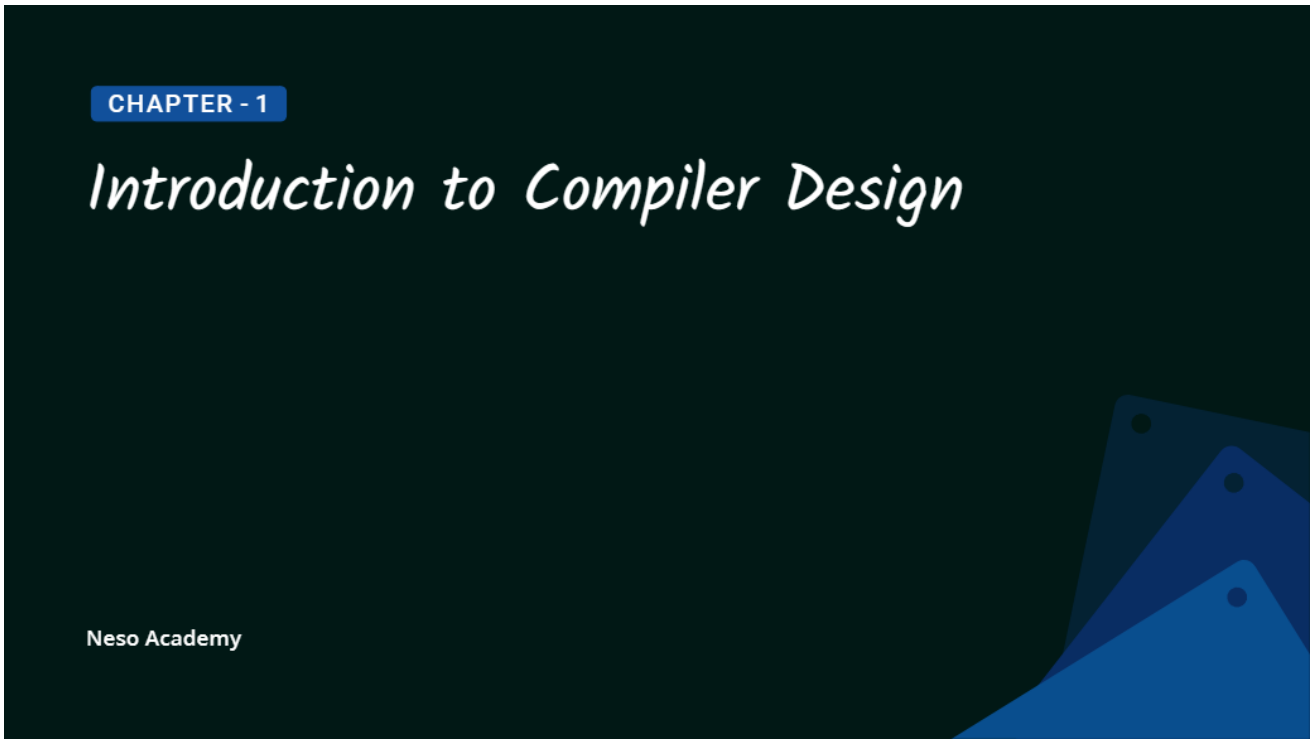
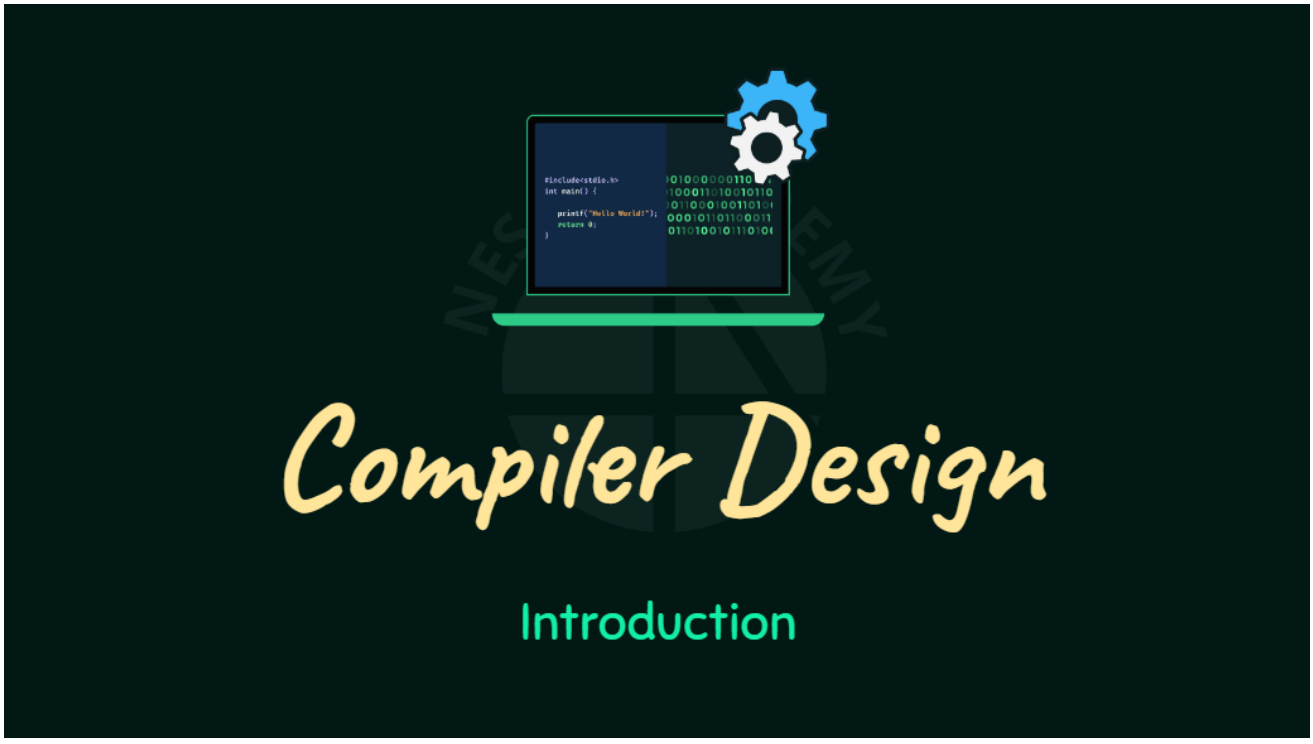
Introduction to Compiler Design | Neso Academy

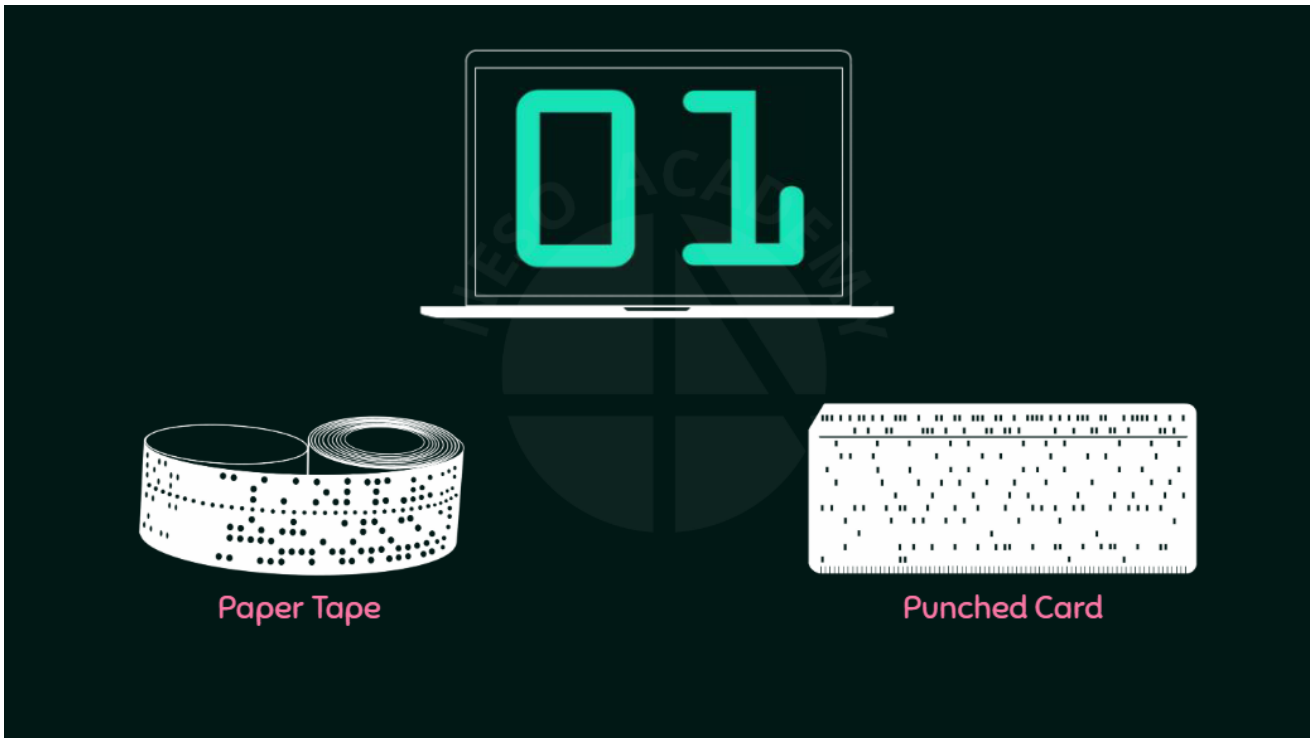
nesoacademy.org/cs/12-compiler-design/ppts/01-introduction-to-compiler-design



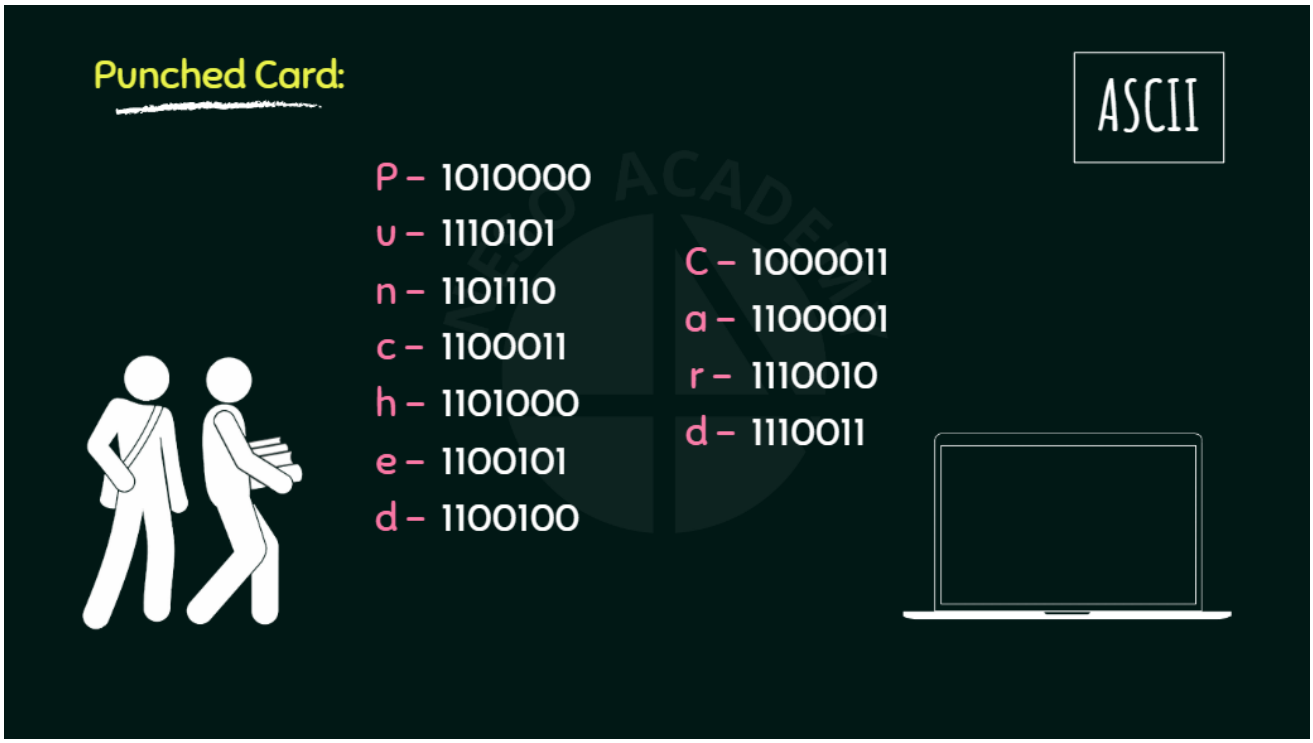
Introduction to Compiler DesignNeso AcademyCHAPTER-1



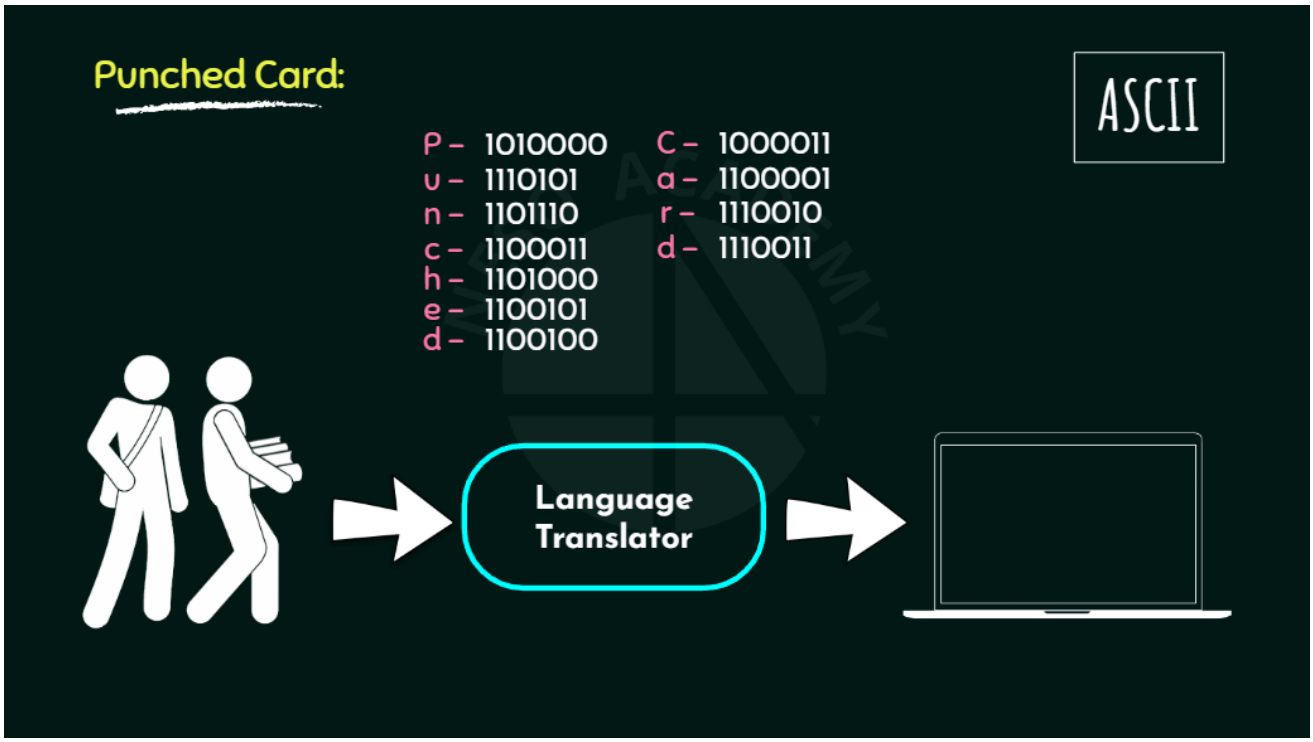
Compiler DesignIntroduction



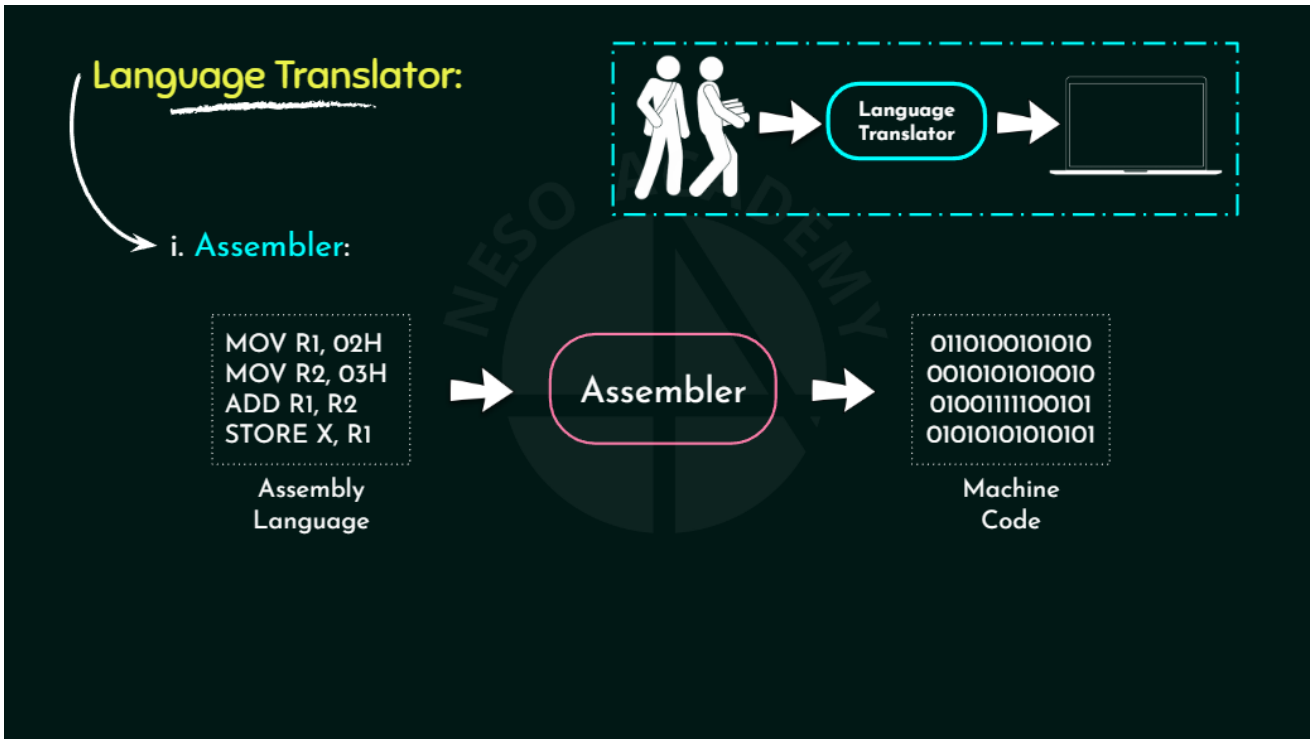
Paper TapePunched Card



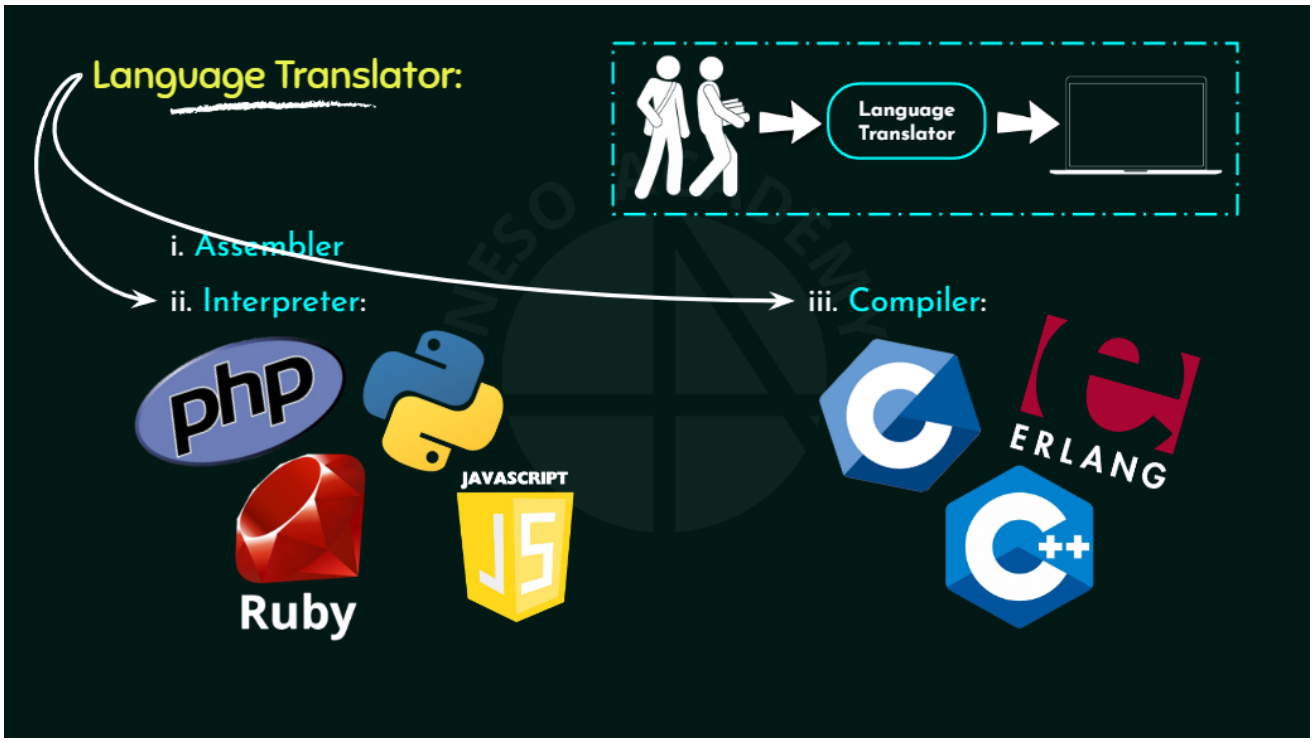
Punched Card:ASCIIP - 1010000 u - 1110101n - 1101110c - 1100011h - 1101000e - 1100101d - 1100100C - 1000011a - 1100001r - 1110010d - 1110011



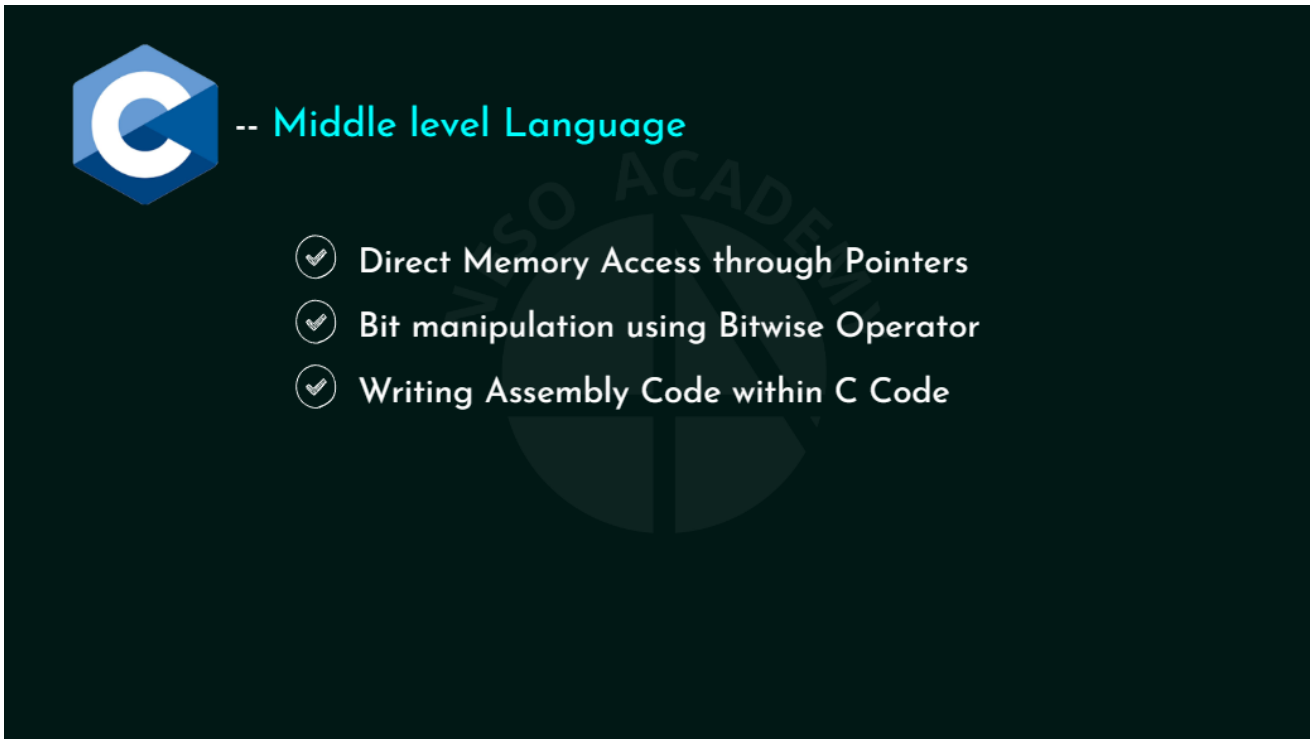
Punched Card:ASCIIP - 1010000 u - 1110101n - 1101110c - 1100011h - 1101000e - 1100101d - 1100100C - 1000011a - 1100001r - 1110010d - 1110011LanguageTranslator



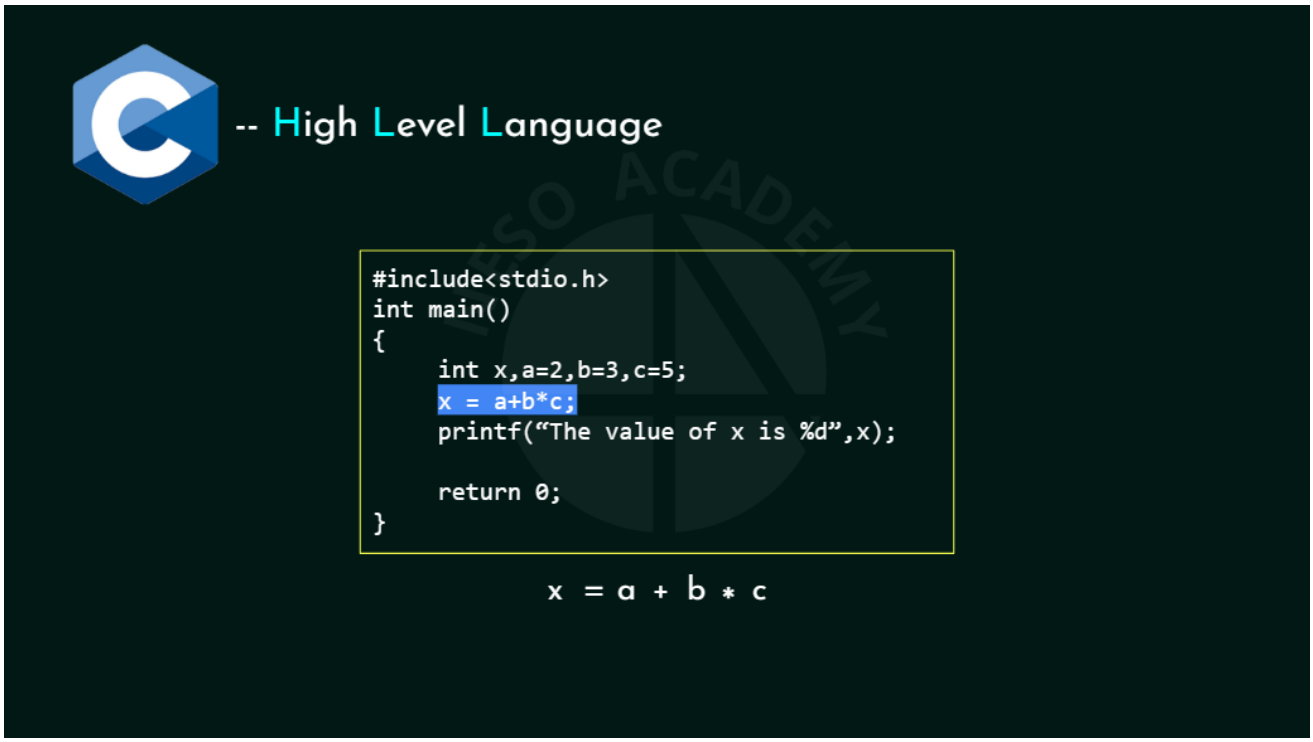
Language Translator:i. Assembler:AssemblerAssemblyLanguage MOV R1, 02HMOV R2, 03HADD R1, R2STORE X, R1MachineCode0110100101010001010101001001001111100101010101010101Language Translator



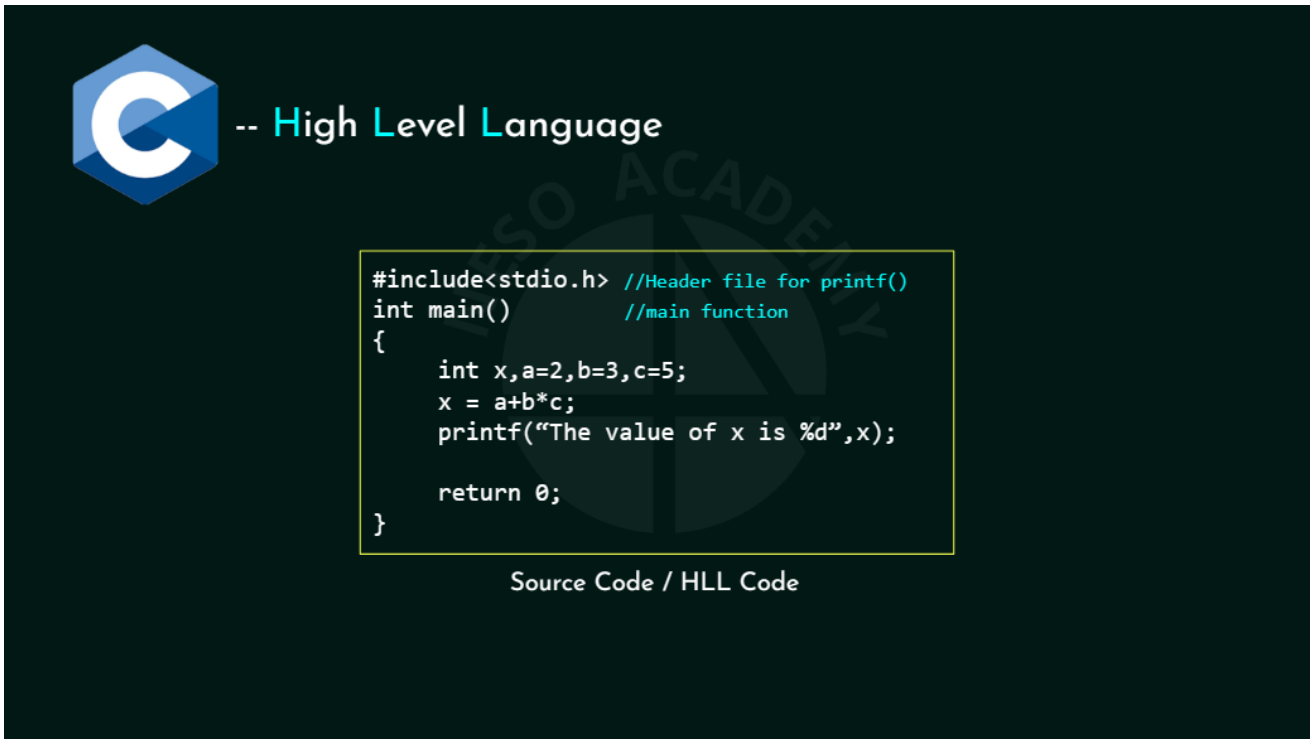
Language Translator:i. Assemblerii. Interpreter:iii. Compiler:Language Translator



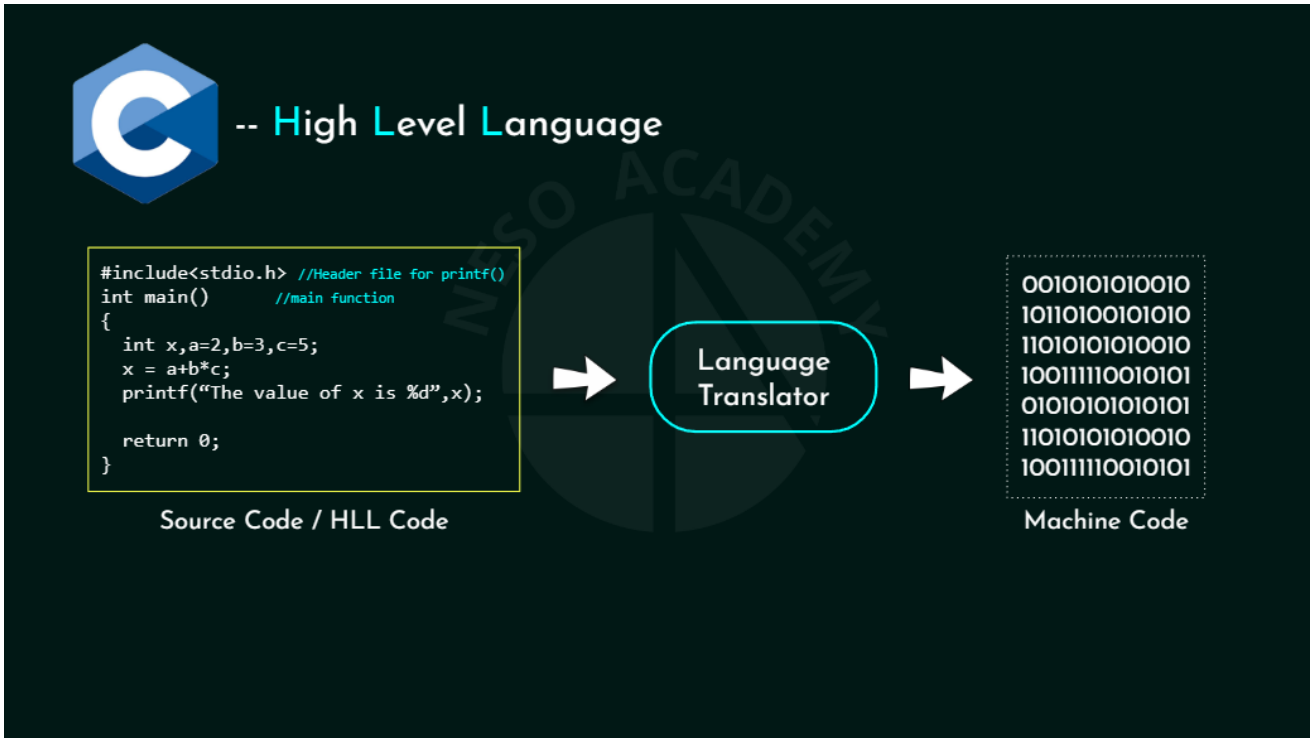
-- Middle level LanguageDirect Memory Access through PointersBit manipulation using Bitwise Operator Writing Assembly Code within C Code



