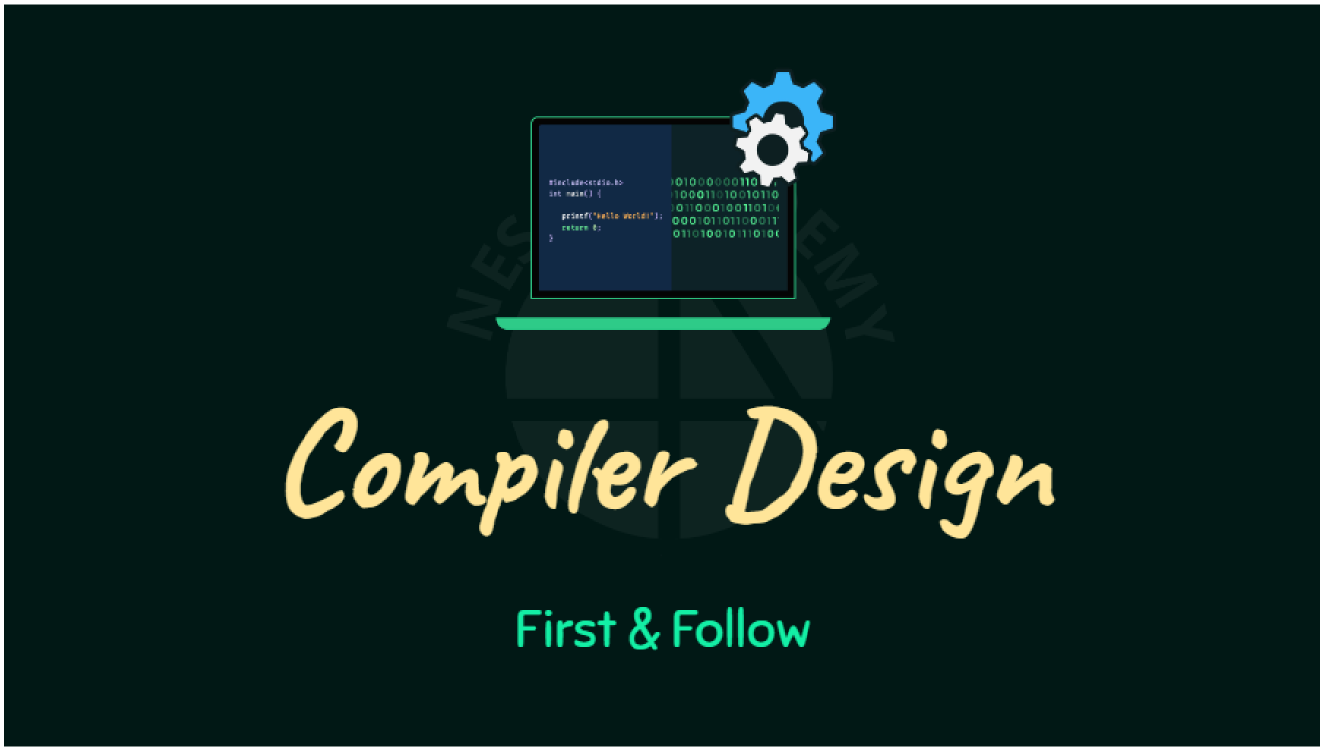
   During the process of derivation, the terminal(s) that could follow the non-terminal are to be considered as FOLLOW() of the non-terminal.

e.g.: S → ABC
A → a
B → b I ε
C → c

FOLLOW(S): { $ }
FOLLOW(A): { b, c }
FOLLOW(B): { c }
FOLLOW(C): { $ }

cLL(1) Parser:2.FOLLOW():During the process of derivation, the terminal(s) that could follow the non-terminal are to be considered as FOLLOW() of the non-terminal.S → ABCA → a B → b I εC → ce.g.:FOLLOW(S): SABCε{ $ }$$FOLLOW(A): { b, c }{ c }FOLLOW(B):

FOLLOW(C): { $ }


Compiler Design

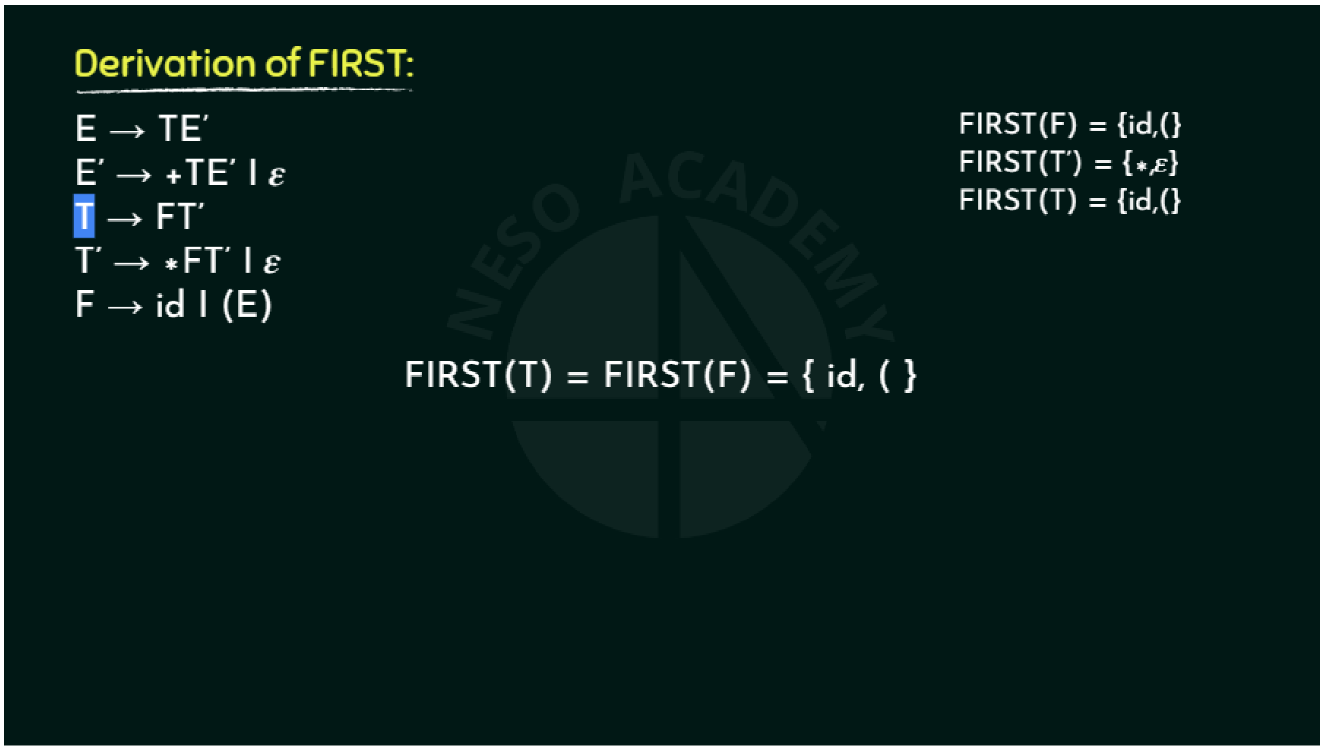
First & Follow

Compiler DesignFirst & Follow

Outcome

☆ Step by step derivation of FIRST and FOLLOW.

Outcome☆Step by step derivation of FIRST and FOLLOW.



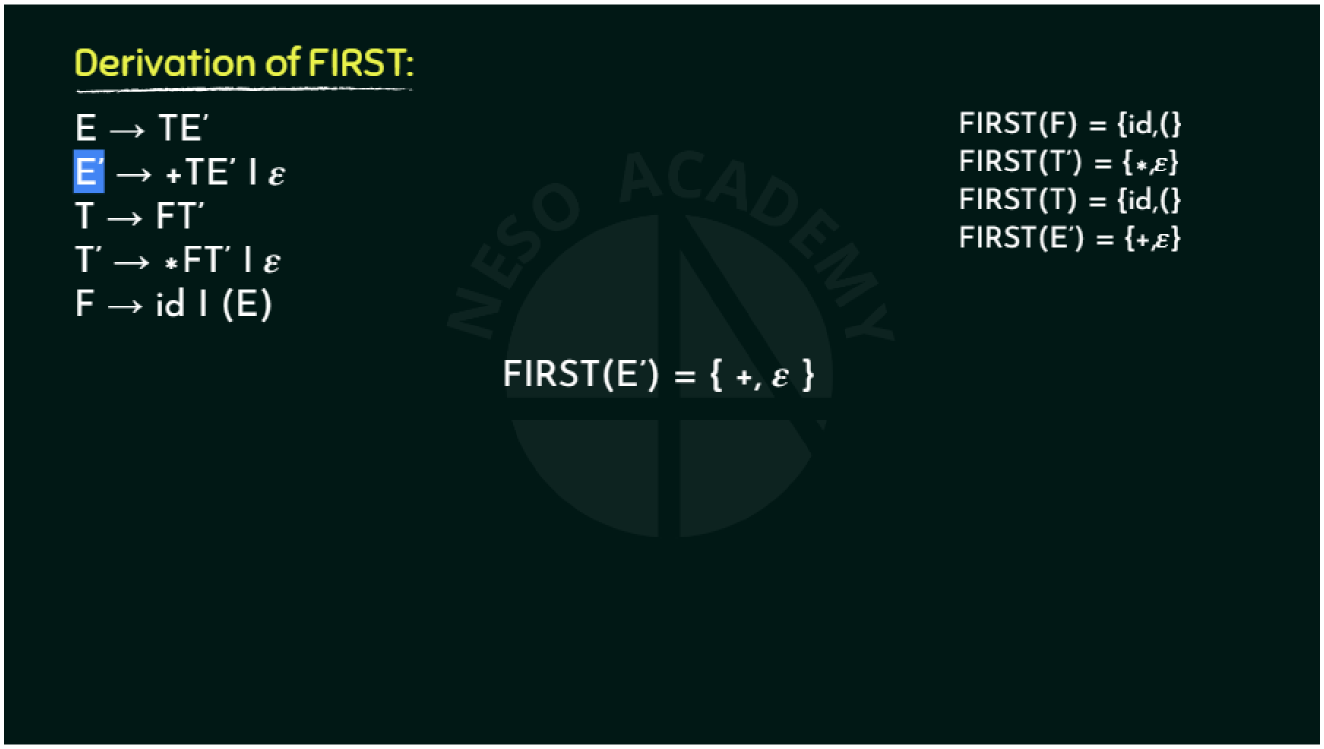
Derivation of FIRST:

E → TE′
E′ → +TE′ | ε
T → FT′
T′ → ∗FT′ | ε
F → id | (E)

FIRST(F) = {id,(}
FIRST(T′) = {∗,ε}
FIRST(T) = {id,(}

FIRST(T) = FIRST(F) = { id, (}

Derivation of FIRST:E → TE'E' → +TE' I εT → FT'T' → +FT' I εF → id I (E)*FIRST(T)FIRST(F) = {id,(}= FIRST(F)= { id, (}FIRST(T) = {id,(}FIRST(T') = {+,ε}*



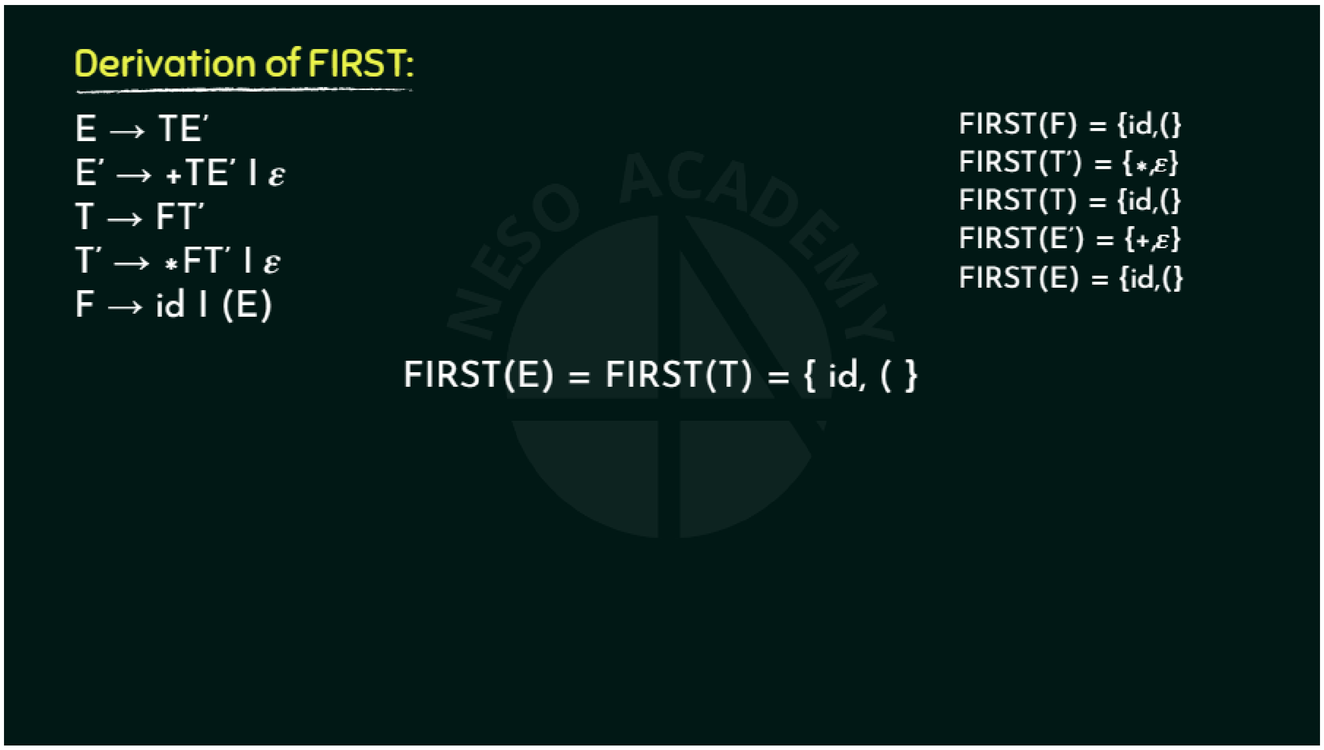
Derivation of FIRST:

E → TE′
E′ → +TE′ | ε
T → FT′
T′ → ∗FT′ | ε
F → id | (E)

FIRST(F) = {id,(}
FIRST(T′) = {∗,ε}
FIRST(T) = {id,(}
FIRST(E′) = {+,ε}

FIRST(E′) = { +, ε }

Derivation of FIRST:E → TE'E' → +TE' I εT → FT'T' → +FT' I εF → id I (E)*FIRST(E')FIRST(F) = {id,(}FIRST(T) = {id,(}FIRST(T') = {+,ε}*= { +, ε } FIRST(E') = {+,ε}



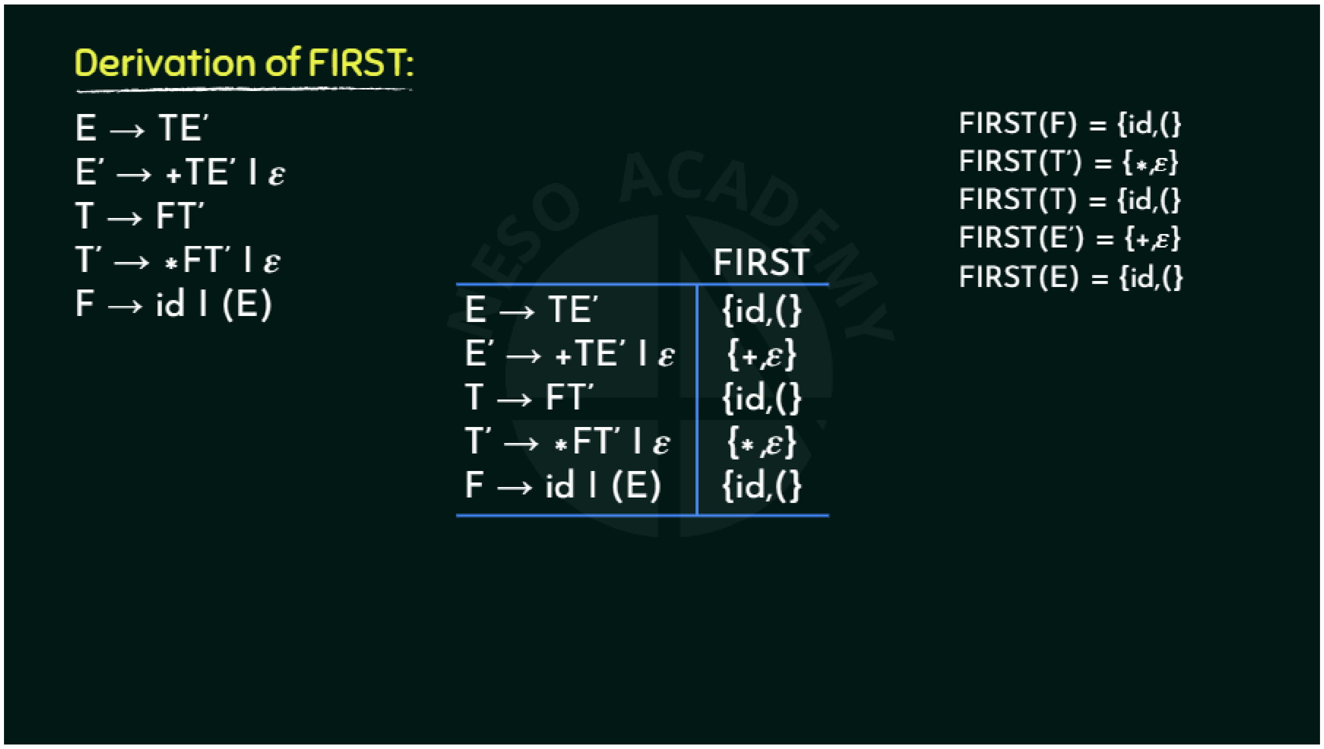
Derivation of FIRST:

E → TE′
E′ → +TE′ | ε
T → FT′
T′ → ∗FT′ | ε
F → id | (E)

FIRST(E) = FIRST(T) = { id, (}

FIRST(F) = {id,(}
FIRST(T′) = {∗,ε}
FIRST(T) = {id,(}
FIRST(E′) = {+,ε}
FIRST(E) = {id,(}

Derivation of FIRST:E → TE'E' → +TE' I εT → FT'T' → +FT' I εF → id I (E)*FIRST(E)FIRST(F) = {id,(}FIRST(T) = {id,(}FIRST(T') = {+,ε}*FIRST(E') = {+,ε}= FIRST(T)= { id, (}FIRST(E) = {id,(}



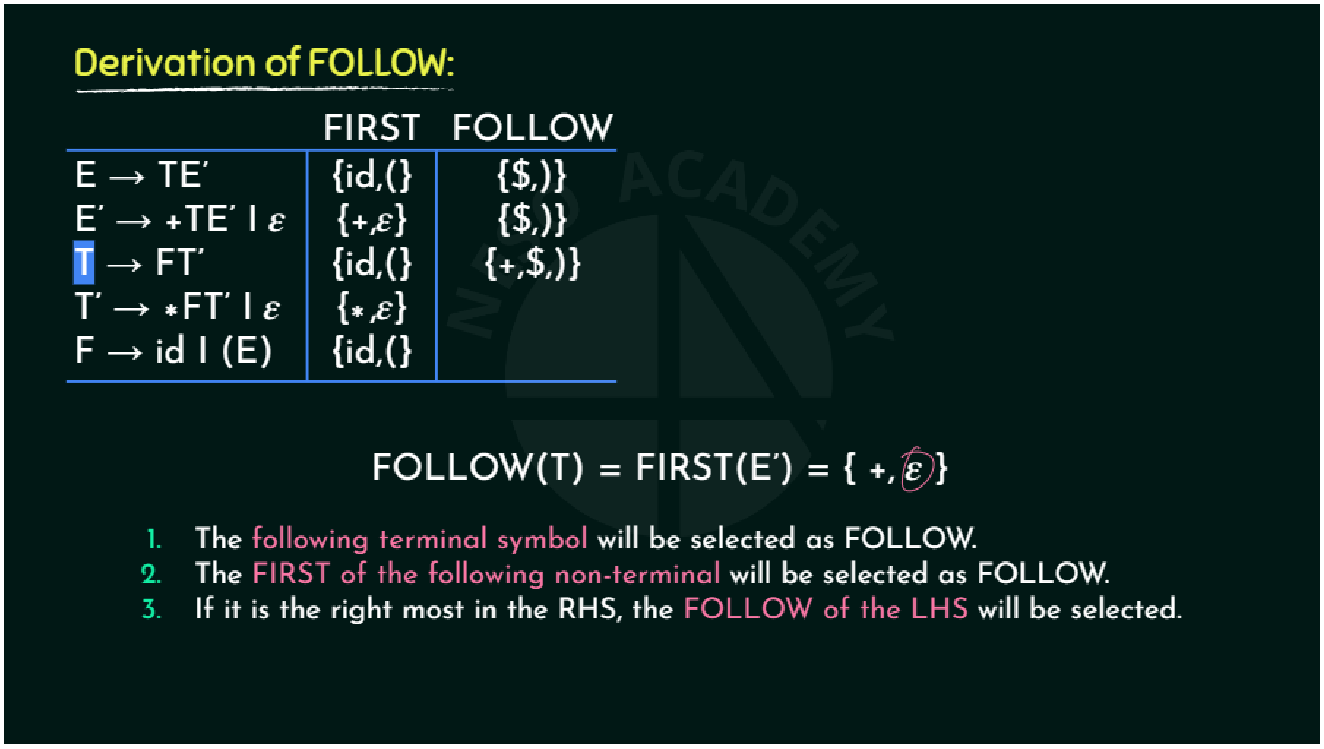
Derivation of FIRST:

E → TE′
E′ → +TE′ | ε
T → FT′
T′ → ∗FT′ | ε
F → id | (E)

	FIRST
E → TE′	{id,(}
E′ → +TE′ | ε	{+,ε}
T → FT′	{id,(}
T′ → ∗FT′ | ε	{∗,ε}
F → id | (E)	{id,(}

FIRST(F) = {id,(}
FIRST(T′) = {∗,ε}
FIRST(T) = {id,(}
FIRST(E′) = {+,ε}
FIRST(E) = {id,(}

Derivation of FIRST:E → TE'E' → +TE' I εT → FT'T' → +FT' I εF → id I (E)*FIRST(F) = {id,(}FIRST(T) = {id,(}FIRST(T') = {+,ε}*FIRST(E') = {+,ε}FIRST(E) = {id,(}E → TE'E' → +TE' I εT → FT'T' → +FT' I εF → id I (E)*{id,(}{+,ε}{id,(}{+,ε}{id,(}FIRST*



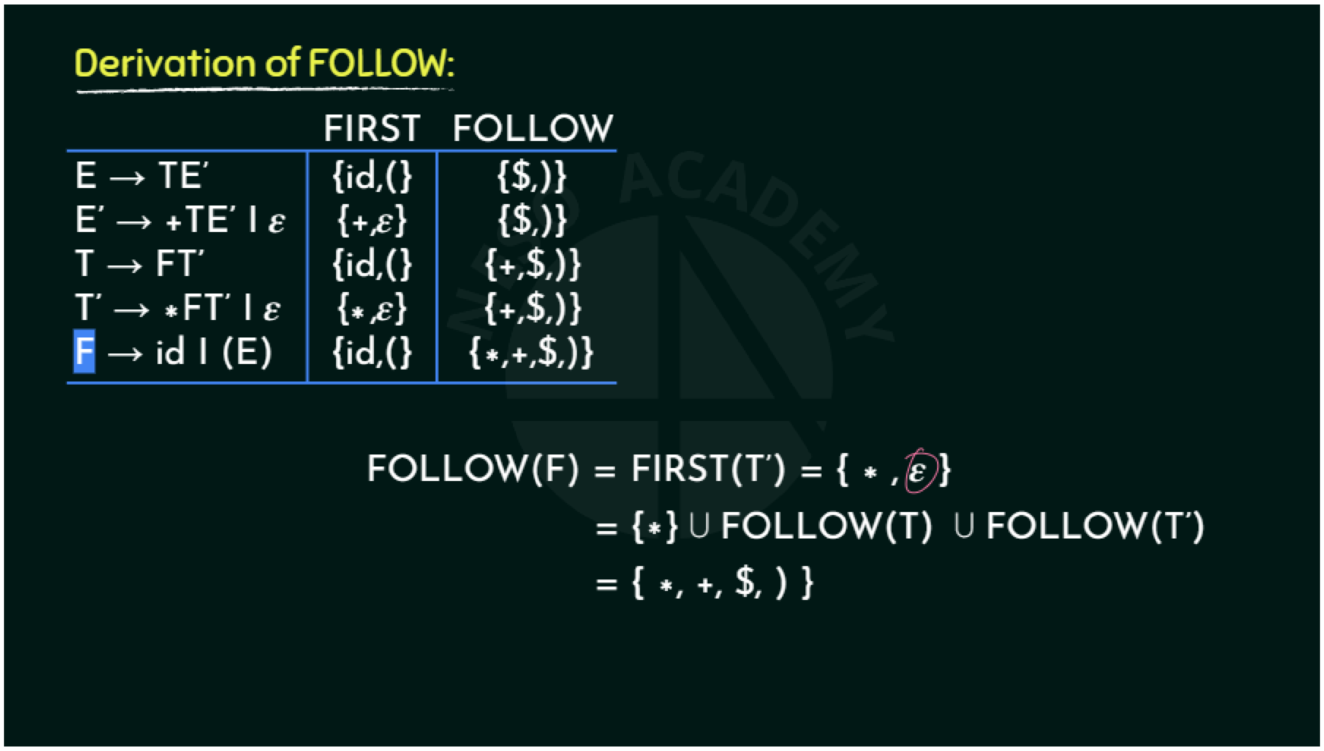
Derivation of FOLLOW:

	FIRST	FOLLOW
E → TE′	{id,(}	{$,)}
E′ → +TE′ I ε	{+,ε}	{$,)}
T → FT′	{id,(}	{+,$,)}
T′ → ∗FT′ I ε	{∗,ε}	
F → id I (E)	{id,(}	

FOLLOW(T) = FIRST(E′) = { +, ε }

1. The following terminal symbol will be selected as FOLLOW.
2. The FIRST of the following non-terminal will be selected as FOLLOW.
3. If it is the right most in the RHS, the FOLLOW of the LHS will be selected.