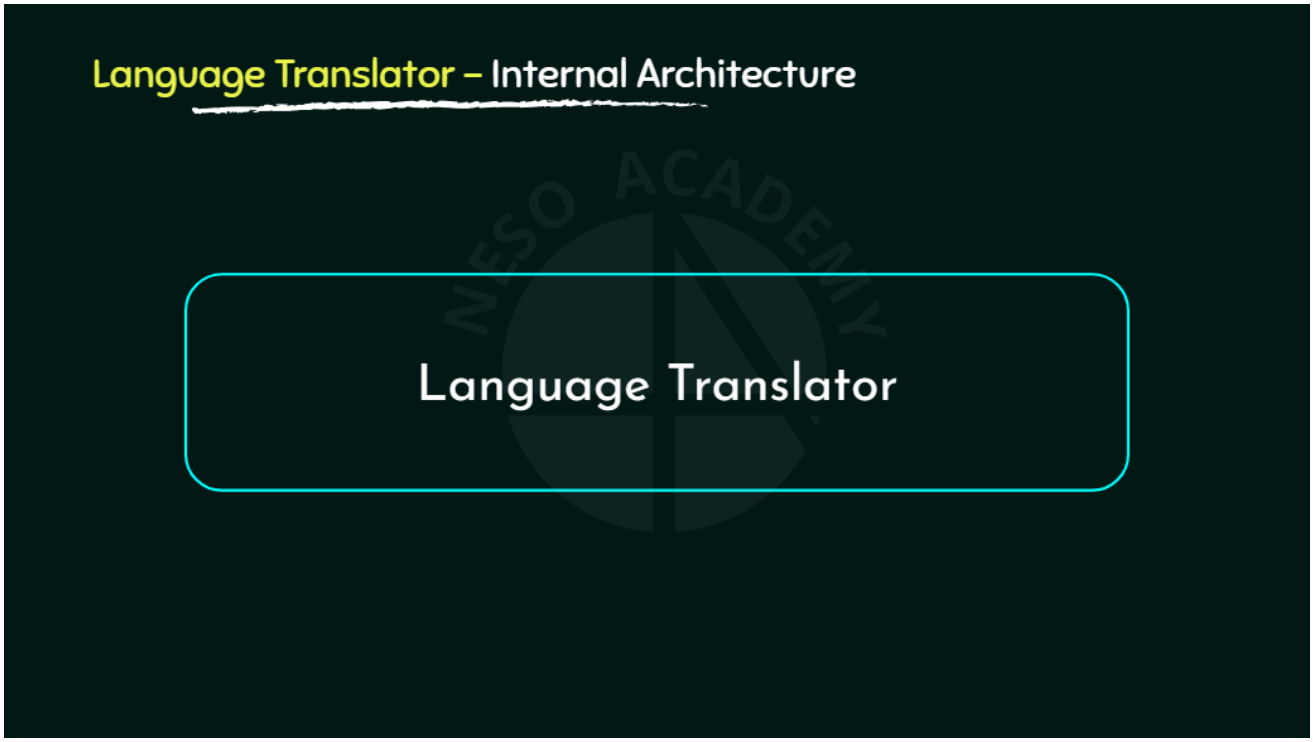
-- High Level Language#include<stdio.h>int main(){int x,a=2,b=3,c=5;x = a+b*c;printf("The value of x is %d",x);return 0;} x=a+b*c



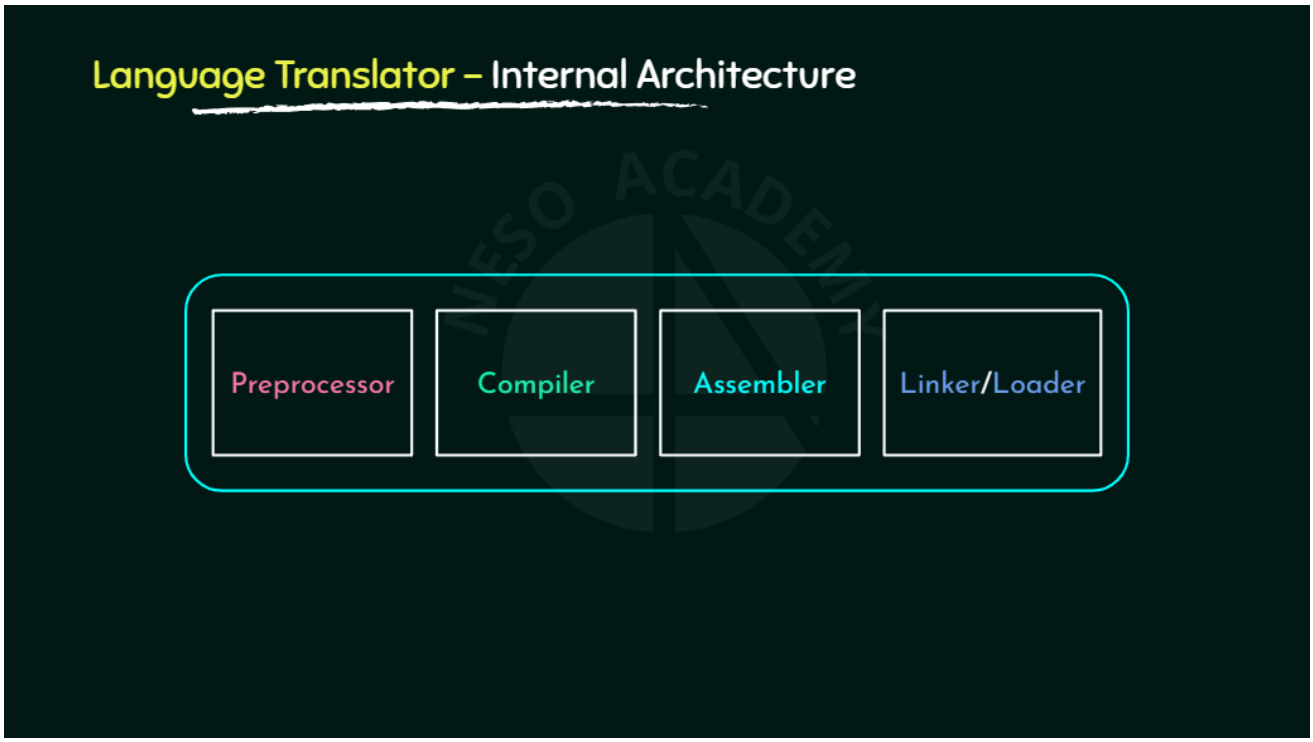
-- High Level Language#include<stdio.h> //Header file for printf()int main() //main function{int x,a=2,b=3,c=5;x = a+b*c;printf("The value of x is %d",x);return 0;} Source Code/ HLL Code



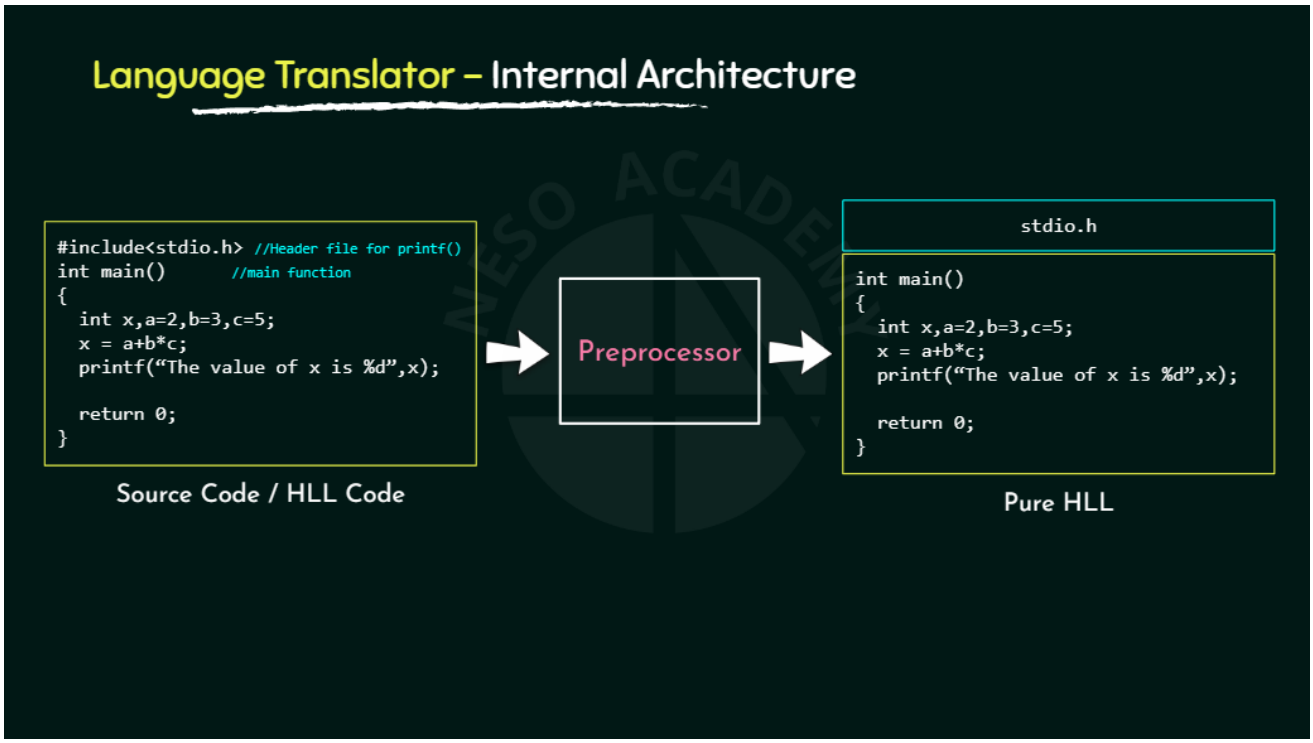
-- High Level LanguageSource Code / HLL Code#include<stdio.h> //Header file for printf()int main() //main function{ int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}Language TranslatorMachine Code0010101010010101101001010101101010101001010011111001010101010101010101011101010101001010011111001010101



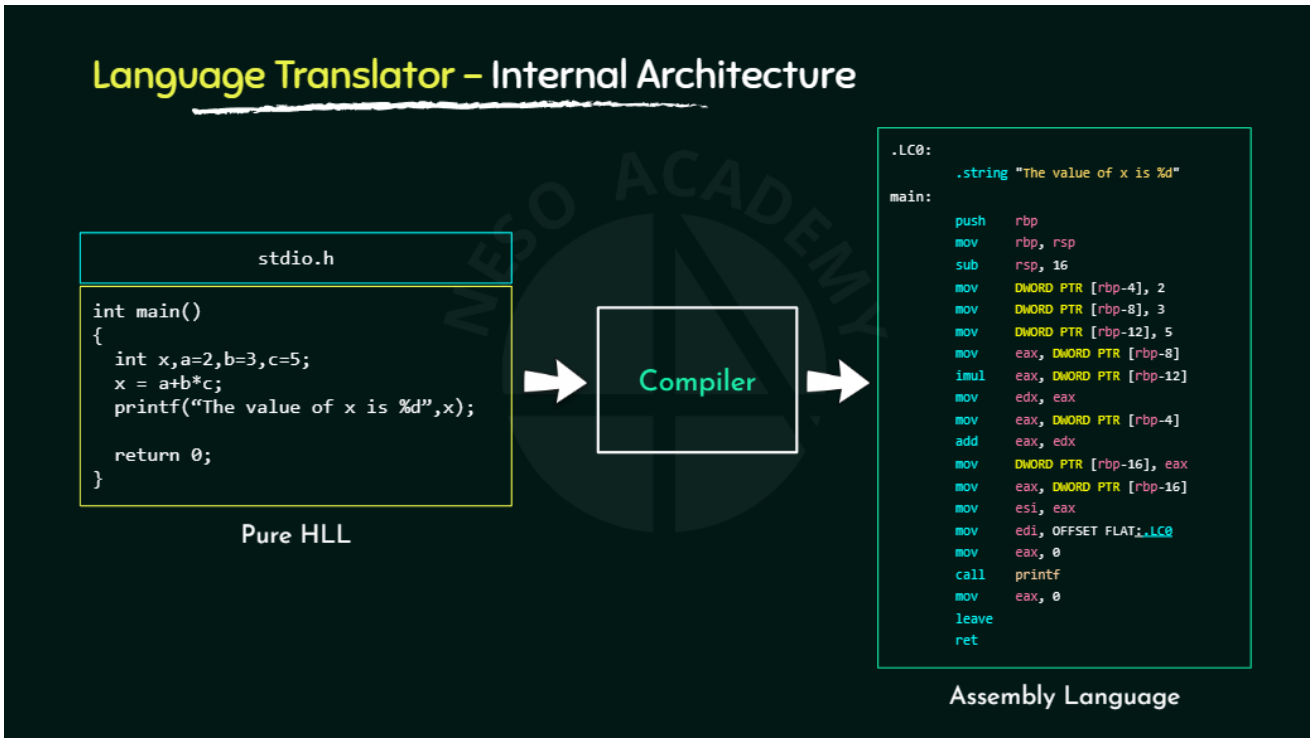
Language Translator - Internal ArchitectureLanguage Translator



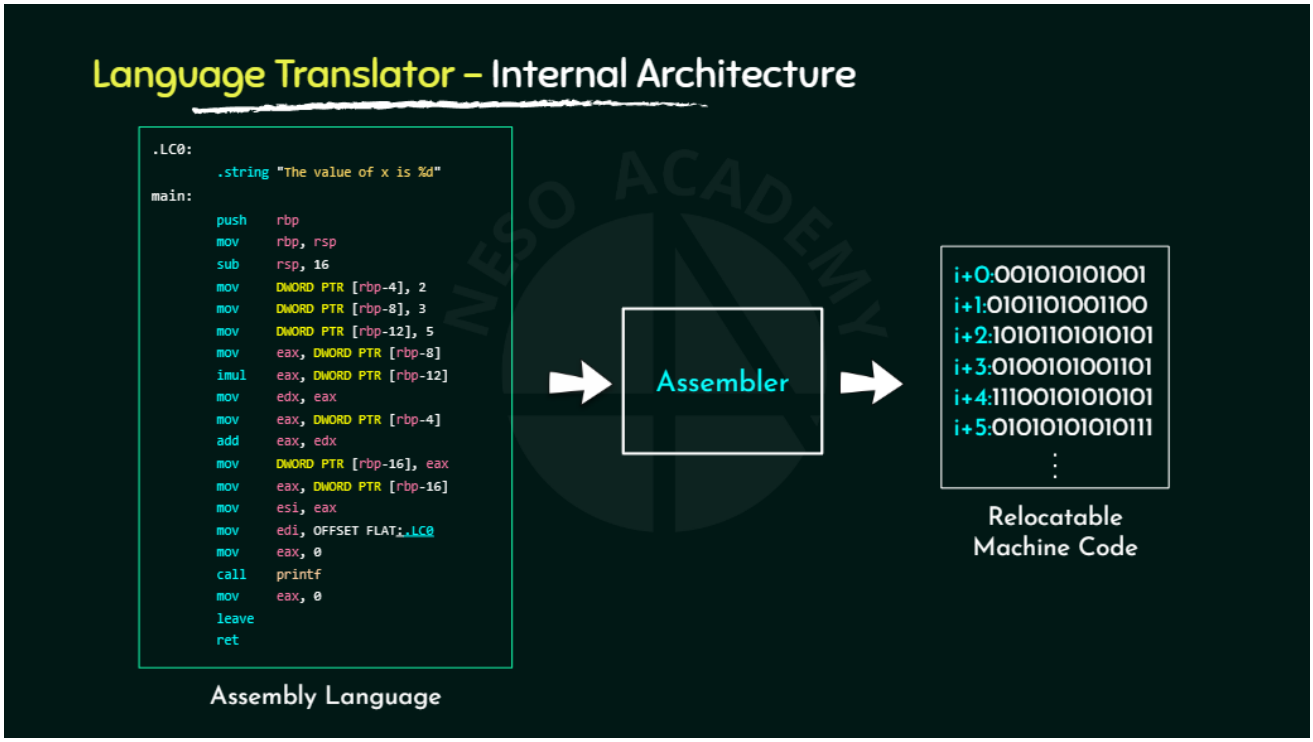
Language Translator - Internal ArchitecturePreprocessorCompilerAssemblerLinker/Loader



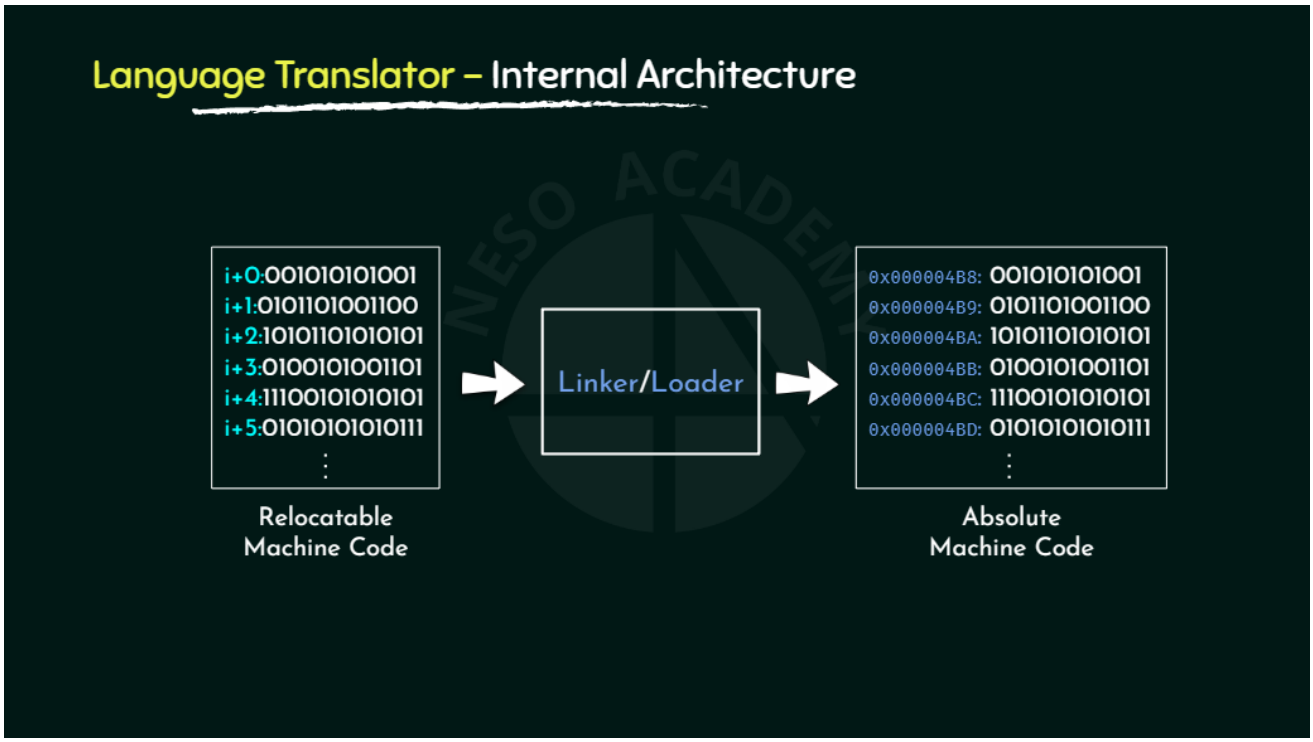
Language Translator - Internal ArchitectureSource Code / HLL Code#include<stdio.h> //Header file for printf()int main() //main function{ int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}Preprocessorint main() { int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}stdio.hPure HLL



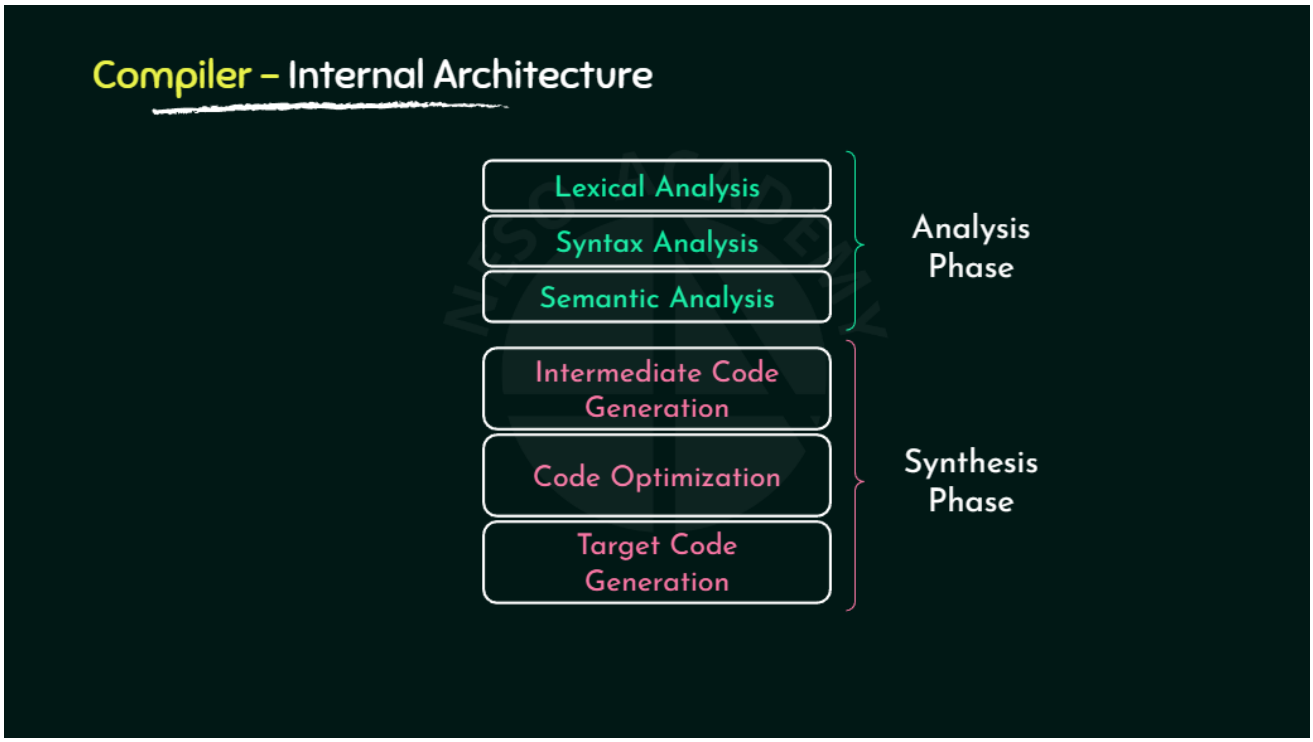
int main() { int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}stdio.hPure HLL Language Translator - Internal ArchitectureCompiler.LC0:.string "The value of x is %d"main:push rbpmov rbp, rspsub rsp, 16mov DWORD PTR [rbp-4], 2mov DWORD PTR [rbp-8], 3mov DWORD PTR [rbp-12], 5mov eax, DWORD PTR [rbp-8]imul eax, DWORD PTR [rbp-12]mov edx, eaxmov eax, DWORD PTR [rbp-4]add eax, edxmov DWORD PTR [rbp-16], eaxmov eax, DWORD PTR [rbp-16]mov esi, eaxmov edi, OFFSET FLAT:.LC0mov eax, 0call printfmov eax, 0leaveretAssembly Language



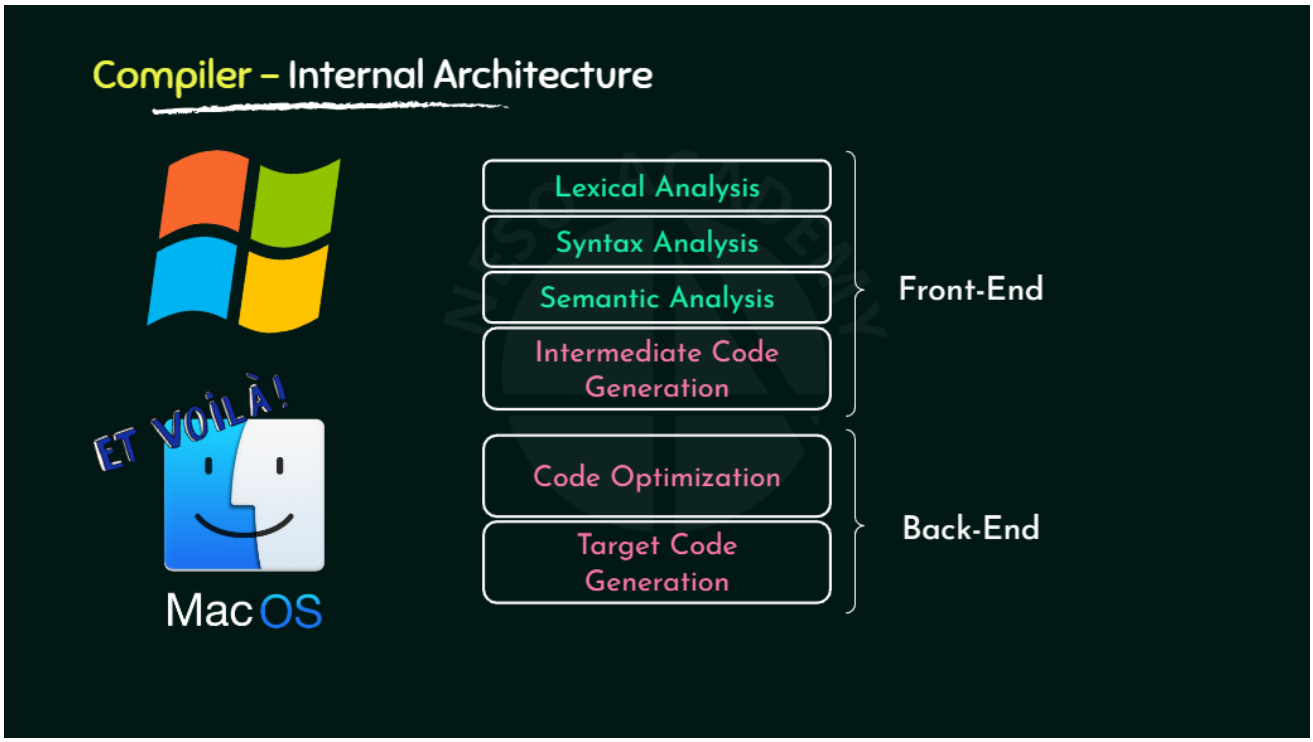
Language Translator - Internal ArchitectureAssembler.LC0:.string "The value of x is %d"main:push rbpmov rbp, rspsub rsp, 16mov DWORD PTR [rbp-4], 2mov DWORD PTR [rbp-8], 3mov DWORD PTR [rbp-12], 5mov eax, DWORD PTR [rbp-8]imul eax, DWORD PTR [rbp-12]mov edx, eaxmov eax, DWORD PTR [rbp-4]add eax, edxmov DWORD PTR [rbp-16], eaxmov eax, DWORD PTR [rbp-16]mov esi, eaxmov edi, OFFSET FLAT:.LC0mov eax, 0call printfmov eax, 0leaveretAssembly Language Relocatable Machine Codei+0:001010101001i+1:0101101001100i+2:10101101010101i+3:0100101001101i+4:11100101010101i+5:01010101010111



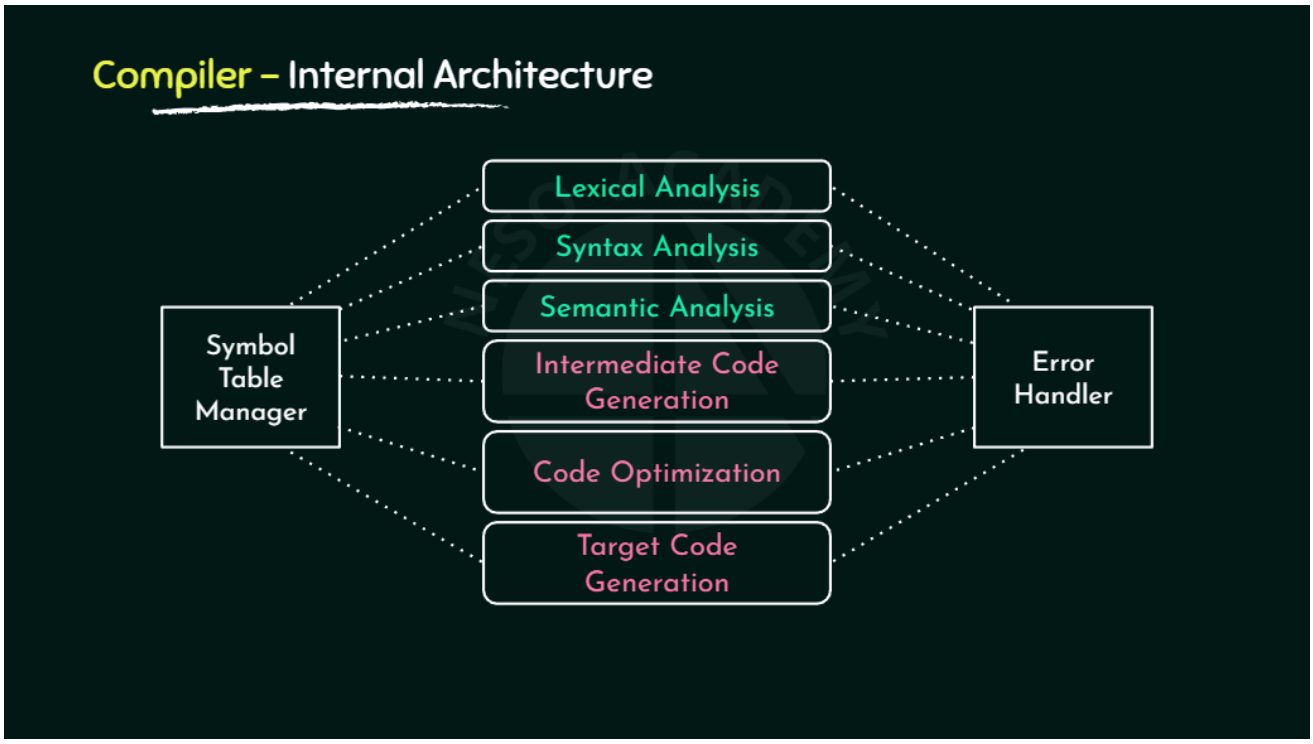
Linker/LoaderLanguage Translator - Internal ArchitectureRelocatable Machine Codei+0:001010101001i+1:0101101001100i+2:10101101010101i+3:0100101001101i+4:11100101010101i+5:01010101010111Absolute Machine Code0x000004B8: 0010101010010x000004B9: 01011010011000x000004BA: 101011010101010x000004BB: 010010100110100x000004BC: 111001010101010x000004BD: 01010101010111



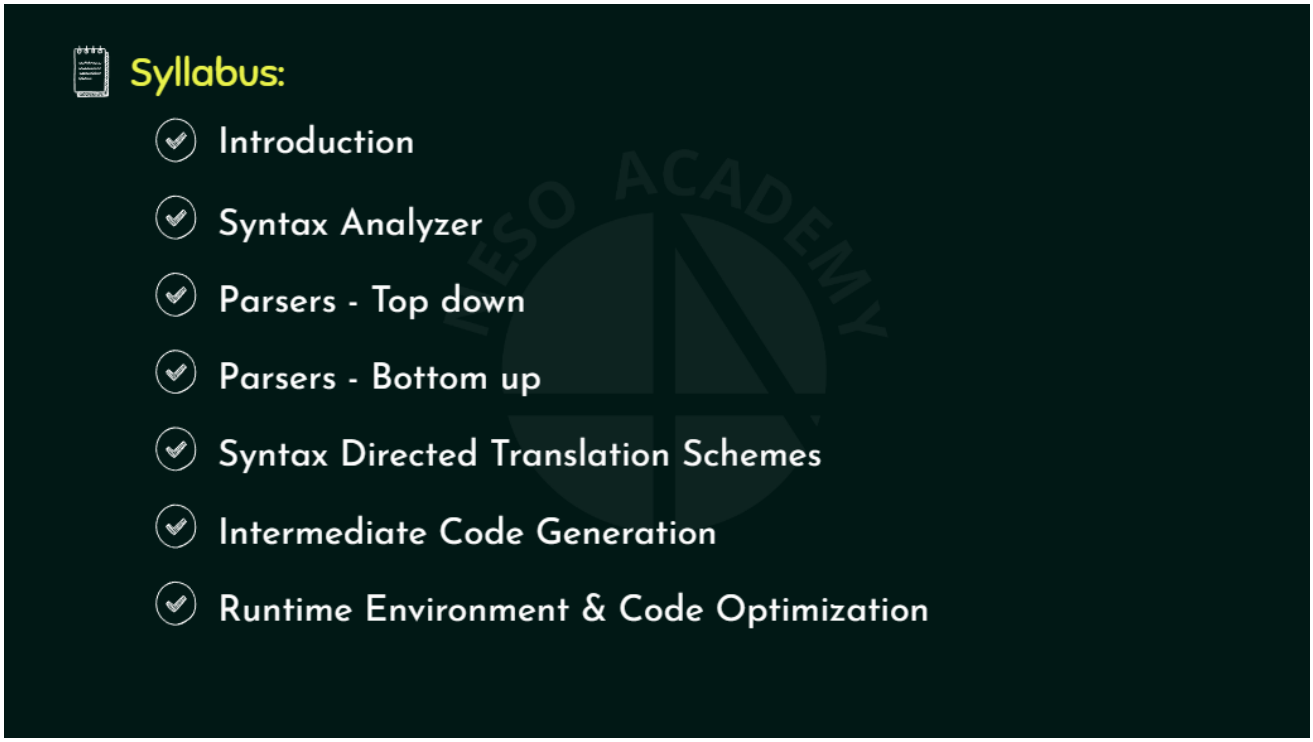
Compiler - Internal ArchitectureLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationAnalysis PhaseSynthesis Phase



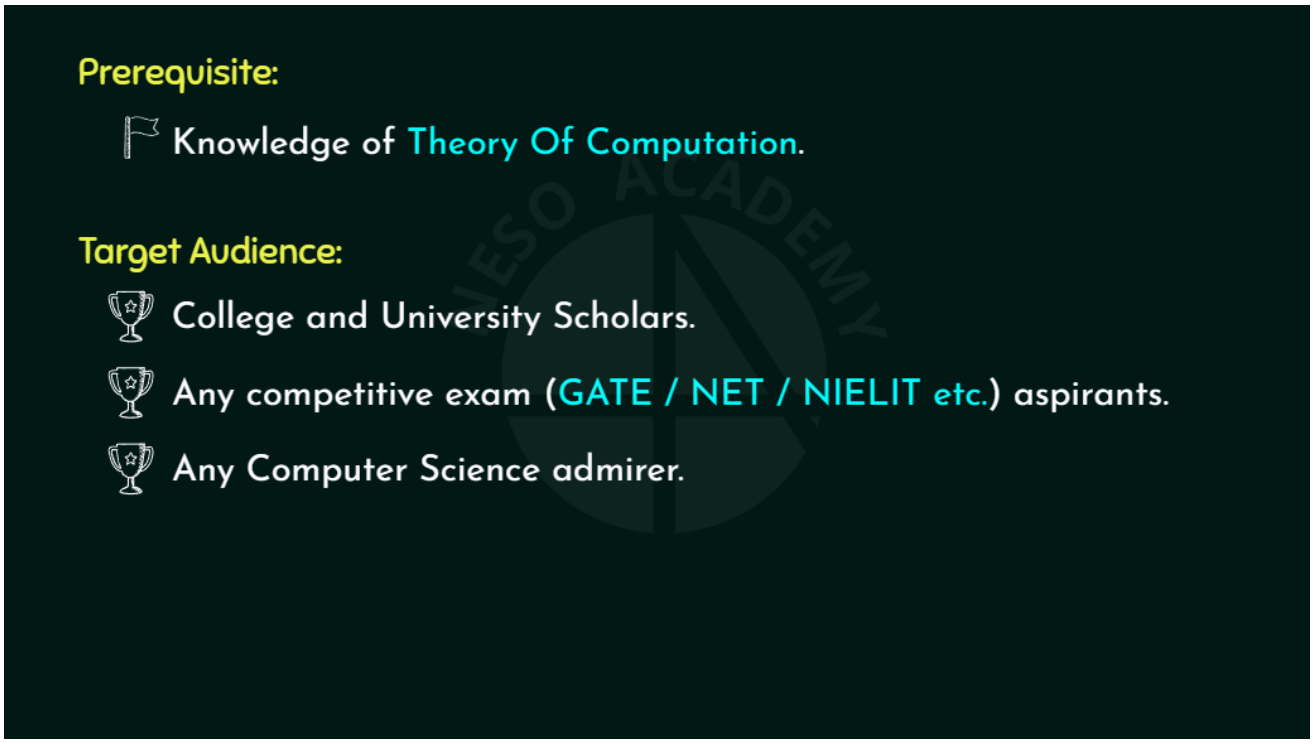
Compiler - Internal ArchitectureLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationFront-EndBack-End



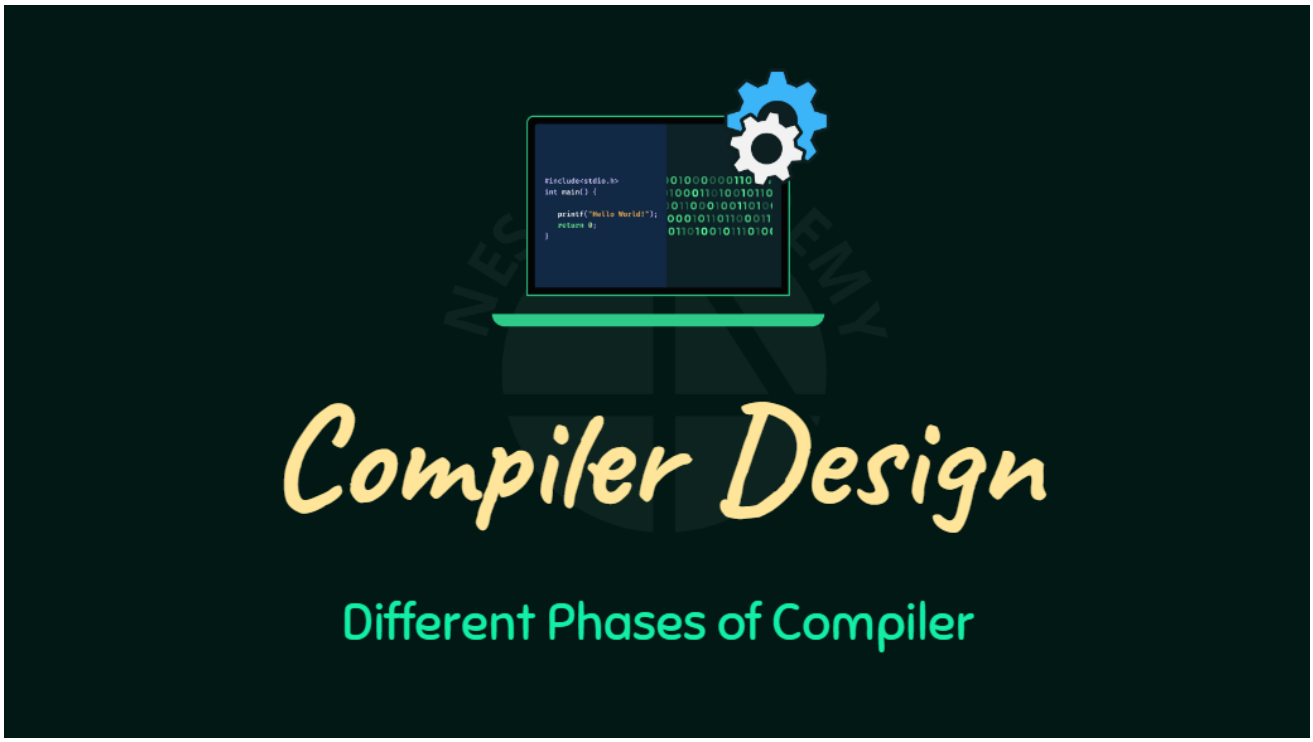
Compiler - Internal ArchitectureLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationSymbol Table ManagerError Handler



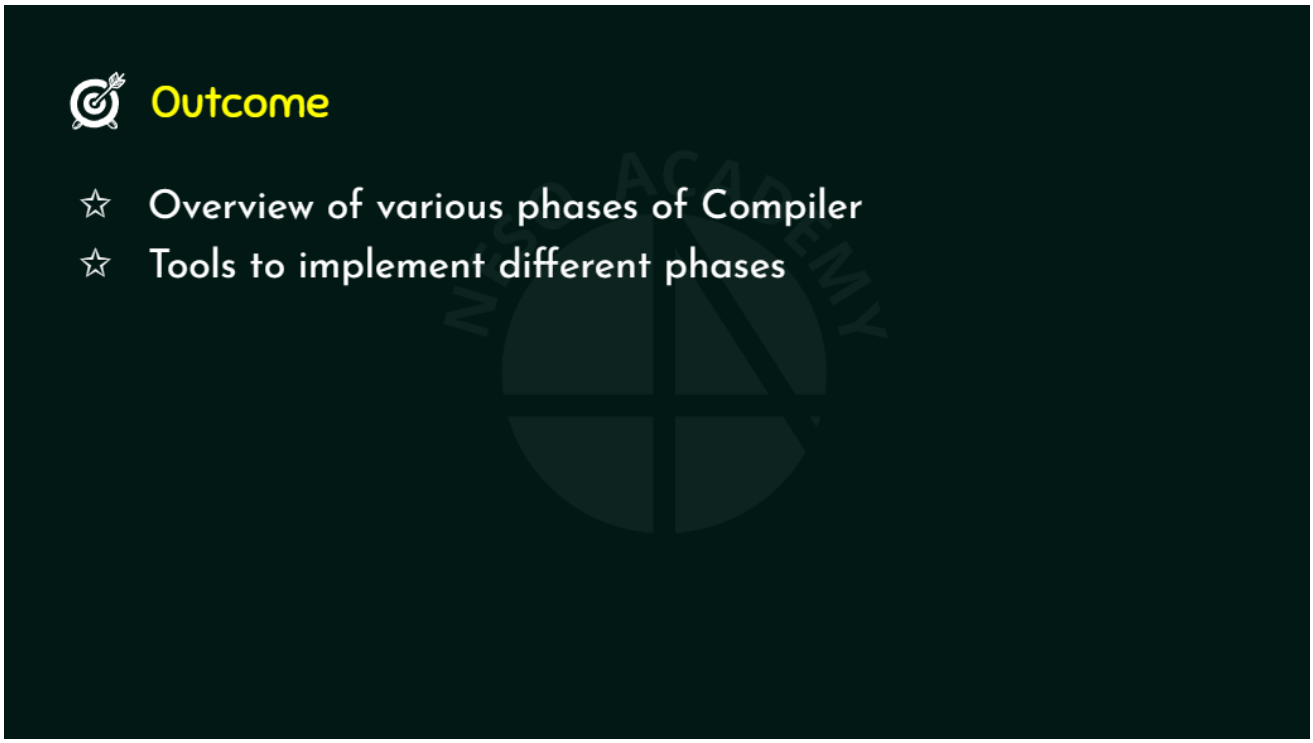
Syllabus:IntroductionSyntax AnalyzerParsers - Top downParsers - Bottom upSyntax Directed Translation SchemesIntermediate Code GenerationRuntime Environment & Code Optimization



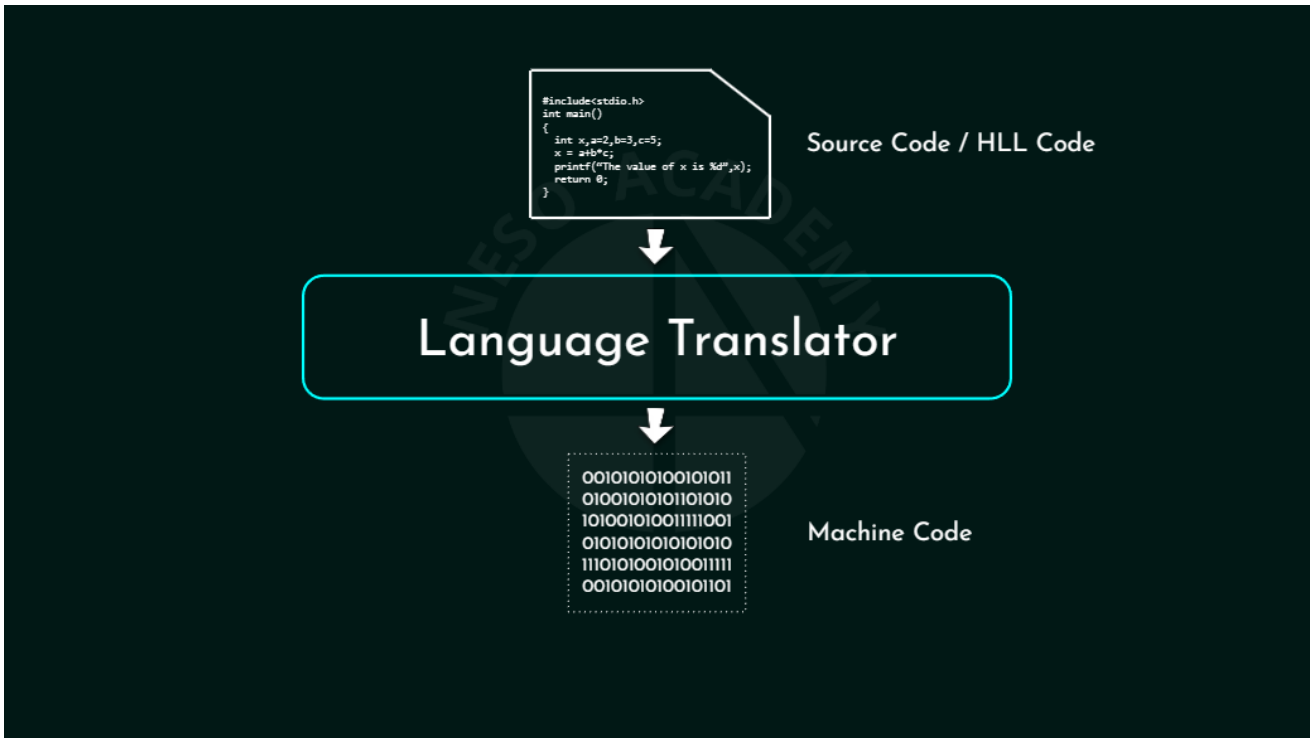
Prerequisite:Knowledge of Theory Of Computation.Target Audience:College and University Scholars.Any competitive exam (GATE / NET / NIELIT etc.) aspirants.Any Computer Science admirer.



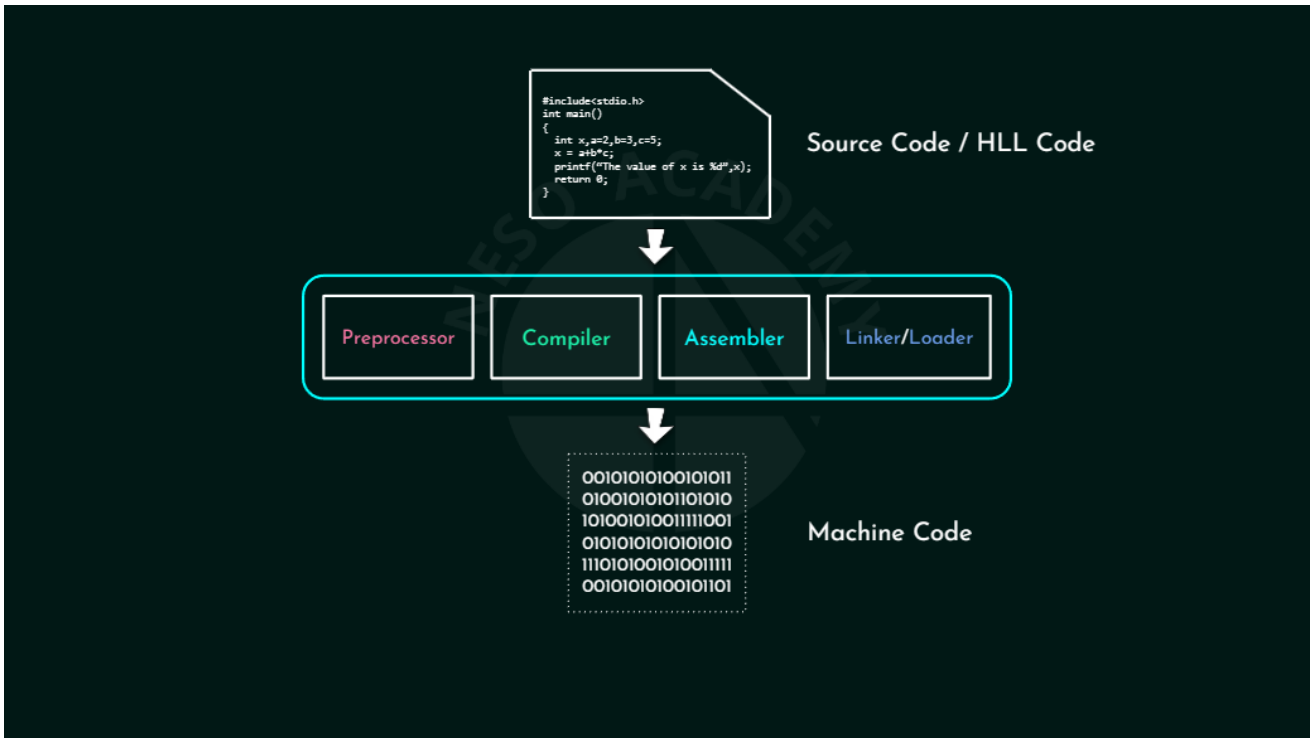
Compiler DesignDifferent Phases of Compiler



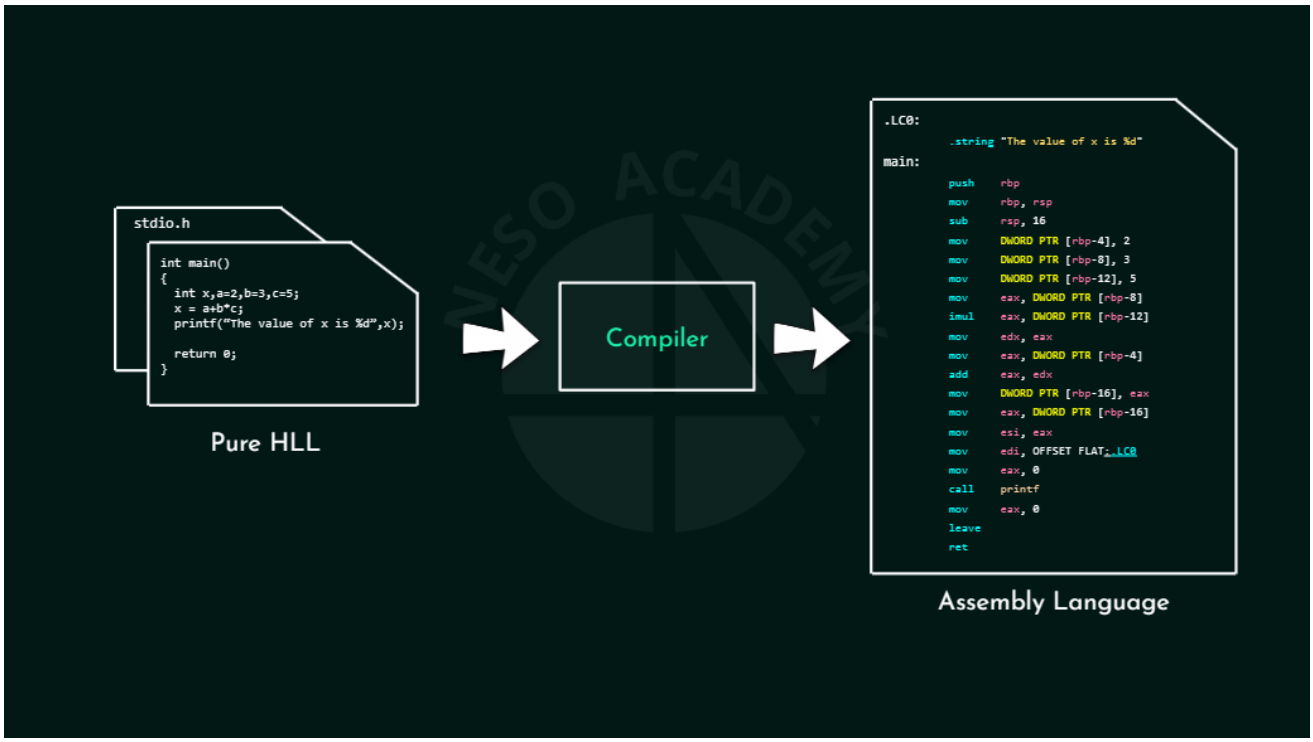
Outcome☆Tools to implement different phases☆Overview of various phases of Compiler



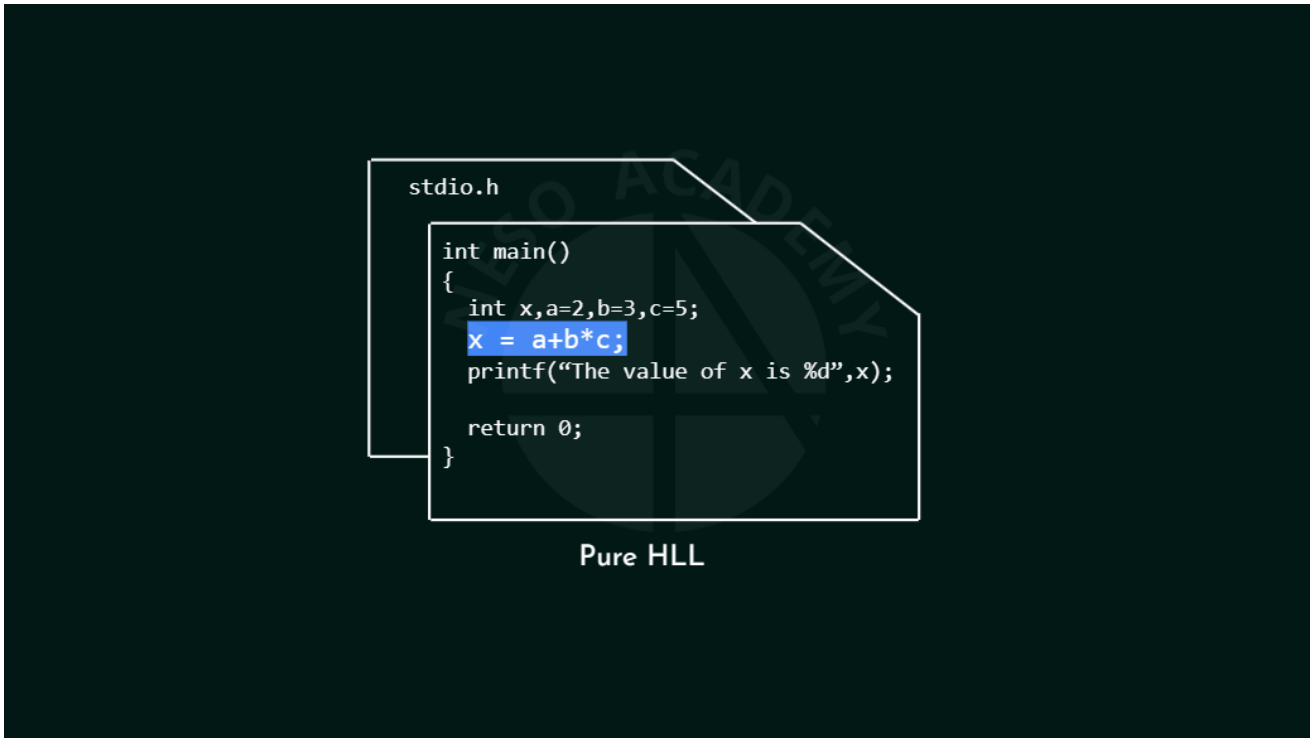
Machine Code0010101010010101101010010101011010101010010100111110010101010101010101011101010010100111110010101010010110
1Source Code / HLL Code#include<stdio.h>int main() { int x,a=2,b=3,c=5; x = a+b*c; printf(“The value of x is %d”,x); return 0;}Language Translator



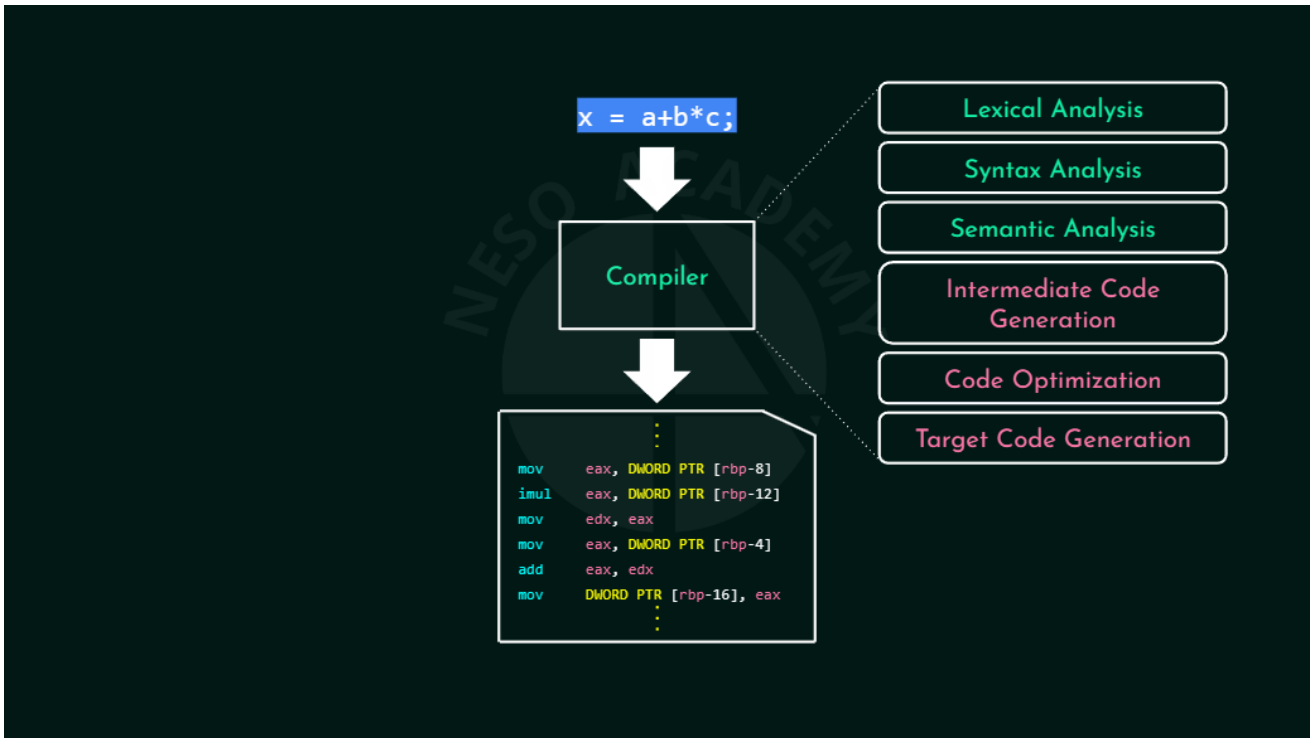
PreprocessorCompilerAssemblerLinker/LoaderSource Code / HLL CodeMachine Code0010101010010101101010010101011010101010010100111110010101010101010101011101010010100111110010101010010110
1#include<stdio.h>int main() { int x,a=2,b=3,c=5; x = a+b*c; printf(“The value of x is %d”,x); return 0;}



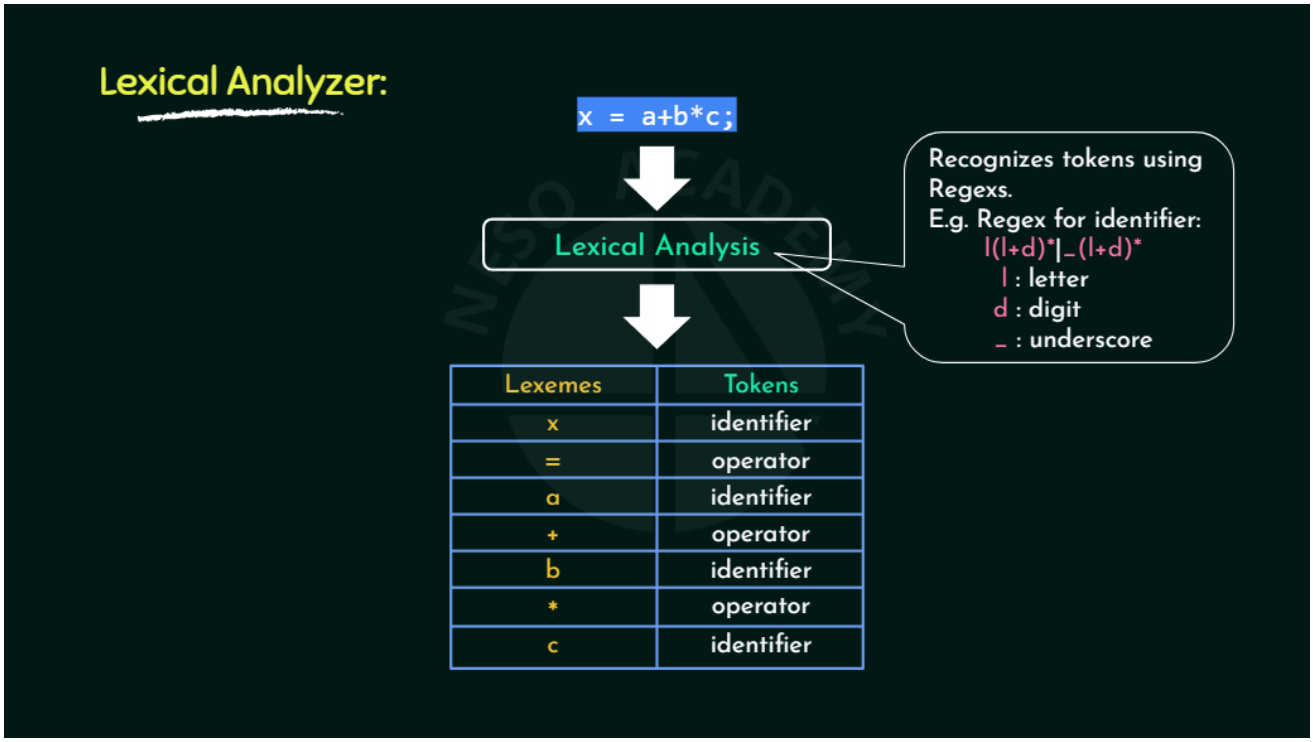
Pure HLLstdio.hint main() { int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}.LC0:.string "The value of x is %d"main:push rbpmov rbp, rspsub rsp, 16mov DWORD PTR [rbp-4], 2mov DWORD PTR [rbp-8], 3mov DWORD PTR [rbp-12], 5mov eax, DWORD PTR [rbp-8]imul eax, DWORD PTR [rbp-12]mov edx, eaxmov eax, DWORD PTR [rbp-4]add eax, edxmov DWORD PTR [rbp-16], eaxmov eax, DWORD PTR [rbp-16]mov esi, eaxmov edi, OFFSET FLAT:.LC0mov eax, 0call printfmov eax, 0leaveretAssembly Language Compiler



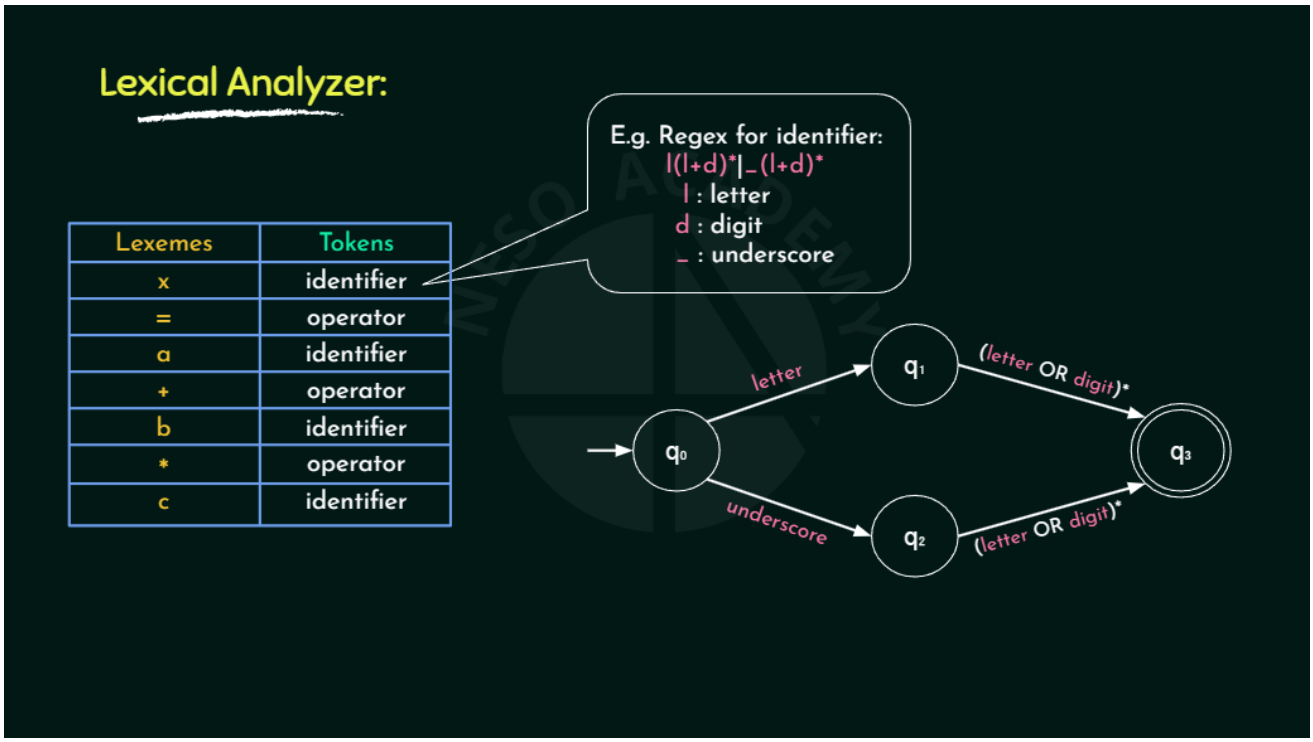
Pure HLLstdio.hint main() { int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}