= { +, ε },)}Derivation of FOLLOW:E → TE’E’ → +TE’ I εT → FT’T’ → +FT’ I εF → id I (E)*FIRSTFOLLOW{${$,)}{+,$,)}FOLLOW(T)= FIRST(E’)1.The following terminal symbol will be selected as FOLLOW.2.The FIRST of the following non-terminal will be selected as FOLLOW.3.If it is the right most in the RHS, the FOLLOW of the LHS will be selected.{id,(}{+,ε}{id,(}{+,ε}{id,(}*



Derivation of FOLLOW:

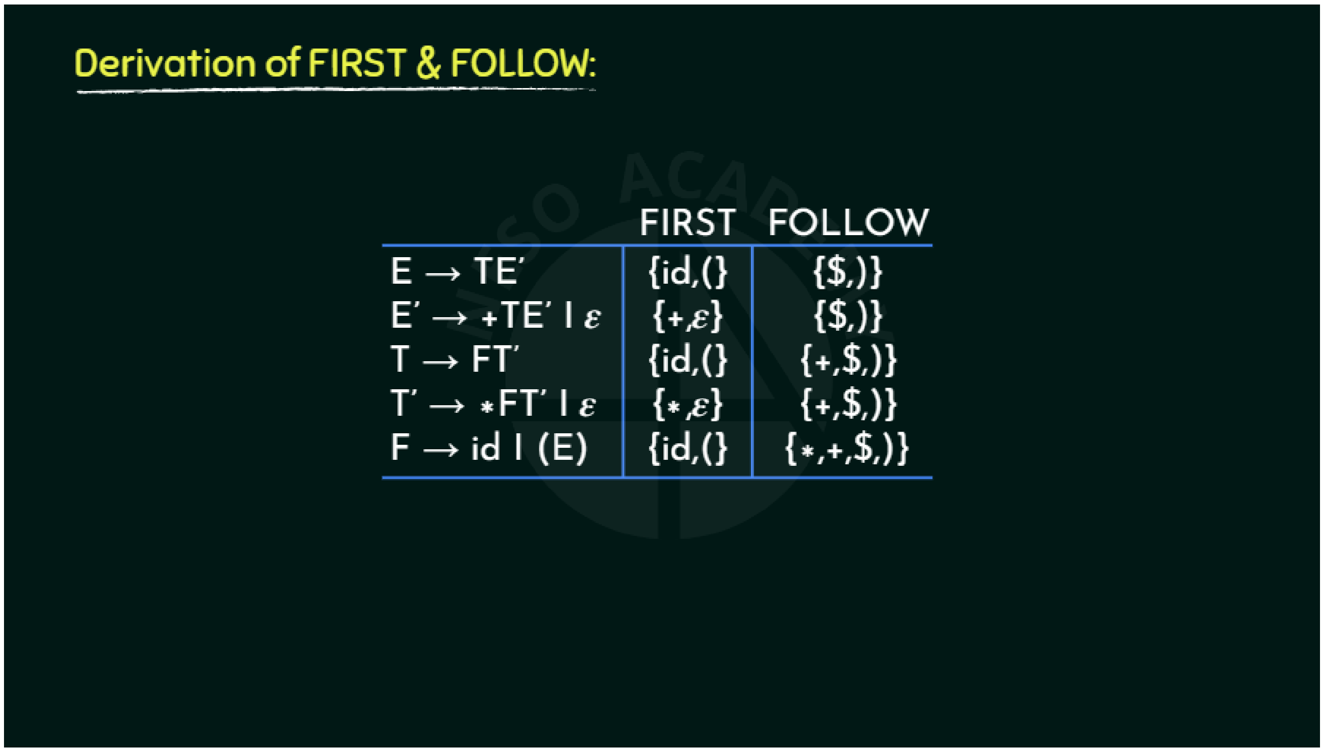
	FIRST	FOLLOW
E → TE′	{id,(}	{$,)}
E′ → +TE′ I ε	{+,ε}	{$,)}
T → FT′	{id,(}	{+,$,)}
T′ → ∗FT′ I ε	{∗,ε}	{+,$,)}
F → id I (E)	{id,(}	{∗,+,$,)}

FOLLOW(F) = FIRST(T′) = { ∗ , ε }

= {∗} ∪ FOLLOW(T) ∪ FOLLOW(T′)

= { ∗, +, $,) }

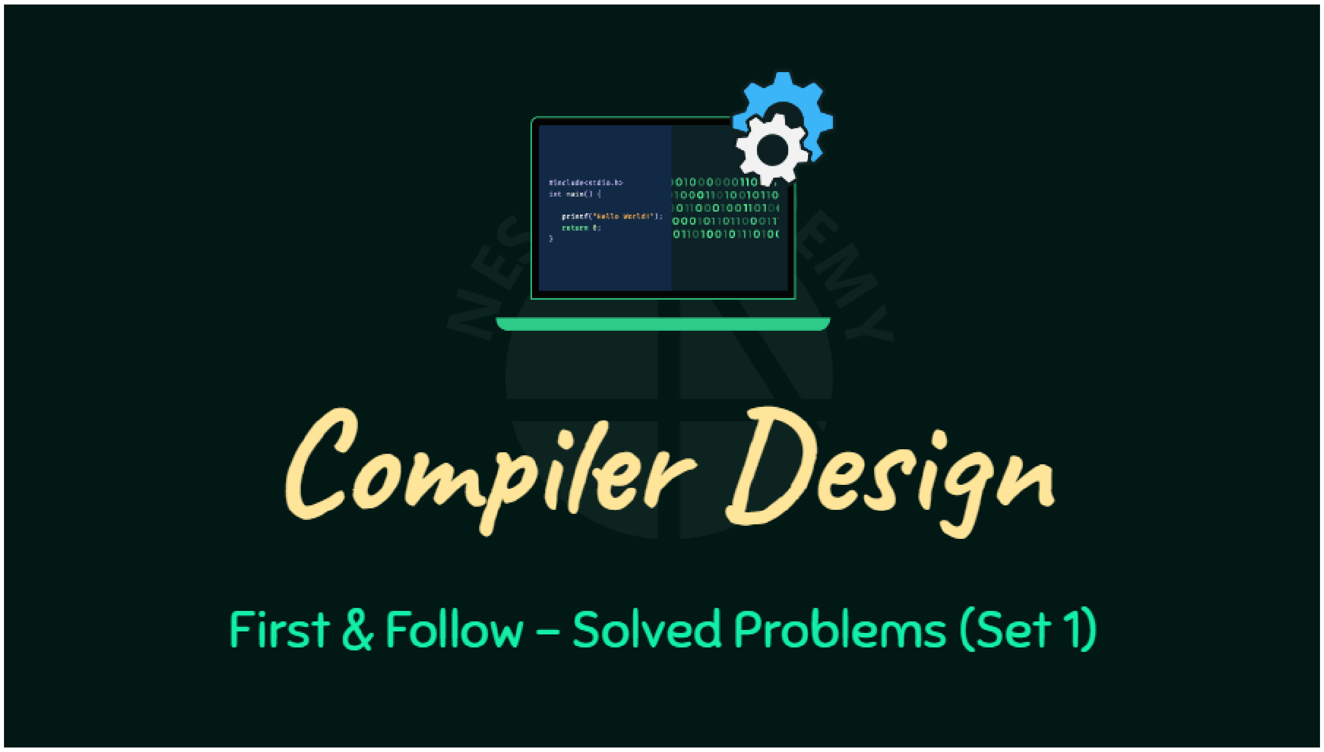
FOLLOW(F),)}Derivation of FOLLOW:E → TE'E' → +TE' I εT → FT'T' → +FT' I εF → id I (E)*FIRSTFOLLOW{${$,)}{+,$,)}{+,$,)}{+*, ε }= { +*= FIRST(T'),+,$,)}= {+}*∪ FOLLOW(T)∪ FOLLOW(T') = { +, +, $,) }*{id,(}{+,ε}{id,(}{+,ε}{id,(}*



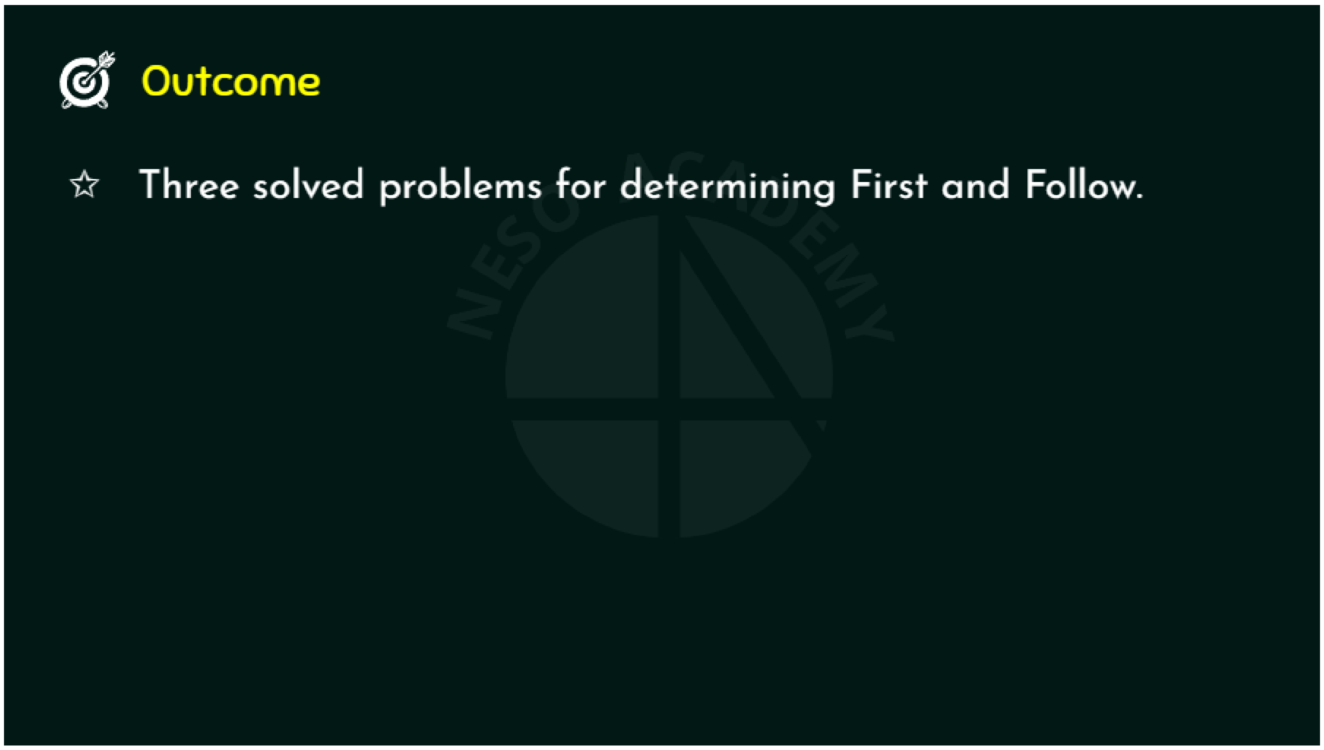
Derivation of FIRST & FOLLOW:

	FIRST	FOLLOW
E → TE'	{id,(}	{$,)}
E' → +TE' I ε	{+,ε}	{$,)}
T → FT'	{id,(}	{+,$,)}
T' → ∗FT' I ε	{∗,ε}	{+,$,)}
F → id I (E)	{id,(}	{∗,+,$,)}

Derivation of FIRST & FOLLOW:,)}E → TE'E' → +TE' I εT → FT'T' → +FT' I εF → id I (E)*FIRSTFOLLOW{${$,)}{+,$,)}{+,$,)}{+*,+,$,)}{id,(}{+,ε}{id,(}{+,ε}{id,(}*



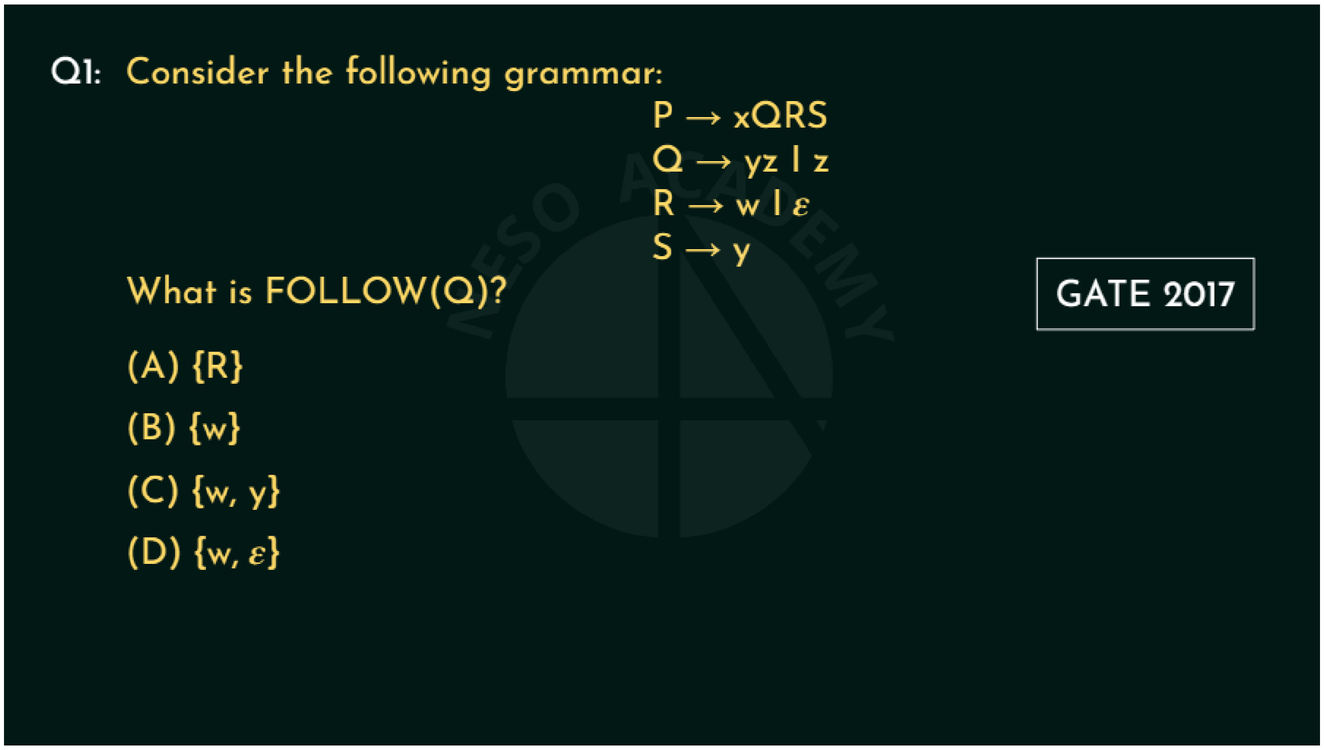
Compiler DesignFirst & Follow - Solved Problems (Set 1)



Outcome

☆ Three solved problems for determining First and Follow.

Outcome☆Three solved problems for determining First and Follow.



Q1: Consider the following grammar:

$P \rightarrow xQRS$
$Q \rightarrow yz \mid z$
$R \rightarrow w \mid \varepsilon$
$S \rightarrow y$

What is FOLLOW(Q)?

GATE 2017

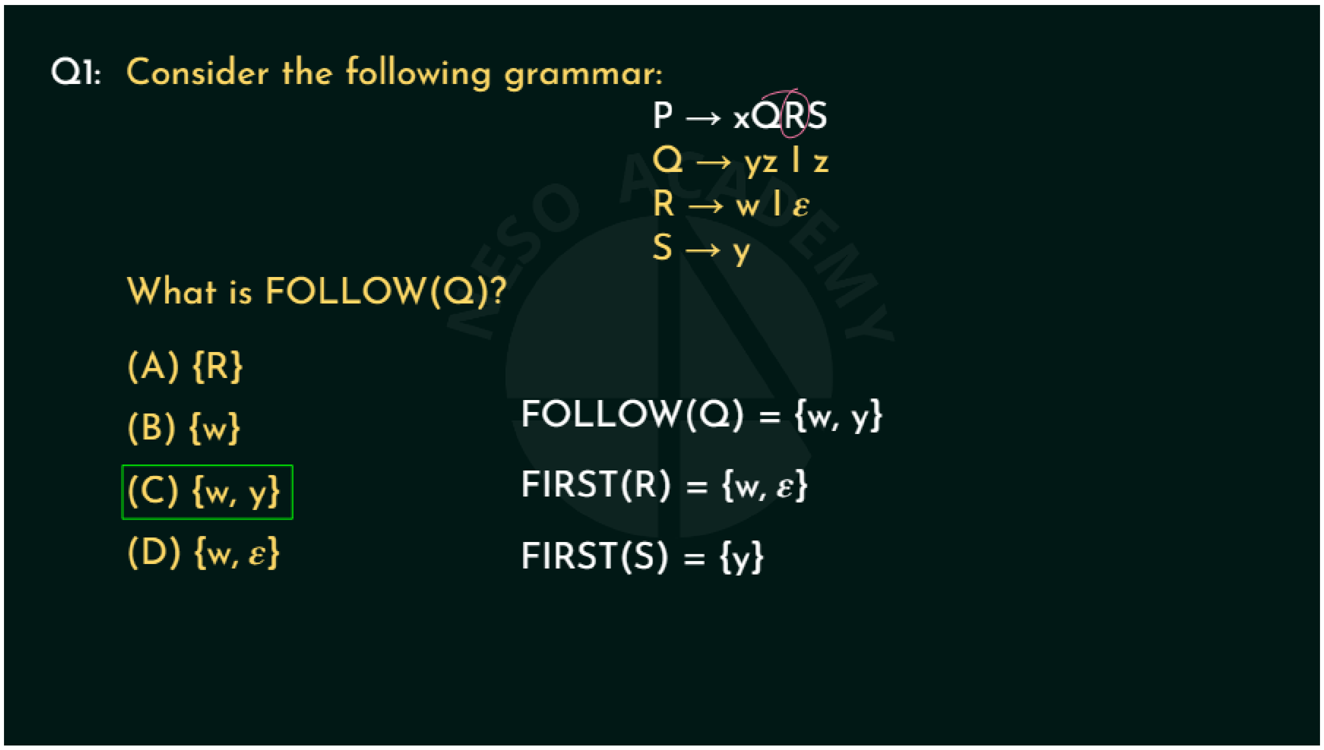
(A) {R}

(B) {w}

(C) {w, y}

(D) {w, ε}

Q1:Consider the following grammar:P → xQRSQ → yz I zR → w I εS → yWhat is FOLLOW(Q)?(A) {R}(B) {w}(C) {w, y}(D) {w, ε}GATE 2017



Q1: Consider the following grammar:

$P \rightarrow xQRS$
$Q \rightarrow yz \mid z$
$R \rightarrow w \mid \varepsilon$
$S \rightarrow y$

What is FOLLOW(Q)?

(A) {R}

(B) {w}

[✓ (C) {w, y}]

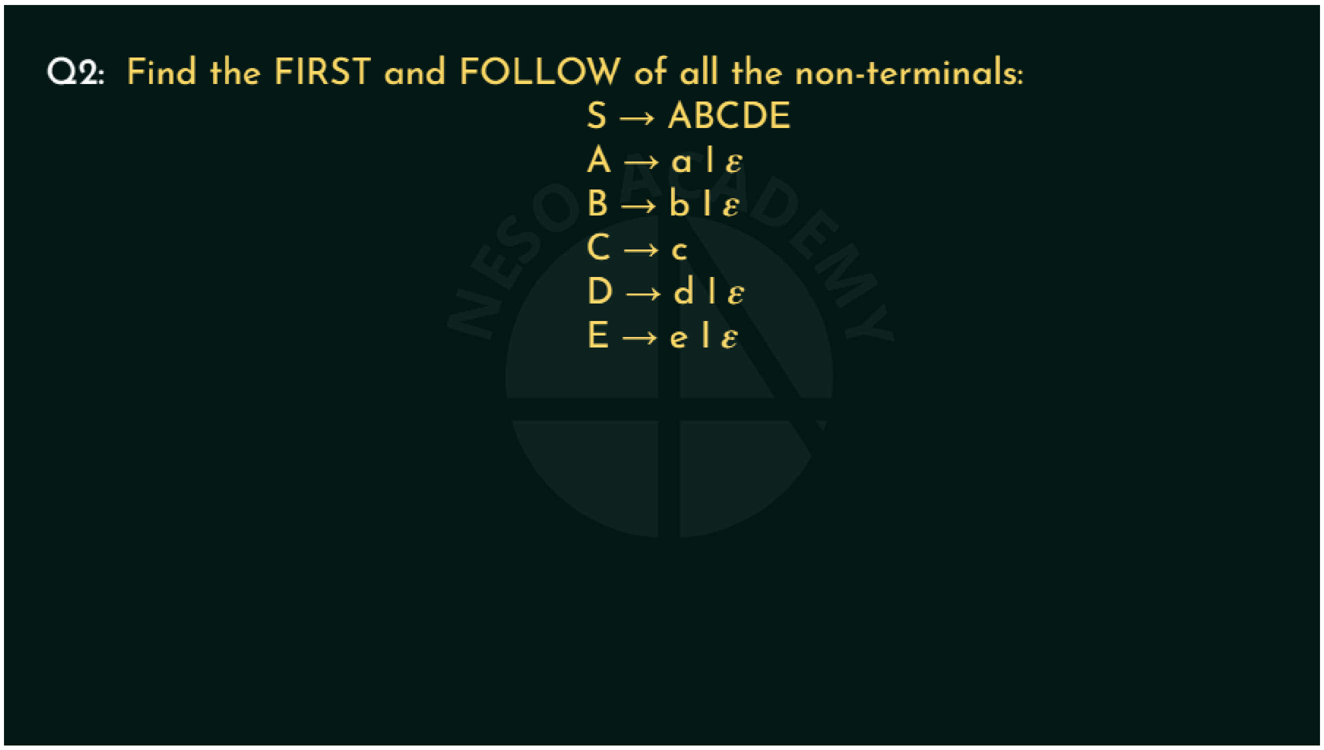
(D) {w, ε}

FOLLOW(Q) = {w, y}

FIRST(R) = {w, ε}

FIRST(S) = {y}

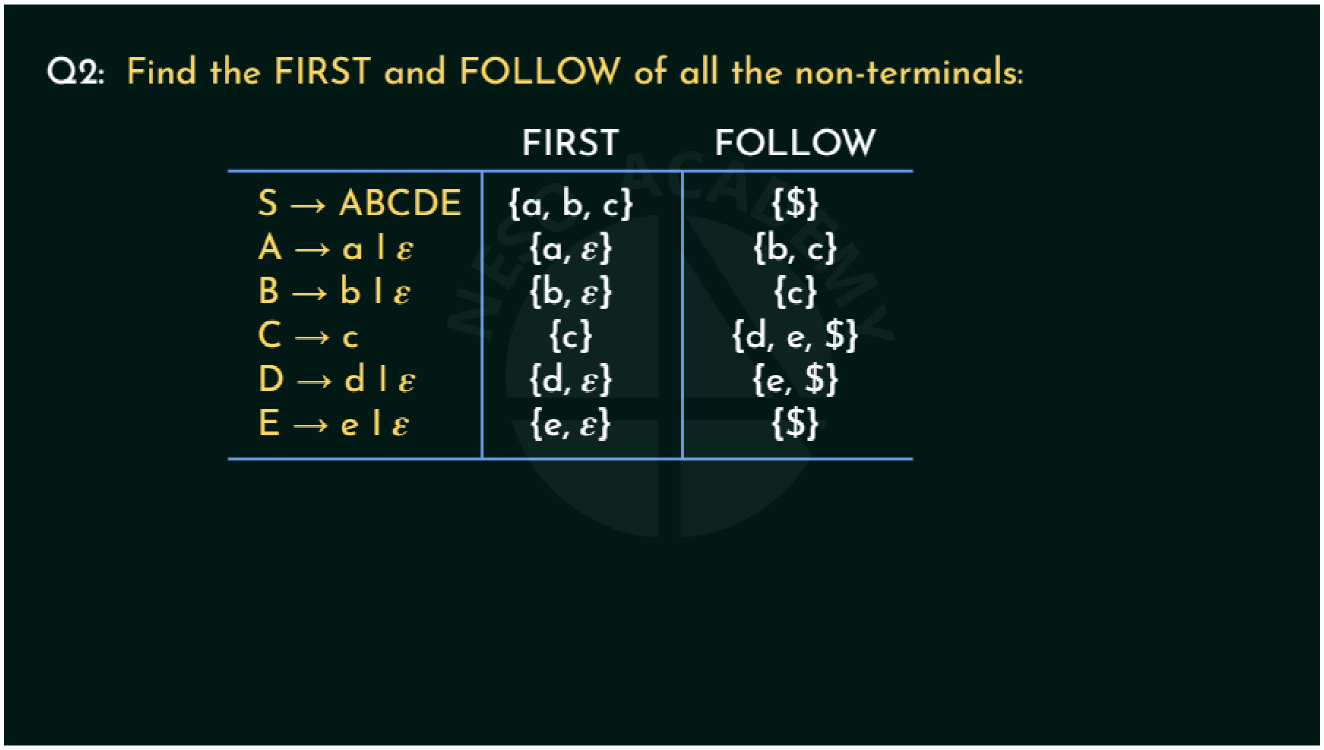
Q1:Consider the following grammar:P → xQRSQ → yz I zR → w I εS → yWhat is FOLLOW(Q)?(A) {R}(B) {w}(C) {w, y}(D) {w, ε}FOLLOW(Q)FIRST(R)= {w, ε}= {w, y}FIRST(S)= {y}



Q2: Find the FIRST and FOLLOW of all the non-terminals:

$S \rightarrow ABCDE$
$A \rightarrow a \mid \varepsilon$
$B \rightarrow b \mid \varepsilon$
$C \rightarrow c$
$D \rightarrow d \mid \varepsilon$
$E \rightarrow e \mid \varepsilon$

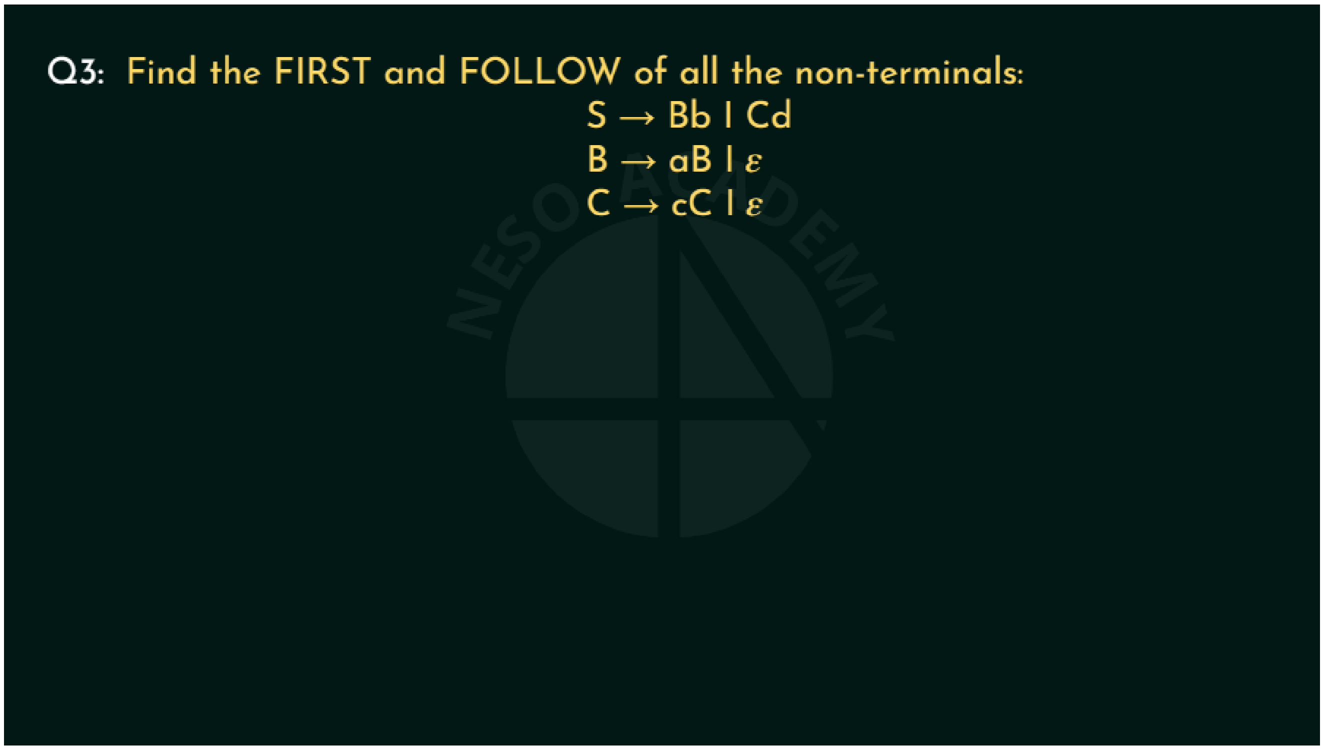
Q2:Find the FIRST and FOLLOW of all the non-terminals:S → ABCDEA → a I εB → b I εC → cD → d I εE → e I ε



Q2: Find the FIRST and FOLLOW of all the non-terminals:

	FIRST	FOLLOW
S → ABCDE	{a, b, c}	{$}
A → a I ε	{a, ε}	{b, c}
B → b I ε	{b, ε}	{c}
C → c	{c}	{d, e, $}
D → d I ε	{d, ε}	{e, $}
E → e I ε	{e, ε}	{$}

Find the FIRST and FOLLOW of all the non-terminals:S → ABCDEA → a I εB → b I εC → cD → d I εE → e I εFIRSTFOLLOW, $} {aQ2:{e, ε} {d, ε} {b, ε} {a, ε} {c} , b, c} {$} {b, c} {c} {d, e, $} {e{$}



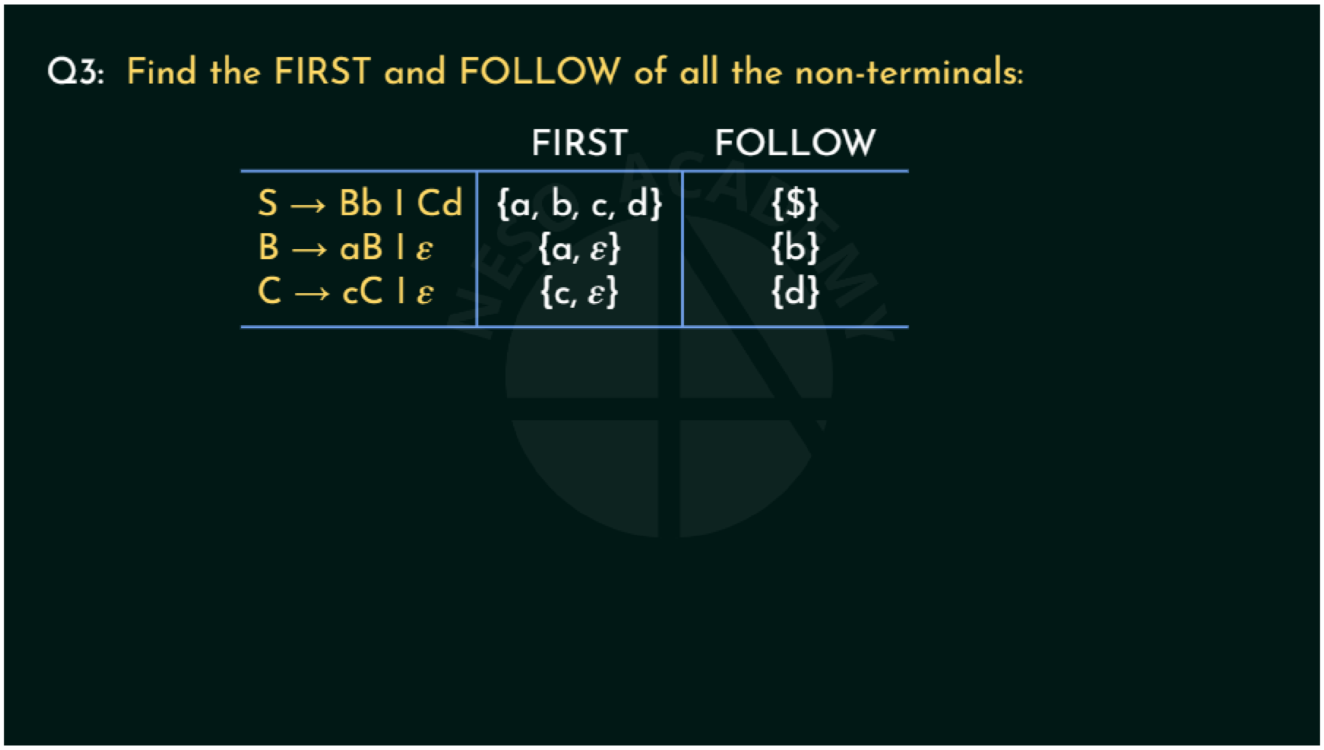
Q3: Find the FIRST and FOLLOW of all the non-terminals:

S → Bb I Cd

B → aB I ε

C → cC I ε

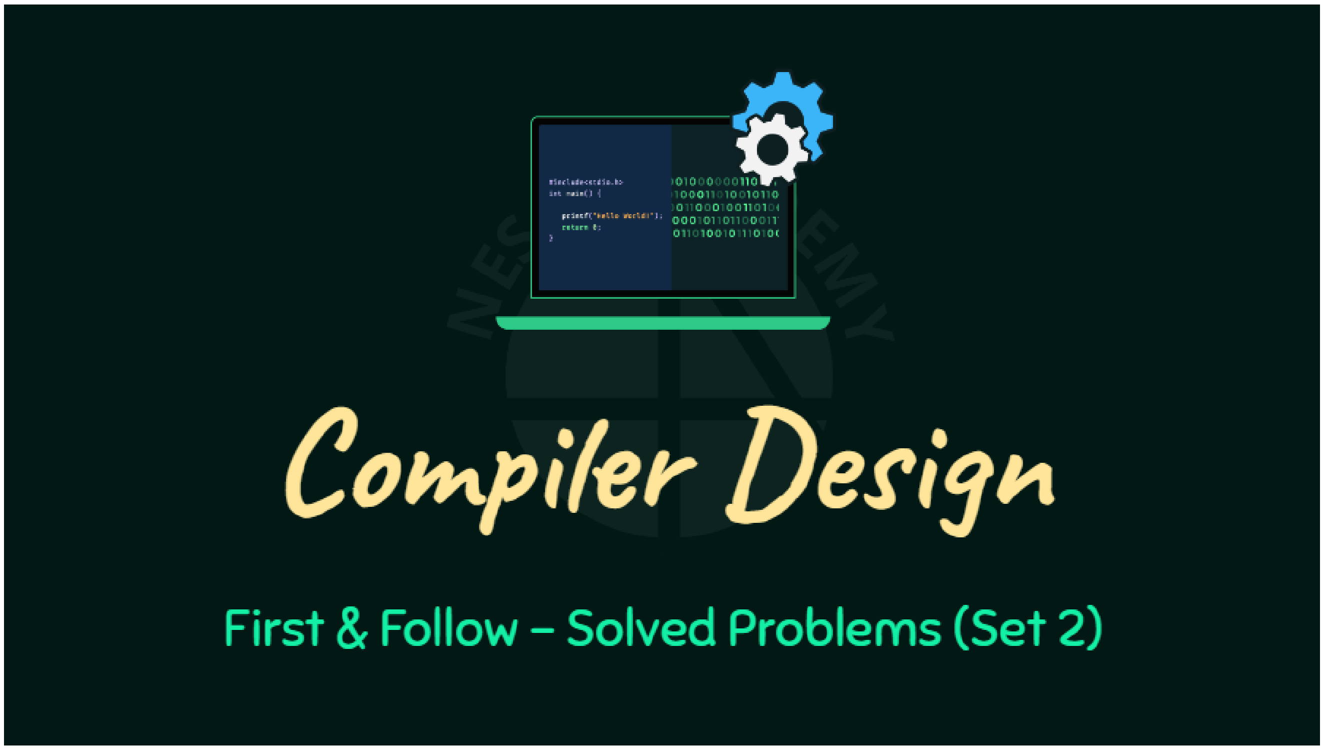
Q3:Find the FIRST and FOLLOW of all the non-terminals:S → Bb I CdB → aB I εC → cC I ε



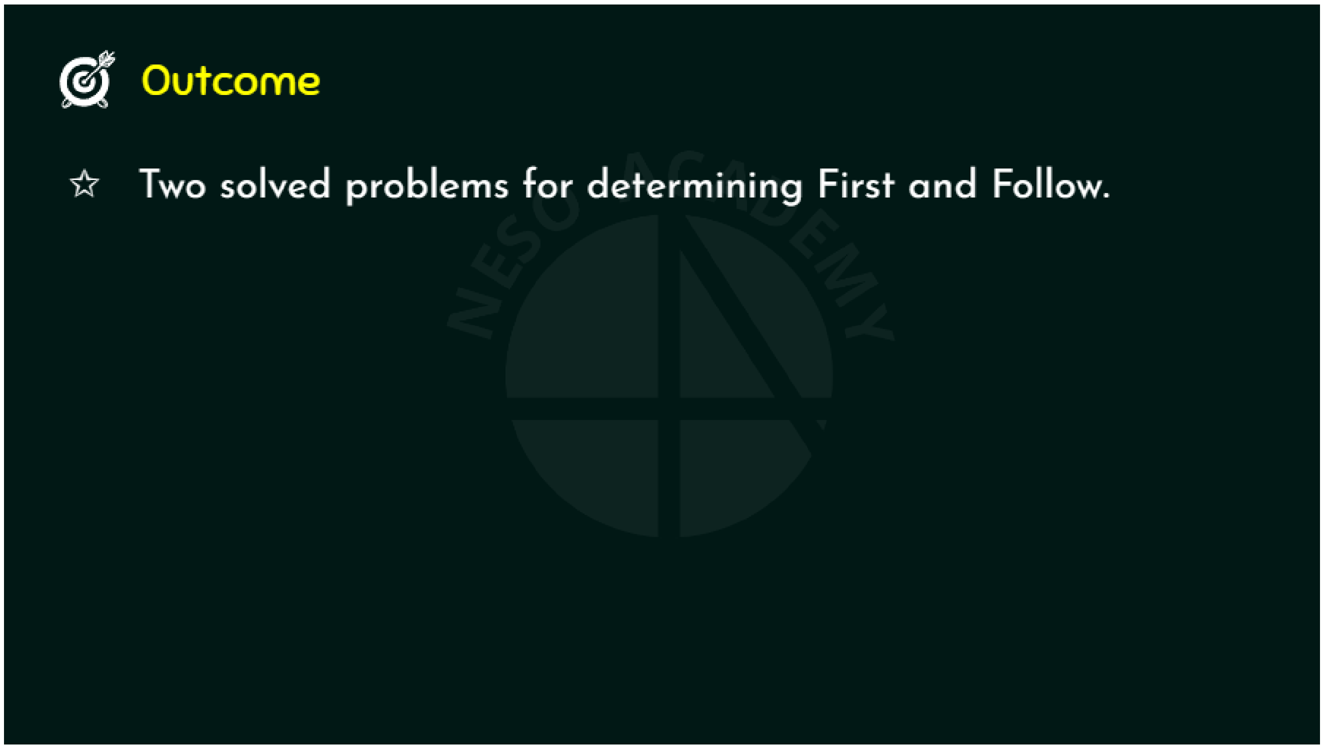
Q3: Find the FIRST and FOLLOW of all the non-terminals:

	FIRST	FOLLOW
S → Bb I Cd	{a, b, c, d}	{$}
B → aB I ε	{a, ε}	{b}
C → cC I ε	{c, ε}	{d}

Find the FIRST and FOLLOW of all the non-terminals:S → Bb I CdB → aB I εC → cC I εFIRSTFOLLOWQ3:{c, ε} {a, ε} {a, b, c, d} {$} {d} {b}



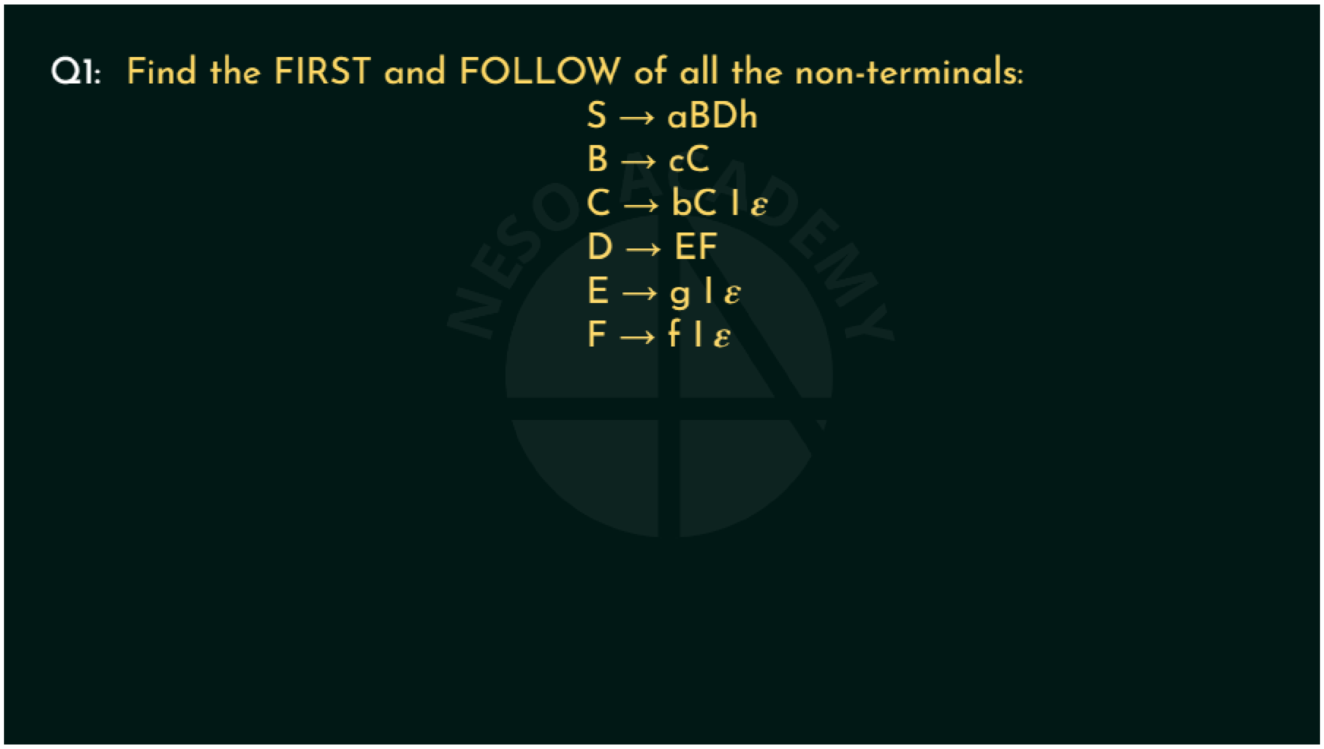
Compiler DesignFirst & Follow - Solved Problems (Set 2)



Outcome

☆ Two solved problems for determining First and Follow.

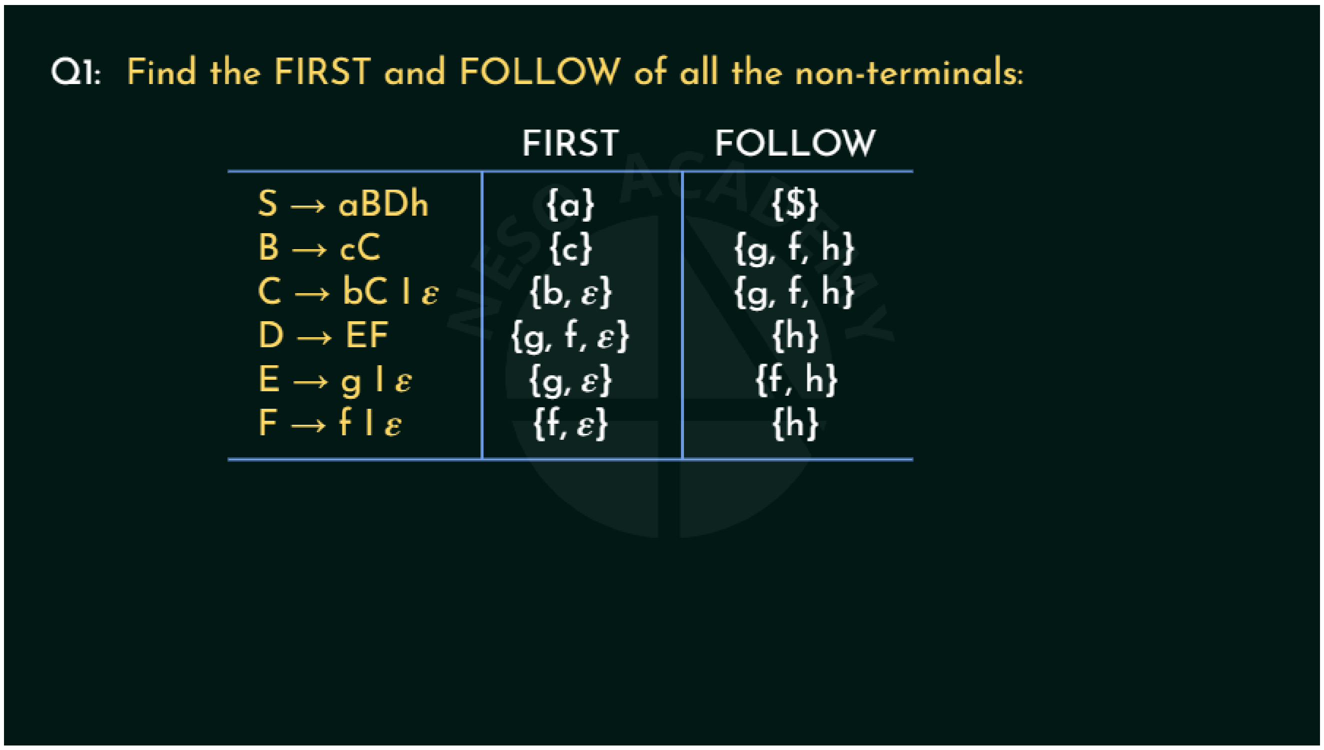
Outcome☆Two solved problems for determining First and Follow.