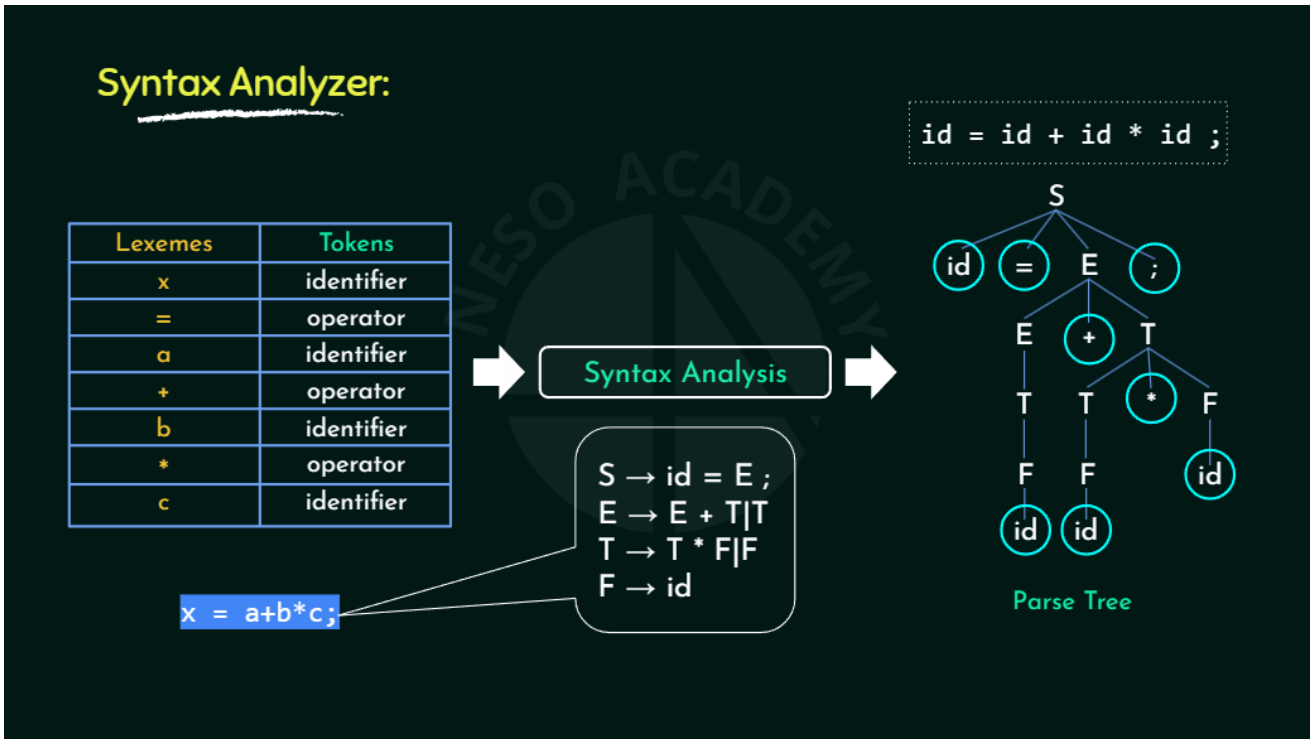
mov eax, DWORD PTR [rbp-8]imul eax, DWORD PTR [rbp-12]mov edx, eaxmov eax, DWORD PTR [rbp-4]add eax, edxmov DWORD PTR [rbp-16], eaxx = a+b*c;CompilerLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code Generation



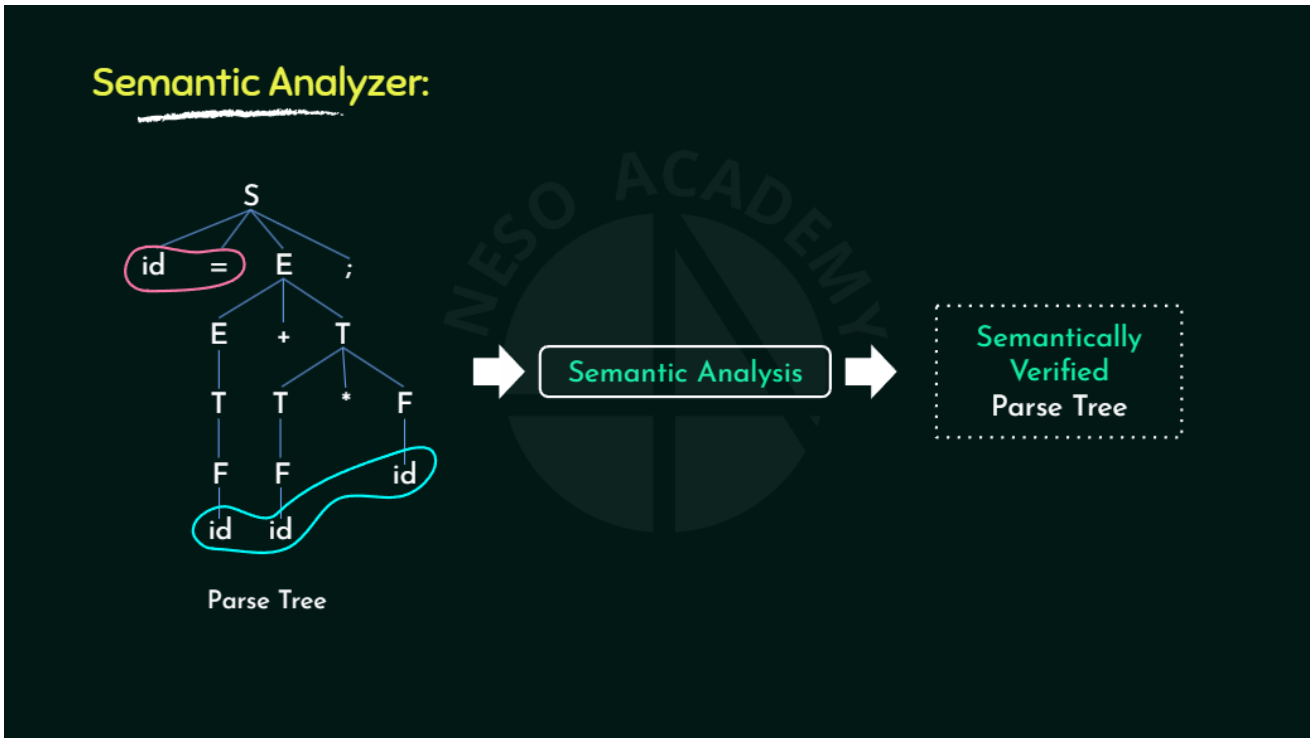
Lexical AnalysisLexical Analyzer:x = a+b*c;LexemesTokensxidentifier=operatoraidentifier+operatorbidentifier*operatorcidentifierRecognizes tokens using Regexs. E.g. Regex for identifier: l(l+d)*|_(l+d)* l : letterd : digit _ : underscore



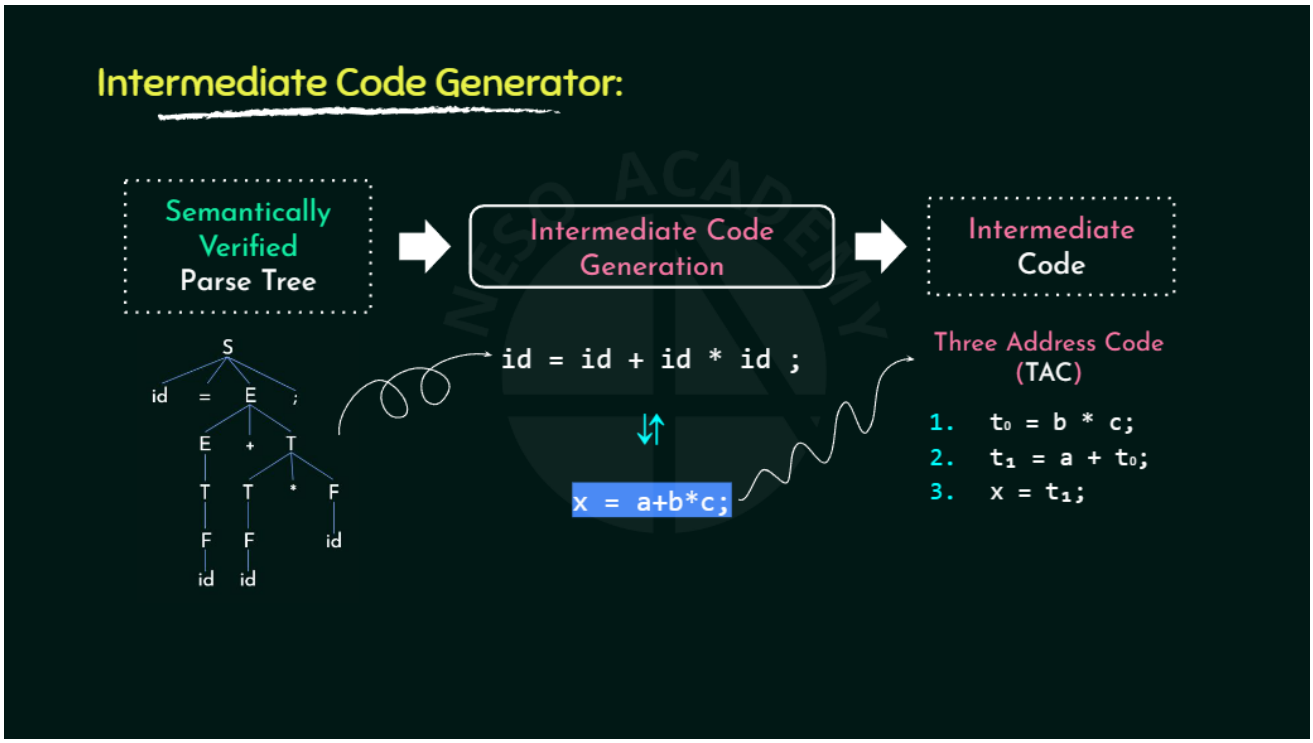
Lexical Analyzer:E.g. Regex for identifier: l(l+d)*|_(l+d)* l : letterd : digit _ : underscoreq₀q₁q₂q₃(letter OR digit)*(letter OR digit)*letterunderscoreLexemesTokensxidentifier=operatoraidentifier+operatorbidentifier*operatorcidentifier



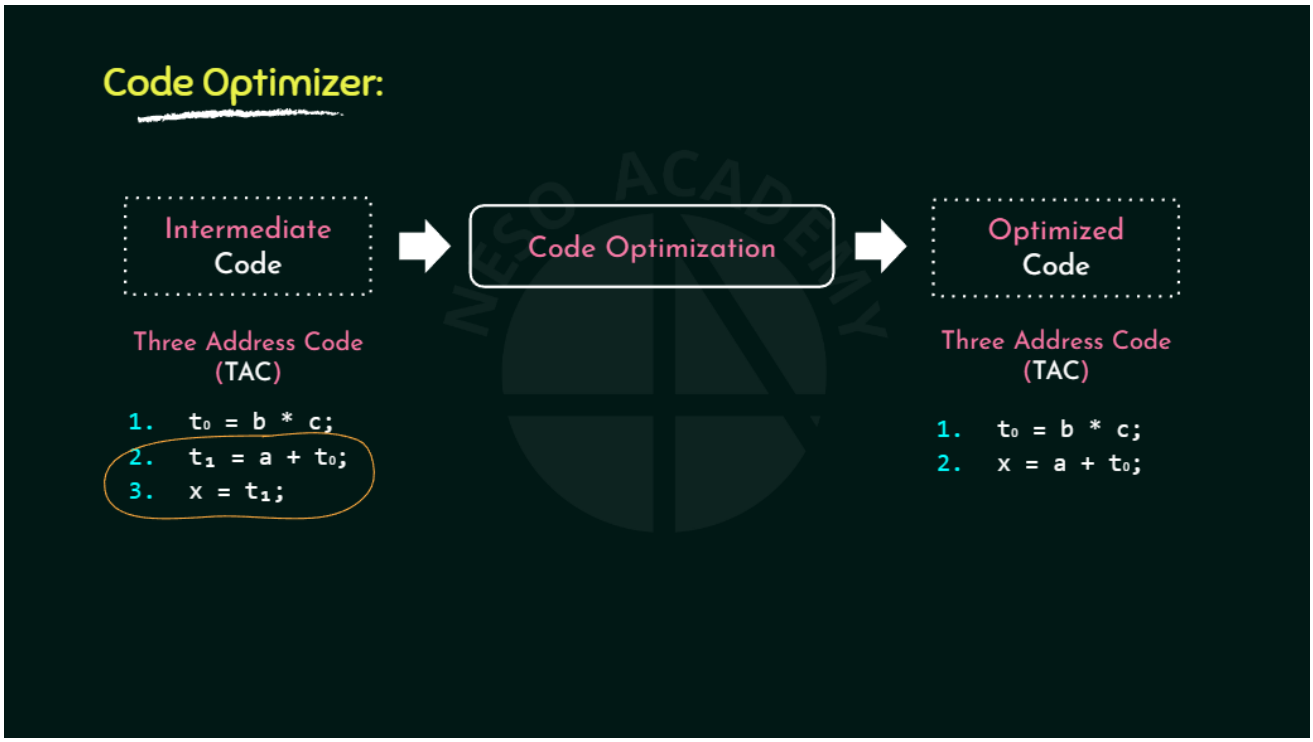
Syntax AnalysisSyntax Analyzer:LexemesTokensxidentifier=operatoraidentifier+operatorbidentifier*operatorcidentifierx = a+b*c;S → id = E ;E → E + T|TT → T * F|FF → idSid=E;E+TT*FTFFidididParse Tree id = id + id * id ;



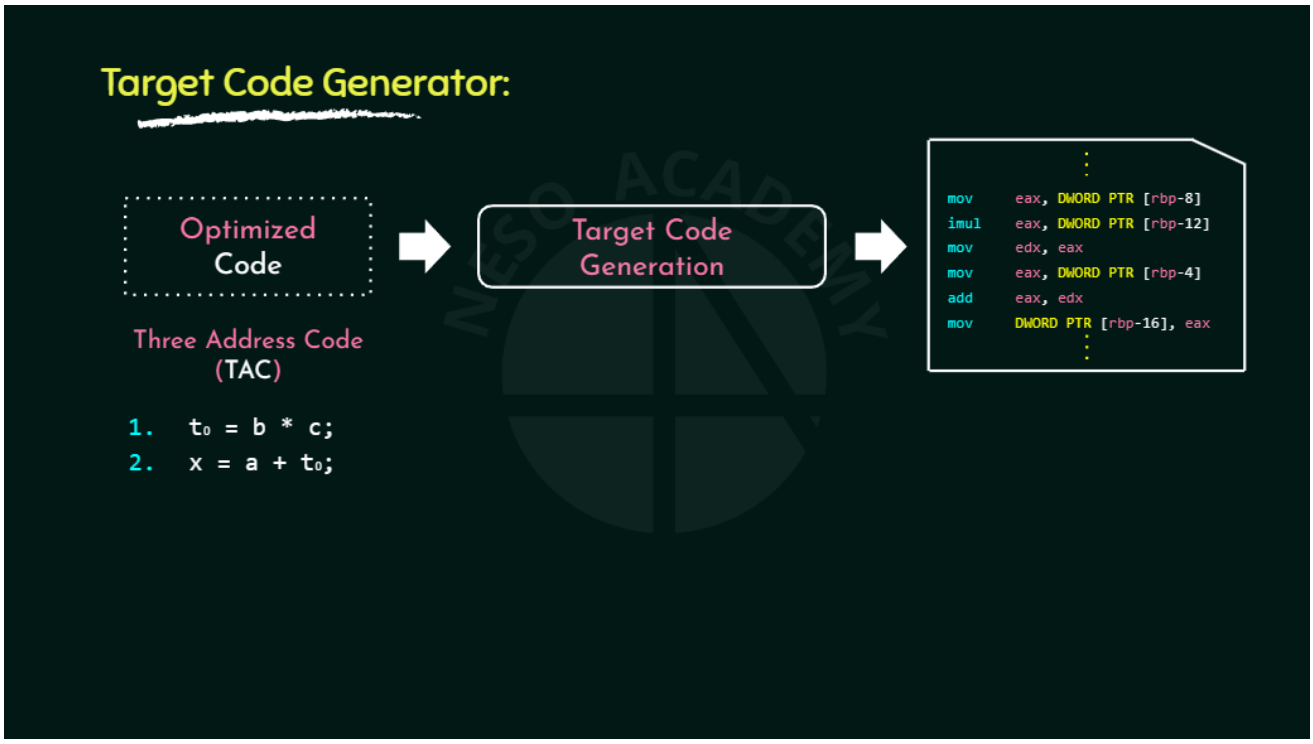
Semantic AnalysisSemantic Analyzer:Sid=E;E+TT*FTFFidididParse Tree Semantically VerifiedParse Tree



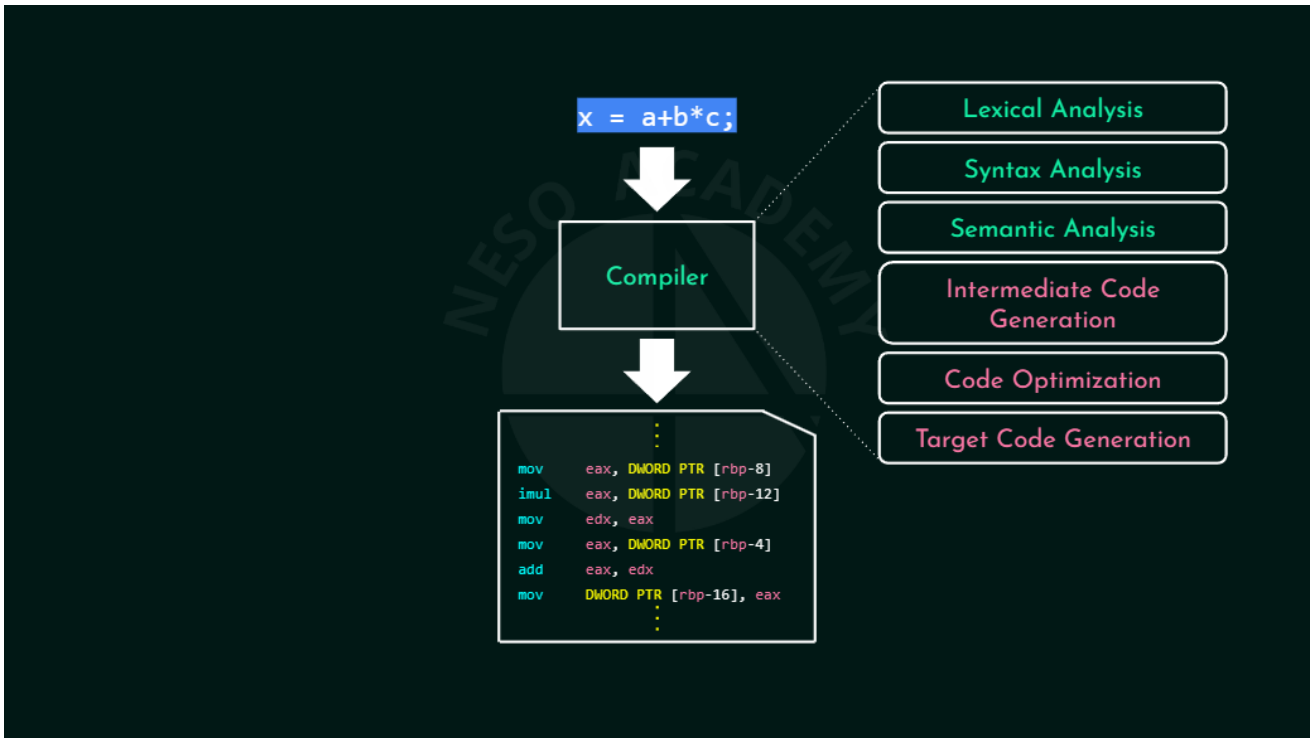
Intermediate Code Generator:Semantically VerifiedParse Tree Intermediate Code GenerationIntermediate Code x = a+b*c;id = id + id * id ;⇆Three Address Code (TAC)1.t_0 = b * c;2.t_1 = a + t_0;3.x = t_1;



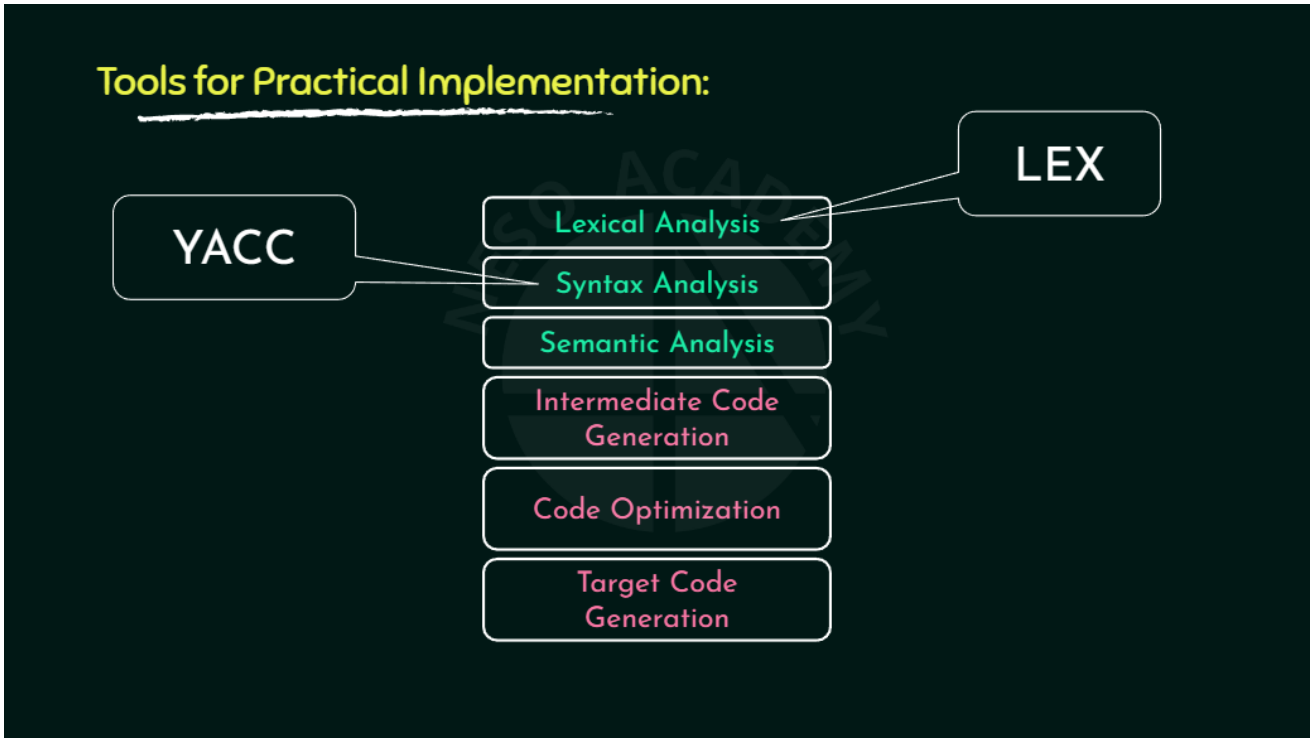
Code OptimizationIntermediate Code Three Address Code (TAC)1.t_0 = b * c;2.t_1 = a + t_0;3.x = t_1;Code Optimizer:Optimized Code Three Address Code (TAC)1.t_0 = b * c;2.x = a + t_0;



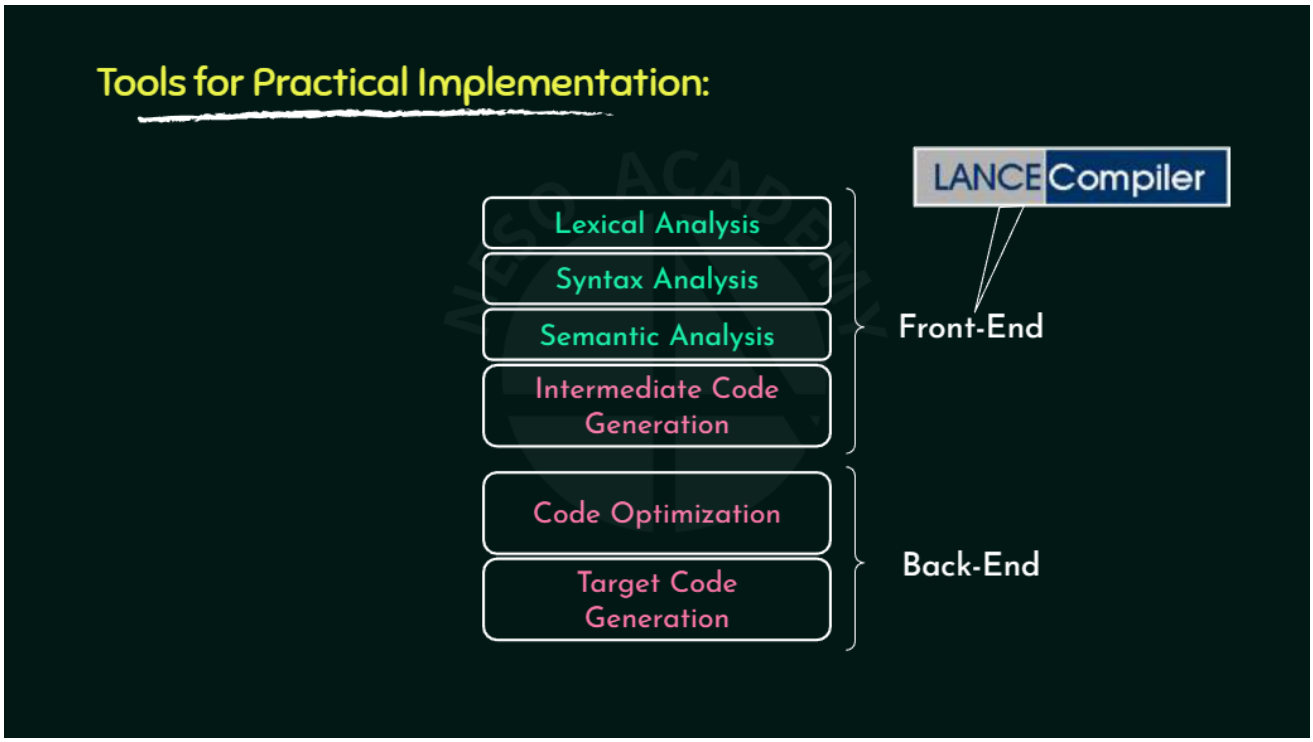
Optimized Code Three Address Code (TAC)1.t_0 = b * c;2.x = a + t_0;Target Code Generator:Target Code Generationmov eax, DWORD PTR [rbp-8]imul eax, DWORD PTR [rbp-12]mov edx, eaxmov eax, DWORD PTR [rbp-4]add eax, edxmov DWORD PTR [rbp-16], eax



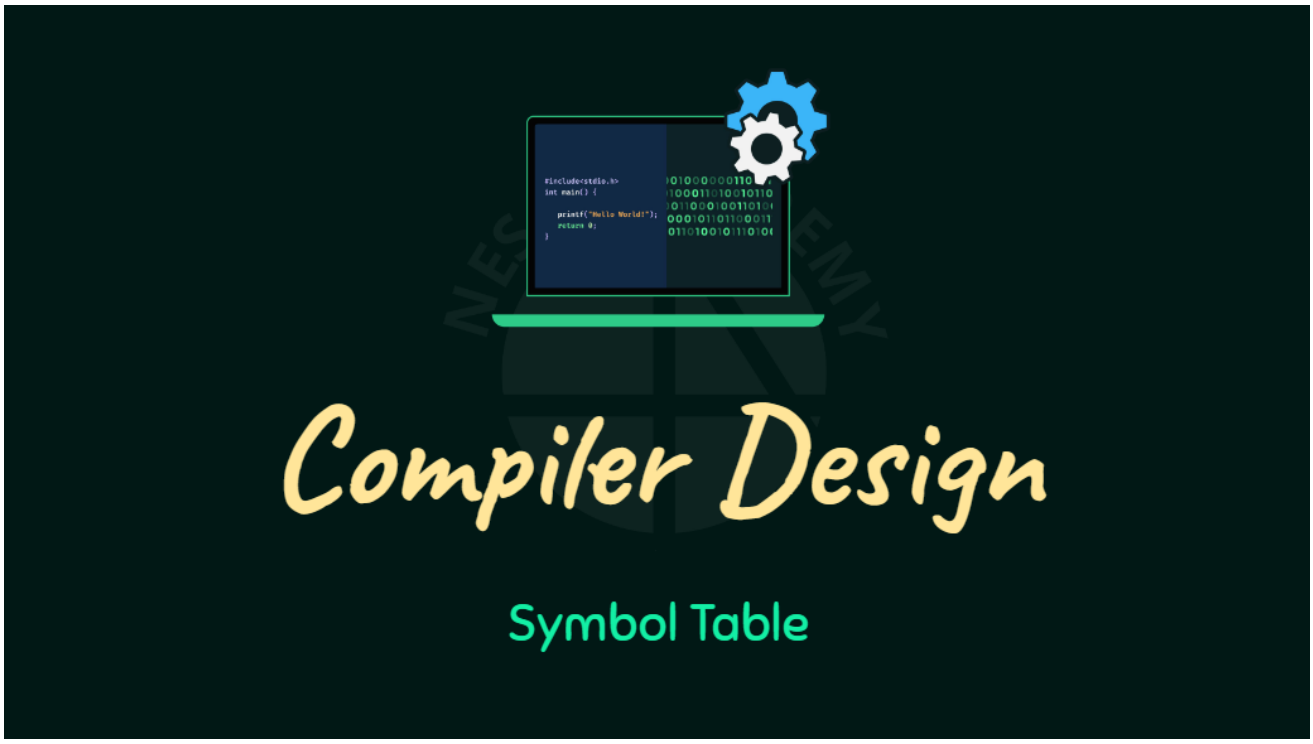
mov eax, DWORD PTR [rbp-8]imul eax, DWORD PTR [rbp-12]mov edx, eaxmov eax, DWORD PTR [rbp-4]add eax, edxmov DWORD PTR [rbp-16], eaxx = a+b*c;CompilerLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code Generation



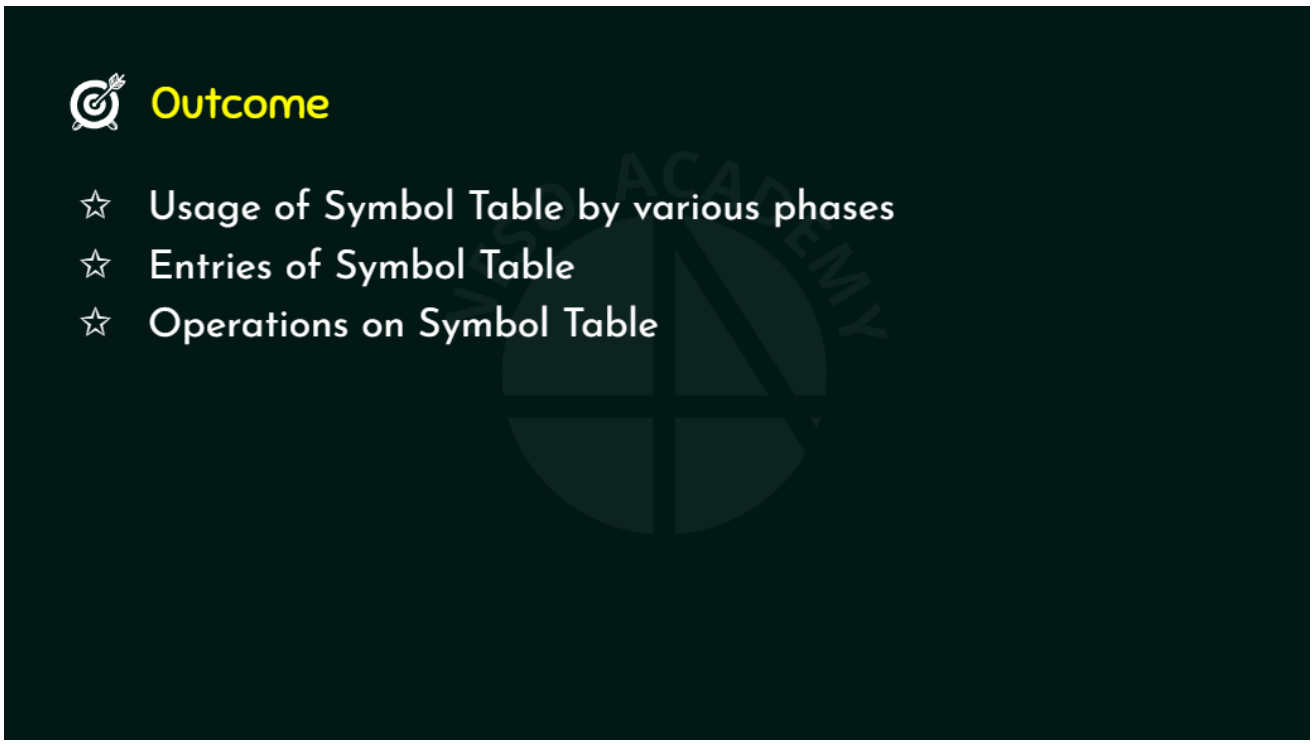
Lexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationTools for Practical Implementation:LEXYACC