Q1: Find the FIRST and FOLLOW of all the non-terminals:

$S \rightarrow aBDh$
$B \rightarrow cC$
$C \rightarrow bC \mid \varepsilon$
$D \rightarrow EF$
$E \rightarrow g \mid \varepsilon$
$F \rightarrow f \mid \varepsilon$

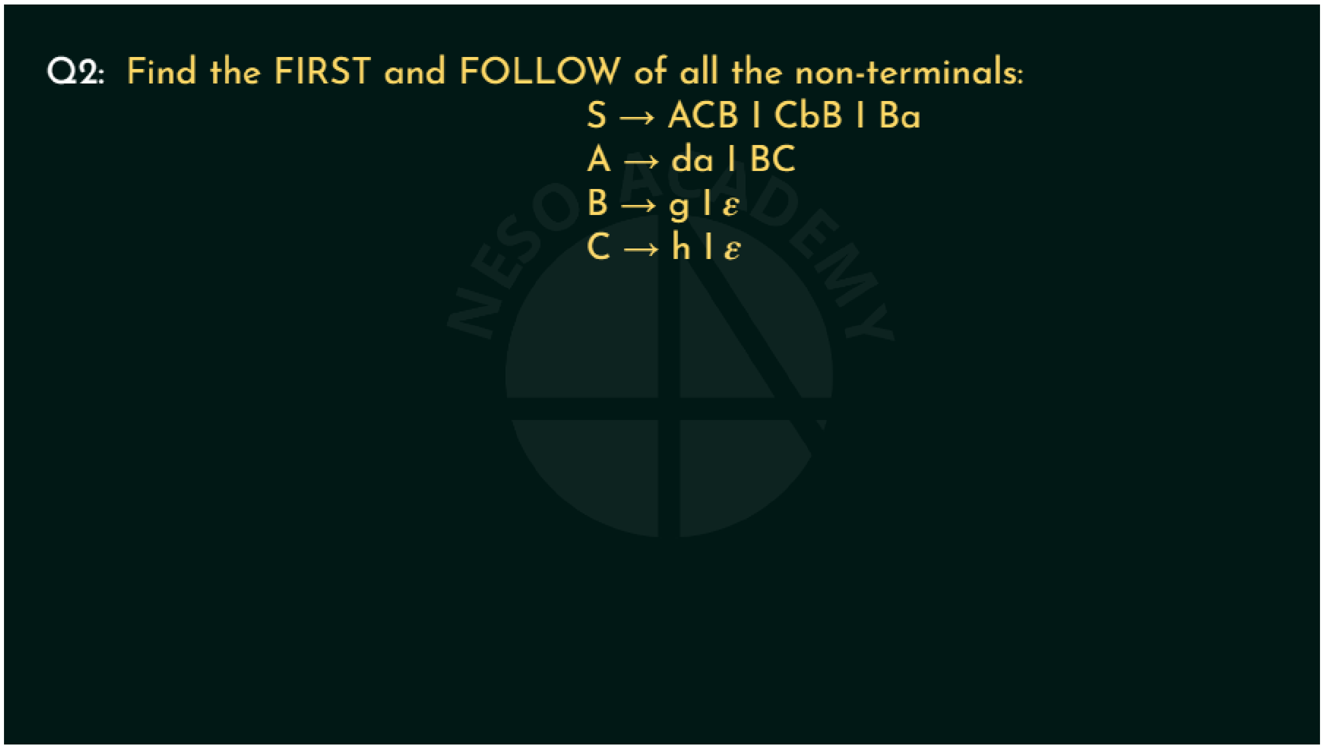
Q1:Find the FIRST and FOLLOW of all the non-terminals:S → aBDhB → cCC → bC I εD → EFE → g I εF → f I ε



Q1: Find the FIRST and FOLLOW of all the non-terminals:

	FIRST	FOLLOW
S → aBDh	{a}	{$}
B → cC	{c}	{g, f, h}
C → bC I ε	{b, ε}	{g, f, h}
D → EF	{g, f, ε}	{h}
E → g I ε	{g, ε}	{f, h}
F → f I ε	{f, ε}	{h}

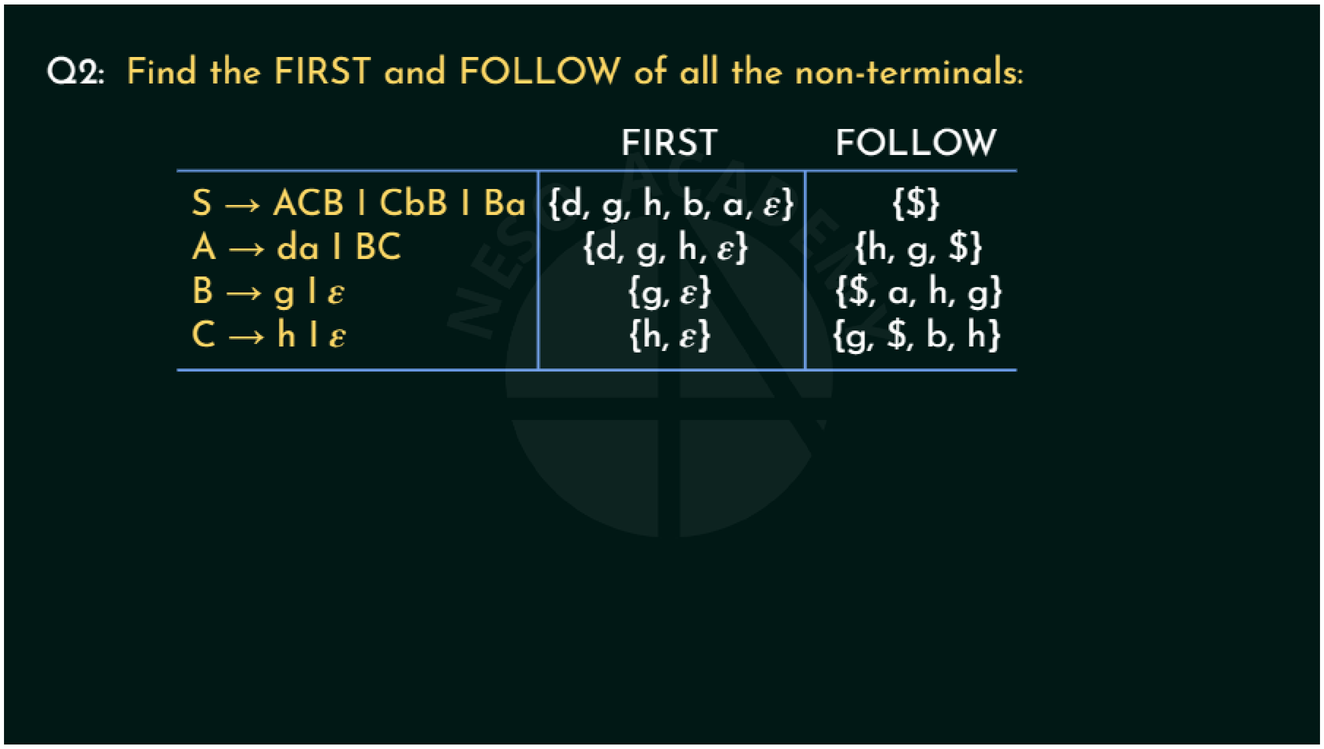
Find the FIRST and FOLLOW of all the non-terminals:S → aBDhB → cCC → bC I εD → EFE → g I εF → f I εFIRSTFOLLOWQ1:{f, ε} {g, ε} {b, ε} {c} {$} , h} {f{h} {g, f, ε} {a} {g, h} , f {g, h} , f {h}



Q2: Find the FIRST and FOLLOW of all the non-terminals:

S → ACB I CbB I Ba
A → da I BC
B → g I ε
C → h I ε

Q2:Find the FIRST and FOLLOW of all the non-terminals:S → ACB I CbB I BaA → da I BCB → g I εC → h I ε

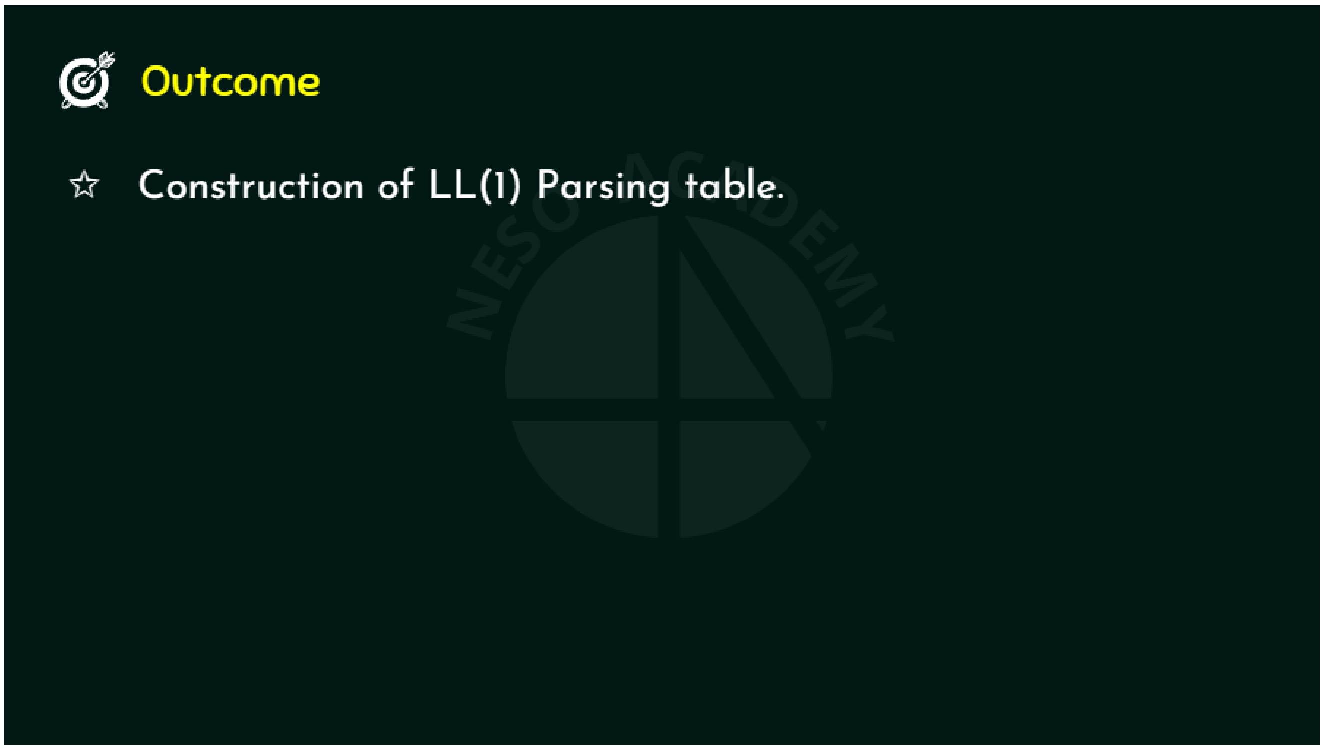


Q2: Find the FIRST and FOLLOW of all the non-terminals:

	FIRST	FOLLOW
S → ACB I CbB I Ba	{d, g, h, b, a, ε}	{$}
A → da I BC	{d, g, h, ε}	{h, g, $}
B → g I ε	{g, ε}	{$, a, h, g}
C → h I ε	{h, ε}	{g, $, b, h}

FIRSTFOLLOWFind the FIRST and FOLLOW of all the non-terminals:S → ACB I CbB I BaA → da I BCB → g I εC → h I εQ2:{g, ε} {$} {h, ε} {d, g, h, ε} {d, g, h, b, a, ε} {h, g, $} {$, a, h, g} {g, $, b, h}

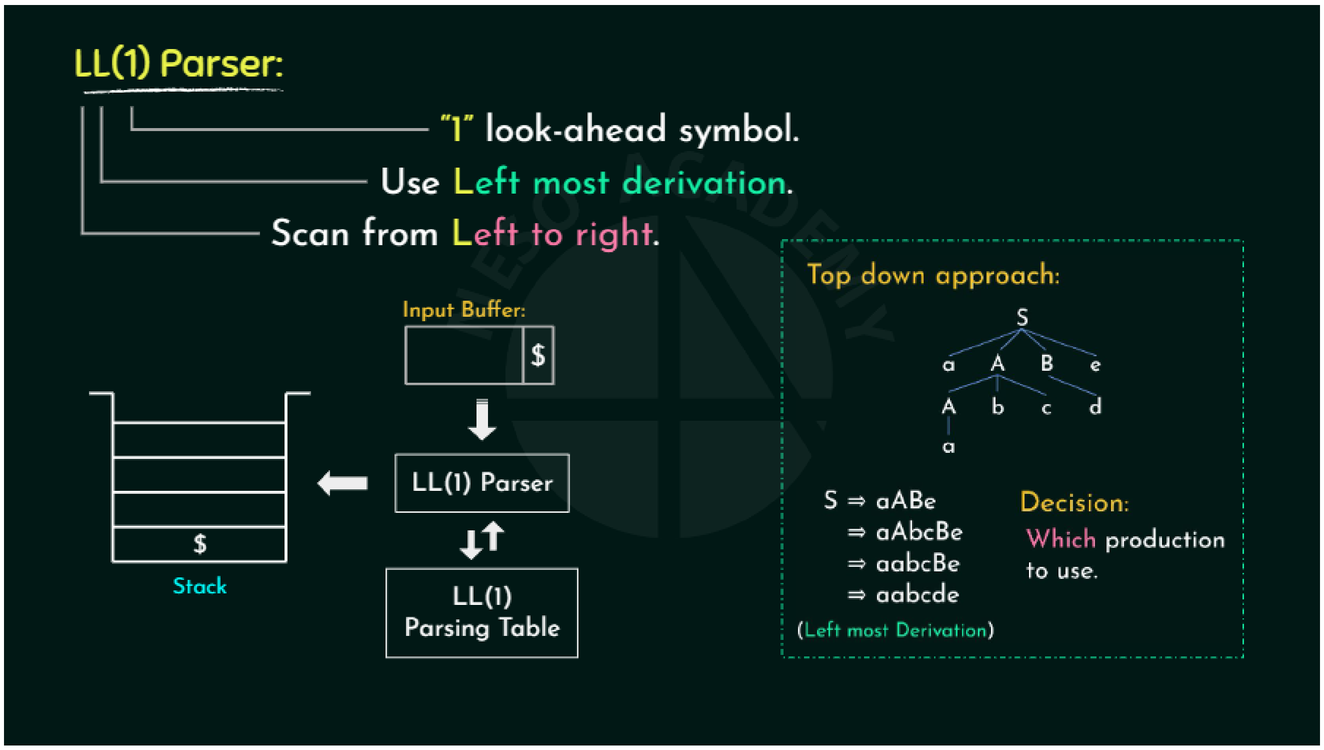
Compiler DesignLL(1) Parsing Table



Outcome

☆ Construction of LL(1) Parsing table.

Outcome☆Construction of LL(1) Parsing table.



LL(1) Parser:

"1" look-ahead symbol.

Use Left most derivation.

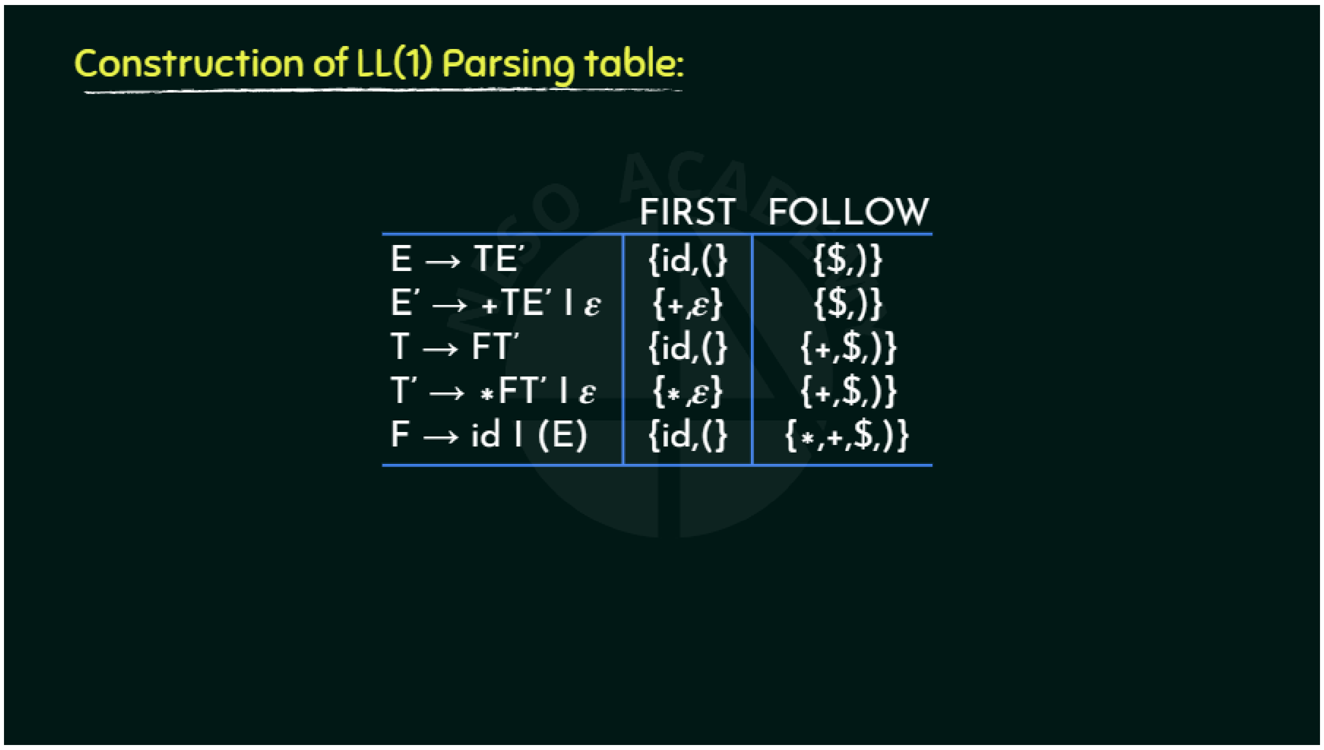
Scan from Left to right.

$$S \Rightarrow aABe \Rightarrow aAbcBe \Rightarrow aabcBe \Rightarrow aabcde$$

(Left most Derivation)

Decision: Which production to use.

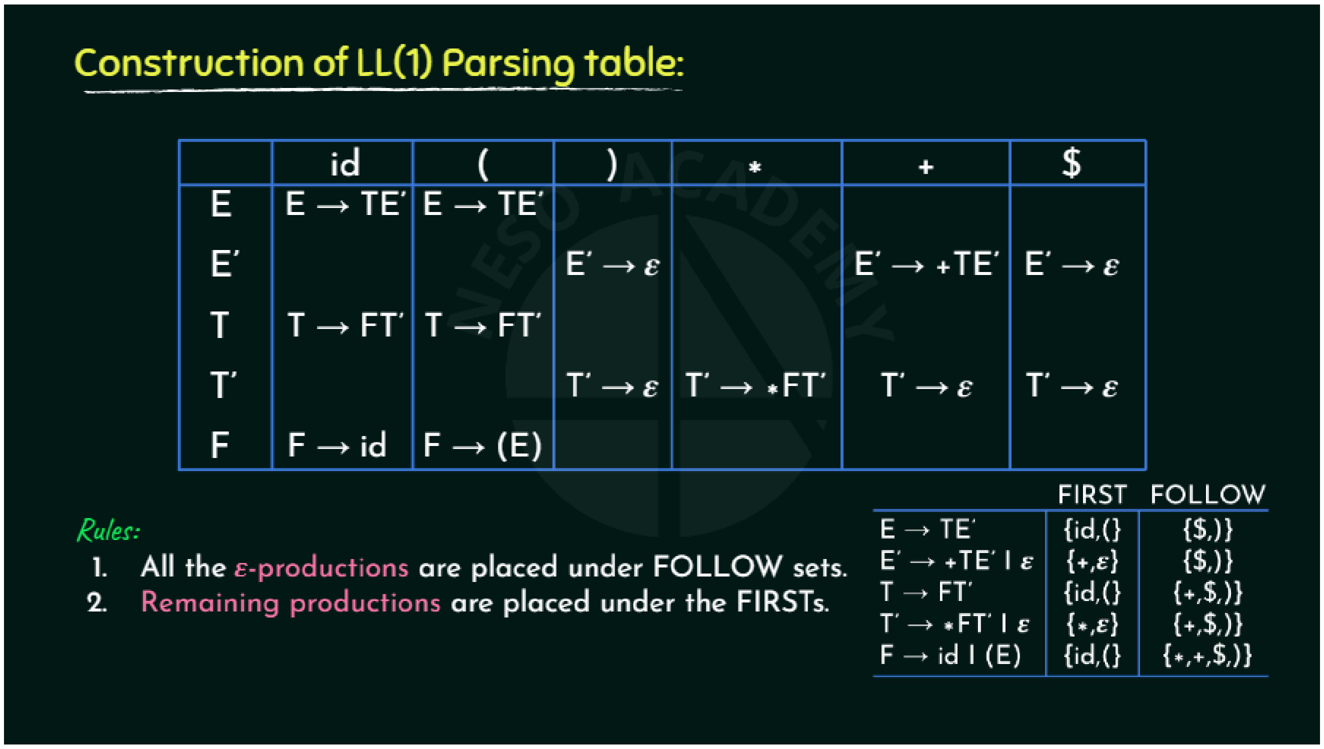
LL(1) Parser:$$LL(1) ParserLL(1) Parsing TableInput Buffer:StackScan from Left to right.Use Left most derivation."1" look-ahead symbol.



Construction of LL(1) Parsing table:

	FIRST	FOLLOW
E → TE′	{id,(}	{$,)}
E′ → +TE′ I ε	{+,ε}	{$,)}
T → FT′	{id,(}	{+,$,)}
T′ → ∗FT′ I ε	{∗,ε}	{+,$,)}
F → id I (E)	{id,(}	{∗,+,$,)}

Construction of LL(1) Parsing table:,)}E → TE’E’ → +TE’ I εT → FT’T’ → +FT’ I εF → id I (E)*FIRSTFOLLOW{${$,)}{+,$,)}{+,$,)}{+*,+,$,)}{id,(}{+,ε}{id,(}{+,ε}{id,(}*



Construction of LL(1) Parsing table:

	id	(	)	∗	+	$
E	E → TE′	E → TE′				
E′			E′ → ε		E′ → +TE′	E′ → ε
T	T → FT′	T → FT′				
T′			T′ → ε	T′ → ∗FT′	T′ → ε	T′ → ε
F	F → id	F → (E)				

Rules:

1. All the ε-productions are placed under FOLLOW sets.
2. Remaining productions are placed under the FIRSTs.

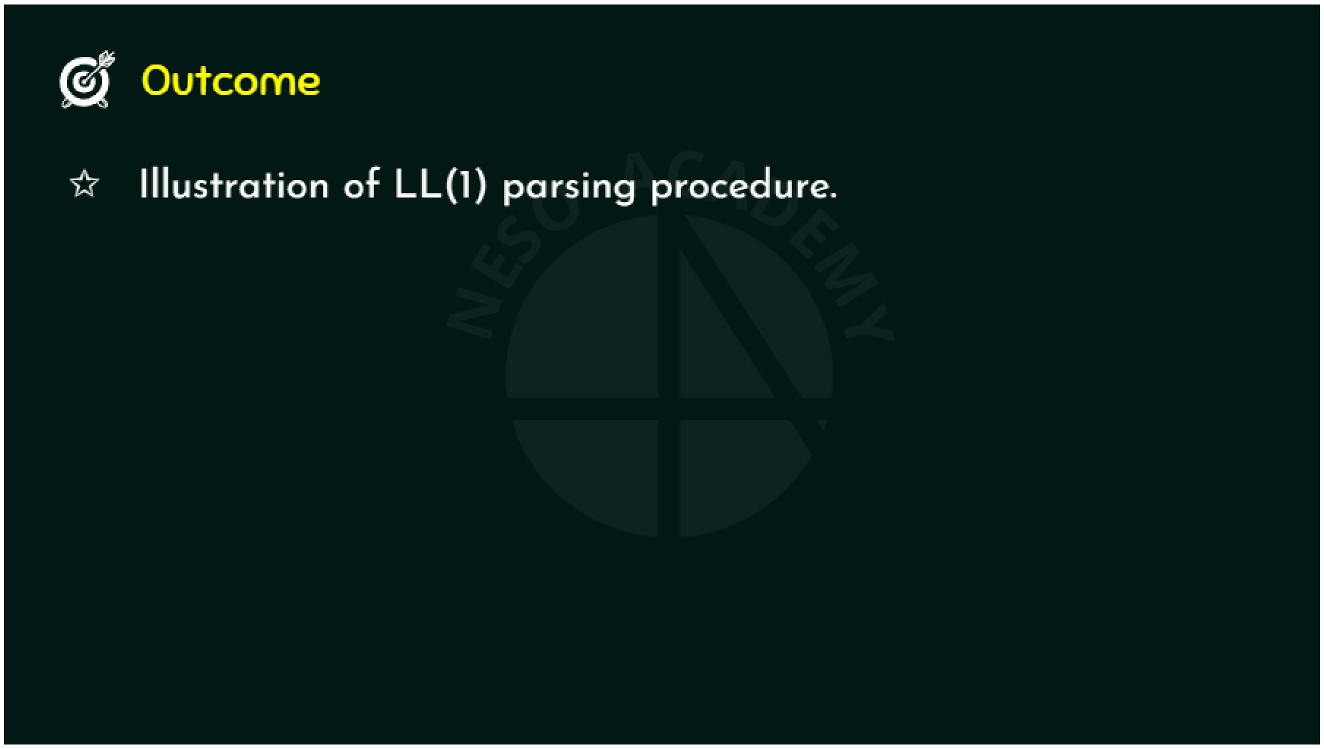
	FIRST	FOLLOW
E → TE′	{id,(}	{$,)}
E′ → +TE′ I ε	{+,ε}	{$,)}
T → FT′	{id,(}	{+,$,)}
T′ → ∗FT′ I ε	{∗,ε}	{+,$,)}
F → id I (E)	{id,(}	{∗,+,$,)}

Construction of LL(1) Parsing table:EE’TT’FF → idE → TE’T → FT’T’ → +FT’*E’ → εT’ → εE’ → +TE’T’ → εE’ → εT’ → εE → TE’T → FT’F → (E)id()*+$Rules:1.All the ε-productions are placed under FOLLOW sets.2.Remaining productions are placed under the FIRSTs.