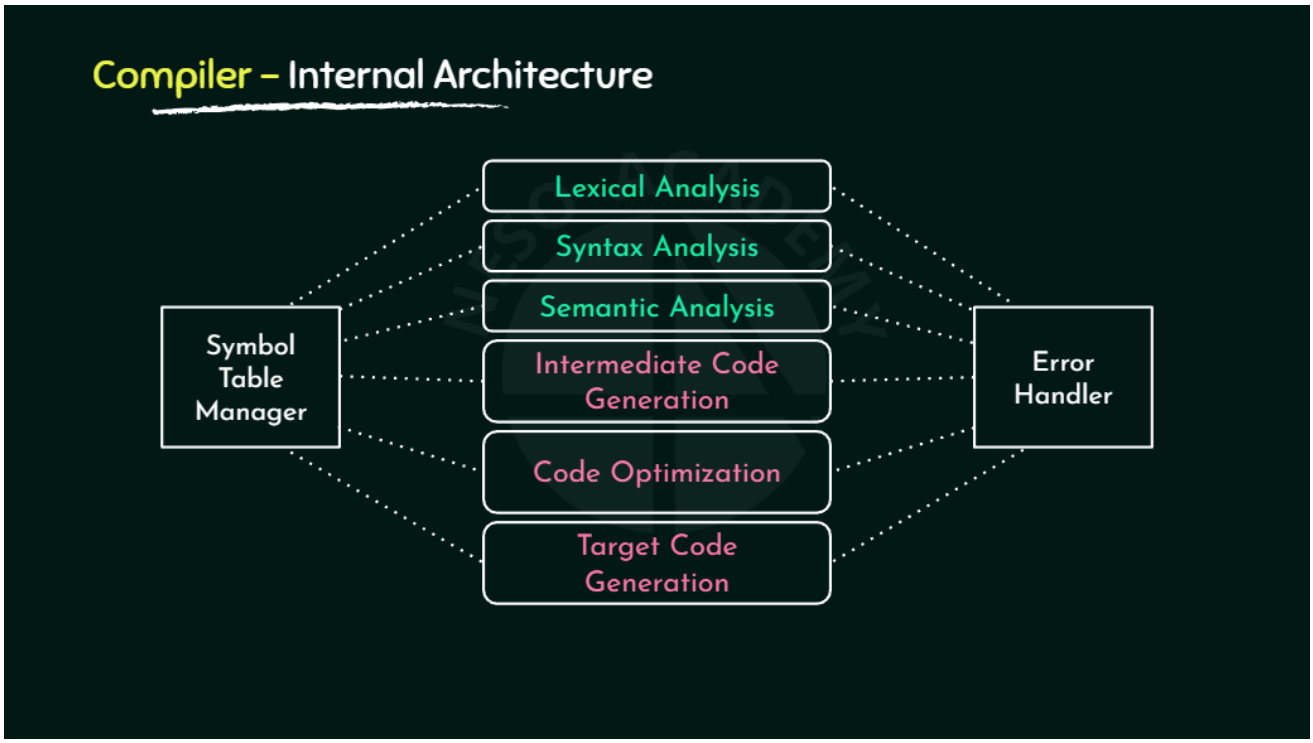
Lexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationFront-EndBack-EndTools for Practical Implementation:



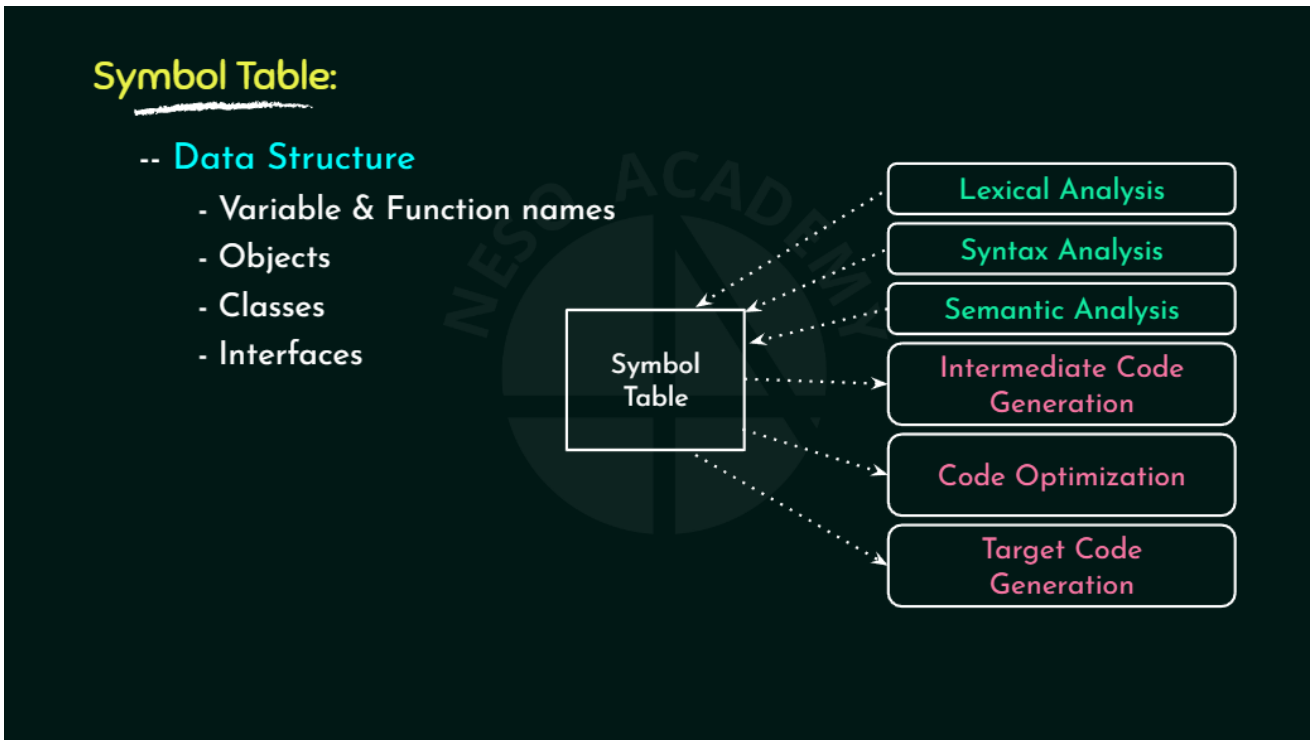
Compiler DesignSymbol Table



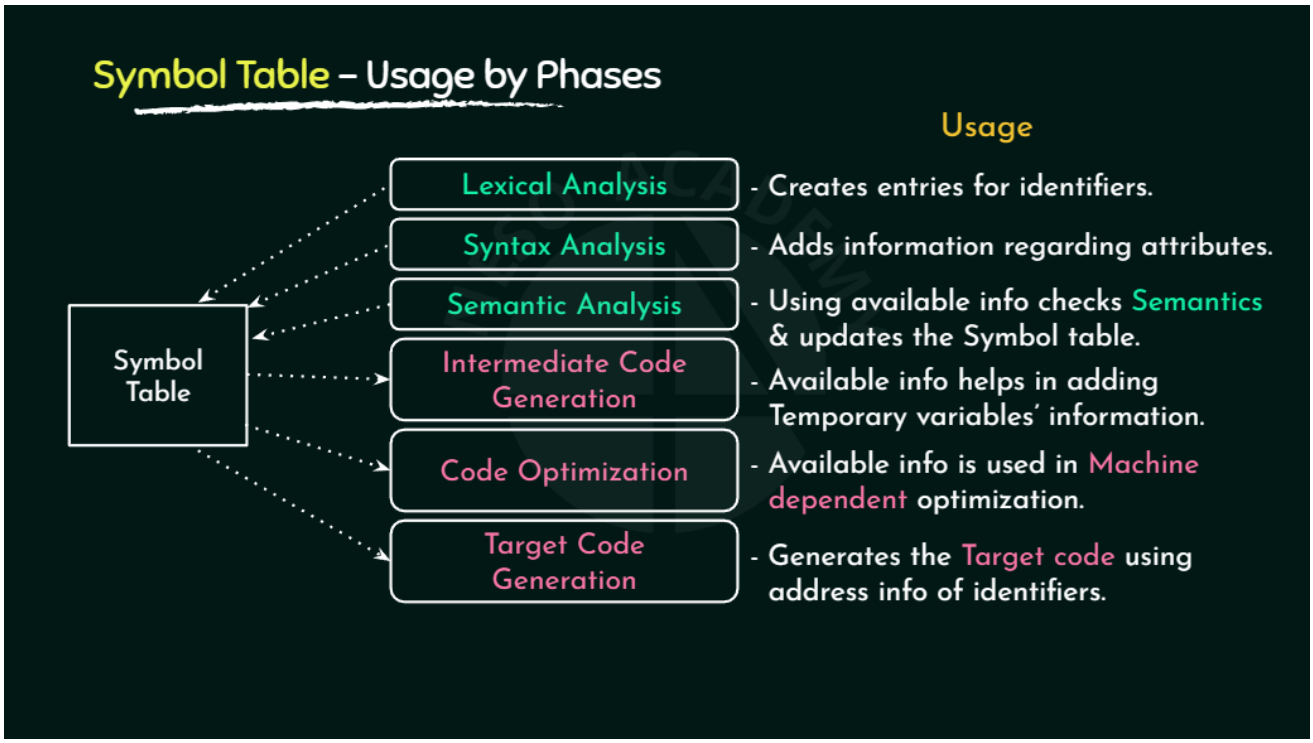
Outcome☆Entries of Symbol Table☆Usage of Symbol Table by various phases☆Operations on Symbol Table



Compiler - Internal ArchitectureLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationSymbol Table ManagerError Handler



Symbol Table:Lexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationSymbol Table -- Data Structure- Variable & Function names- Objects- Classes- Interfaces



Symbol Table - Usage by PhasesLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationSymbol Table Usage- Creates entries for identifiers.- Adds information regarding attributes.- Using available info checks Semantics & updates the Symbol table. - Available info helps in adding Temporary variables' information.- Available info is used in Machine dependent optimization.- Generates the Target code using address info of identifiers.

Symbol Table – Entries

```
int count;
char x[] = "NESO ACADEMY";
```

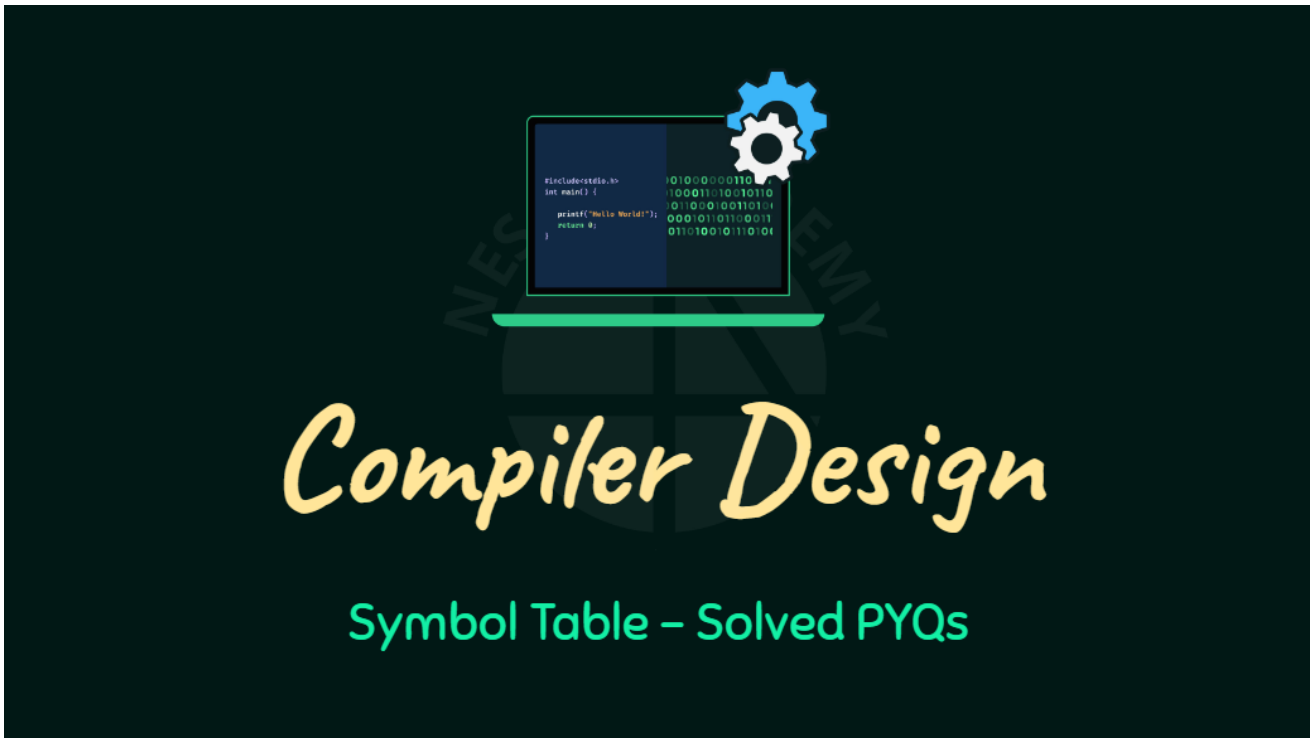
Name	Type	Size	Dimension	Line of Declaration	Line of Usage	Address
count	int	2	0	--	--	--
x	char	12	1	--	--	--

Symbol Table - EntriesNameTypeSizeDimensionLine of DeclarationLine of UsageAddresschar x[] = “NESO ACADEMY”;int count;countint20------xchar121------

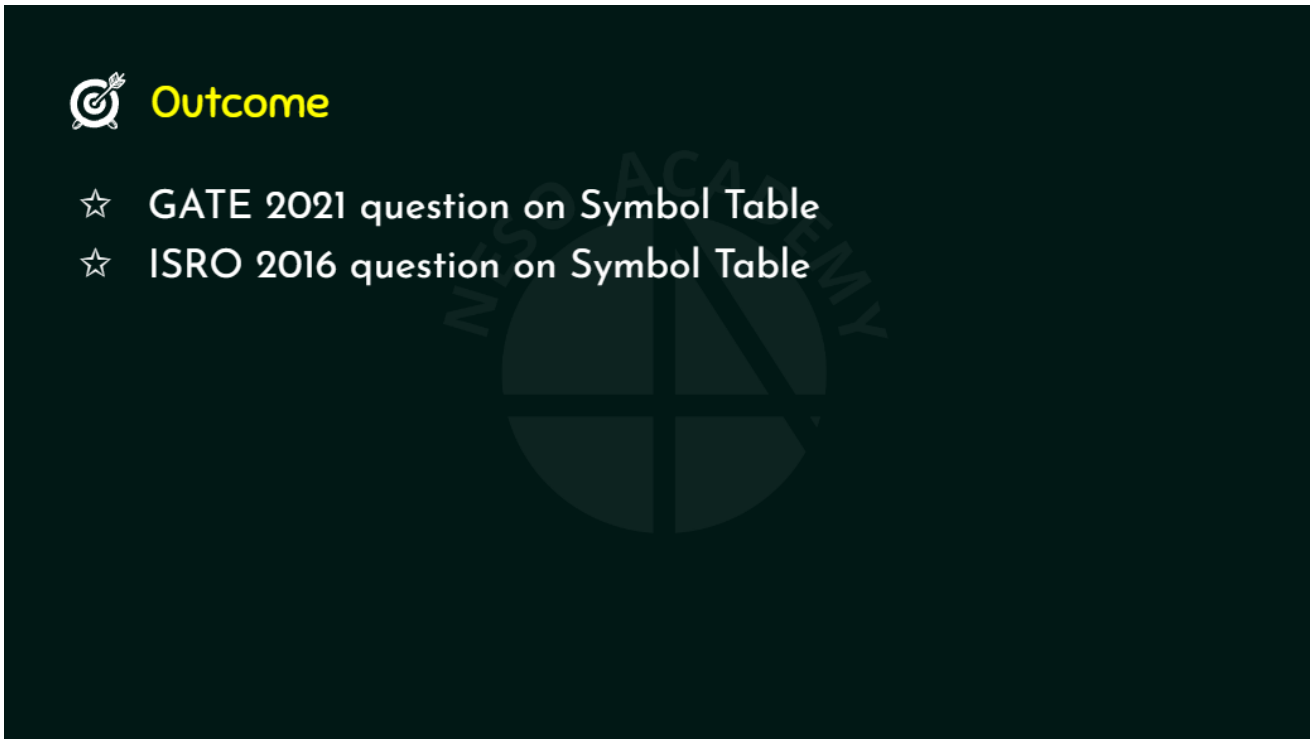
Symbol Table – Operations

- Non-Block Structured Language:
  - -- Contains single instance of the variable declaration.
  - -- Operations:
    - i. Insert() ii. Lookup()
- Block Structured Language:
  - -- Variable declaration may happen multiple times.
  - -- Operations:
    - i. Insert() ii. Lookup() iii. Set() iv. Reset()

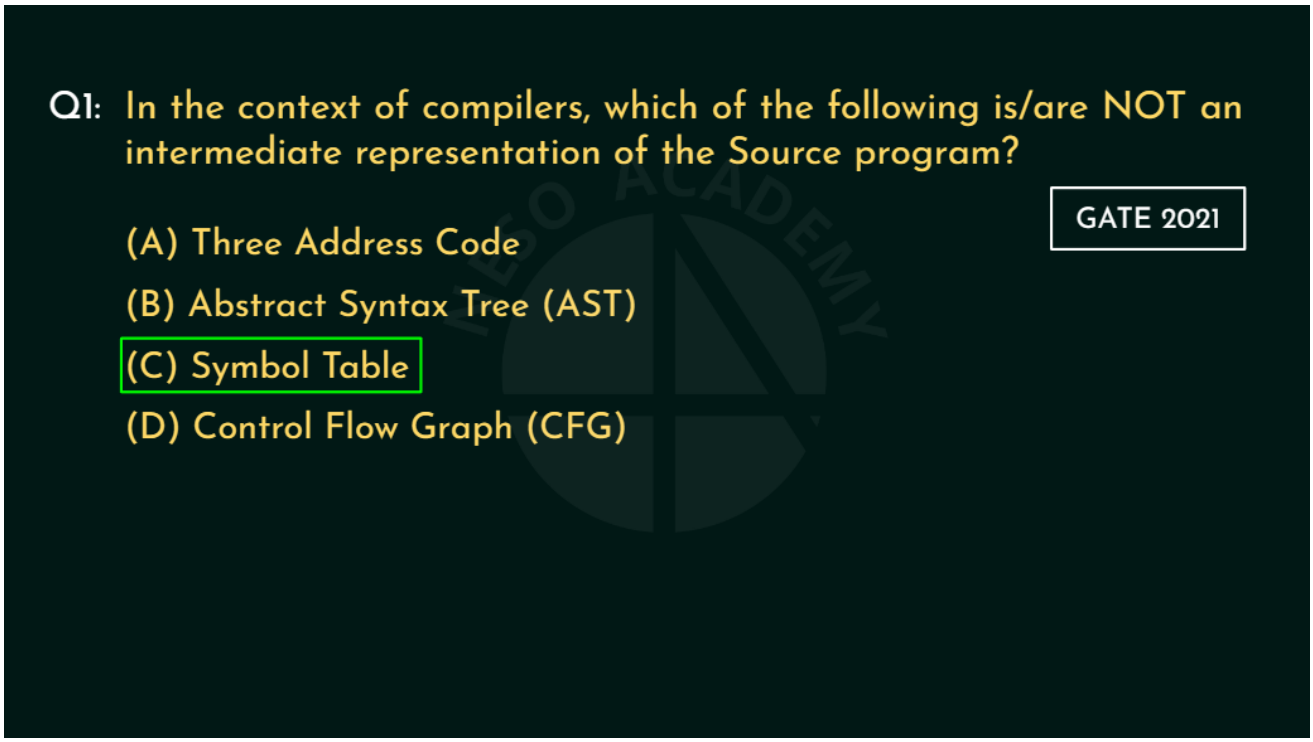
Symbol Table - Operations●Non-Block Structured Language:-- Contains single instance of the variable declaration.-- Operations:i. Insert()ii. Lookup()●Block Structured Language:-- Variable declaration may happen multiple times.-- Operations:i. Insert()ii. Lookup()iii. Set()iv. Reset()



Compiler DesignSymbol Table - Solved PYQs



Outcome☆ISRO 2016 question on Symbol Table☆GATE 2021 question on Symbol Table



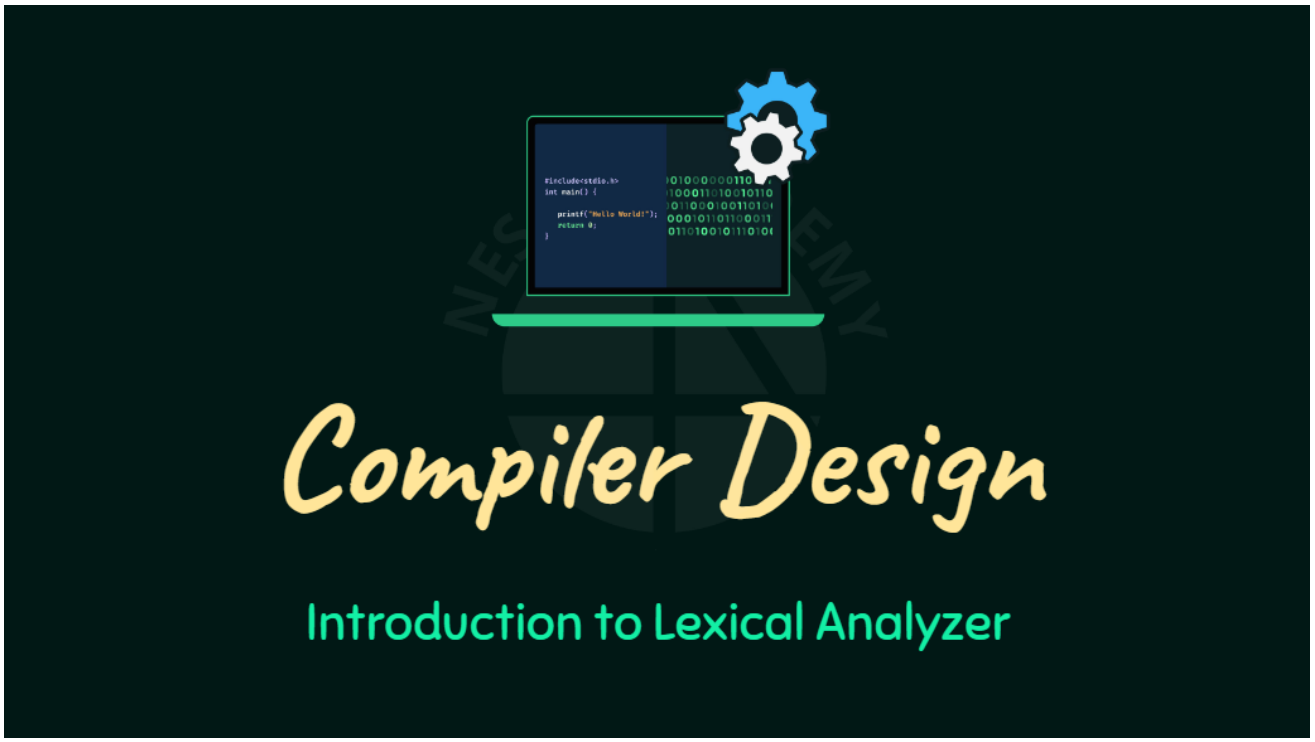
Q1:In the context of compilers, which of the following is/are NOT an intermediate representation of the Source program?(A) Three Address Code(B) Abstract Syntax Tree (AST)(C) Symbol Table(D) Control Flow Graph (CFG)GATE 2021

Q2:Access time of the Symbol table will be logarithmic if it is implemented by(A) Linear List(B) Search Tree(C) Hash Table(D) None of theseISRO 2016

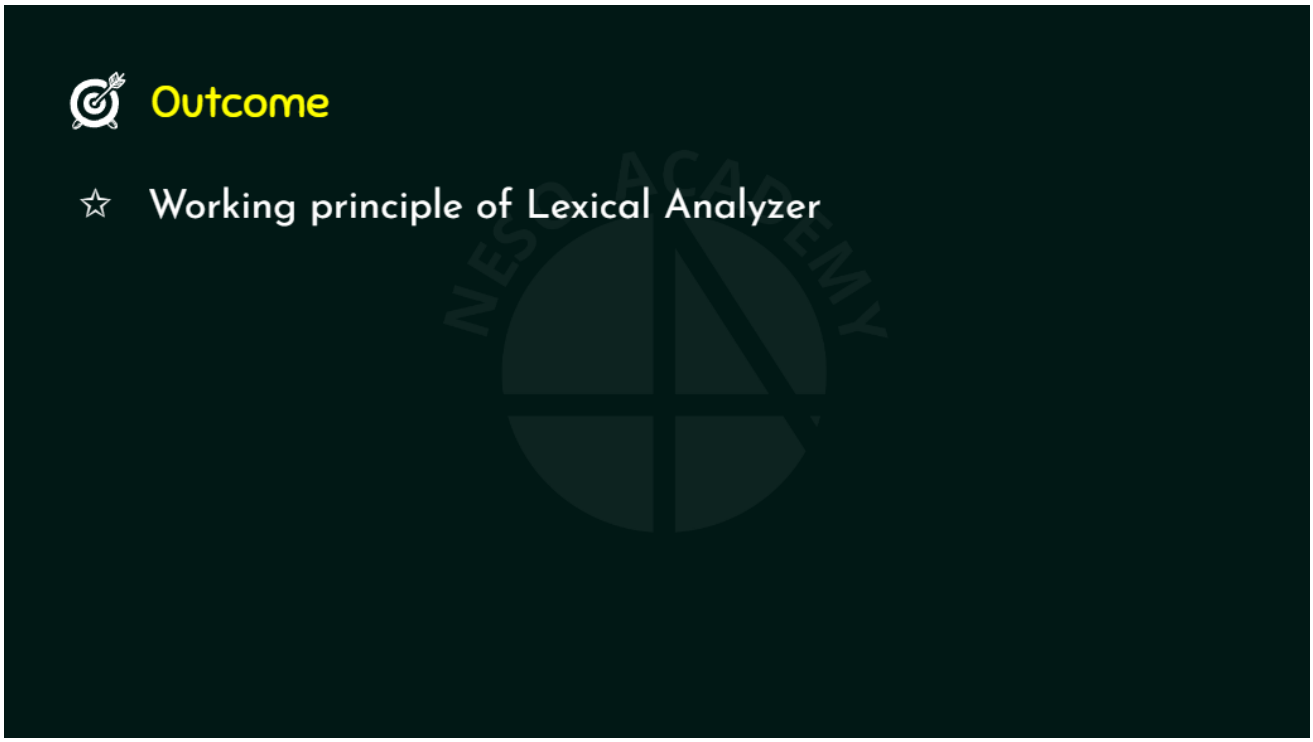
Symbol Table – Various Implementations:

Implementation	Insertion Time	Lookup Time	Disadvantages
A. Linear Lists i. Ordered Lists a. Arrays b. Linked Lists ii. Unordered Lists	 O(n) O(n) O(1)	 O(logn) O(n) O(n)	i. For Ordered Lists, every insertion is preceded by lookup operation. ii. Access time is directly proportional to table size.
B. Search Tree	$O(\log_m n)$	$O(\log_m n)$	Always needs to be balanced.
C. Hash Table	O(1)	O(1)	Too many collisions increases the Time complexity to O(n).

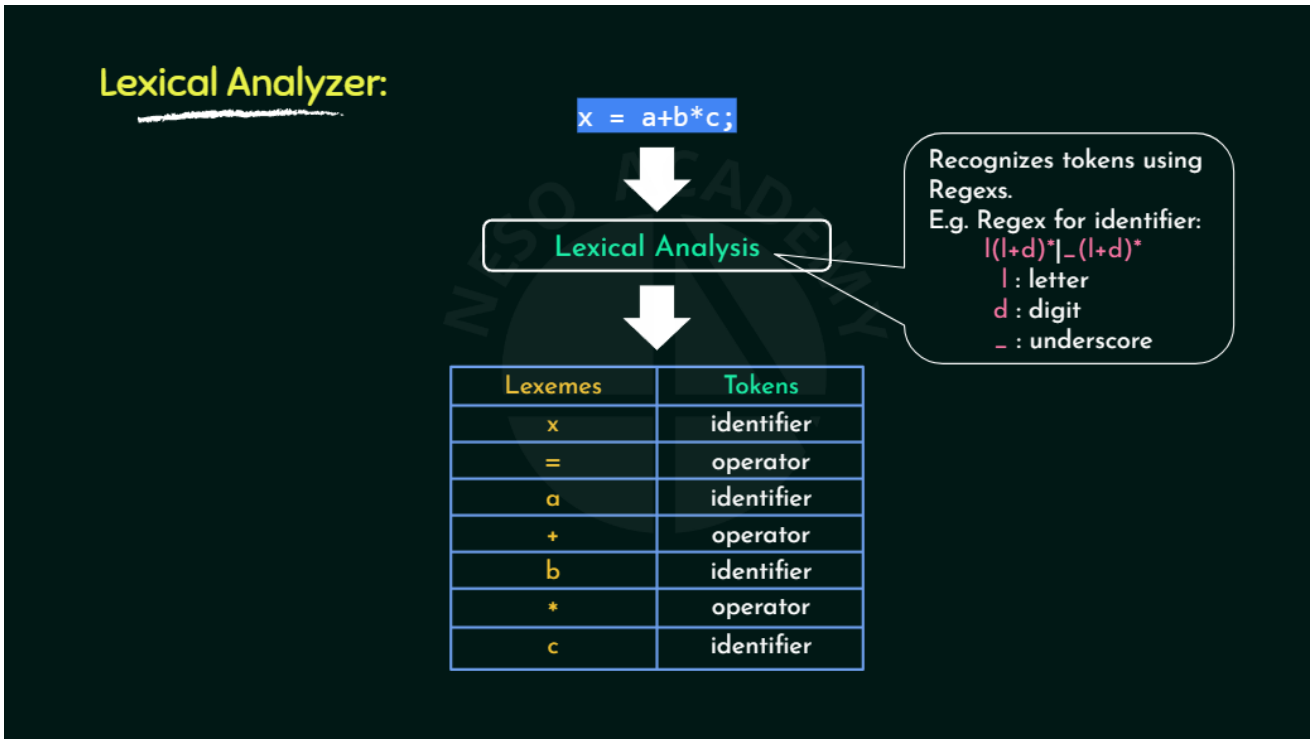
Symbol Table - Various Implementations:ImplementationInsertion TimeLookup TimeDisadvantagesA. Linear Listsi. Ordered Listsii. Unordered Listsa. Arraysb. Linked ListsB. Search TreeC. Hash TableO(n)O(logn)O(n)O(n)O(1)O(n)O(log n)mO(log n)mO(1)O(1)i. For Ordered Lists, every insertion is preceded by lookup operation.ii. Access time is directly proportional to table size.Always needs to be balanced.Too many collisions increases the Time complexity to O(n).



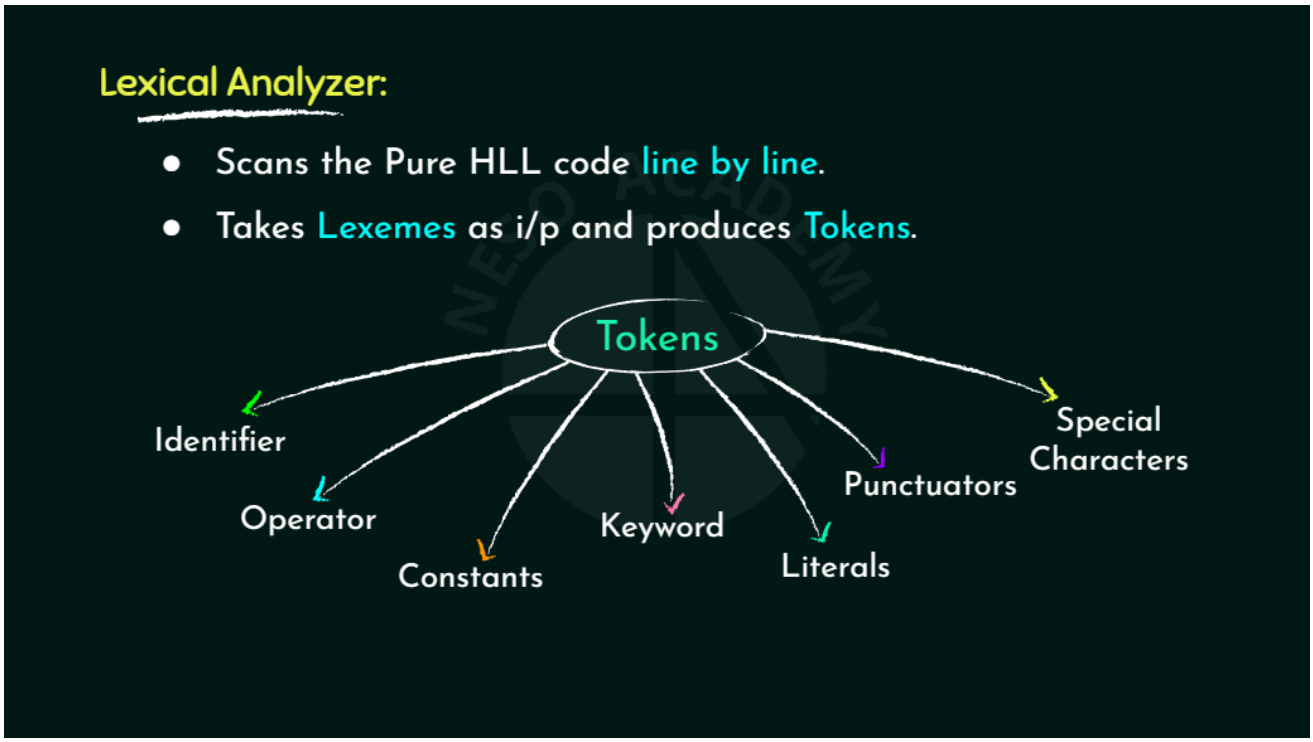
Compiler DesignIntroduction to Lexical Analyzer



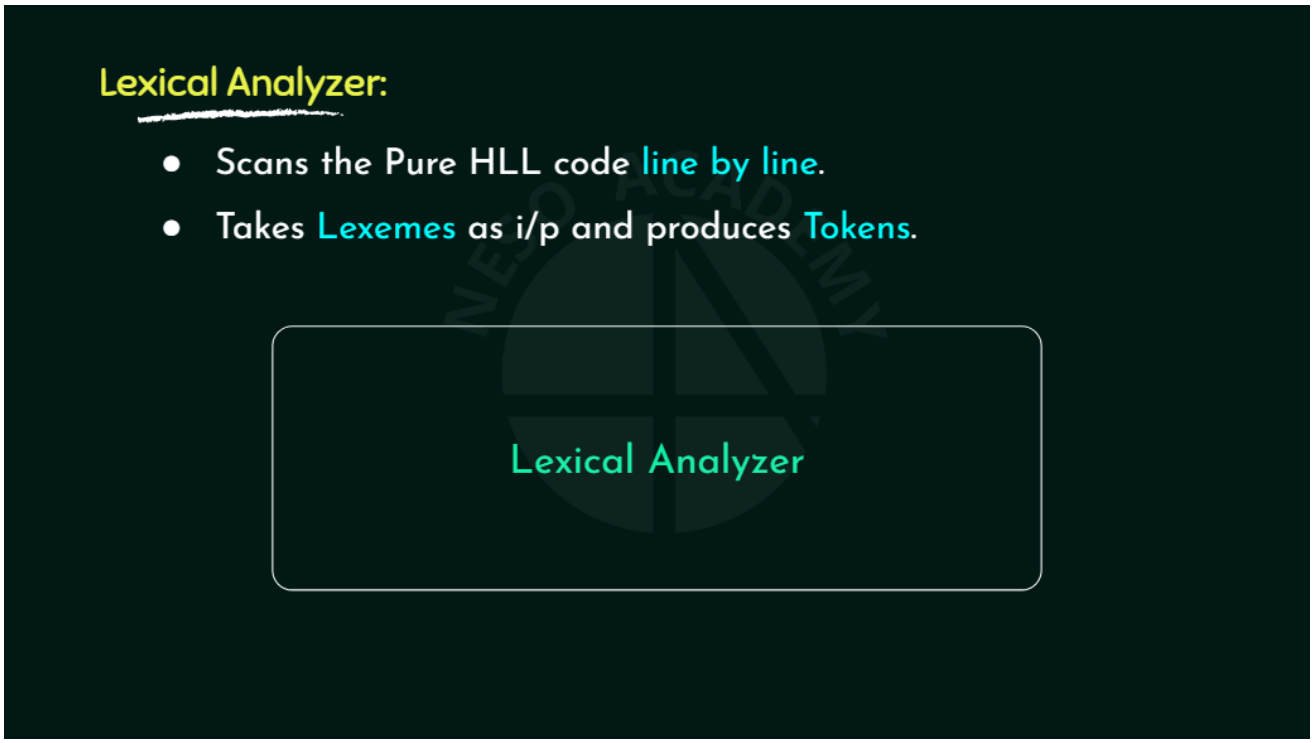
Outcome☆Working principle of Lexical Analyzer



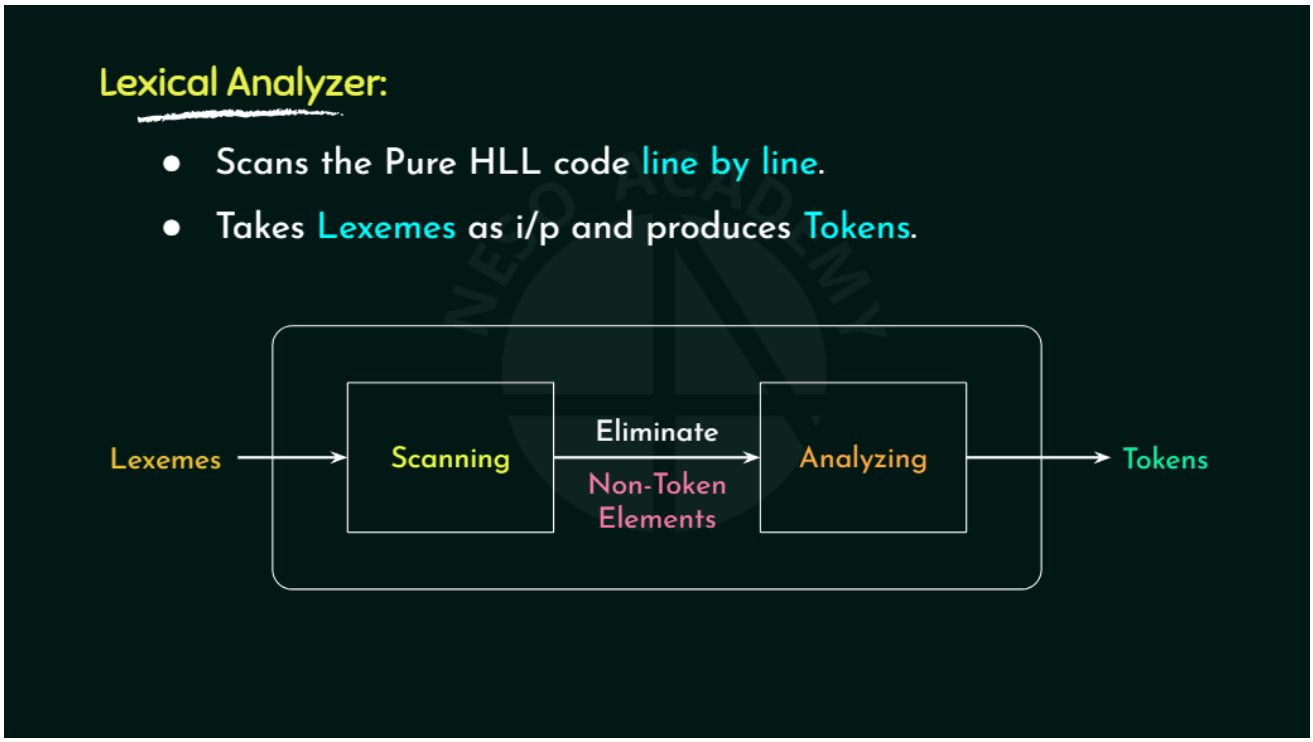
Lexical AnalysisLexical Analyzer:x = a+b*c;LexemesTokensxidentifier=operatoraidentifier+operatorbidentifier*operatorcidentifierRecognizes tokens using Regexs. E.g. Regex for identifier: l(l+d)*|_(l+d)* l : letterd : digit _ : underscore



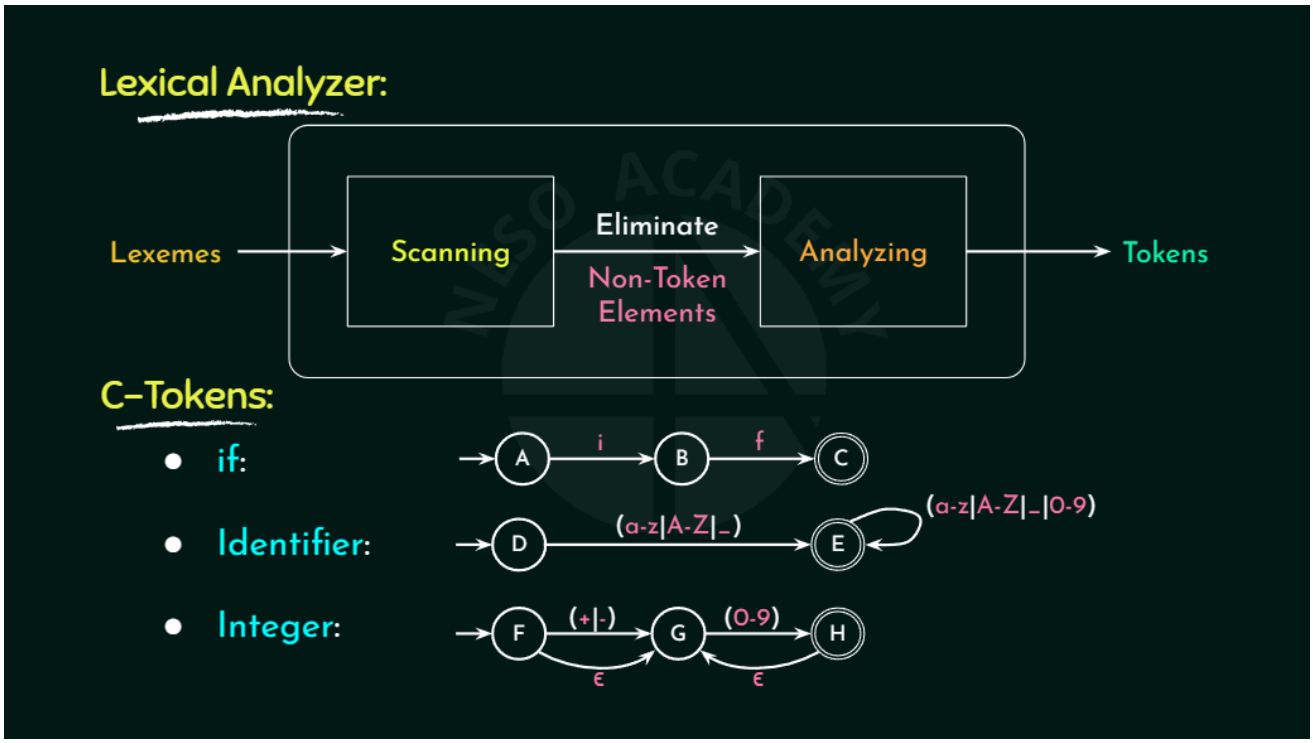
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. TokensIdentifierOperatorConstantsKeywordLiteralsPunctuatorsSpecial Characters



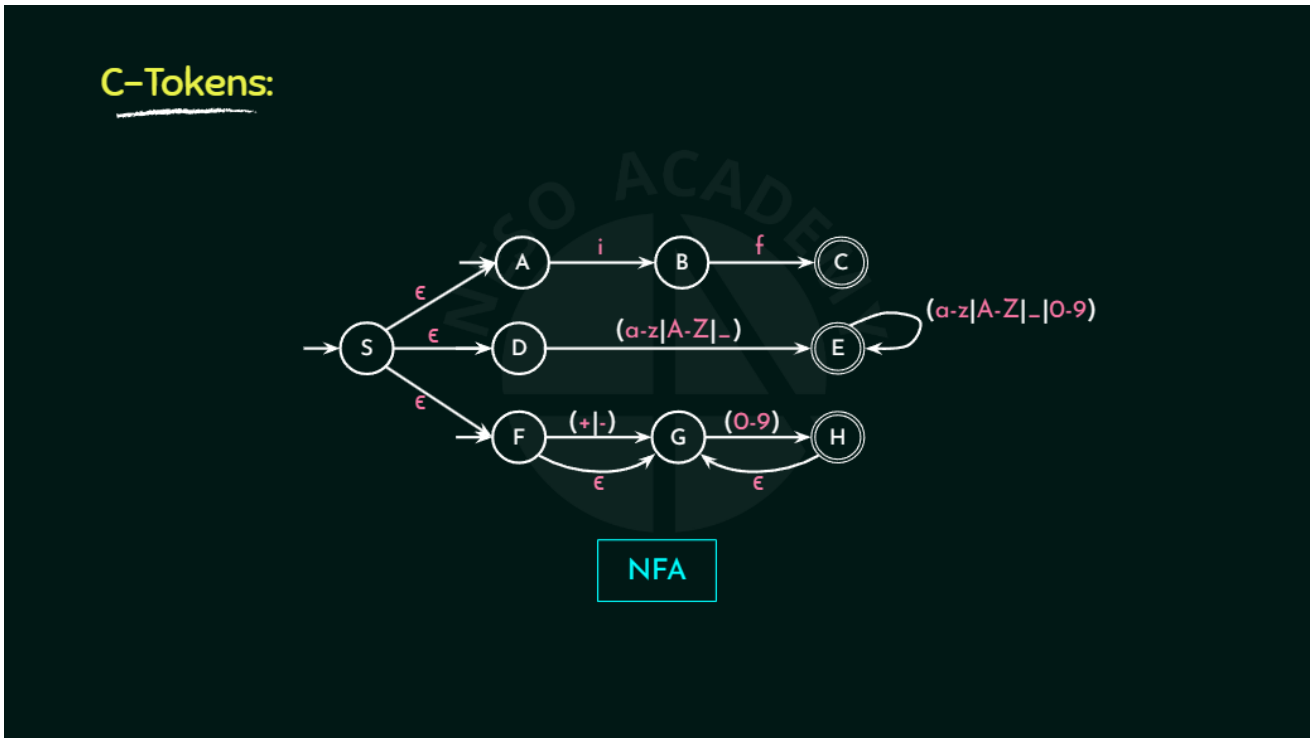
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. Lexical Analyzer



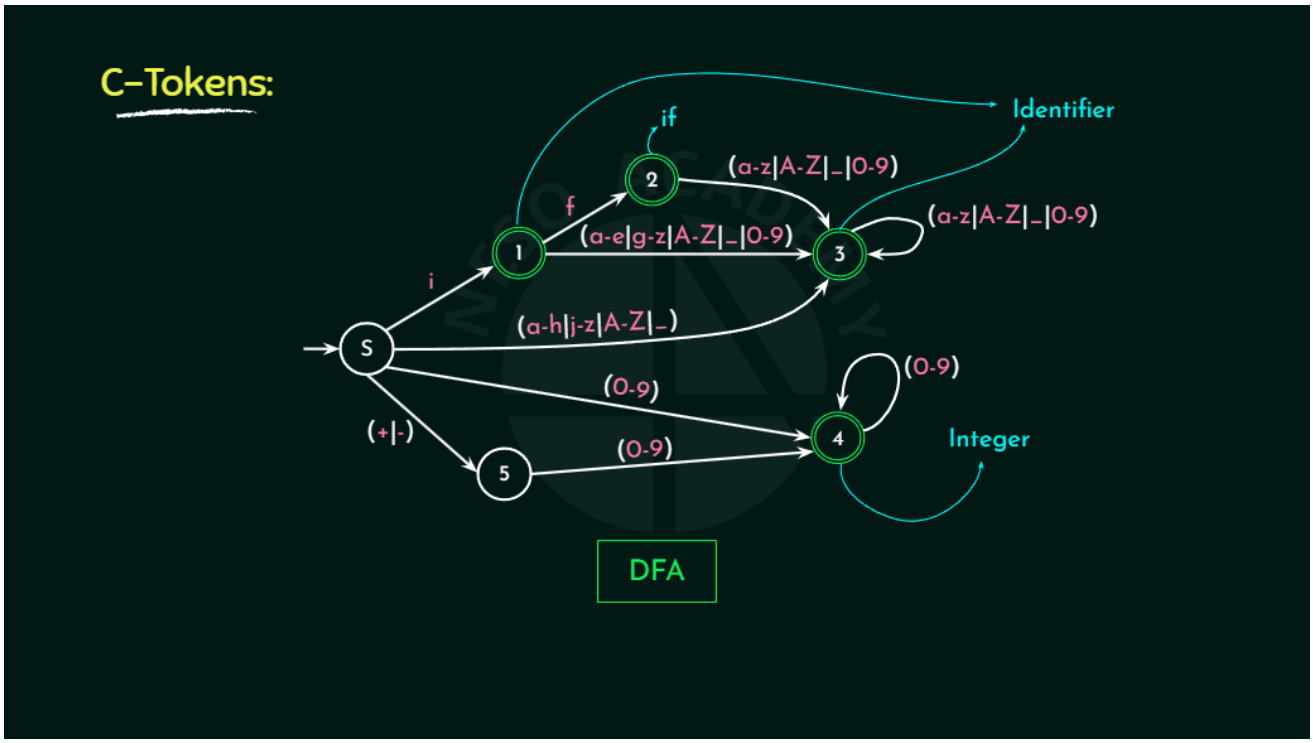
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. ScanningAnalyzingEliminateNon-TokenElementsLexemesTokens



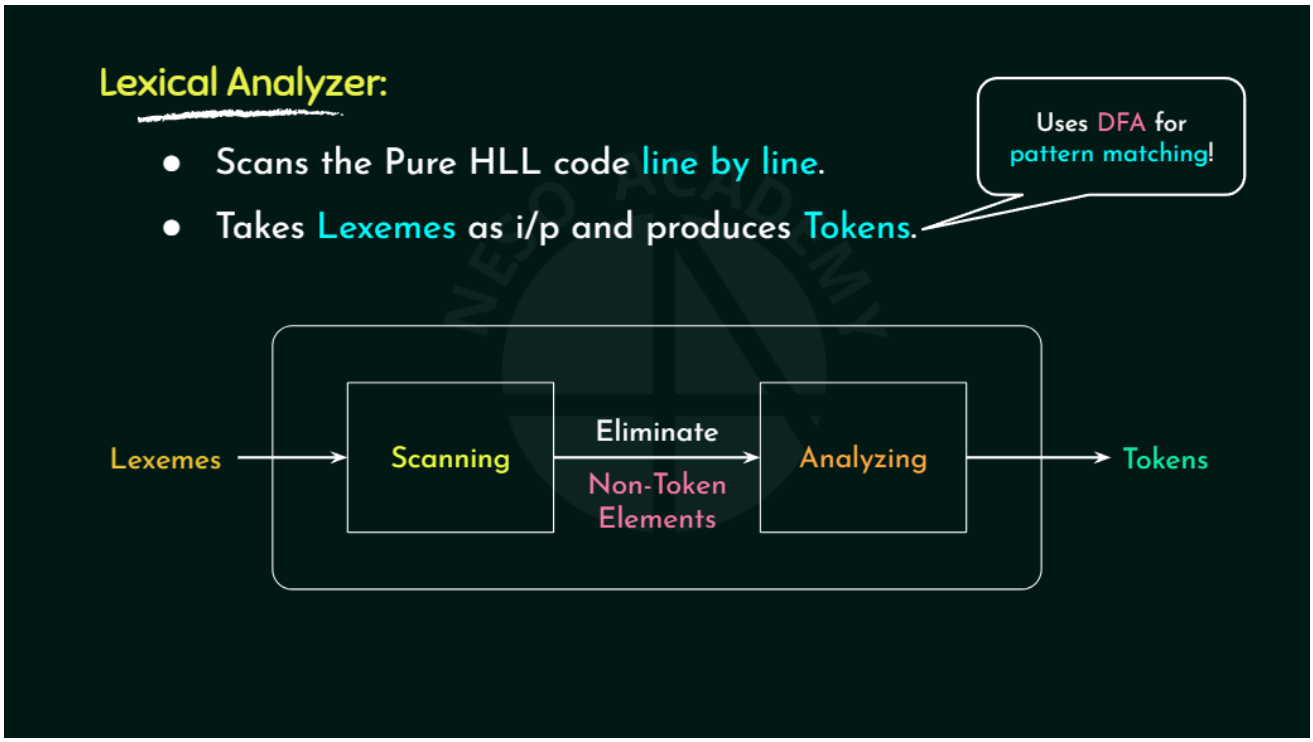
Lexical Analyzer:ScanningAnalyzingEliminateNon-TokenElementsLexemesTokensC-Tokens:●if:●Identifier:●Integer:ABDiCfE(a-z|A-Z|_)(a-z|A-Z|_|0-9)FG(+|-)H(0-9)εε



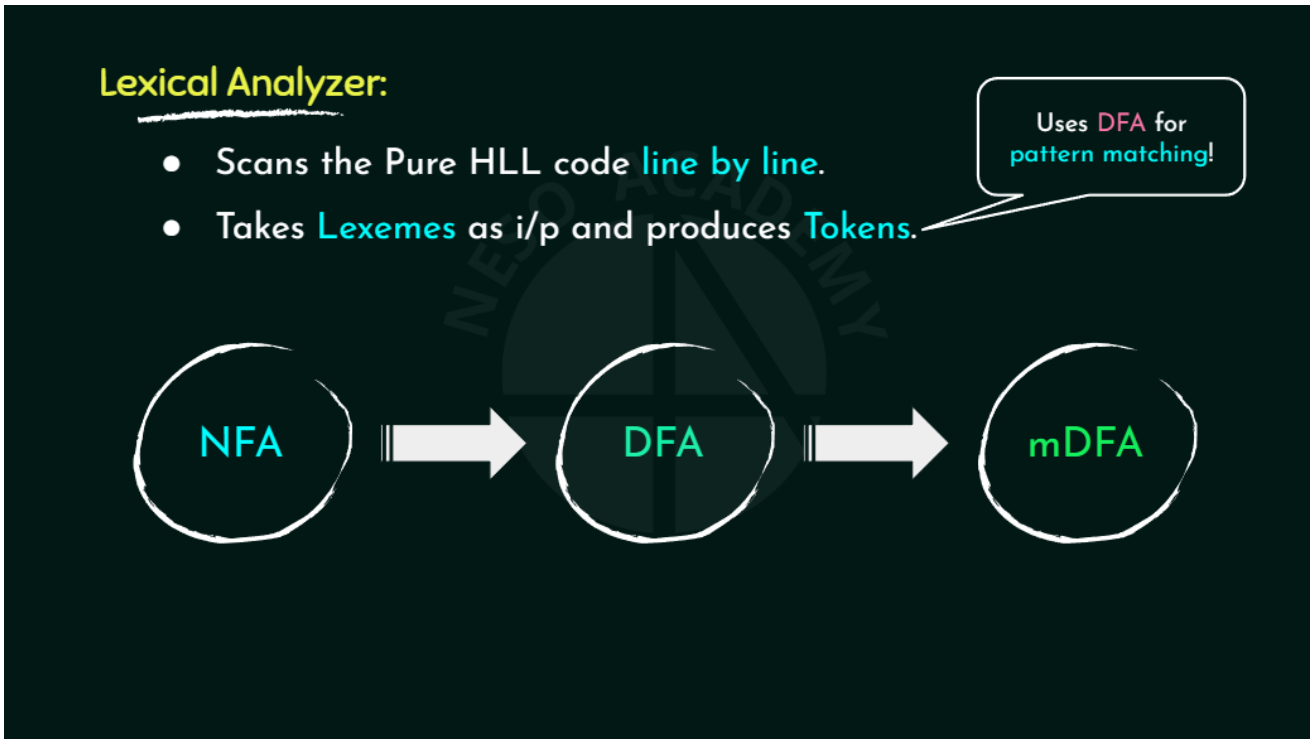
C-Tokens:ABDiCfE(a-z|A-Z|_)(a-z|A-Z|_|0-9)FG(+|-)H(0-9)ɛɛSɛɛɛNFA



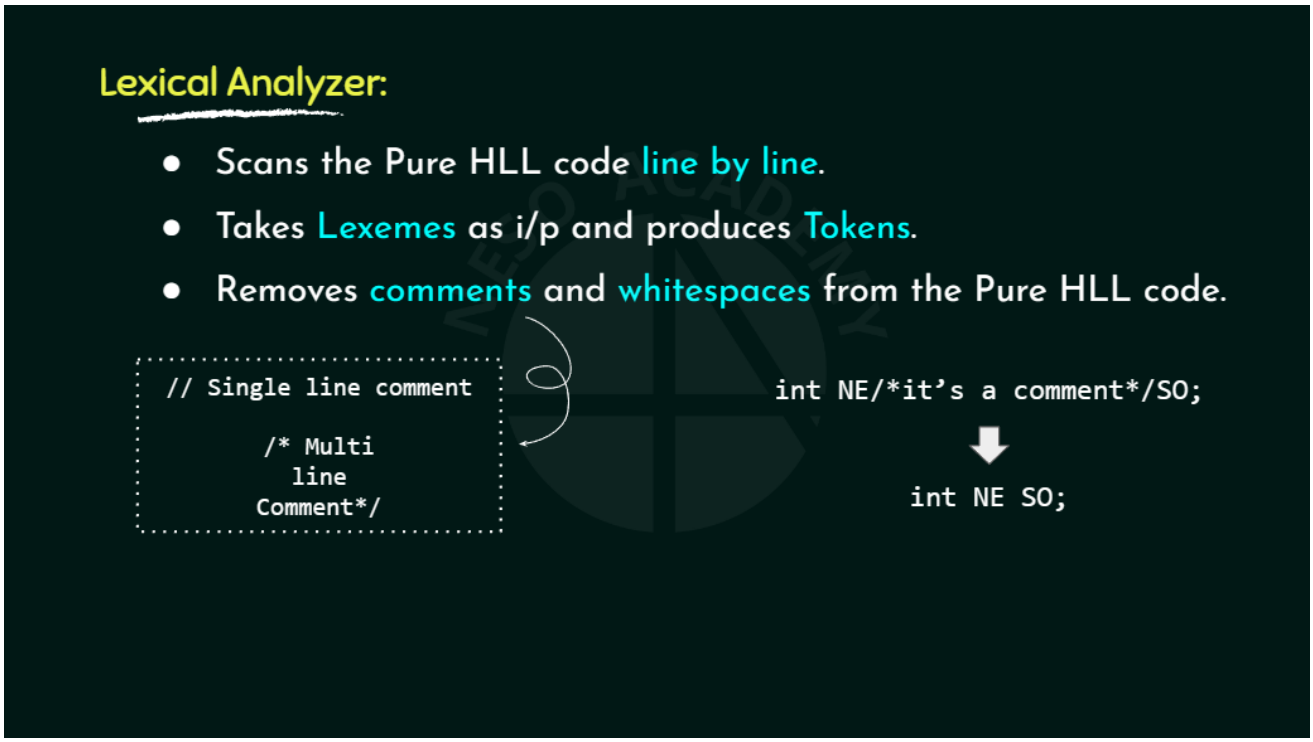
1SC-Tokens:2f3(a-e|g-z|A-Z|_|0-9)(a-z|A-Z|_|0-9)5(+|-)4(0-9)iDFA(a-h|j-z|A-Z|_)(0-9)(a-z|A-Z|_|0-9)ifIdentifierInteger(0-9)



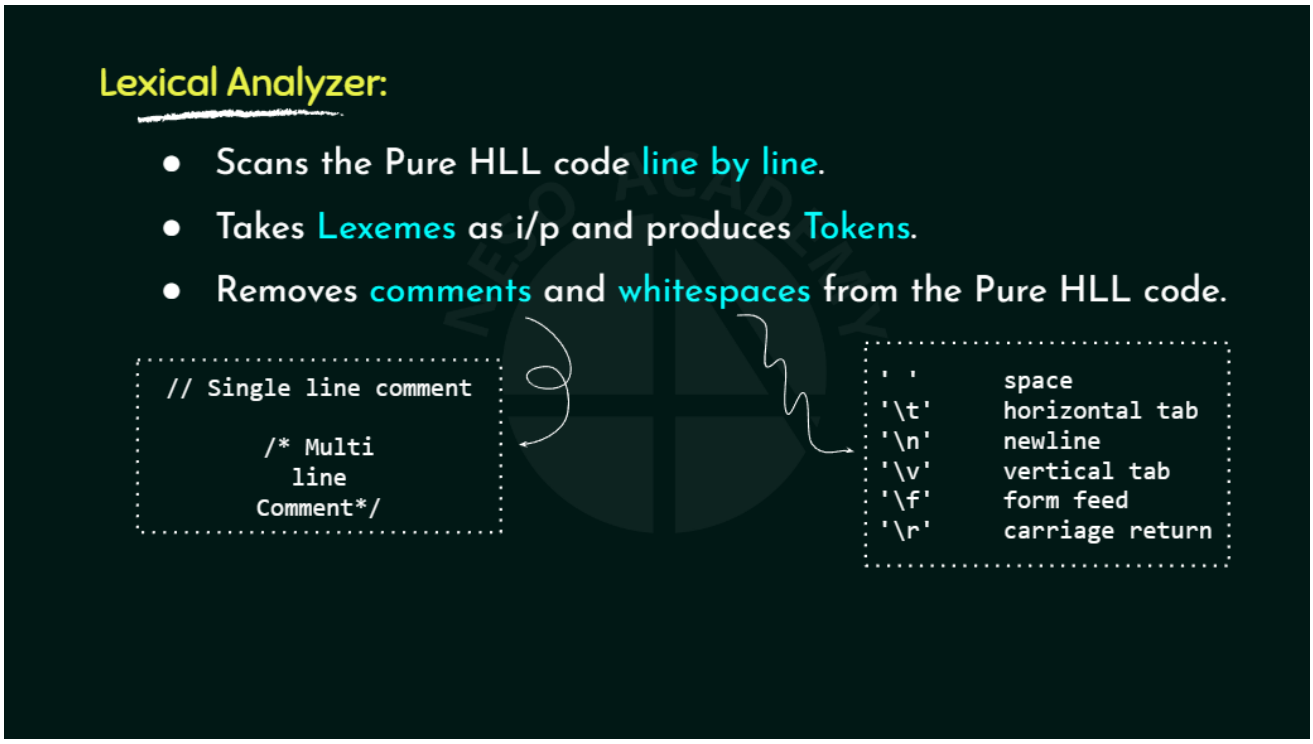
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. ScanningAnalyzingEliminateNon-TokenElementsLexemesTokensUses DFA for pattern matching!



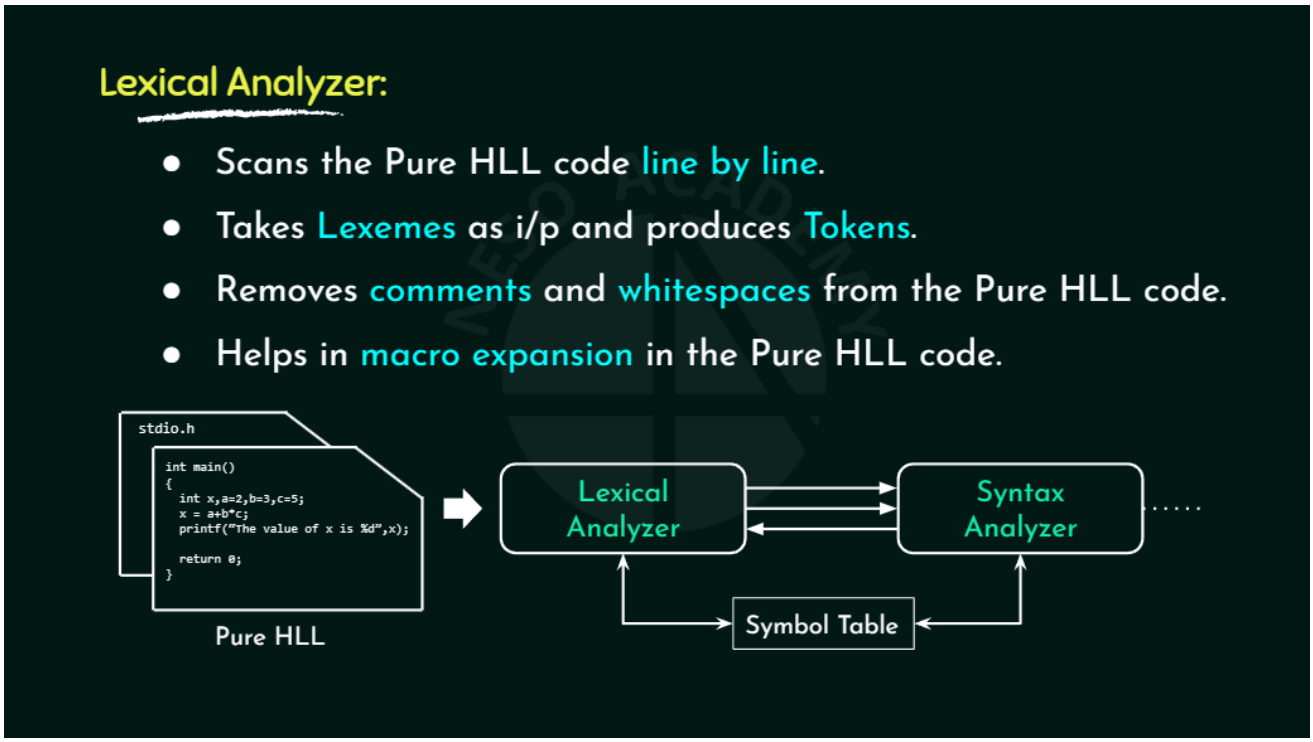
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. Uses DFA for pattern matching!NFADFAmDFA



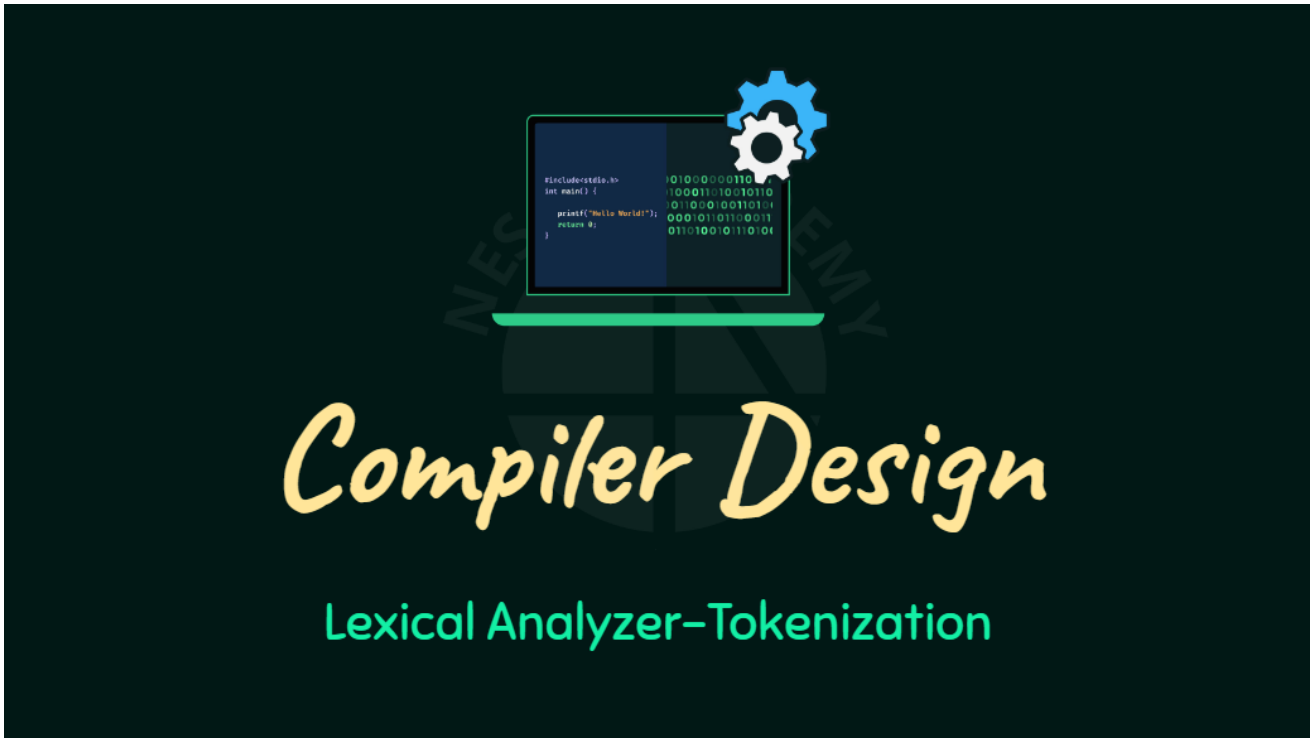
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. ●Removes comments and whitespaces from the Pure HLL code. // Single line comment/* Multiline Comment*/int NE/*it's a comment*/SO;int NE SO;