Compiler Design

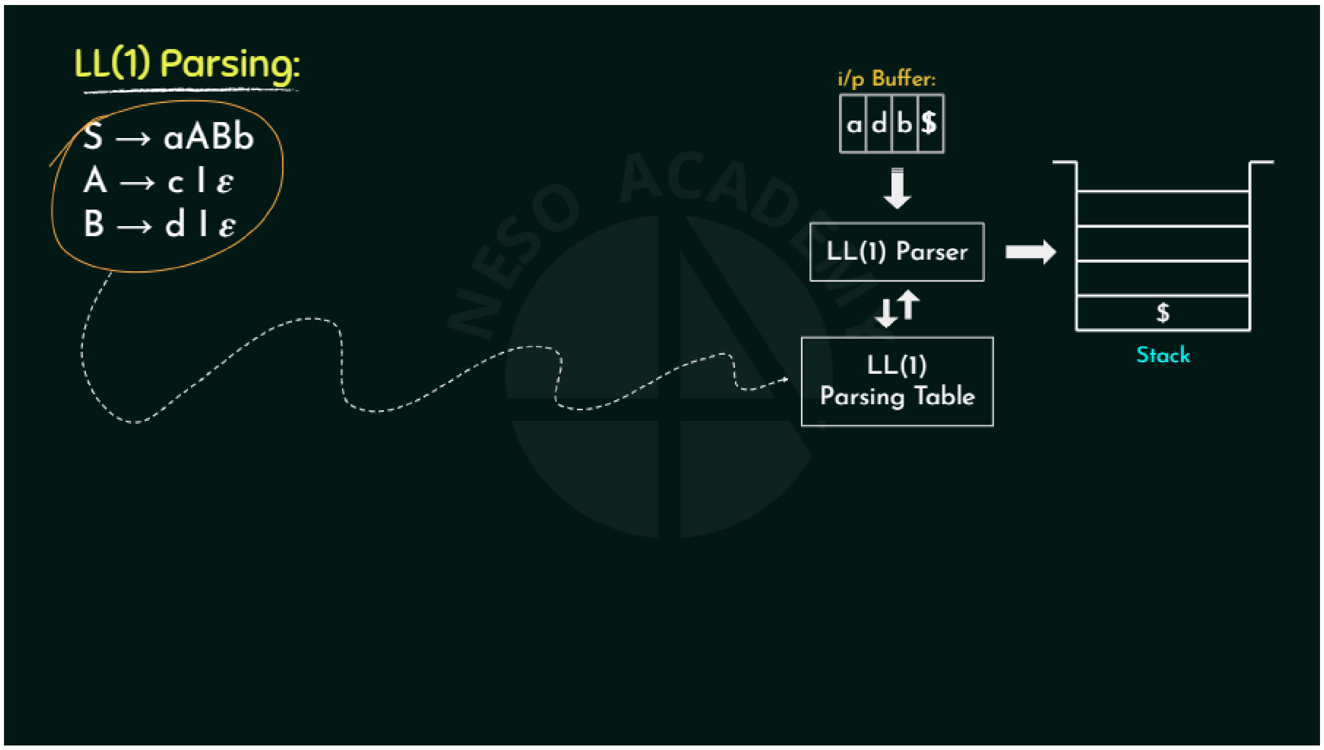
LL(1) Parsing

Compiler DesignLL(1) Parsing


Outcome

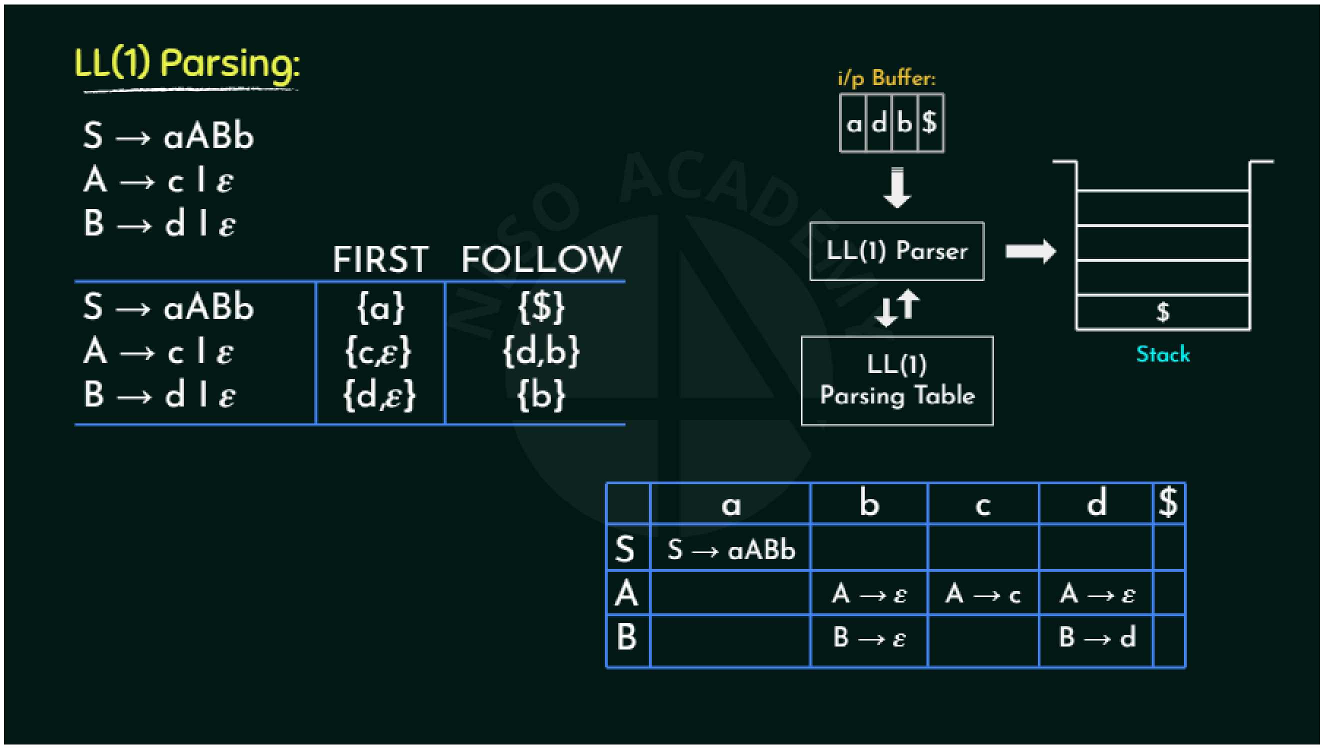
☆ Illustration of LL(1) parsing procedure.

Outcome☆Illustration of LL(1) parsing procedure.


LL(1) Parsing:

S → aABb
A → c | ε
B → d | ε

LL(1) Parsing:$LL(1) ParserLL(1) Parsing TableStackS → aABbA → c I εB → d I εa d b $i/p Buffer:$

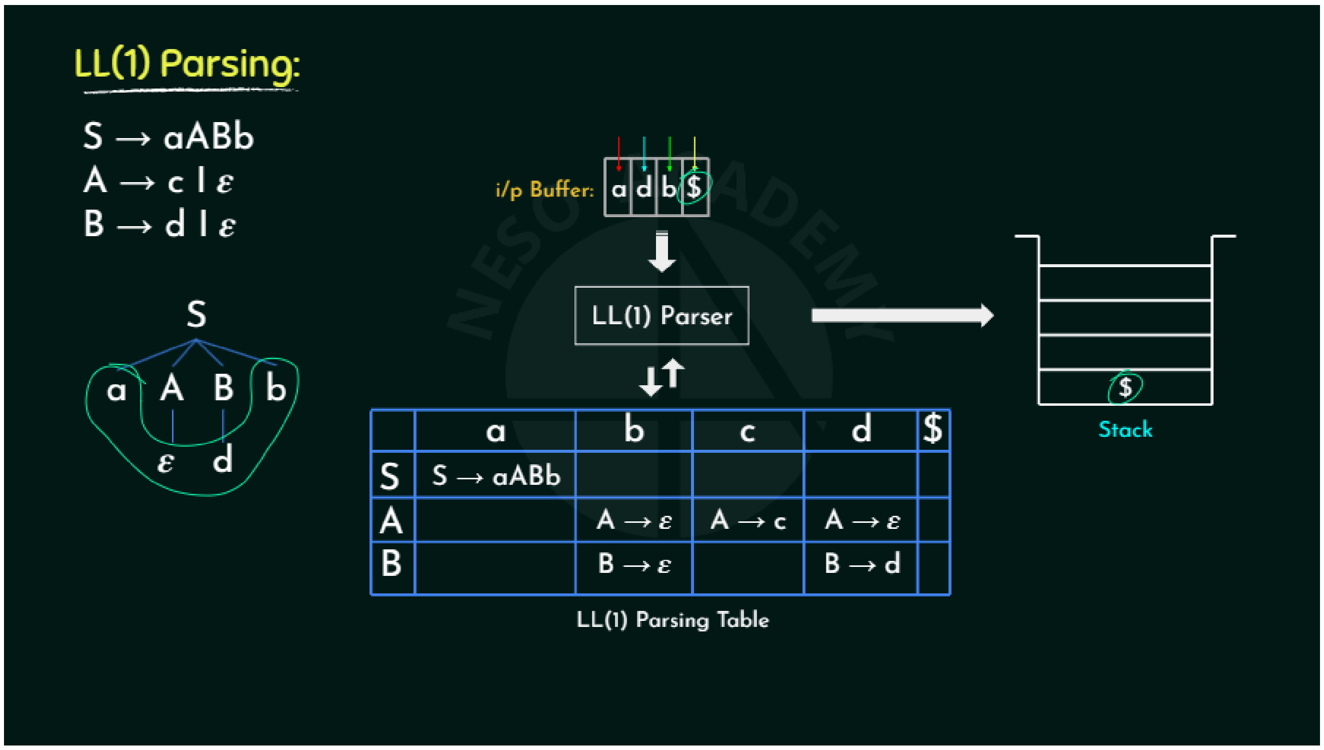

LL(1) Parsing:

S → aABb
A → c | ε
B → d | ε

	FIRST	FOLLOW
S → aABb	{a}	{$}
A → c | ε	{c,ε}	{d,b}
B → d | ε	{d,ε}	{b}

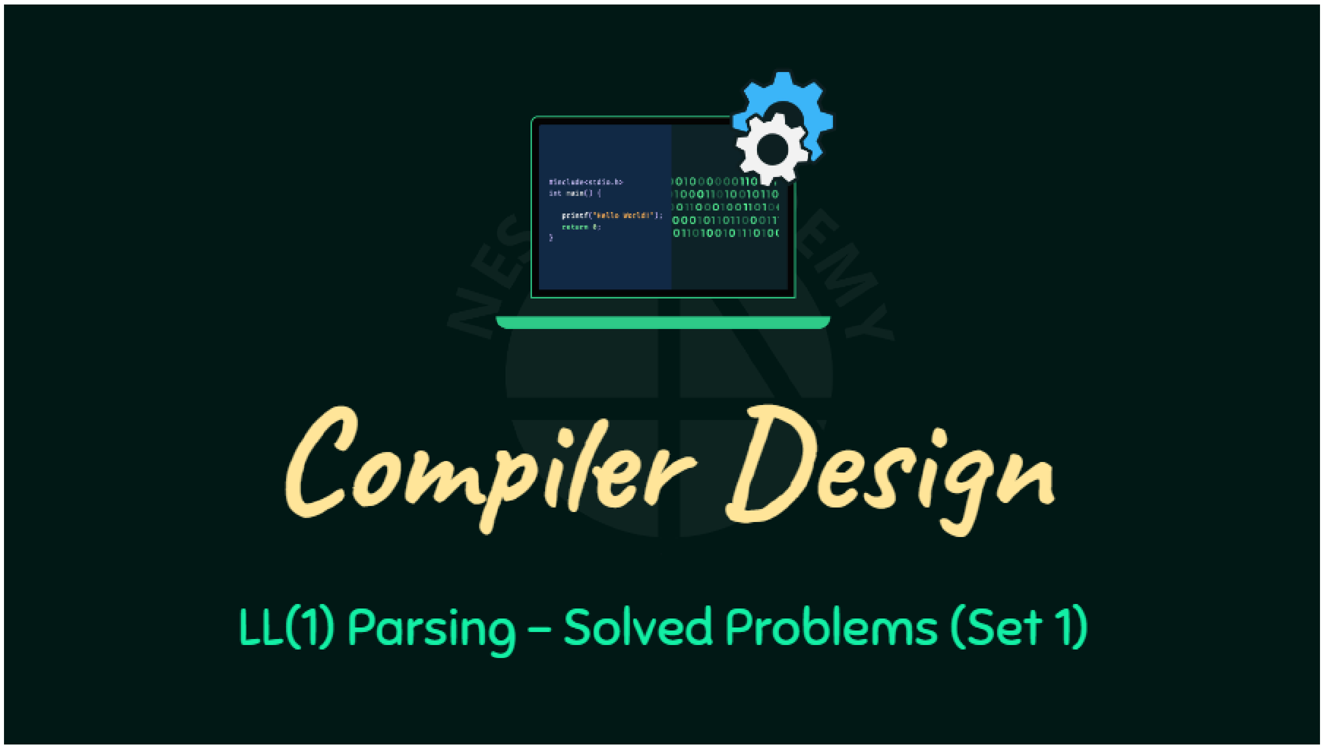
	a	b	c	d	$
S	S → aABb				
A		A → ε	A → c	A → ε	
B		B → ε		B → d	

S → aABbA → εLL(1) Parsing:$LL(1) ParserLL(1) Parsing TableStackS → aABbA → c I εB → d I εa d b $i/p Buffer:{dS → aABbA → c I εB → d I εFIRSTFOLLOW{$}{b}{a},ε}{c{d,ε},b}aSABA → εB → ε$bcdA → cB → d

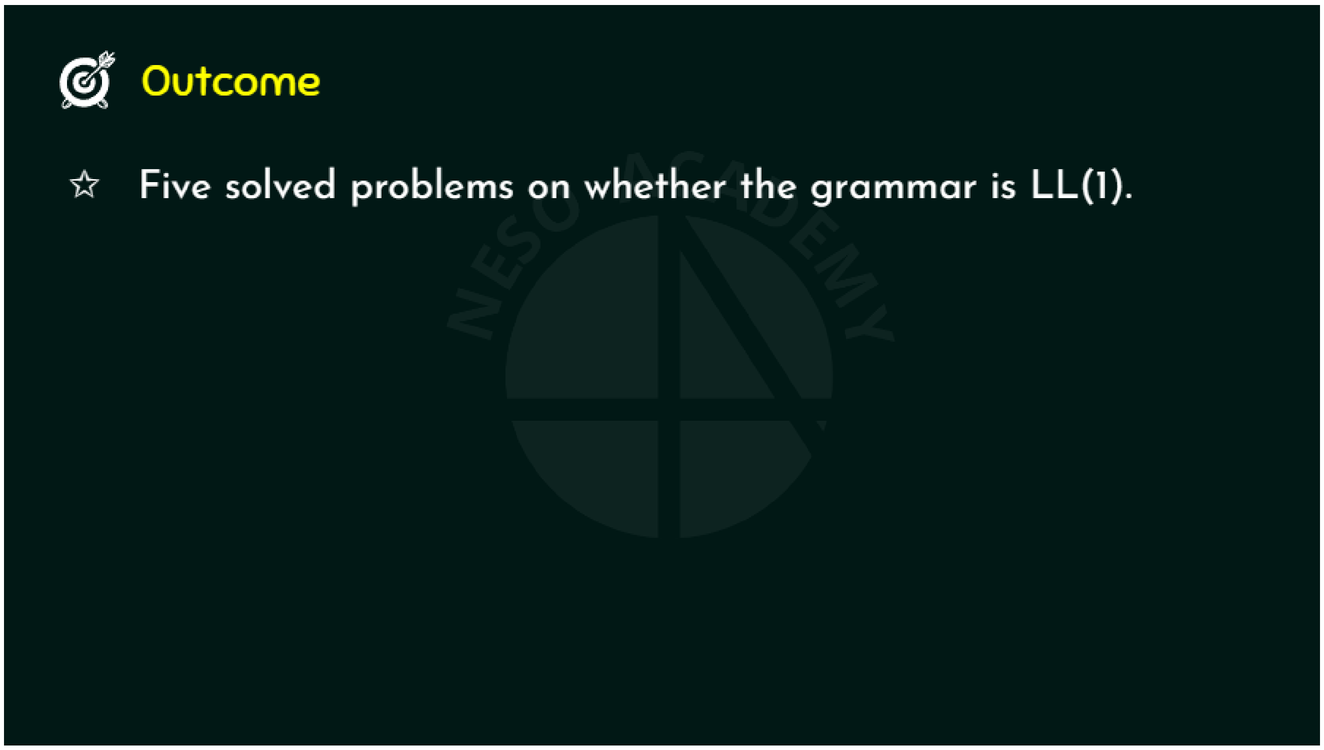


	a	b	c	d	$
S	S → aABb				
A		A → ε	A → c	A → ε	
B		B → ε		B → d	

LL(1) Parsing:$LL(1) ParserLL(1) Parsing TableStackS → aABbA → c I εB → d I εa d b $i/p Buffer:aS → aABbdA → εB → dSAB$cA → cbA → εB → εSaABbεd



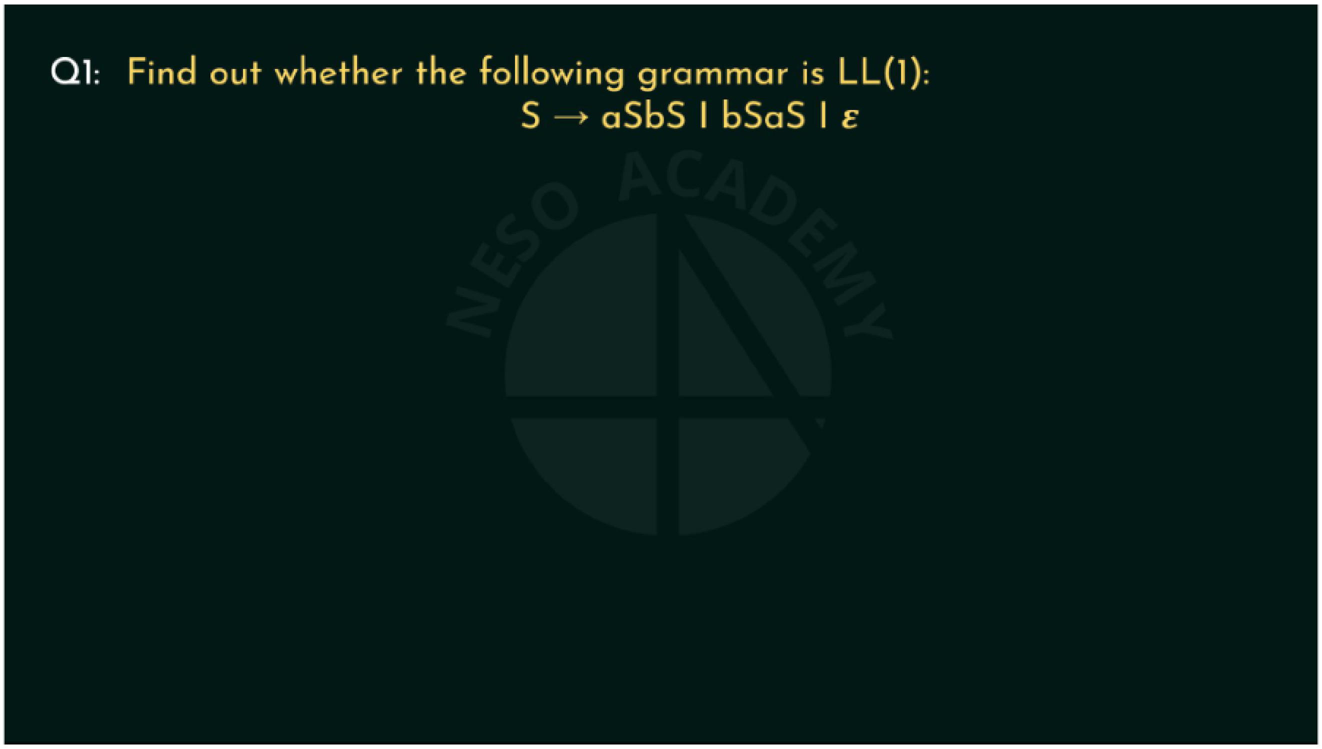
Compiler DesignLL(1) Parsing - Solved Problems (Set 1)



Outcome

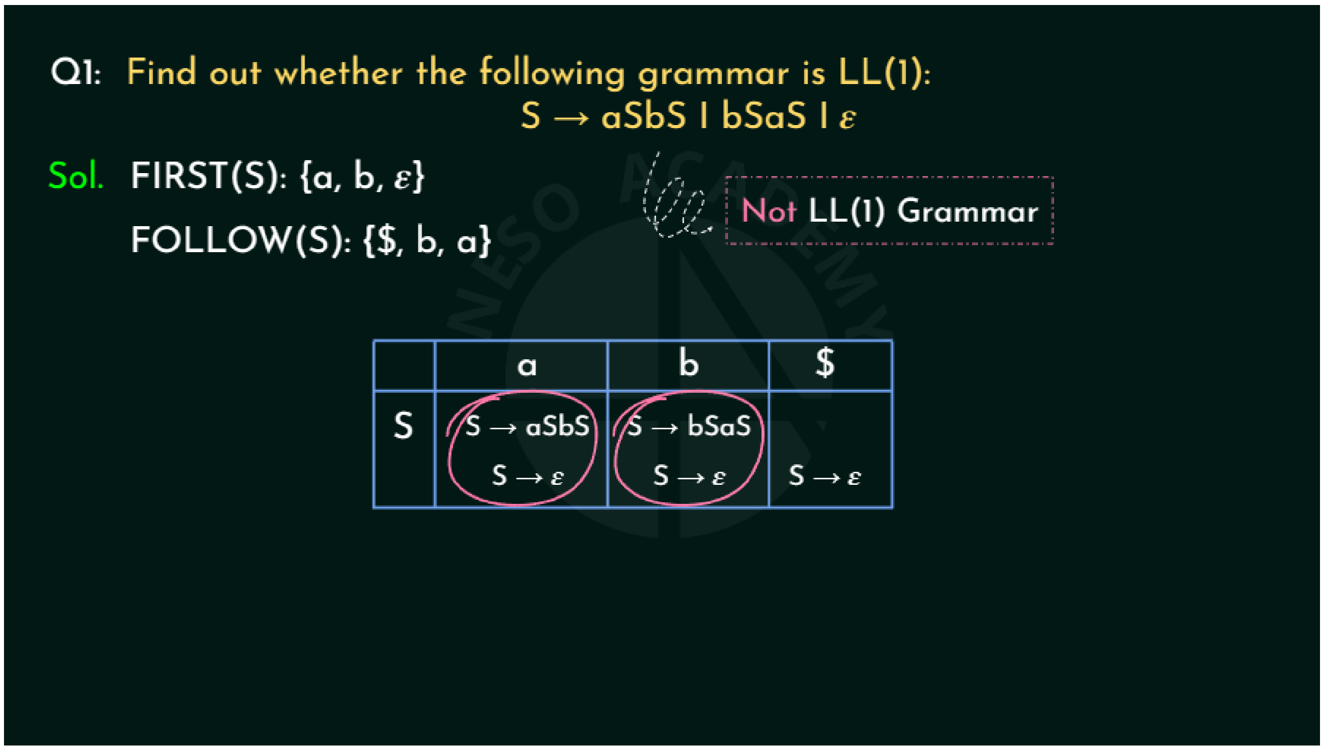
☆ Five solved problems on whether the grammar is LL(1).

Outcome☆Five solved problems on whether the grammar is LL(1).