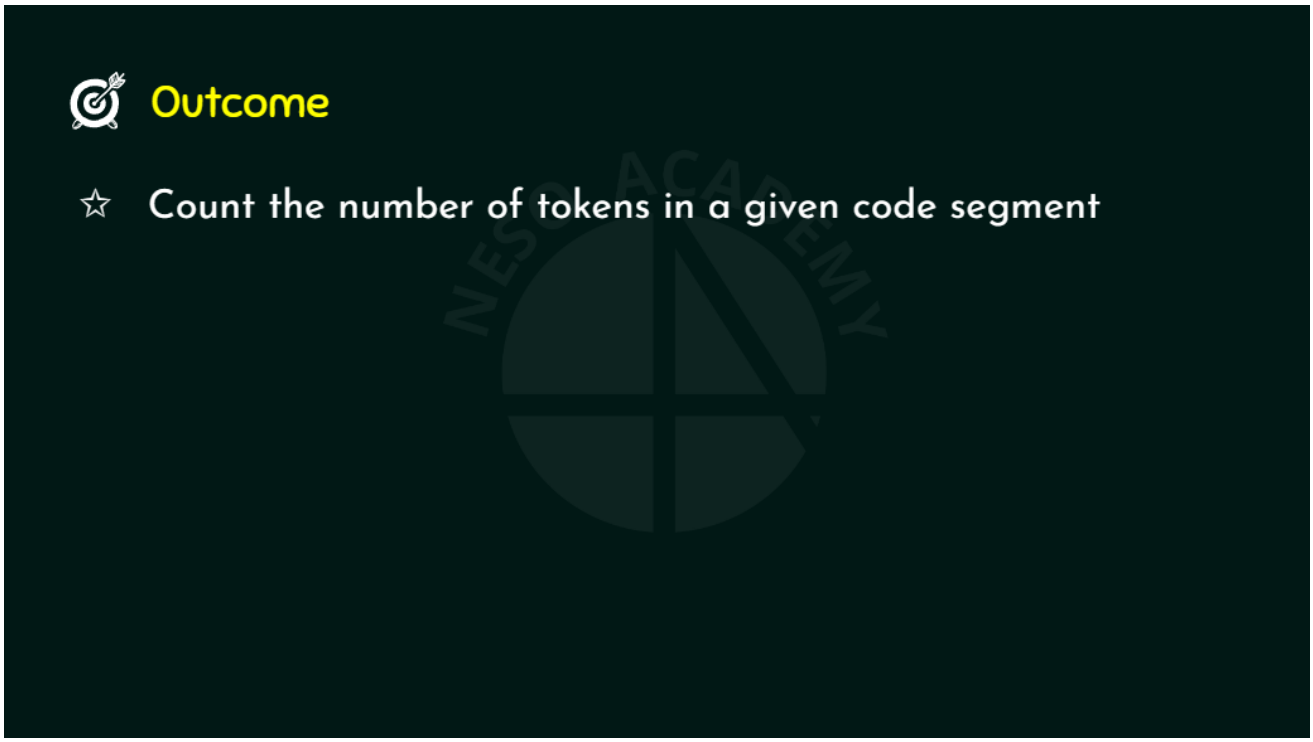
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. ●Removes comments and whitespaces from the Pure HLL code. // Single line comment/* Multiline Comment*/' ' space '\t' horizontal tab '\n' newline'\v' vertical tab '\f' form feed '\r' carriage return



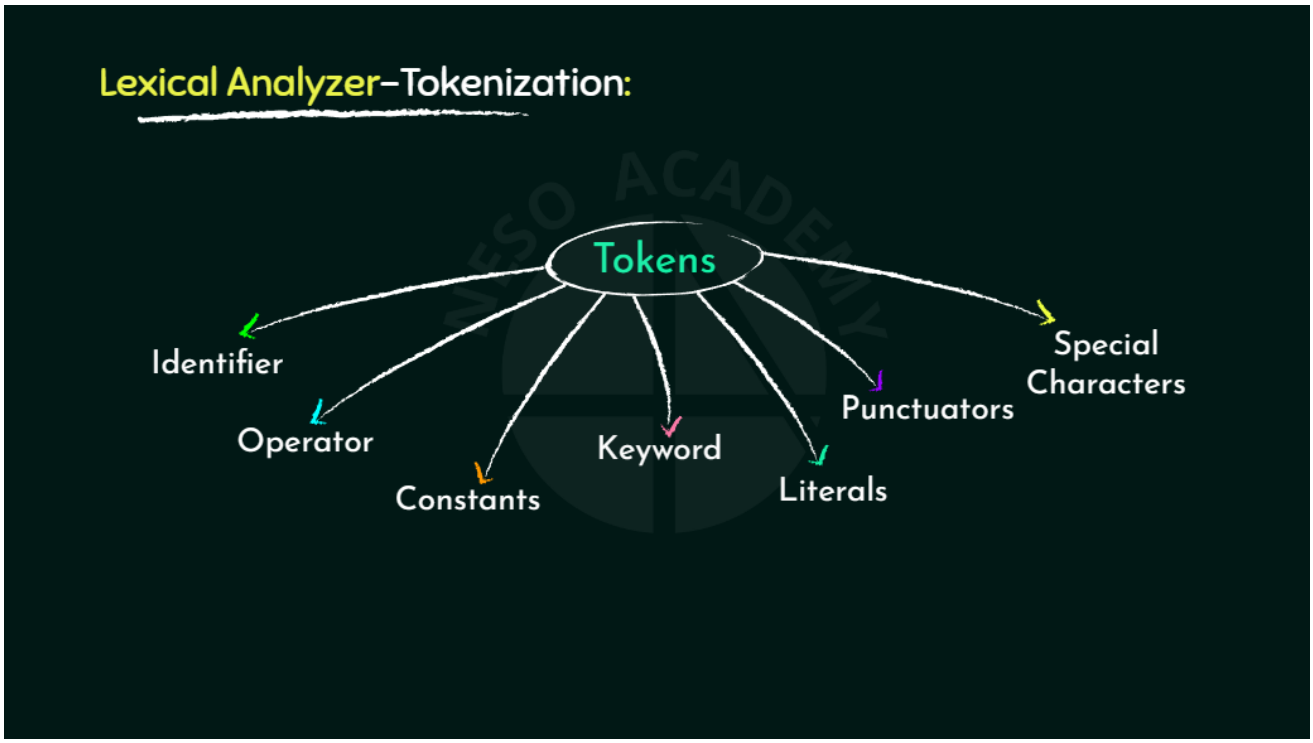
Lexical Analyzer:●Scans the Pure HLL code line by line. ●Takes Lexemes as i/p and produces Tokens. ●Removes comments and whitespaces from the Pure HLL code. ●Helps in macro expansion in the Pure HLL code. Pure HLLstdio.hint main() { int x,a=2,b=3,c=5; x = a+b*c; printf(“The value of x is %d”,x); return 0;}Lexical AnalyzerSymbol TableSyntax Analyzer



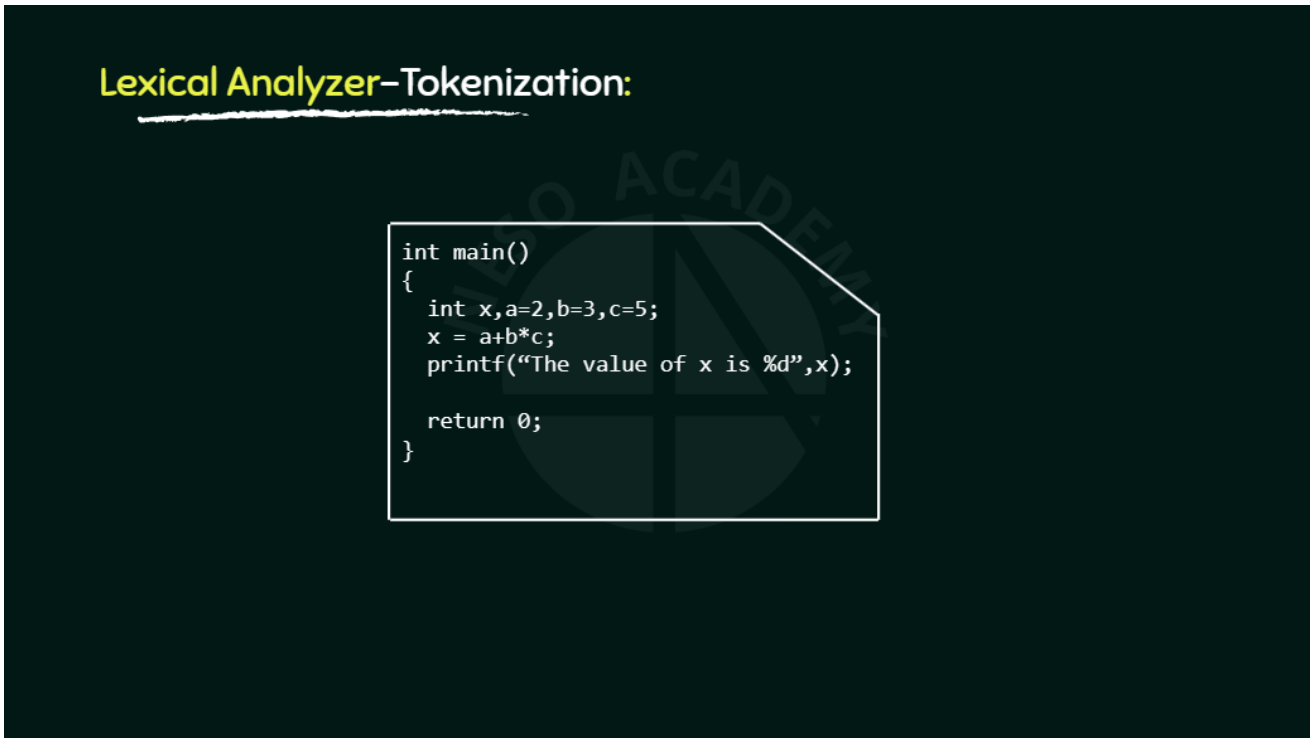
Compiler DesignLexical Analyzer-Tokenization



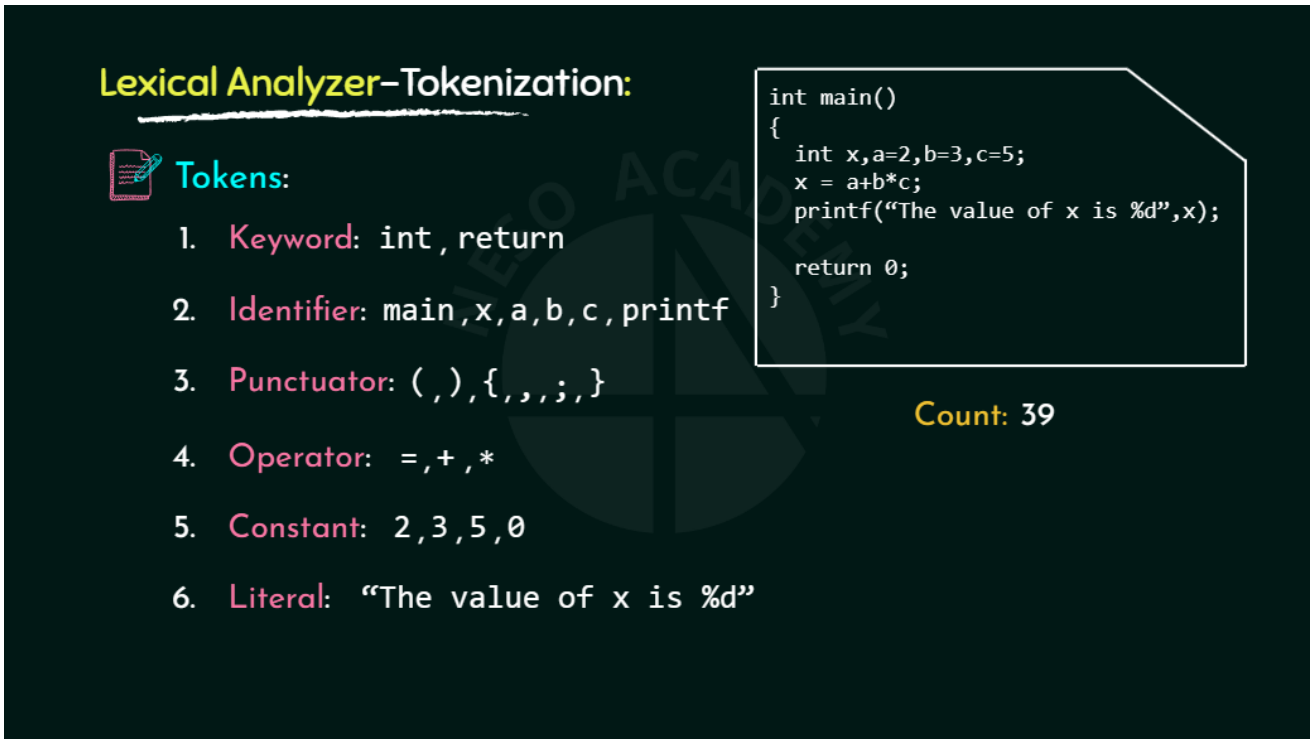
Outcome☆Count the number of tokens in a given code segment



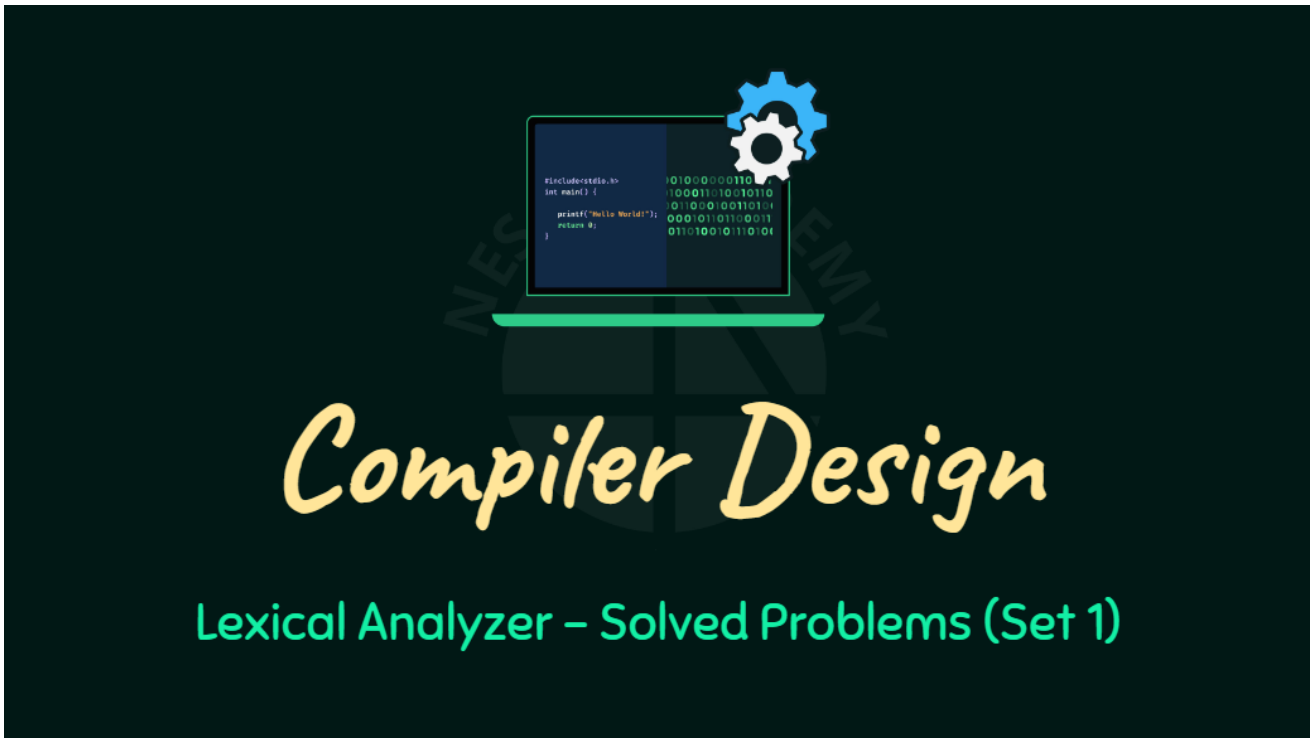
Lexical Analyzer-Tokenization:TokensIdentifierOperatorConstantsKeywordLiteralsPunctuatorsSpecial Characters



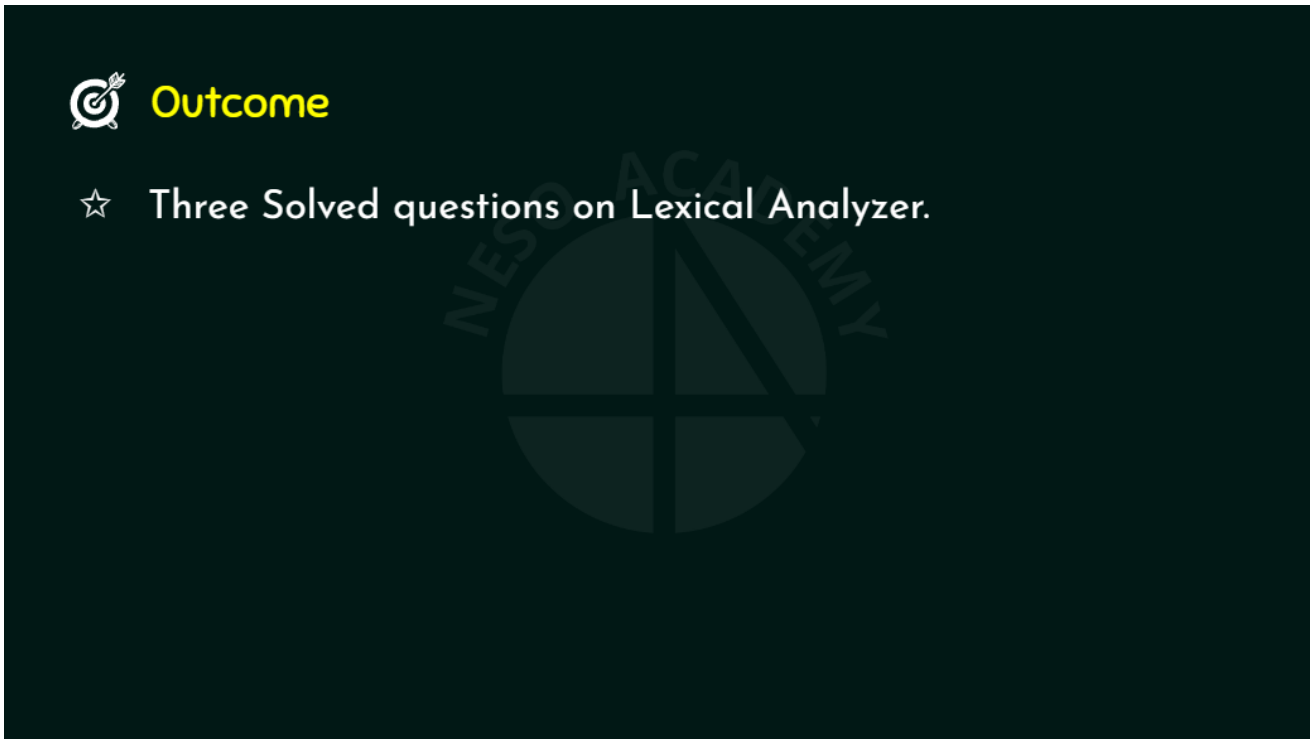
Lexical Analyzer-Tokenization:int main() { int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}



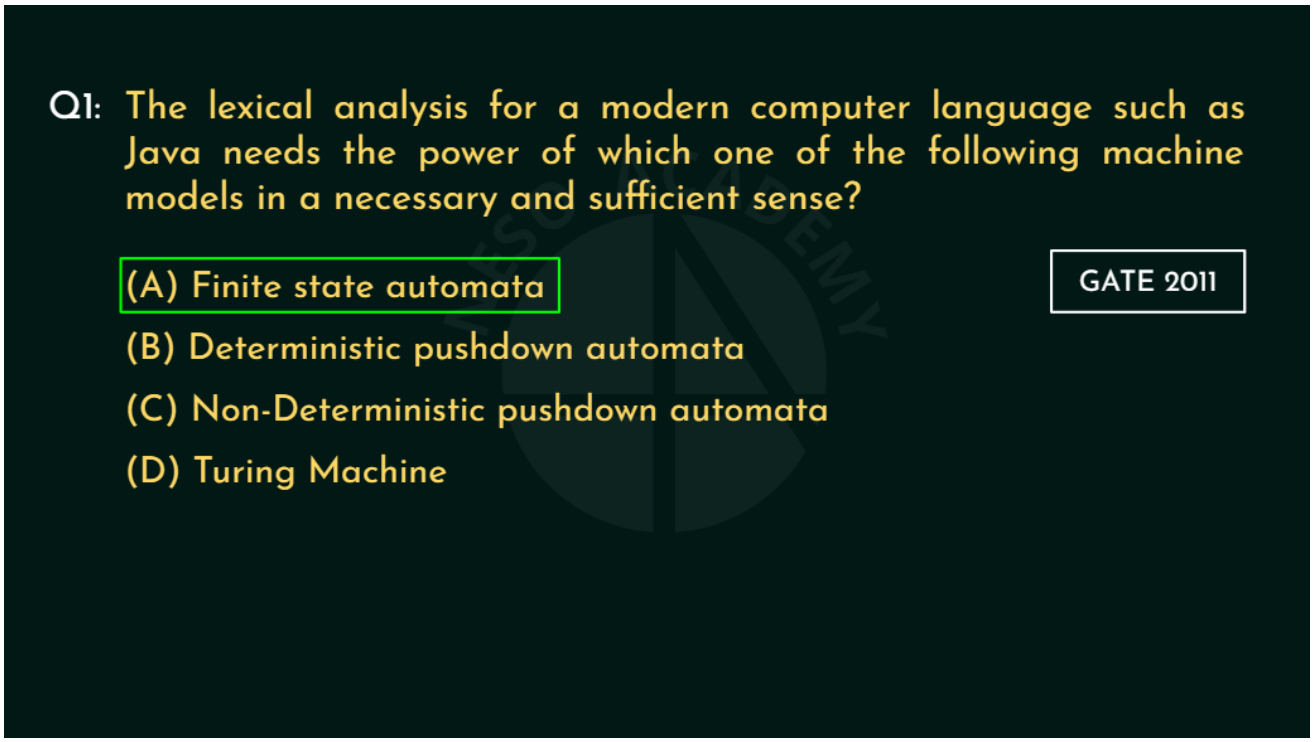
Lexical Analyzer-Tokenization:int main() { int x,a=2,b=3,c=5; x = a+b*c; printf("The value of x is %d",x); return 0;}Tokens: printf,2.Identifier: mainx,a,b,c,4.Operator: =+,*,6.Literal: "The value of x is %d"1.Keyword: intreturn,5.Constant: 23,5,0,3.Punctuator: (),{,,,;,},Count:39



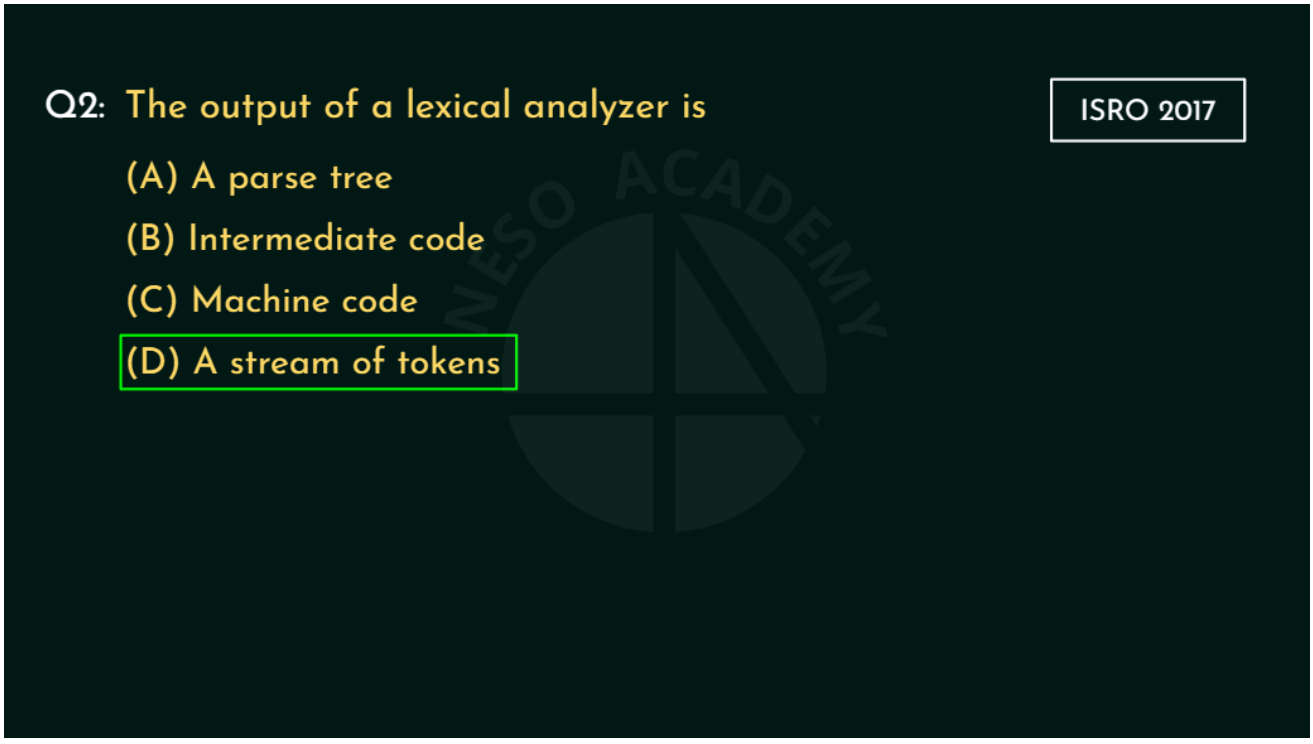
Compiler DesignLexical Analyzer - Solved Problems (Set 1)



Outcome☆Three Solved questions on Lexical Analyzer.



Q1:The lexical analysis for a modern computer language such as Java needs the power of which one of the following machine models in a necessary and sufficient sense?(A) Finite state automata(B) Deterministic pushdown automata(C) Non-Deterministic pushdown automata(D) Turing MachineGATE 2011



Q2:The output of a lexical analyzer is(A) A parse tree(B) Intermediate code(C) Machine code(D) A stream of tokensISRO 2017

Q3: The number of tokens in the following C statement is

```
printf("i=%d, &i=%x",i,&i);
```

GATE 2000

(A) 3

(B) 26

(C) 10

(D) 21

Q3: The number of tokens in the following C statement is

```
printf("i=%d, &i=%x",i,&i);
```

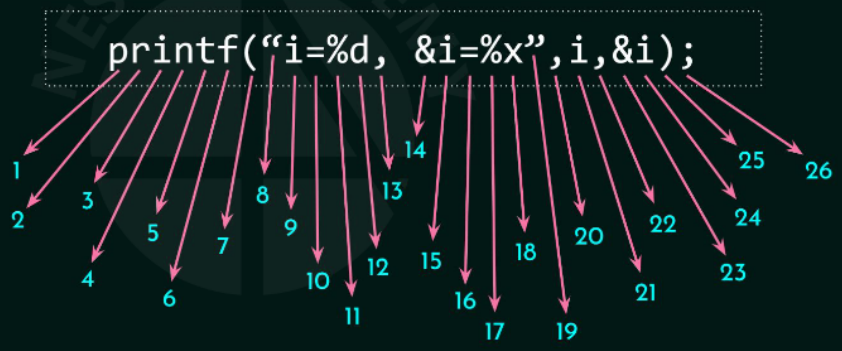
GATE 2000

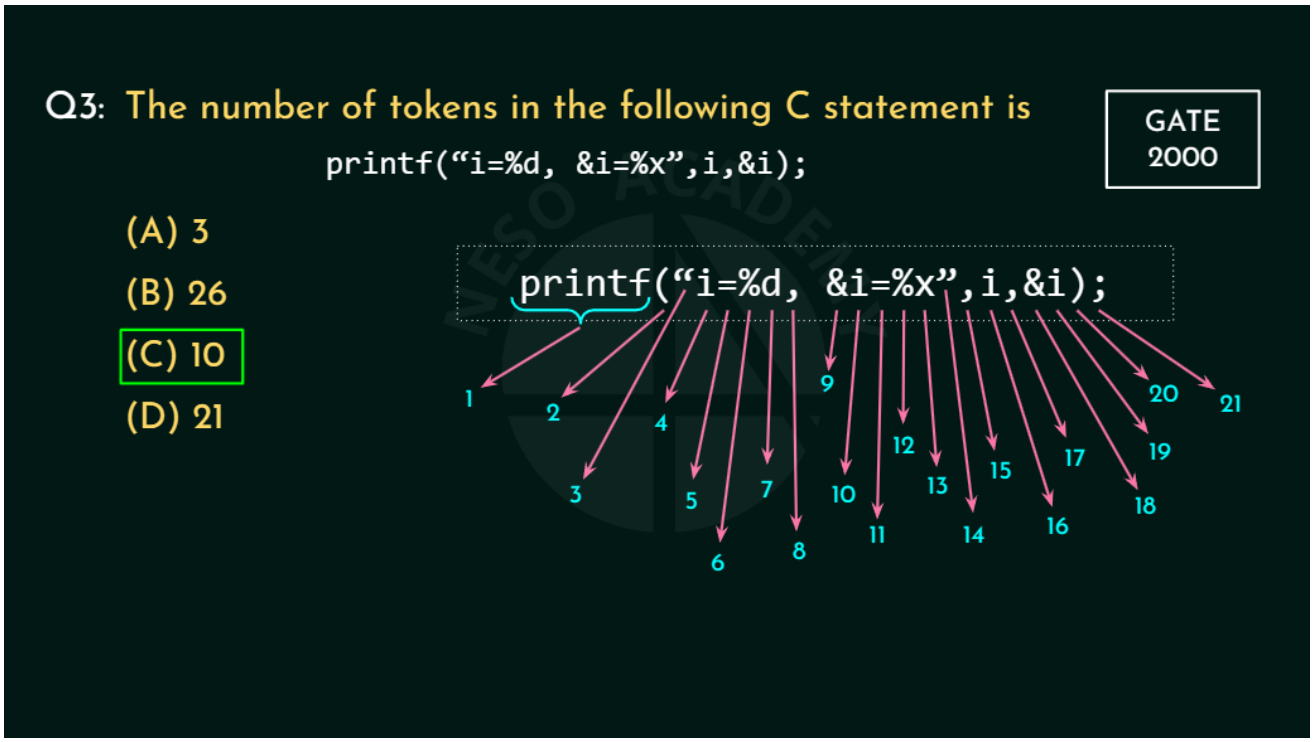
(A) 3

(B) 26

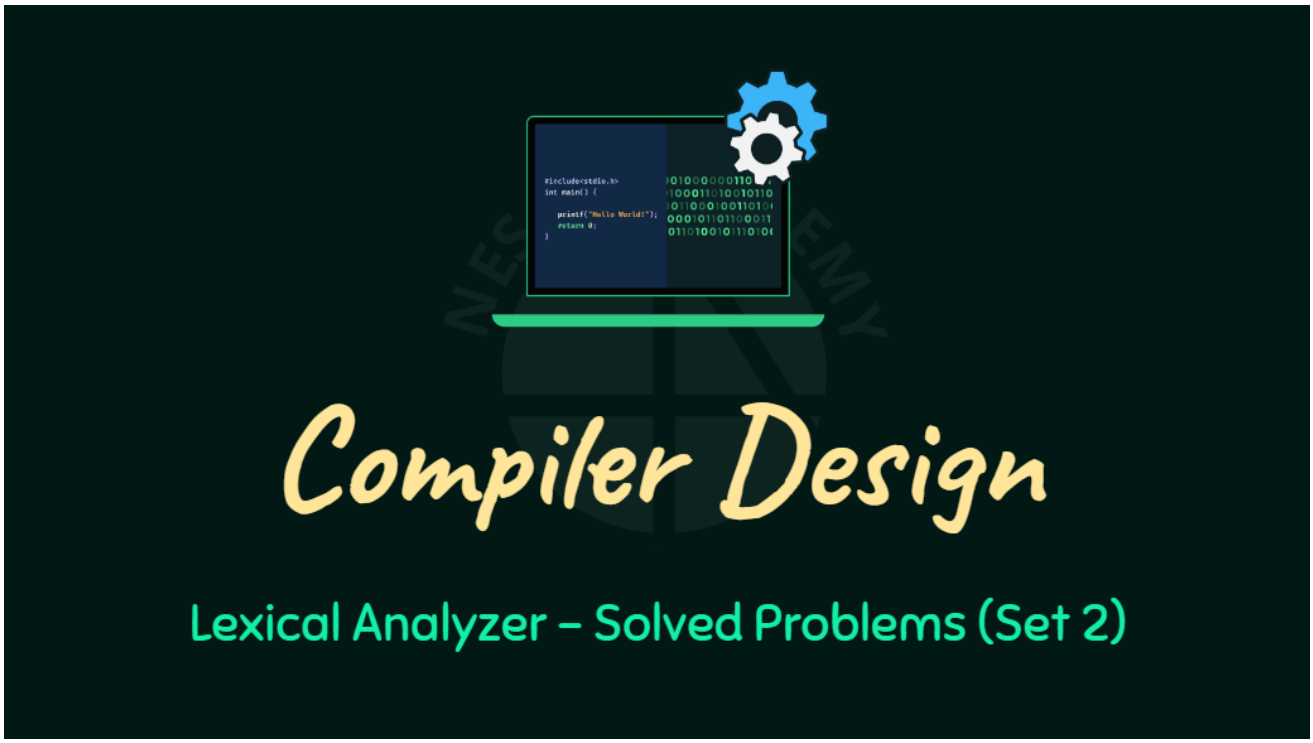
(C) 10

(D) 21

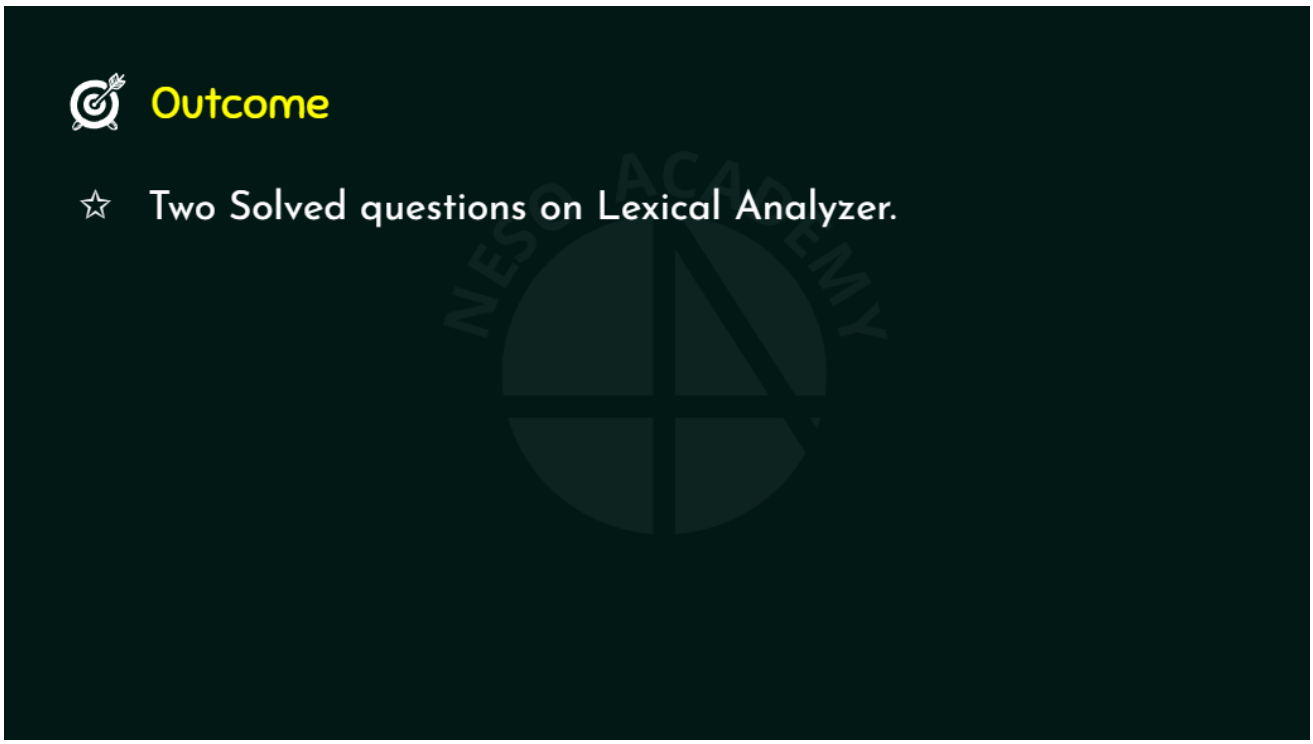




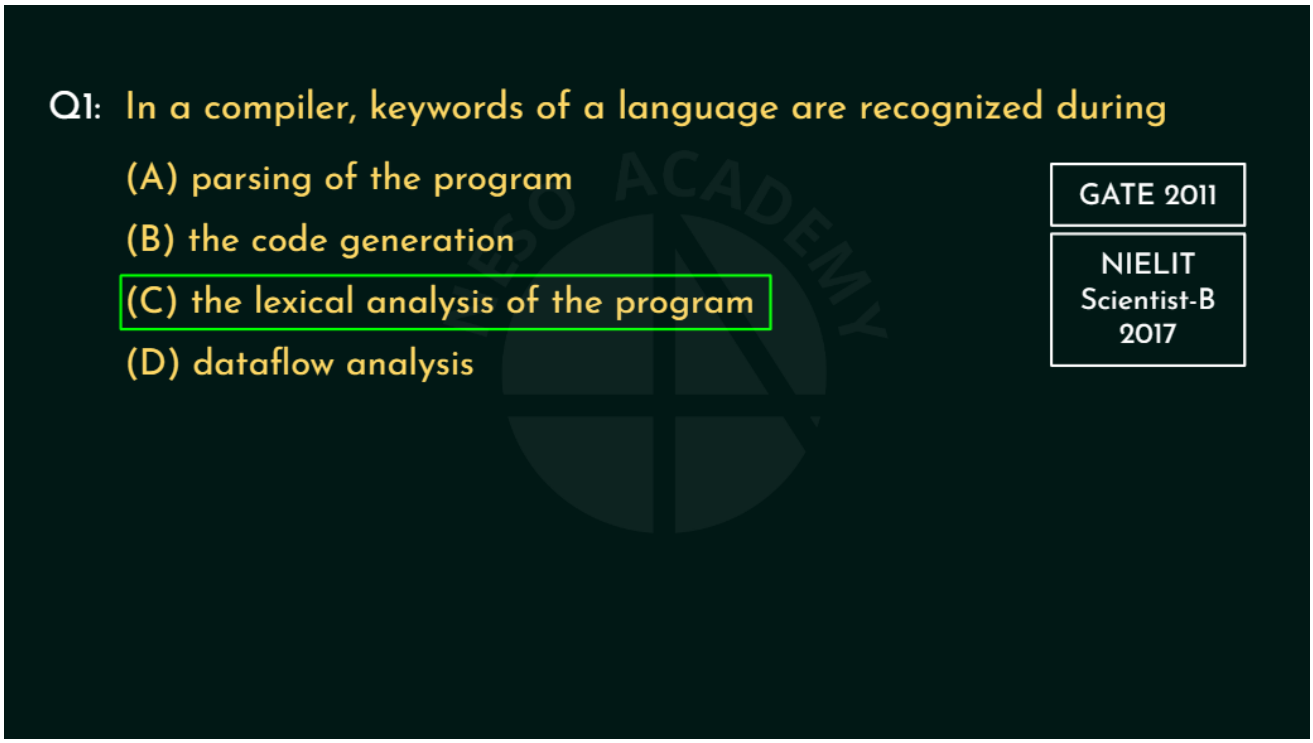
The number of tokens in the following C statement is(A) 3(B) 26(C) 10(D) 21GATE 2000 printf("i=%d, &i=%x",i,&i);Q3:printf("i=%d, &i=%x",i,&i);123456710111213141516171819892021



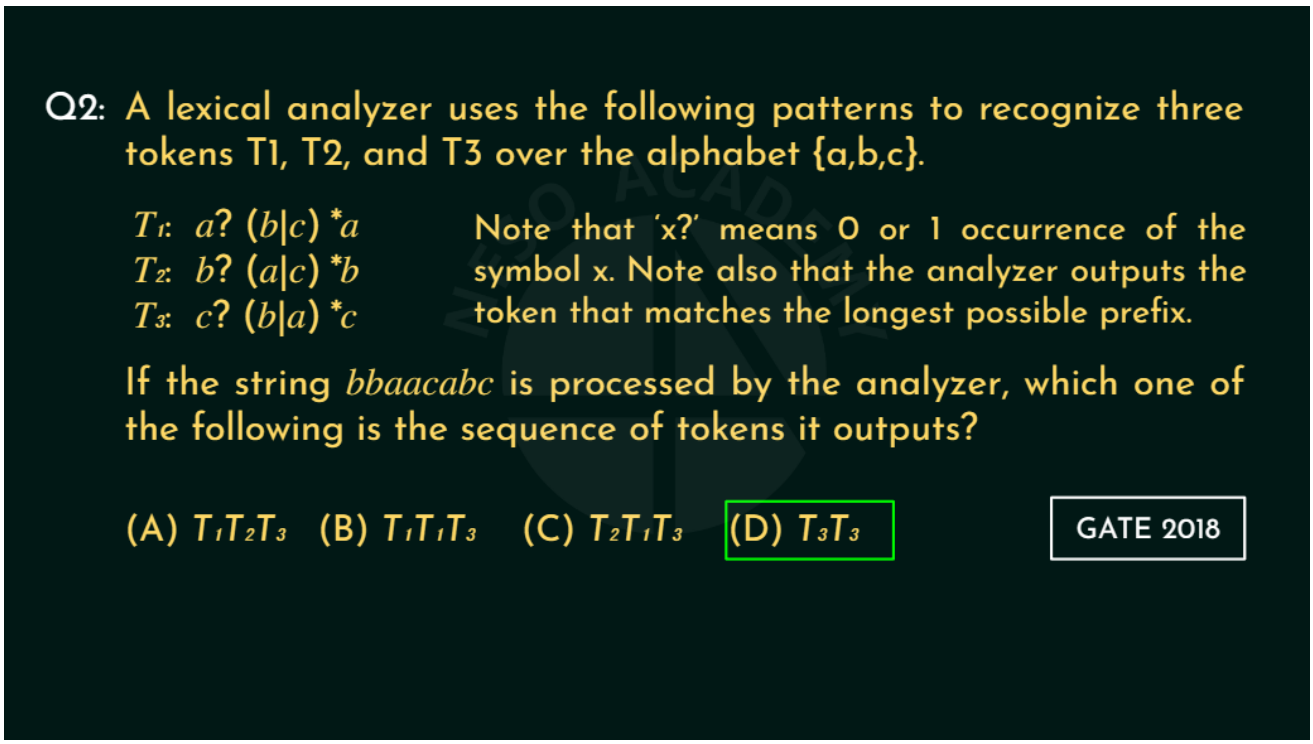
Compiler DesignLexical Analyzer - Solved Problems (Set 2)



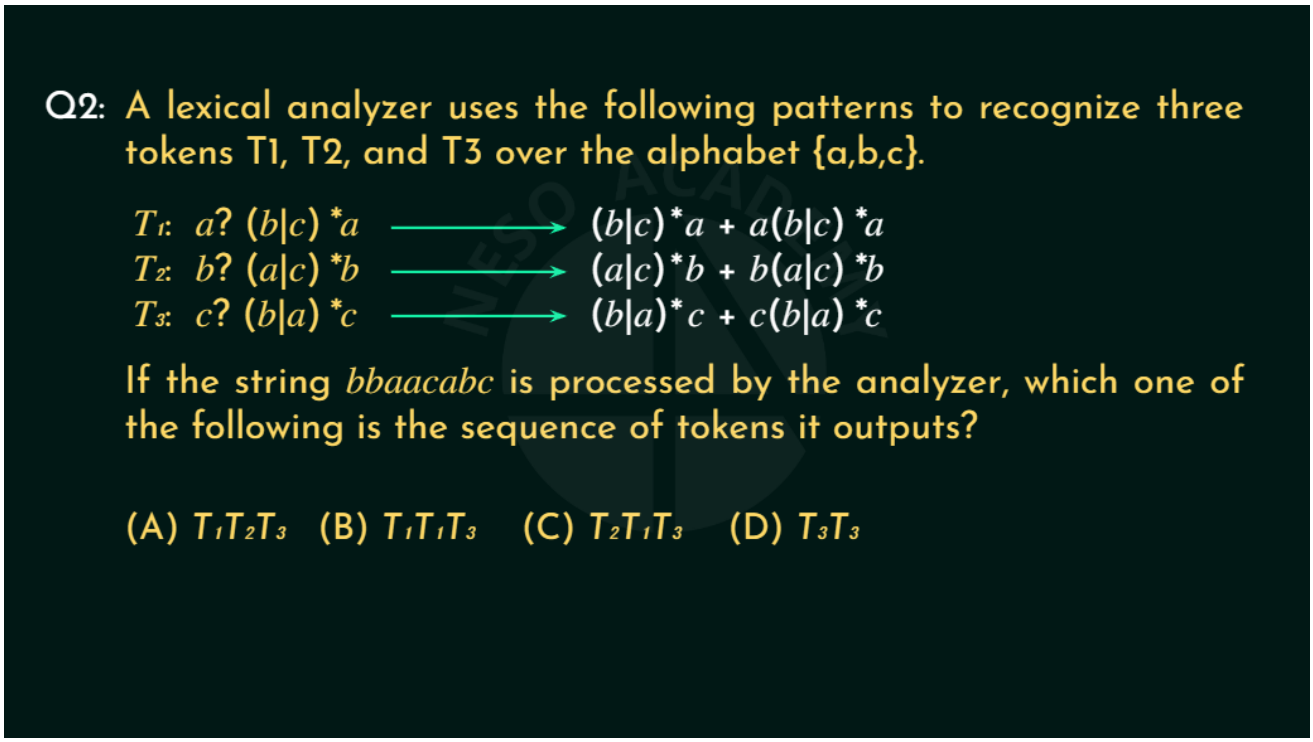
Outcome☆Two Solved questions on Lexical Analyzer.



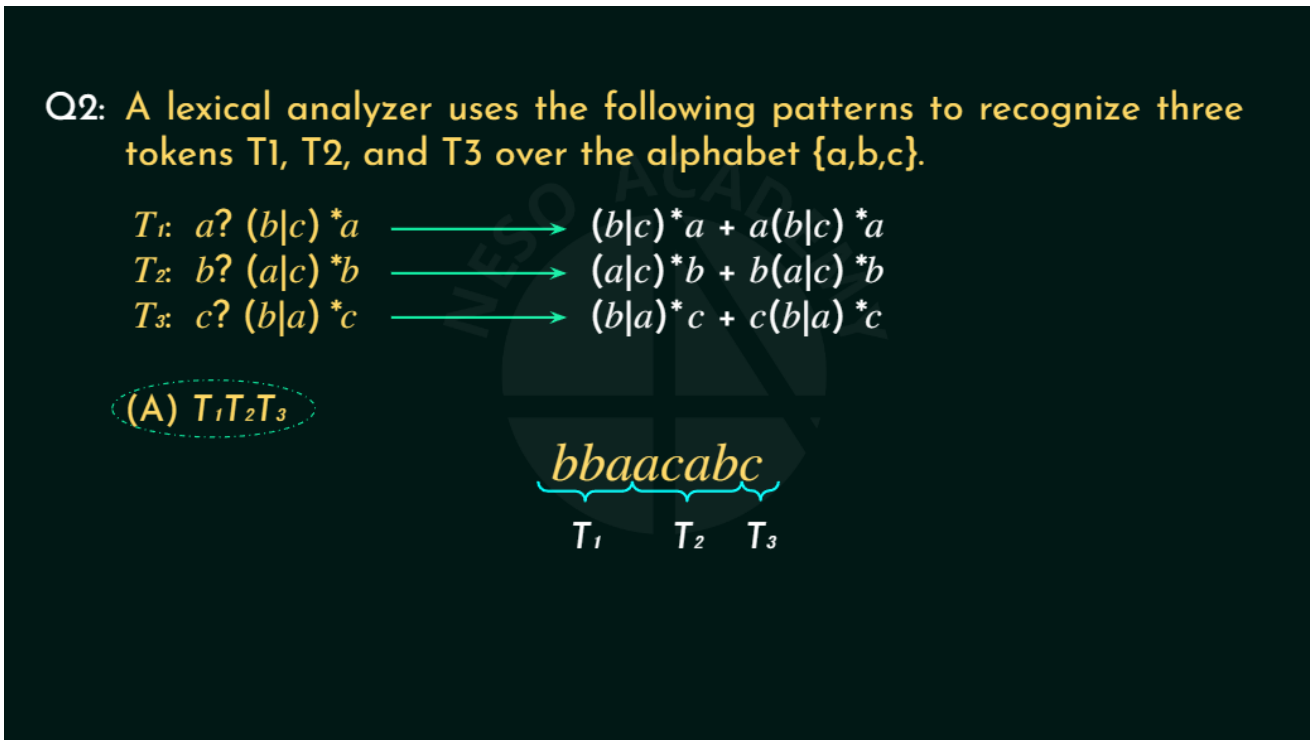
Q1:In a compiler, keywords of a language are recognized during(A) parsing of the program(B) the code generation(C) the lexical analysis of the program(D) dataflow analysisGATE 2011 NIELIT Scientist-B2017



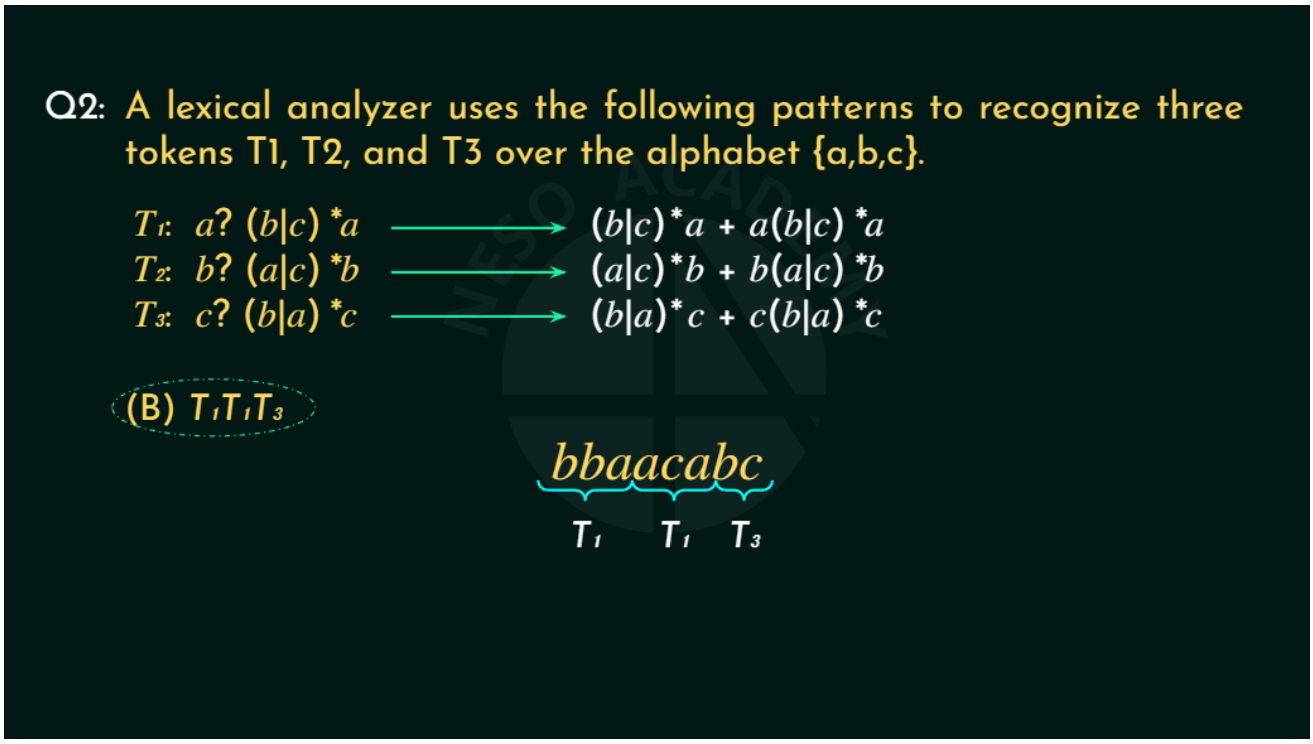
Q2:A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.(A) $T_1T_2T_3$ (B) $T_1T_1T_3$ (C) $T_2T_1T_3$ (D) T_3T_3GATE 2018 T_1: $a?\ (b|c)*a$$T_2$: $b?\ (a|c)*b$$T_3$: $c?\ (b|a)*c$***Note that 'x?' means 0 or 1 occurrence of the symbol x. Note also that the analyzer outputs the token that matches the longest possible prefix.If the string $bbaacabc$ is processed by the analyzer, which one of the following is the sequence of tokens it outputs?



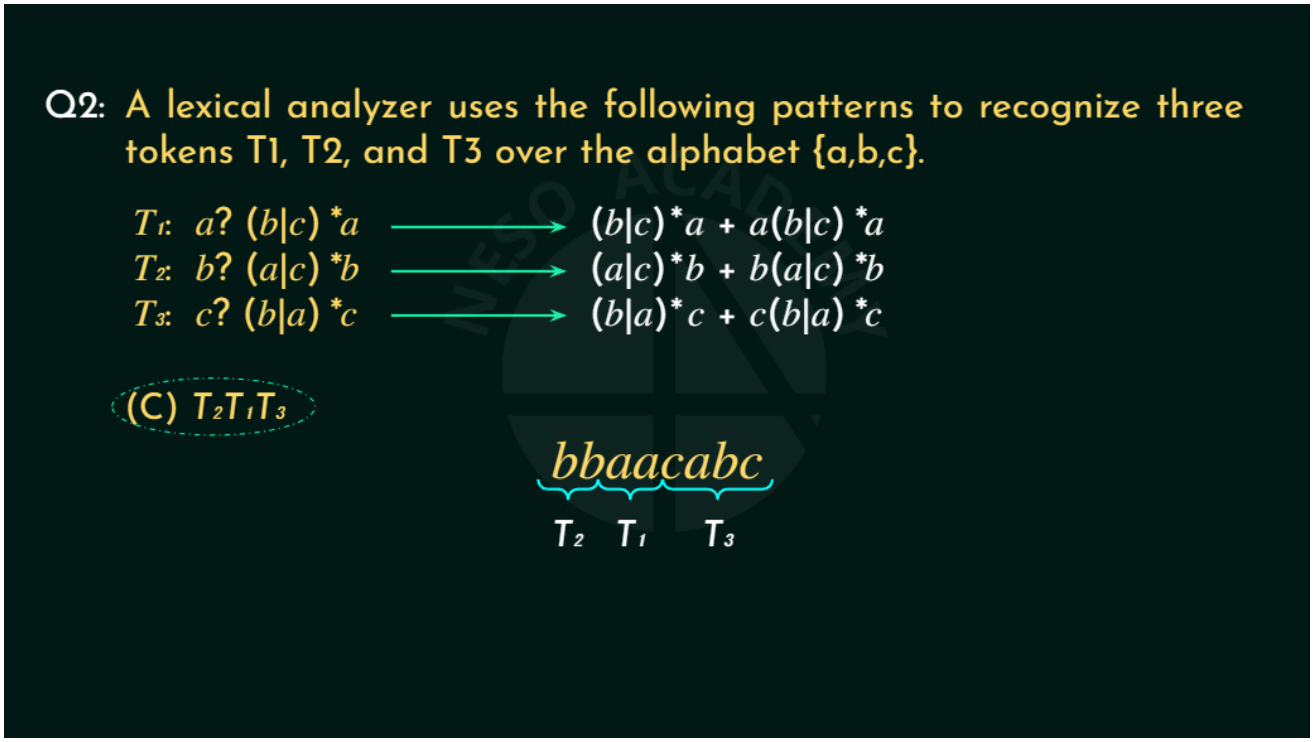
Q2:A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.(A) $T_1T_2T_3$ (B) $T_1T_1T_3$ (C) $T_2T_1T_3$ (D) $T_3T_3T_1$: $a?\ (b|c)*a$$T_2$: $b?\ (a|c)*b$$T_3$: $c?\ (b|a)*c$***If the string $bbaacabc$ is processed by the analyzer, which one of the following is the sequence of tokens it outputs?$(b|c)*a + a(b|c)*a(a|c)*b + b(a|c)*b\ (b|a)*c + c(b|a)*c$******



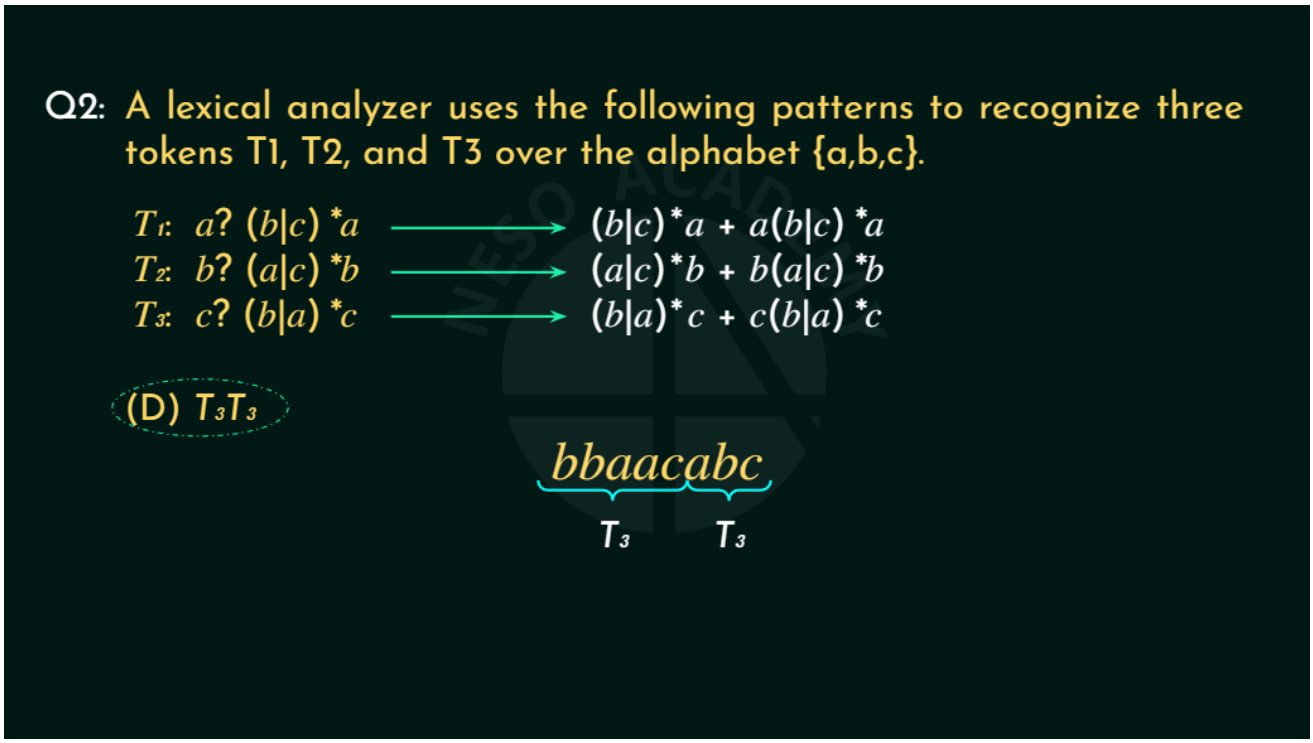
Q2:A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.(A) $T_1T_2T_3$ T_1: $a?\ (b|c)*a$ T_2: $b?\ (a|c)*b$ T_3: $c?\ (b|a)*c$ *** bbaacabc $T_1T_2T_3$ $(b|c)*a + a(b|c)*a(a|c)*b + b(a|c)*b\ (b|a)*c + c(b|a)*c$ ******



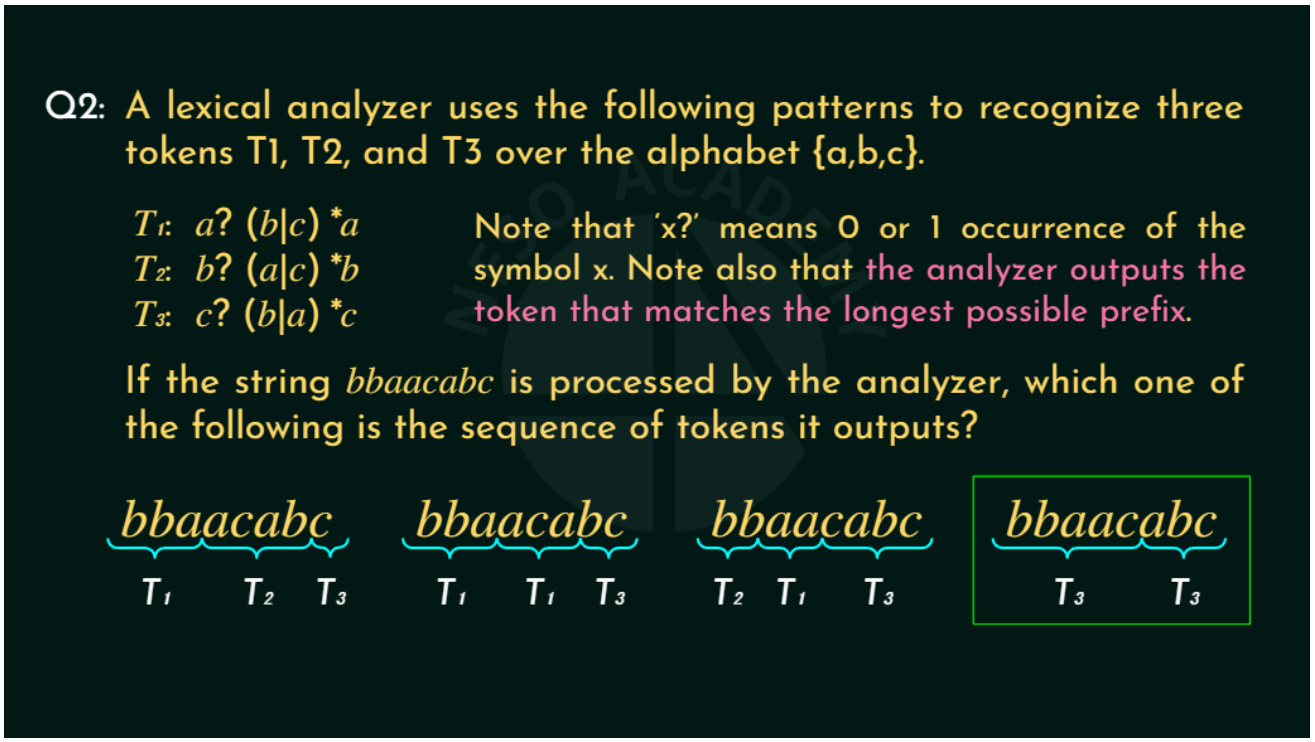
Q2:A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.(B) $T_1T_1T_3$ T_1: $a?\ (b|c)*a$ T_2: $b?\ (a|c)*b$ T_3: $c?\ (b|a)*c$ *** bbaacabc T_1T_3 $(b|c)*a + a(b|c)*a(a|c)*b + b(a|c)*b\ (b|a)*c + c(b|a)*c$ ****** T_1



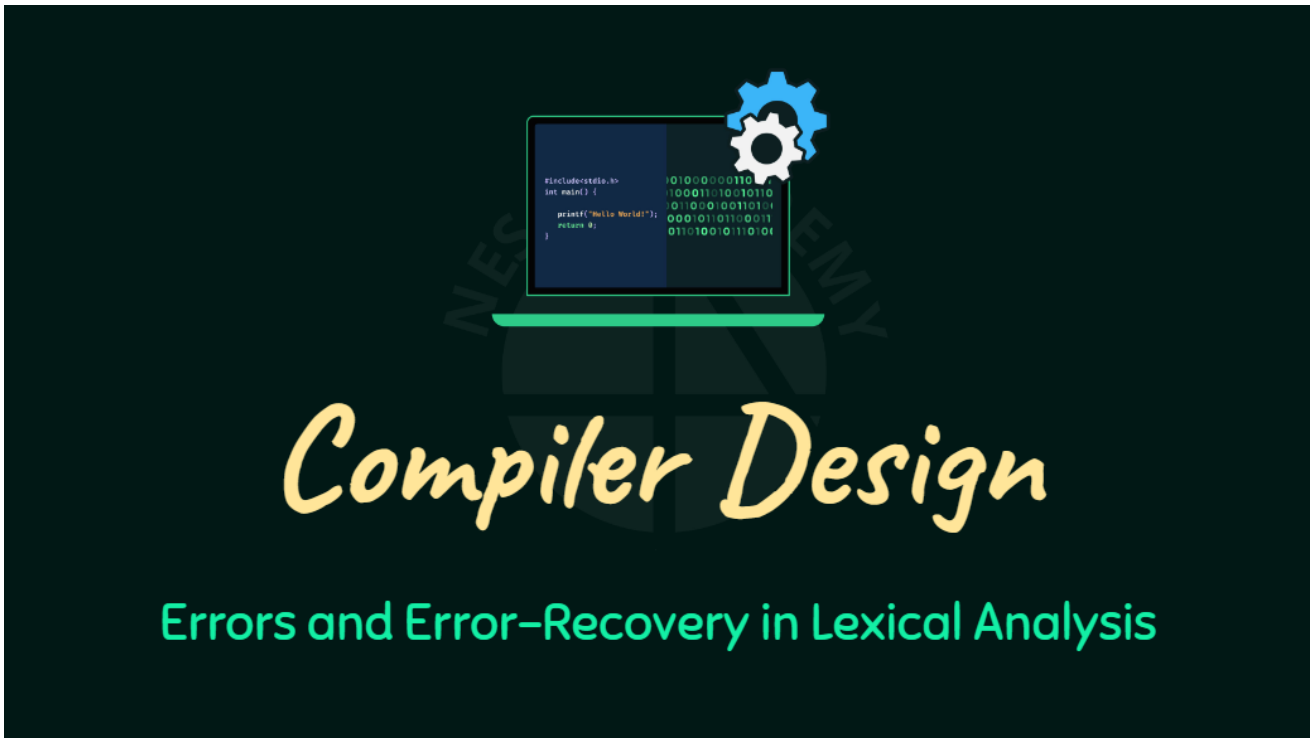
Q2:A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.(C) $T_2T_1T_3$$T_1$: a? $(b|c)*a$$T_2$: b? $(a|c)*b$$T_3$: c? $(b|a)*c$***bbaacabc$(b