Q1: Find out whether the following grammar is LL(1):

$$S \rightarrow aSbS \mid bSaS \mid \varepsilon$$



Q1: Find out whether the following grammar is LL(1):

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

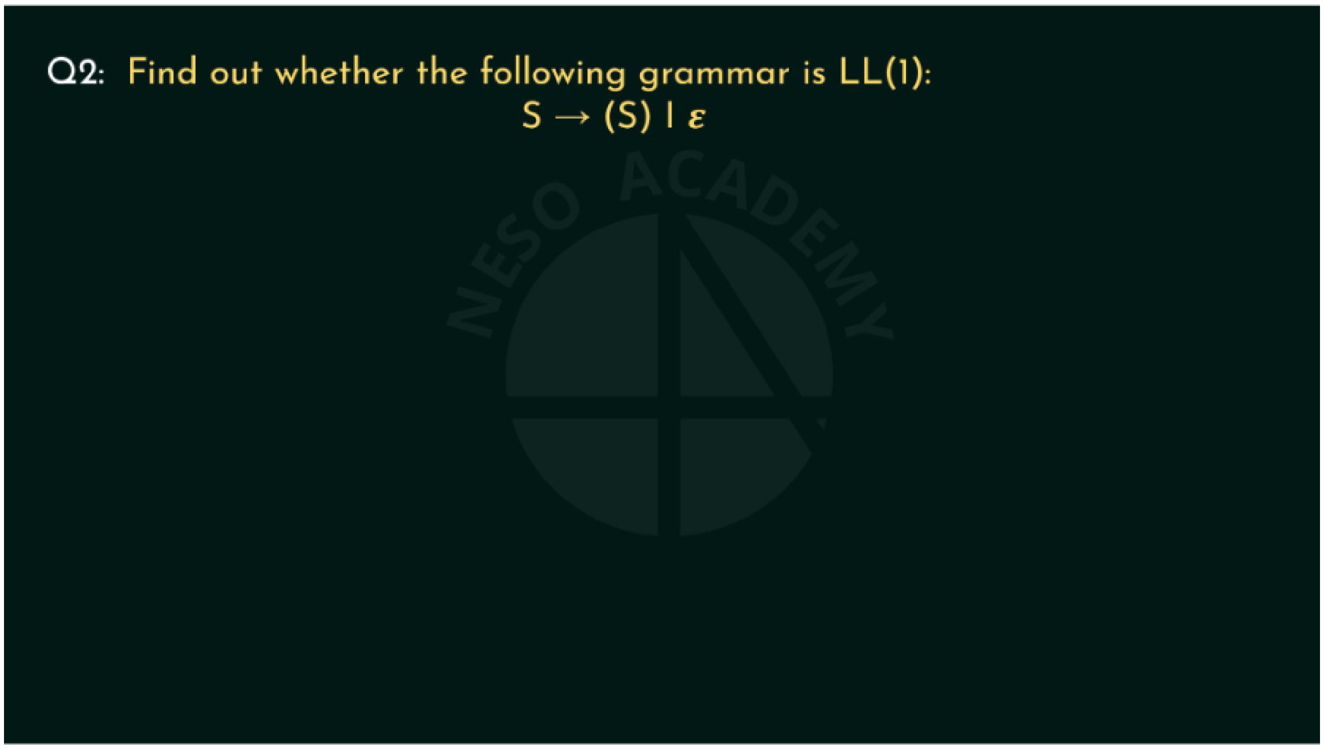
Sol. FIRST(S): $\{a, b, \varepsilon\}$

FOLLOW(S): $\{\$, b, a\}$

Not LL(1) Grammar

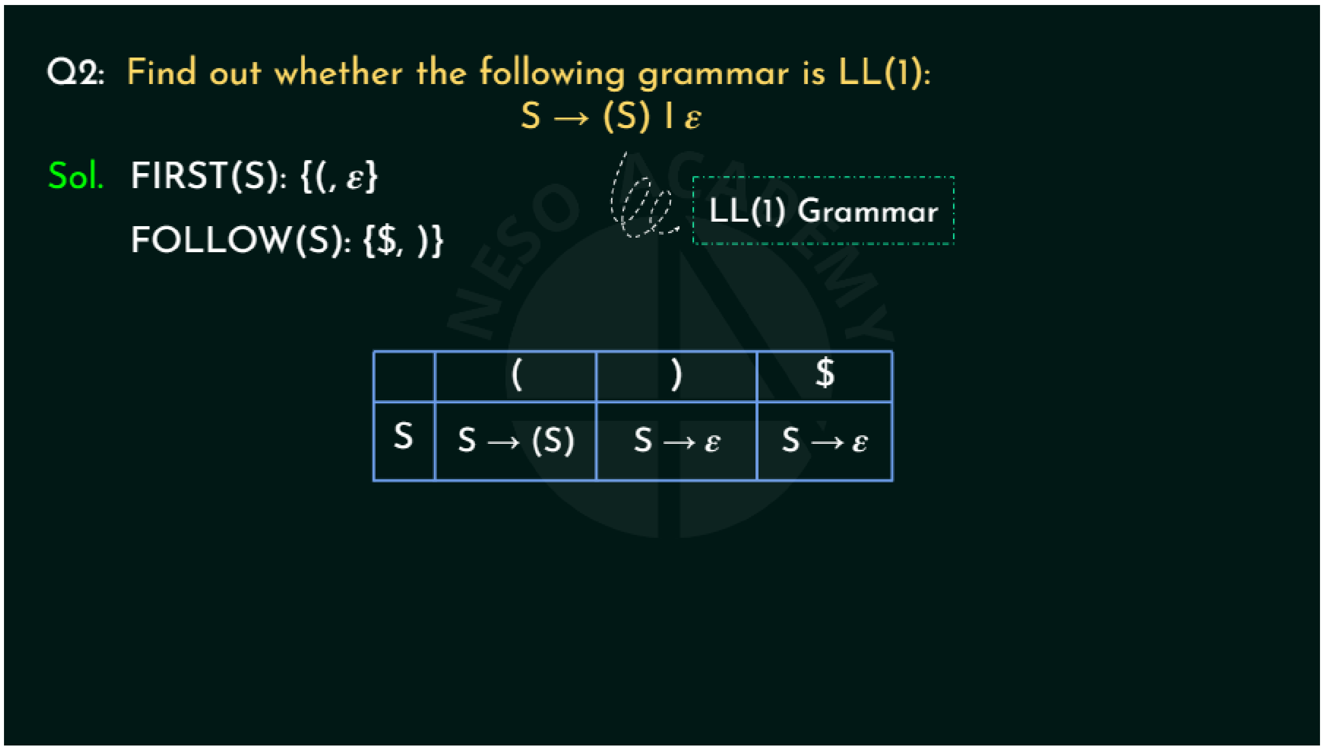
	a	b	$
S	$S \rightarrow aSbS$ $S \rightarrow \varepsilon$	$S \rightarrow bSaS$ $S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$

{$, ε} Find out whether the following grammar is LL(1):S → aSbS I bSaS I ε Q1:Sol.FIRST(S): {a, bFOLLOW(S): , b, a}S → aSbSS → ε S → bSaSS → ε S → ε aS$bNot LL(1) Grammar



Q2: Find out whether the following grammar is LL(1):

$S \rightarrow (S) \mid \varepsilon$



Q2: Find out whether the following grammar is LL(1):

$S \rightarrow (S) \mid \varepsilon$

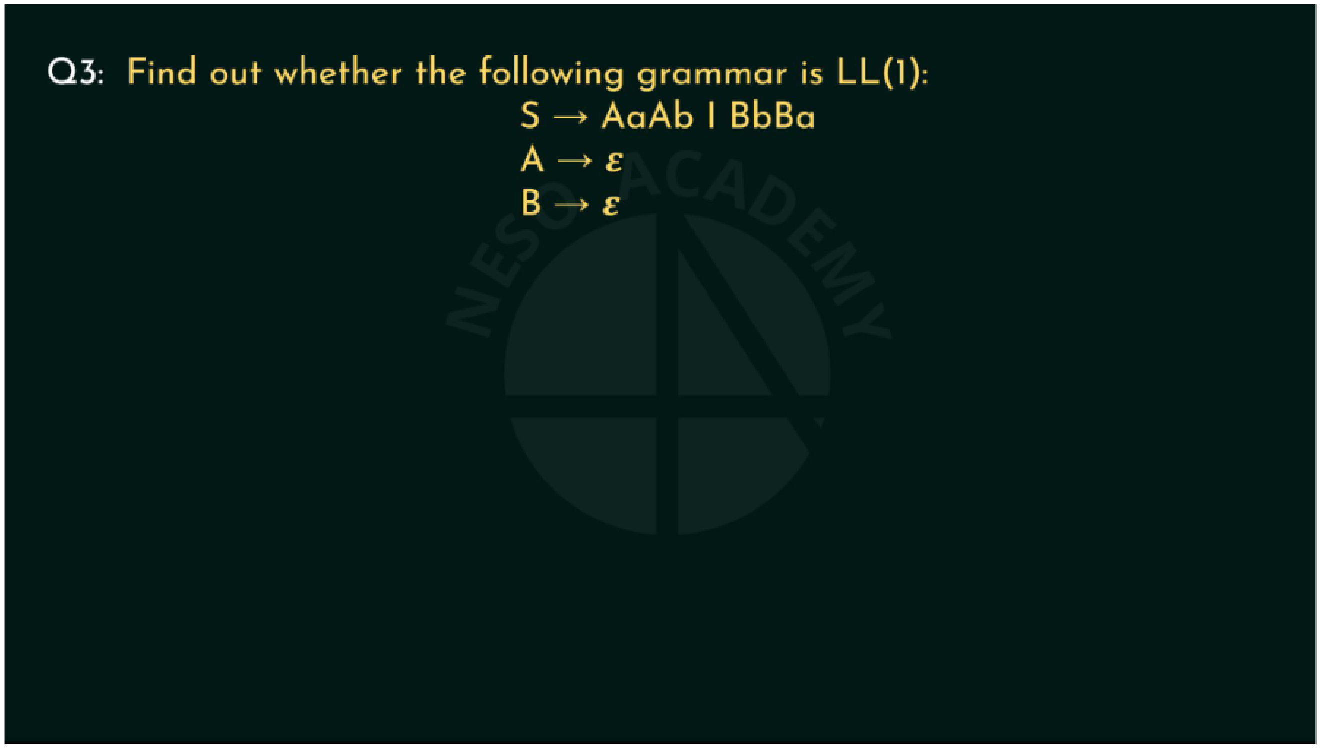
Sol. FIRST(S): {(, ε}

FOLLOW(S): {$,)}

LL(1) Grammar

	(	)	$
S	$S \rightarrow (S)$	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$

FOLLOW(S): {$, ε} Find out whether the following grammar is LL(1):S → (S) I ε
Q2:Sol.FIRST(S): {(,)}S → (S)S → ε S → ε (S$)LL(1) Grammar

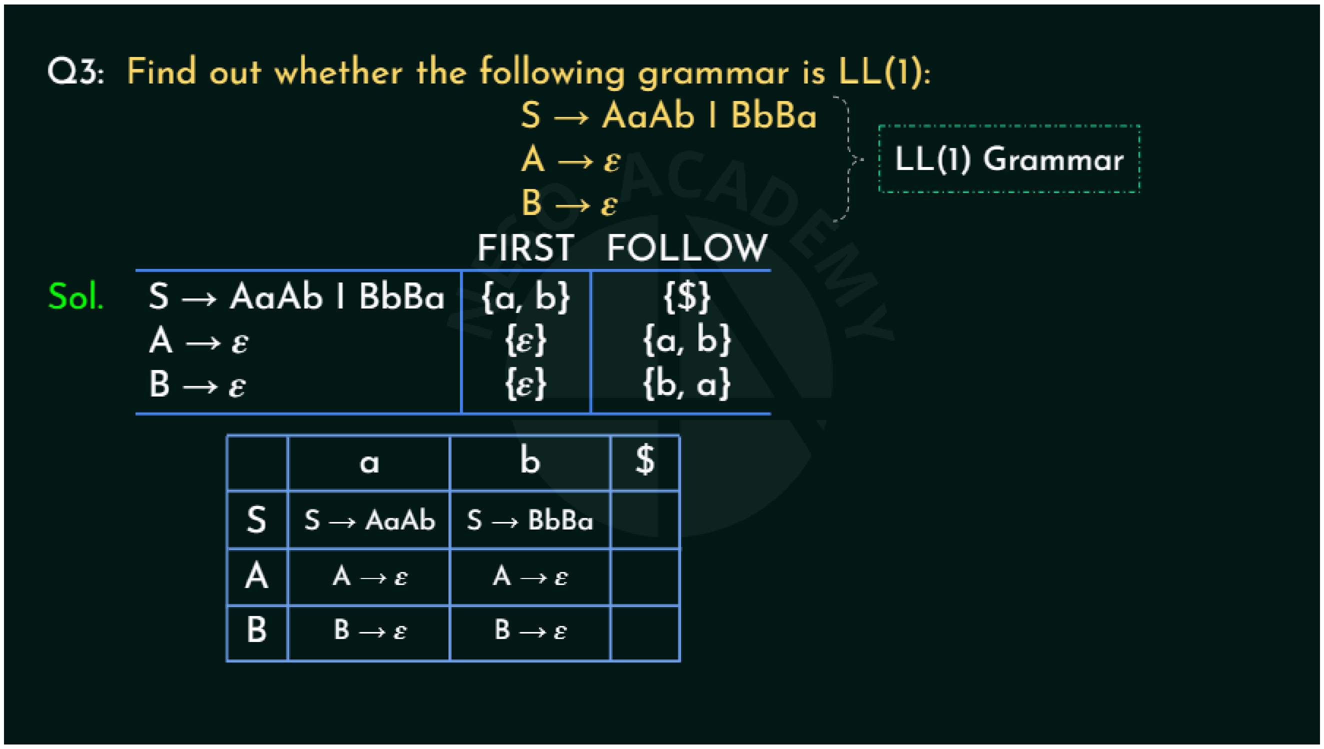


Q3: Find out whether the following grammar is LL(1):

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \varepsilon$

$B \rightarrow \varepsilon$



Q3: Find out whether the following grammar is LL(1):

$S \rightarrow AaAb \mid BbBa$
$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$

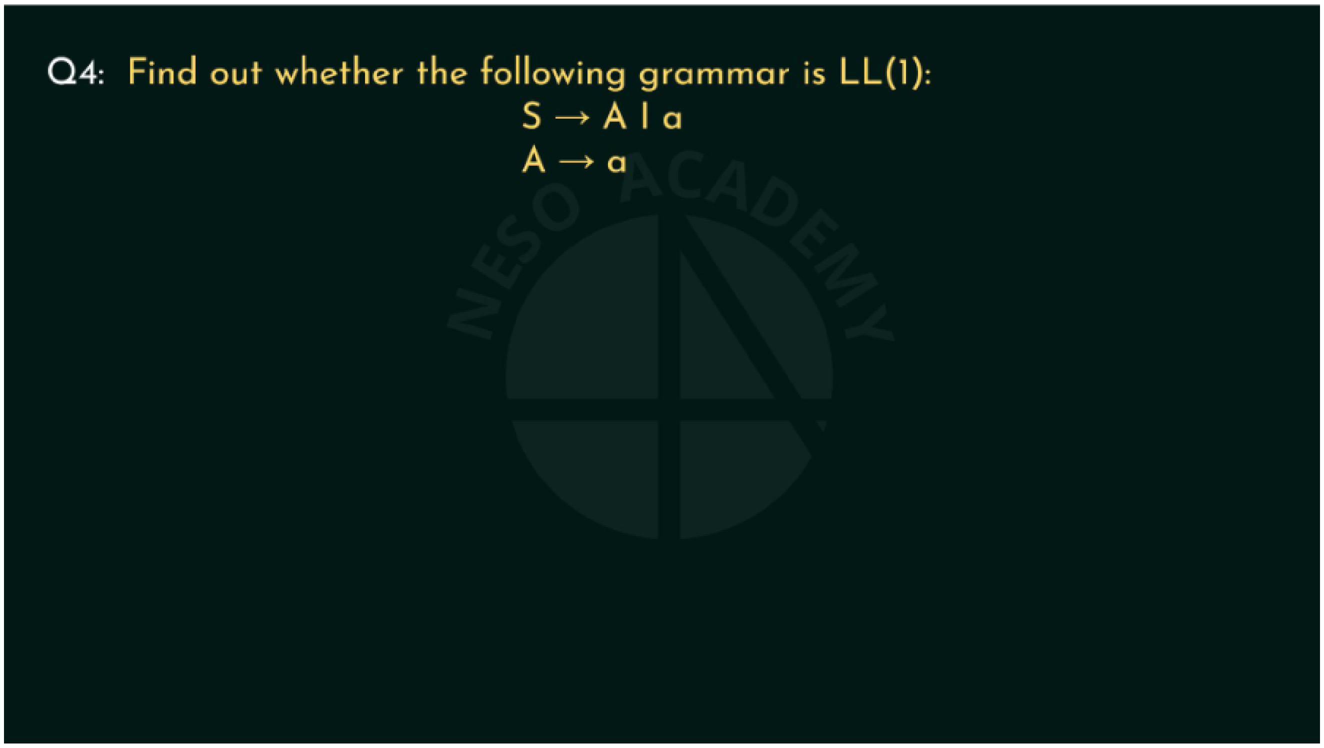
LL(1) Grammar

Sol.

	FIRST	FOLLOW
$S \rightarrow AaAb \mid BbBa$	{a, b}	{\$}
$A \rightarrow \varepsilon$	{ε}	{a, b}
$B \rightarrow \varepsilon$	{ε}	{b, a}

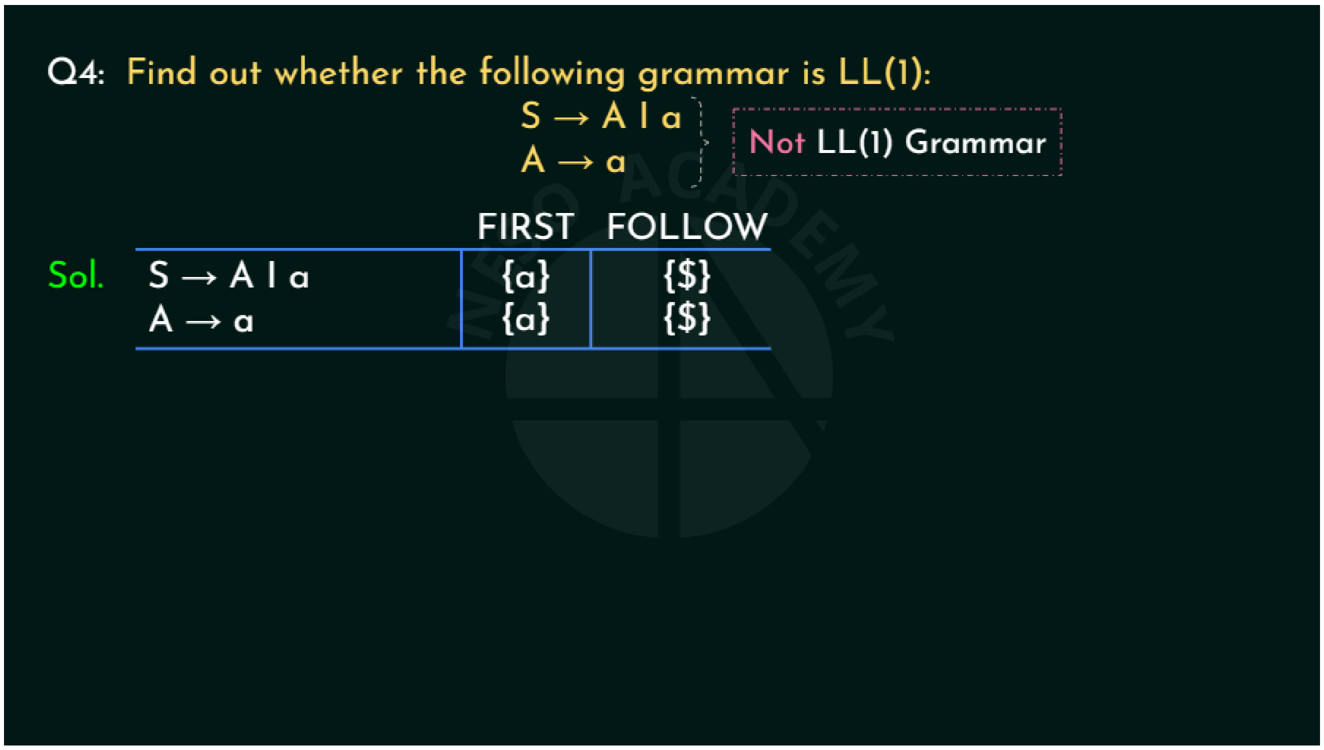
	a	b	$
S	$S \rightarrow AaAb$	$S \rightarrow BbBa$	
A	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$	
B	$B \rightarrow \varepsilon$	$B \rightarrow \varepsilon$	

{aFind out whether the following grammar is LL(1):S → AaAb I BbBaA → εB → εQ3:Sol.S → AaAb I BbBaA → εB → εFIRSTFOLLOW{$}{ε}{ε}, b}{a, b}{b, a}LL(1) GrammarS → AaAbS → BbBa a$bSABA → ε B → ε A → ε B → ε



Q4: Find out whether the following grammar is LL(1):

$S \rightarrow A \mid a$
$A \rightarrow a$



Q4: Find out whether the following grammar is LL(1):

$S \rightarrow A \mid a$

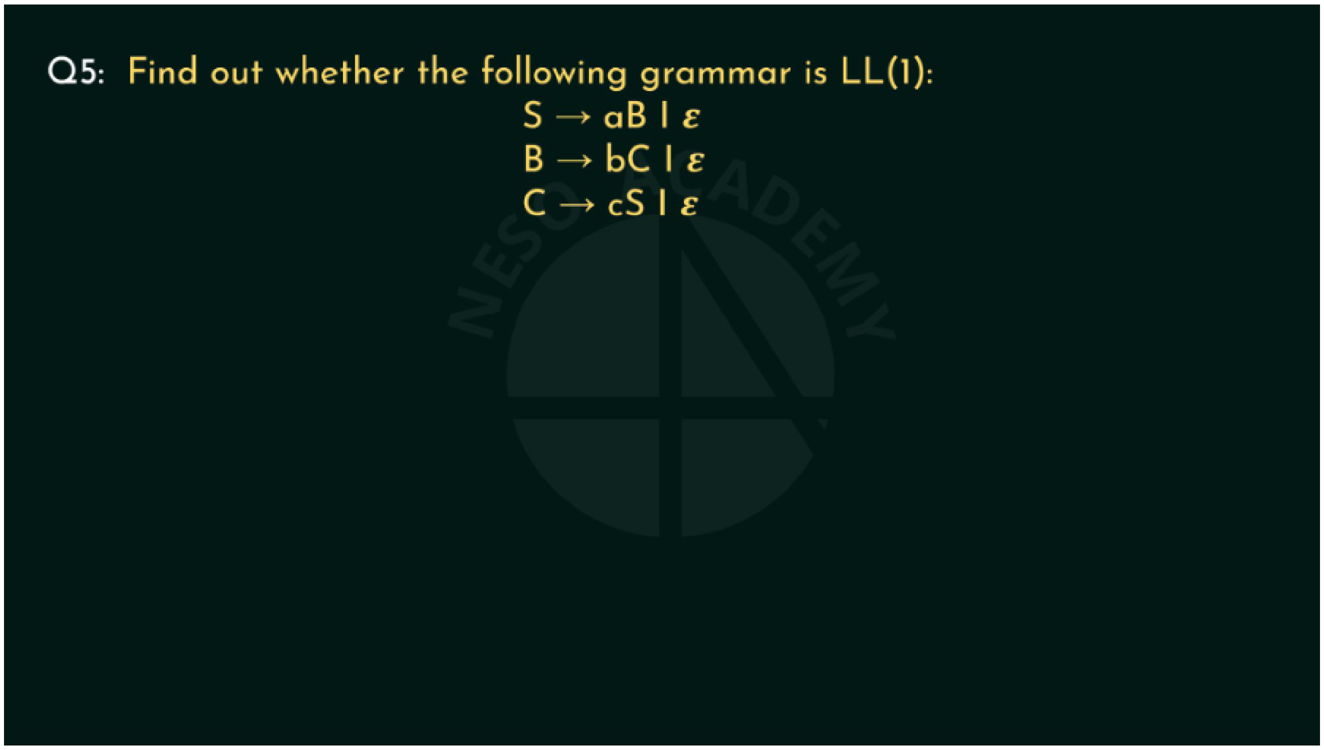
$A \rightarrow a$

Not LL(1) Grammar

Sol.

	FIRST	FOLLOW
$S \rightarrow A \mid a$	{a}	{\$}
$A \rightarrow a$	{a}	{\$}

S → A I aA → aFIRSTFOLLOW{a}Find out whether the following grammar is LL(1):S → A I aA → aQ4:Sol.{$}{a}Not LL(1) Grammar{$}

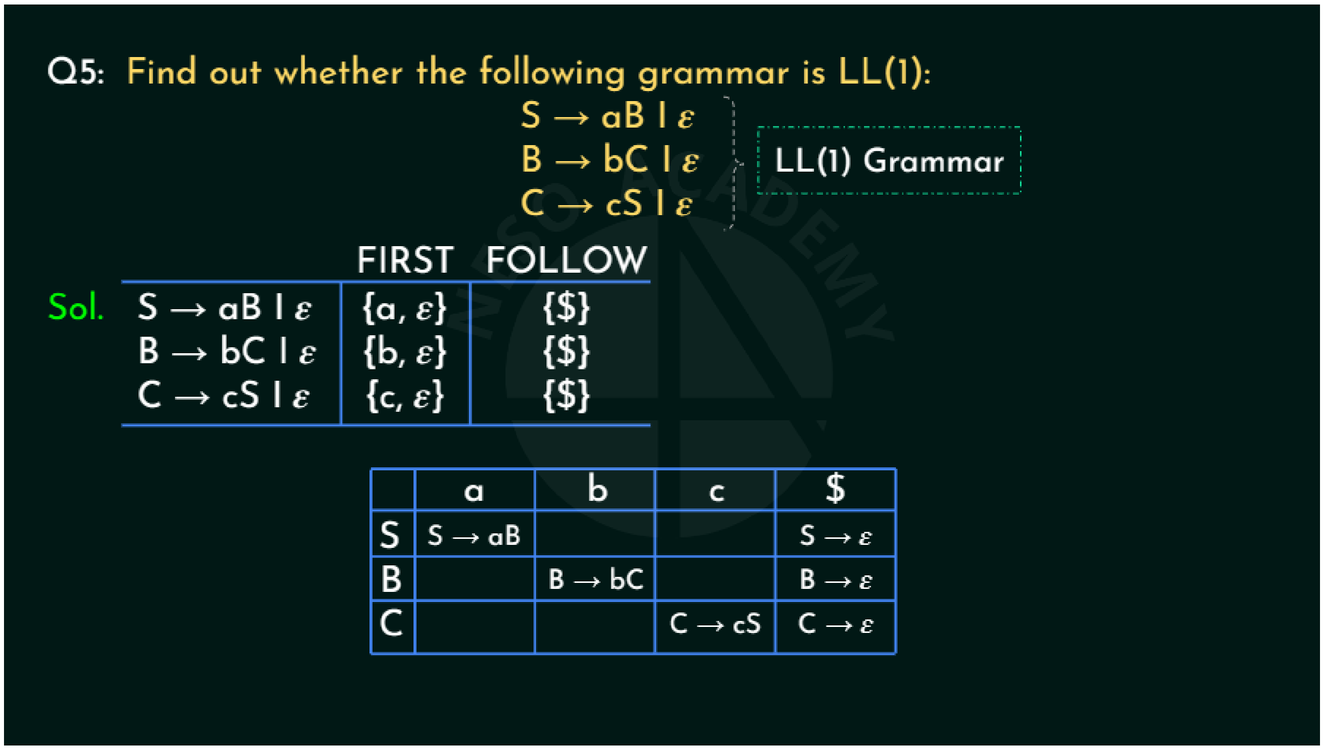


Q5: Find out whether the following grammar is LL(1):

$S \rightarrow aB \mid \varepsilon$

$B \rightarrow bC \mid \varepsilon$

$C \rightarrow cS \mid \varepsilon$



Q5: Find out whether the following grammar is LL(1):

S → aB | ε
B → bC | ε
C → cS | ε

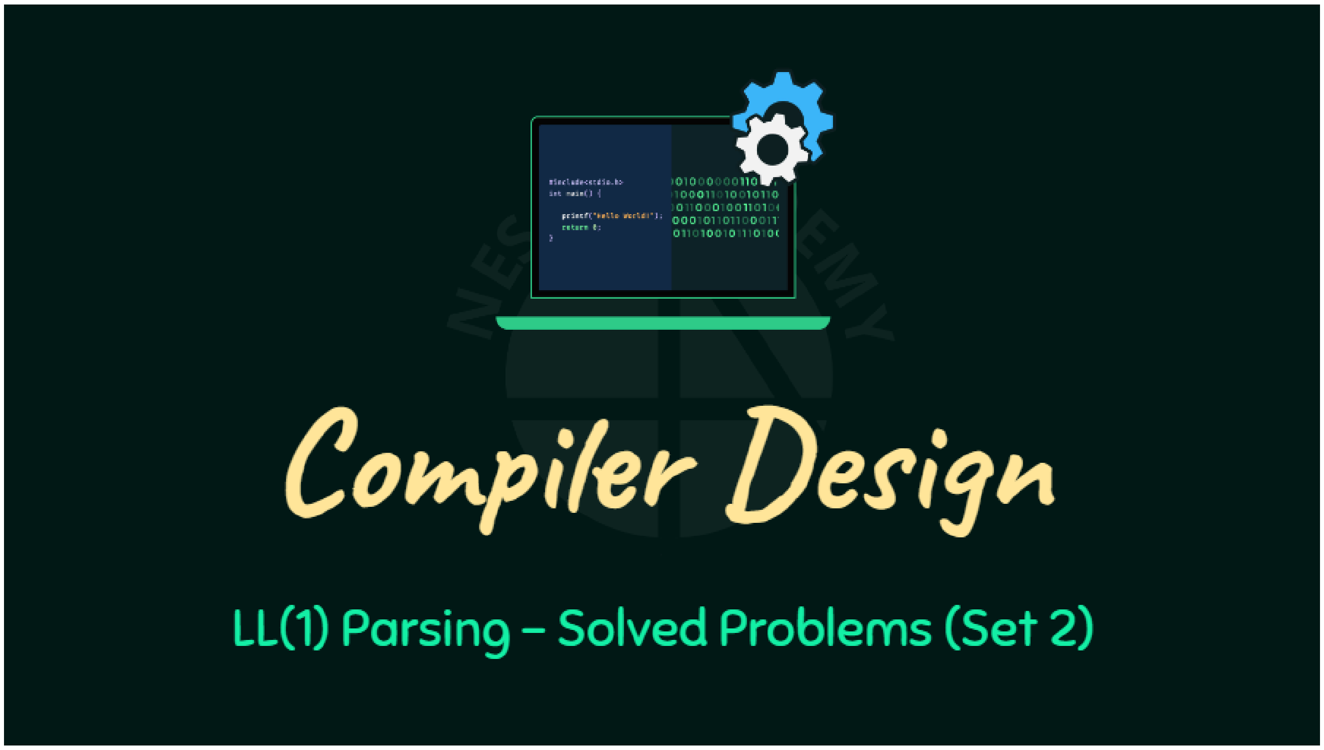
LL(1) Grammar

Sol.

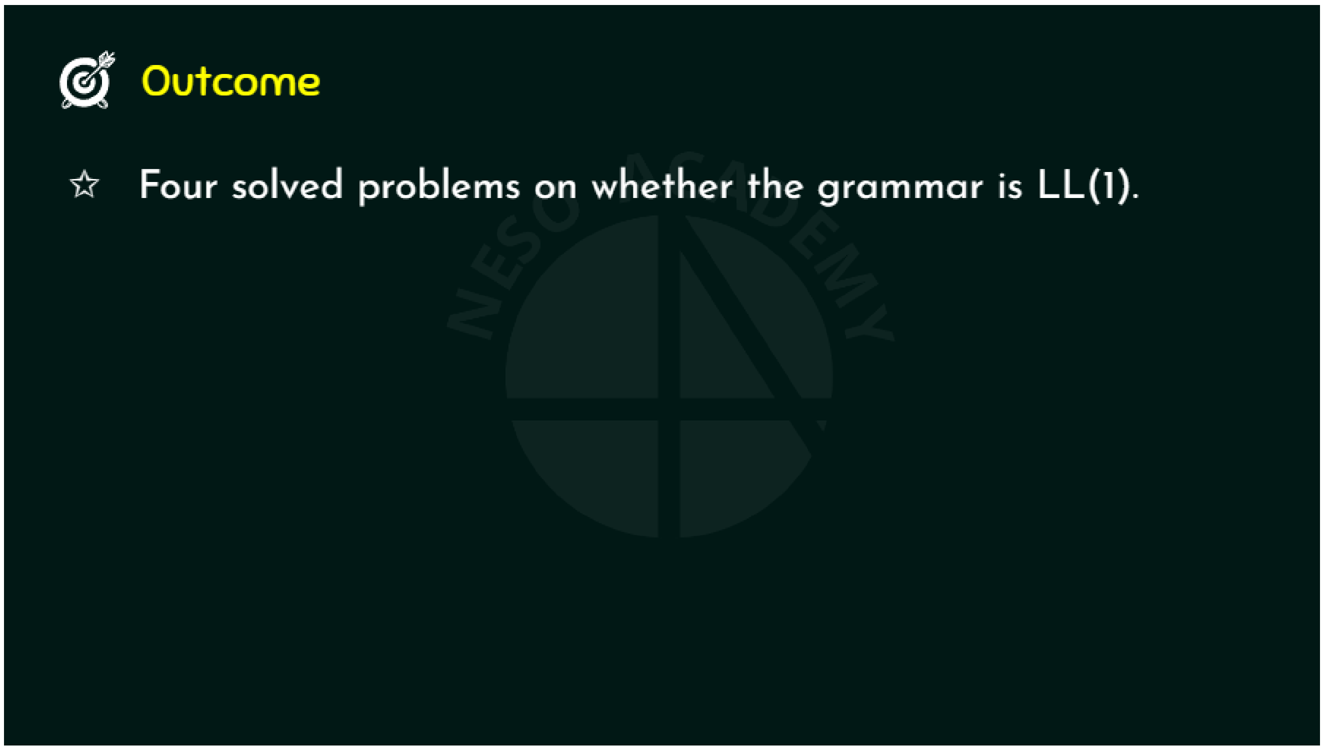
	FIRST	FOLLOW
S → aB | ε	{a, ε}	{$}
B → bC | ε	{b, ε}	{$}
C → cS | ε	{c, ε}	{$}

	a	b	c	$
S	S → aB			S → ε
B		B → bC		B → ε
C			C → cS	C → ε

S → aB I εB → bC I εC → cS I εFIRSTFOLLOW{aFind out whether the following grammar is LL(1):S → aB I εB → bC I εC → cS I εQ5:LL(1) GrammarSol.{$}{b, ε}{c, ε}, ε}{$}{$}S → aBaB → εC → εbc$SBCB → bCC → cSS → ε



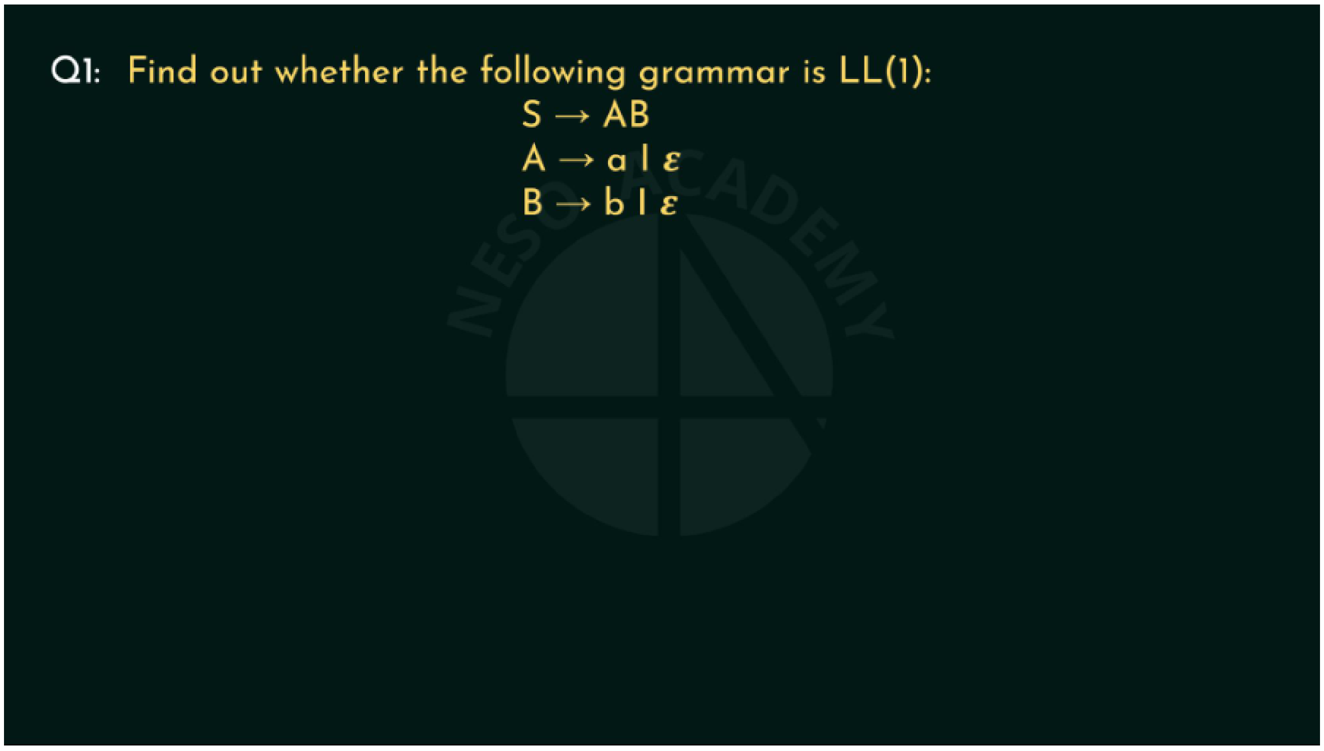
Compiler DesignLL(1) Parsing - Solved Problems (Set 2)



Outcome

☆ Four solved problems on whether the grammar is LL(1).

Outcome☆Four solved problems on whether the grammar is LL(1).

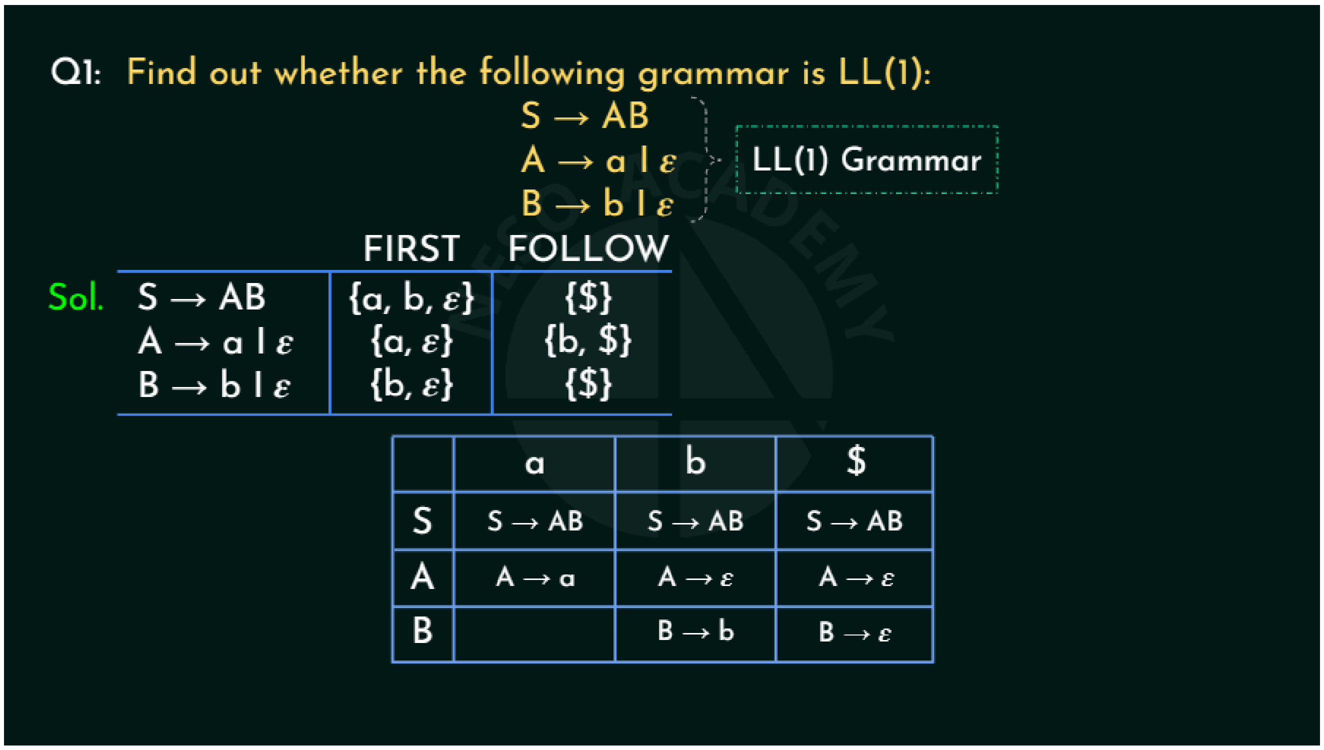


Q1: Find out whether the following grammar is LL(1):

$S \rightarrow AB$

$A \rightarrow a \mid \varepsilon$

$B \rightarrow b \mid \varepsilon$



Q1: Find out whether the following grammar is LL(1):

$S \rightarrow AB$
$A \rightarrow a \mid \varepsilon$
$B \rightarrow b \mid \varepsilon$

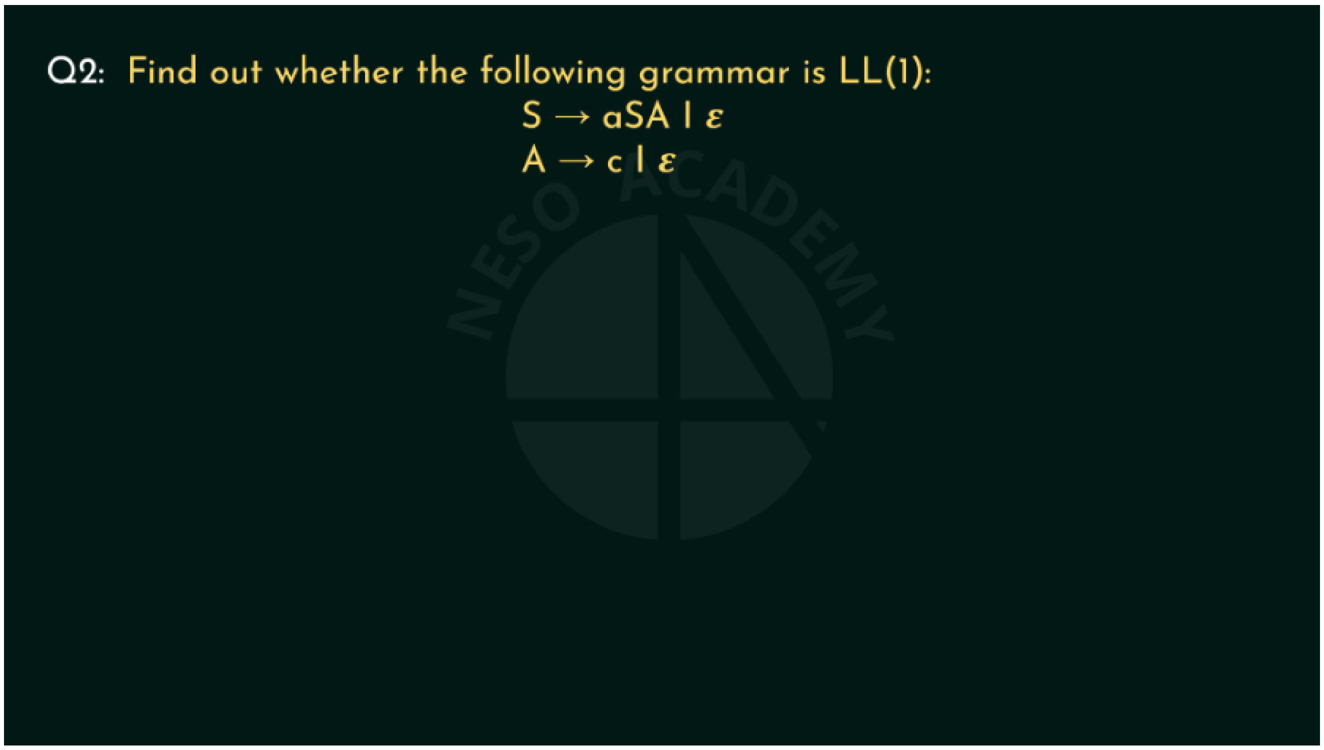
LL(1) Grammar

Sol.

	FIRST	FOLLOW
$S \rightarrow AB$	$\{a, b, \varepsilon\}$	$\{\$\}$
$A \rightarrow a \mid \varepsilon$	$\{a, \varepsilon\}$	$\{b, \$\}$
$B \rightarrow b \mid \varepsilon$	$\{b, \varepsilon\}$	$\{\$\}$

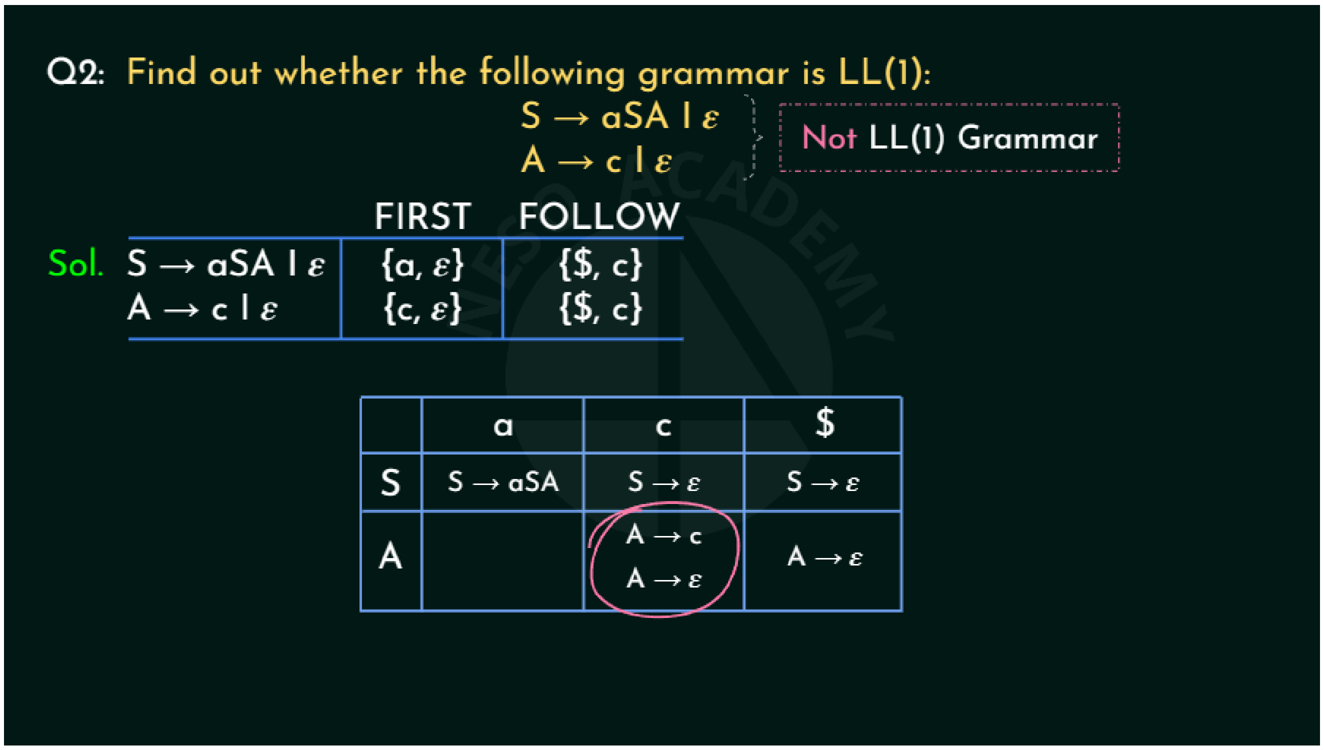
	a	b	\$
S	$S \rightarrow AB$	$S \rightarrow AB$	$S \rightarrow AB$
A	$A \rightarrow a$	$A \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
B		$B \rightarrow b$	$B \rightarrow \varepsilon$

{aS → AB A → a I εB → b I εFIRSTFOLLOW, $}{b, b, ε}Find out whether the following grammar is LL(1):S → AB A → a I εB → b I ε Q1:Sol.{$}{a, ε}{b, ε}{$}S → ABS → AB a$bSABA → ε A → a B → b A → ε B → ε LL(1) GrammarS → AB



Q2: Find out whether the following grammar is LL(1):

$S \rightarrow aSA \mid \varepsilon$
$A \rightarrow c \mid \varepsilon$



Q2: Find out whether the following grammar is LL(1):

$S \to aSA \mid \varepsilon$
$A \to c \mid \varepsilon$

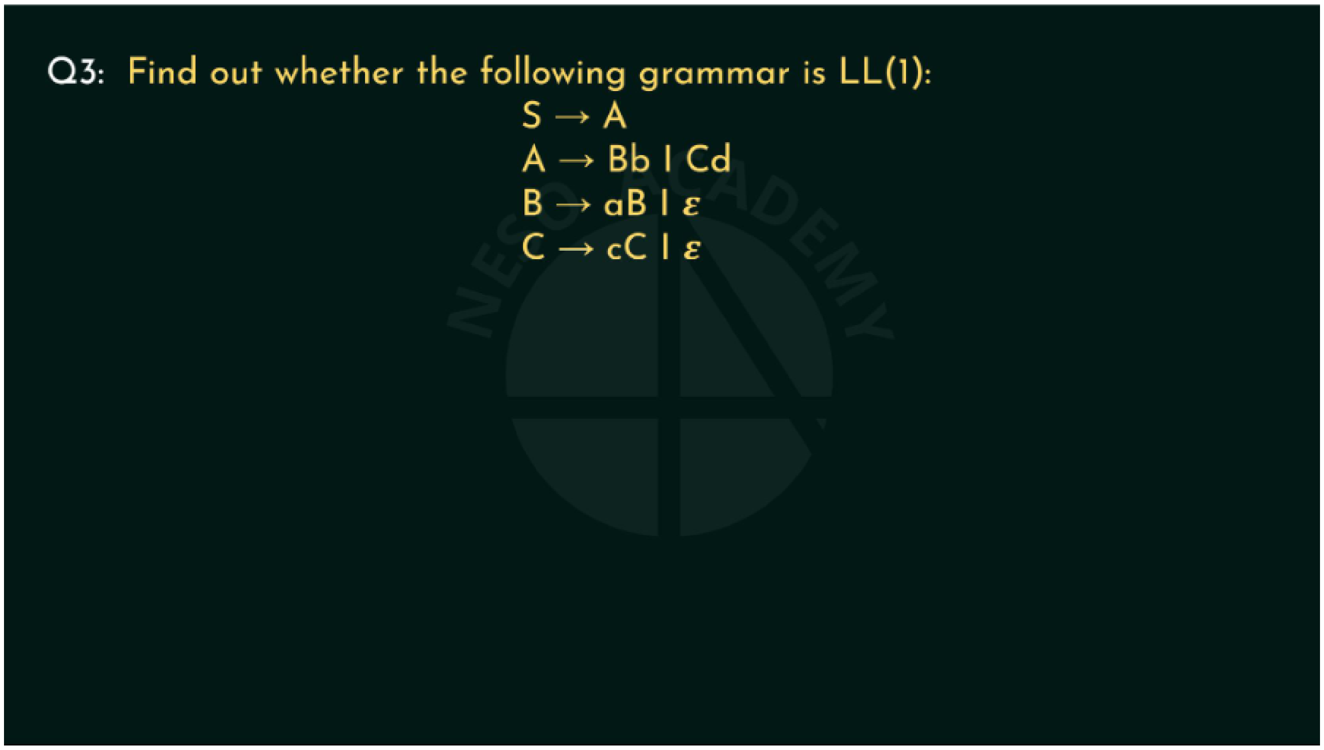
Not LL(1) Grammar

Sol.

	FIRST	FOLLOW
$S \to aSA \mid \varepsilon$	$\{a, \varepsilon\}$	$\{\$, c\}$
$A \to c \mid \varepsilon$	$\{c, \varepsilon\}$	$\{\$, c\}$

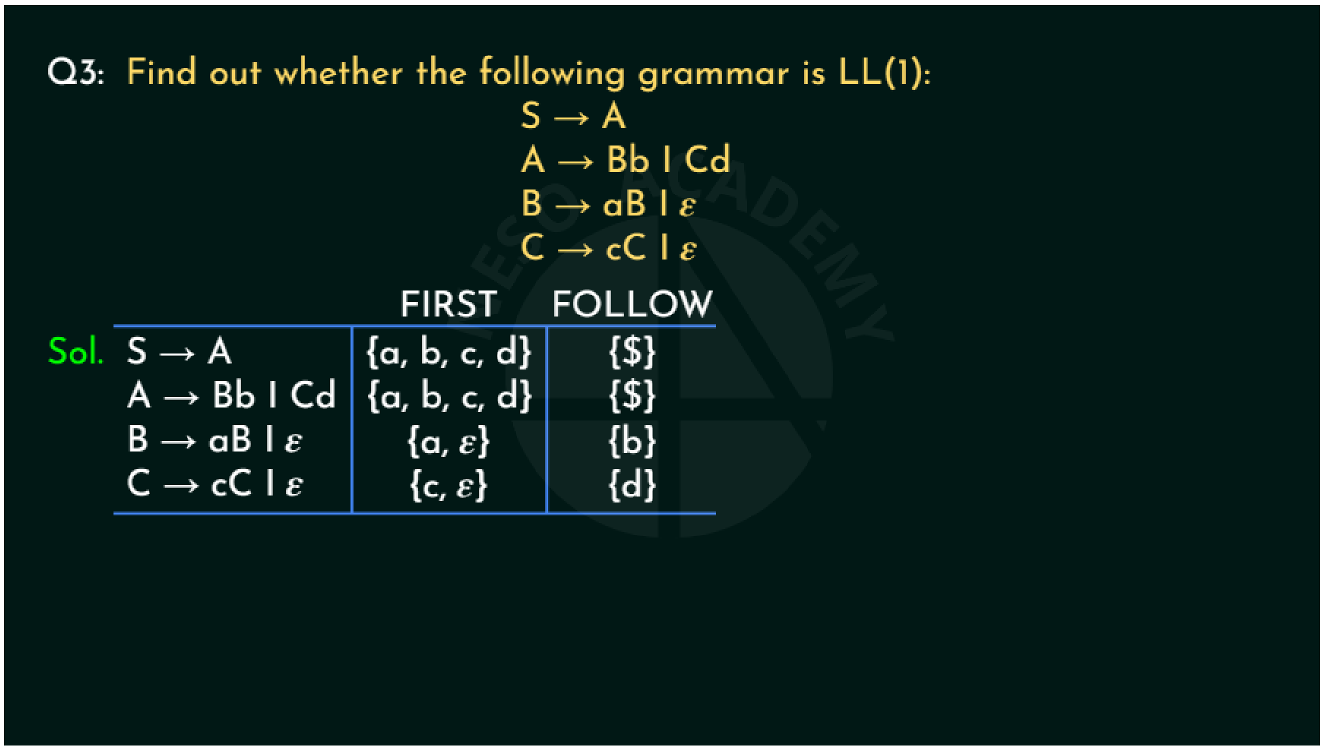
	a	c	\$
S	$S \to aSA$	$S \to \varepsilon$	$S \to \varepsilon$
A		$A \to c$ $A \to \varepsilon$	$A \to \varepsilon$

S → aSA I εA → c I εFIRSTFOLLOWFind out whether the following grammar is LL(1):S → aSA I ε A → c I εQ2:Sol.{\$, c}{a, ε}{c, ε}S → aSAS → ε a\$cSAA → ε A → c A → ε Not LL(1) Grammar{\$, c}S → ε



Q3: Find out whether the following grammar is LL(1):

$S \to A$
$A \to Bb \mid Cd$
$B \to aB \mid \varepsilon$
$C \to cC \mid \varepsilon$



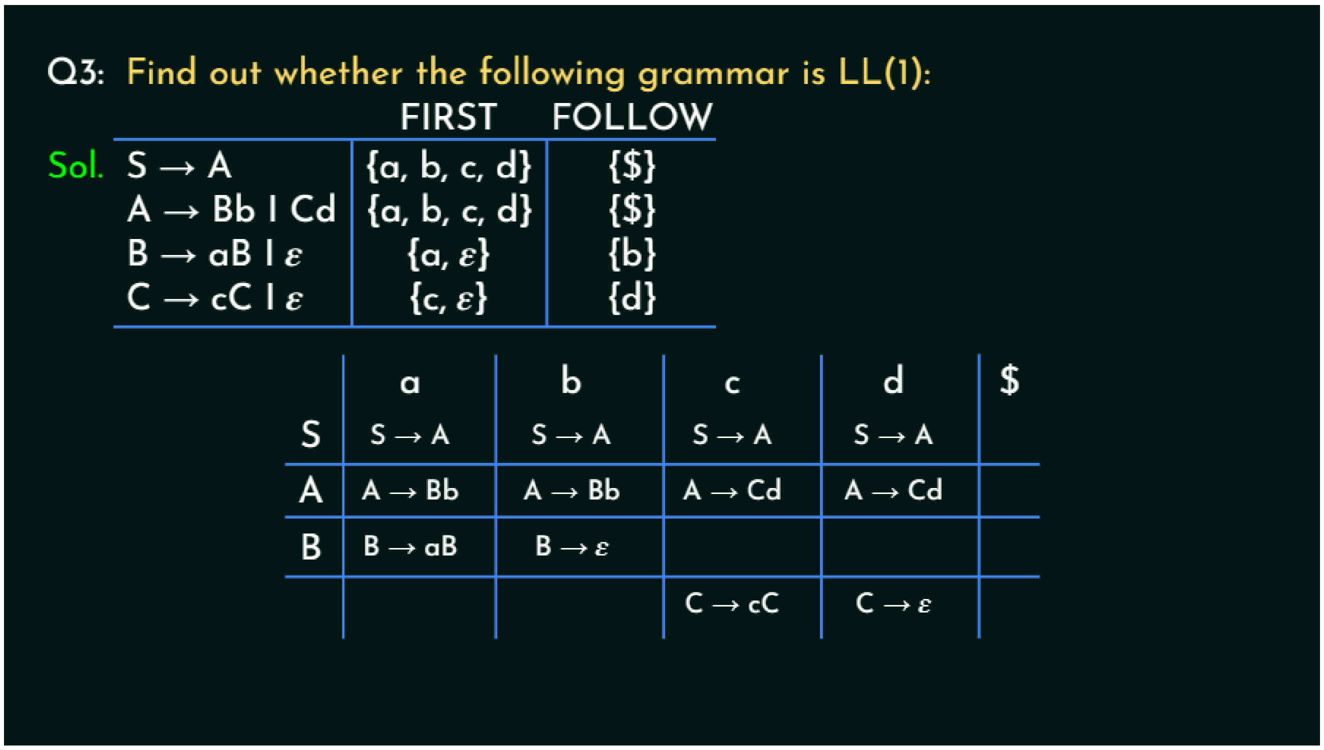
Q3: Find out whether the following grammar is LL(1):

$S \rightarrow A$
$A \rightarrow Bb \mid Cd$
$B \rightarrow aB \mid \varepsilon$
$C \rightarrow cC \mid \varepsilon$

Sol.

	FIRST	FOLLOW
$S \rightarrow A$	{a, b, c, d}	{\$}
$A \rightarrow Bb \mid Cd$	{a, b, c, d}	{\$}
$B \rightarrow aB \mid \varepsilon$	{a, ε}	{b}
$C \rightarrow cC \mid \varepsilon$	{c, ε}	{d}

Find out whether the following grammar is LL(1):S → AA → Bb I CdB → aB I εC → cC I εQ3:Sol.S → AA → Bb I CdB → aB I εC → cC I εFOLLOWFIRST{$}{b}{a, b, c, d}{a, ε}{c, ε}{a, b, c, d}{$}{d}



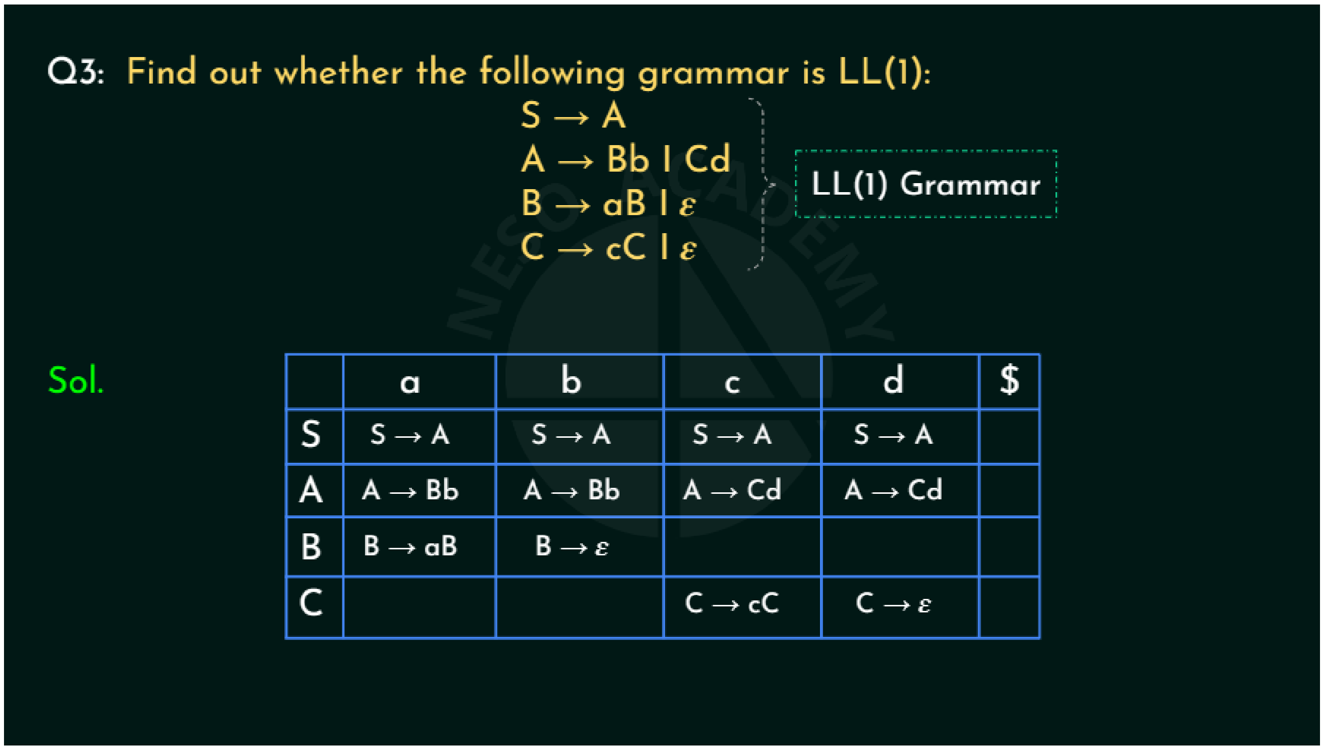
Q3: Find out whether the following grammar is LL(1):

Sol.

	FIRST	FOLLOW
$S \rightarrow A$	{a, b, c, d}	{\$}
$A \rightarrow Bb \mid Cd$	{a, b, c, d}	{\$}
$B \rightarrow aB \mid \varepsilon$	{a, ε}	{b}
$C \rightarrow cC \mid \varepsilon$	{c, ε}	{d}

	a	b	c	d	\$
S	$S \rightarrow A$	$S \rightarrow A$	$S \rightarrow A$	$S \rightarrow A$	
A	$A \rightarrow Bb$	$A \rightarrow Bb$	$A \rightarrow Cd$	$A \rightarrow Cd$	
B	$B \rightarrow aB$	$B \rightarrow \varepsilon$			
			$C \rightarrow cC$	$C \rightarrow \varepsilon$	

Find out whether the following grammar is LL(1):Q3:Sol.S → AA → Bb I CdB → aB I εC → cC I εFOLLOW{$}{b}FIRST{a, b, c, d}{a, ε}{c, ε}{a, b, c, d}{$}{d}S → AS → AS → AS → AA → BbA → CdA → CdB → εC → εA → BbB → aB C → cCa b c d $ S A B C



Q3: Find out whether the following grammar is LL(1):

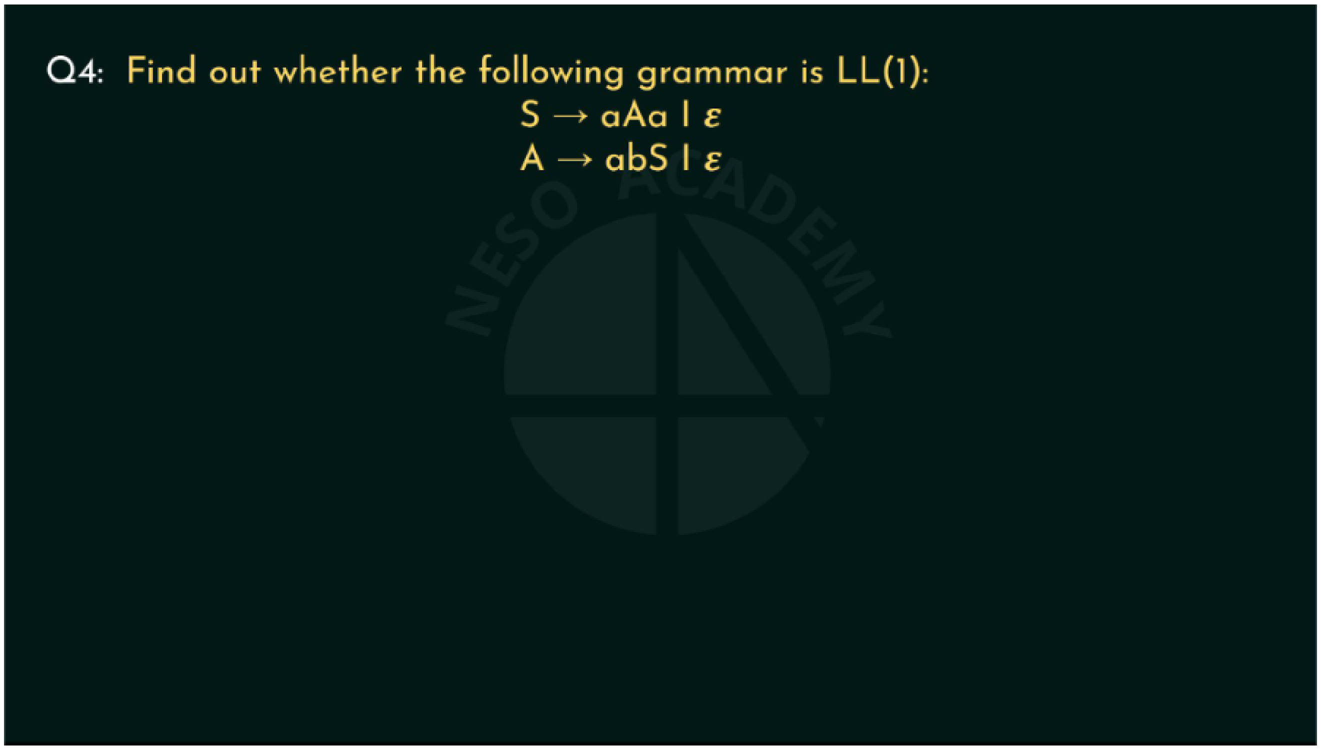
$S \rightarrow A$
$A \rightarrow Bb \mid Cd$
$B \rightarrow aB \mid \varepsilon$
$C \rightarrow cC \mid \varepsilon$

LL(1) Grammar

Sol.

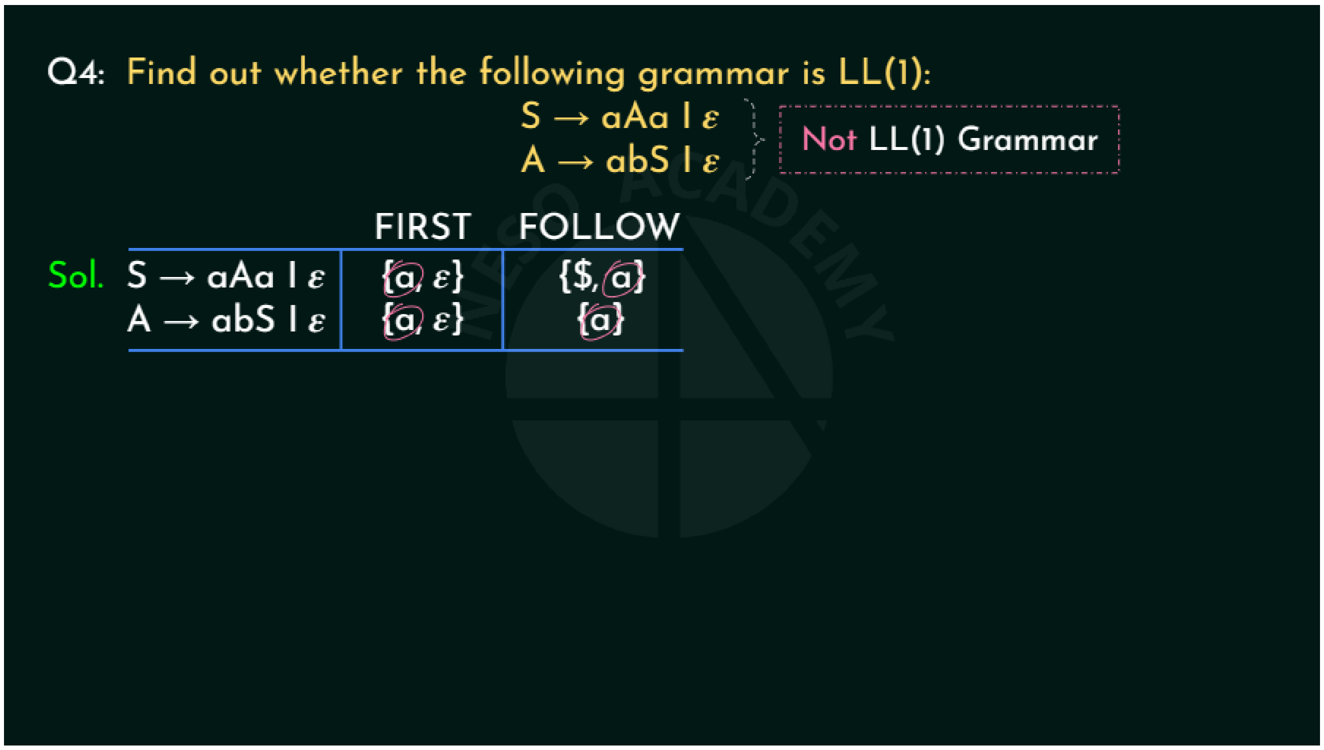
	a	b	c	d	$
S	$S \rightarrow A$	$S \rightarrow A$	$S \rightarrow A$	$S \rightarrow A$	
A	$A \rightarrow Bb$	$A \rightarrow Bb$	$A \rightarrow Cd$	$A \rightarrow Cd$	
B	$B \rightarrow aB$	$B \rightarrow \varepsilon$			
C			$C \rightarrow cC$	$C \rightarrow \varepsilon$	

Find out whether the following grammar is LL(1):S → AA → Bb I CdB → aB I εC → cC I εQ3:Sol.S → AS → AS → AS → AA → BbA → BbA → CdA → CdB → aB B → εC → cCC → εS A B C a b c d $ LL(1) Grammar



Q4: Find out whether the following grammar is LL(1):

$S \rightarrow aAa \mid \varepsilon$
$A \rightarrow abS \mid \varepsilon$



Q4: Find out whether the following grammar is LL(1):

$S \rightarrow aAa \mid \varepsilon$

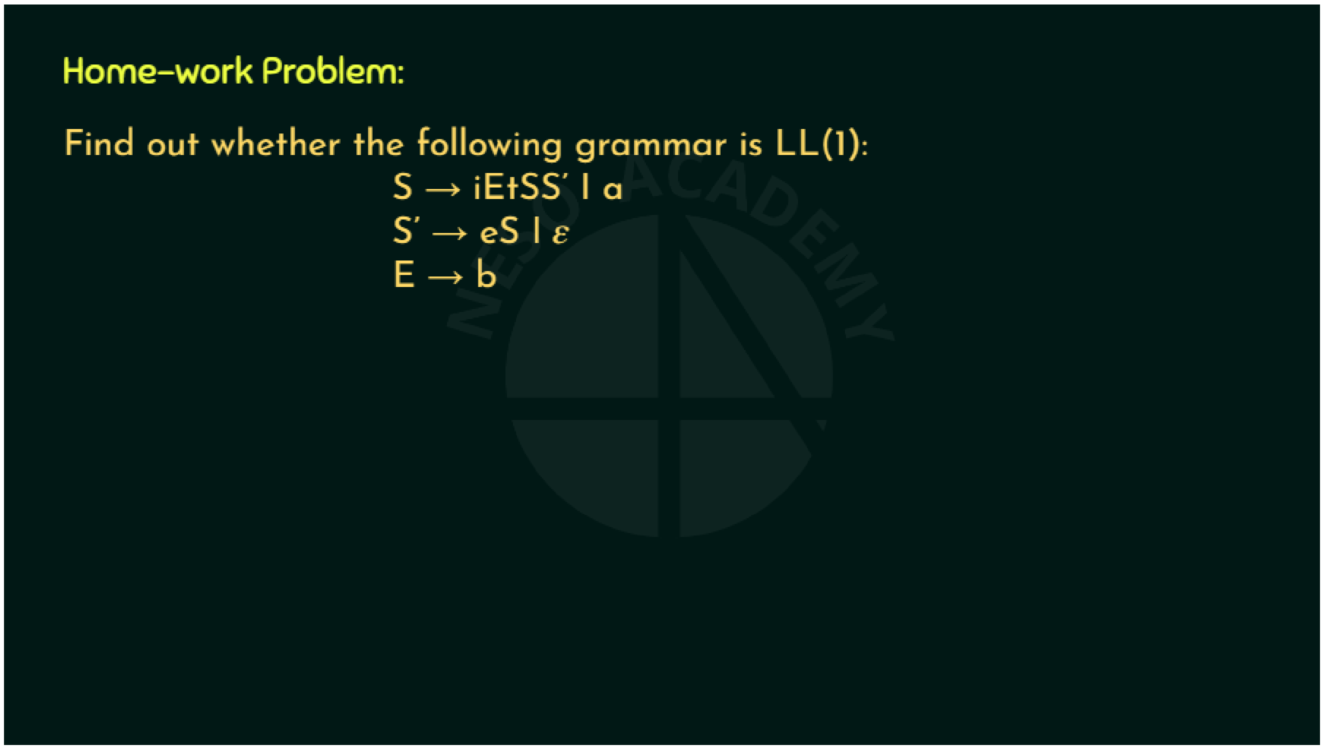
$A \rightarrow abS \mid \varepsilon$

Not LL(1) Grammar

Sol.

	FIRST	FOLLOW
$S \rightarrow aAa \mid \varepsilon$	{a, ε}	{$, a}
$A \rightarrow abS \mid \varepsilon$	{a, ε}	{a}

Find out whether the following grammar is LL(1):S → aAa I ε A → abS I εQ4:Sol.S → aAa I εA → abS I εFIRSTFOLLOW{a}{a, ε}{a, ε}Not LL(1) Grammar{$, a}



Home-work Problem:

Find out whether the following grammar is LL(1):

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \varepsilon$

$E \rightarrow b$

Find out whether the following grammar is LL(1):S → iEtSS' I a S' → eS I εE → bHome-work Problem: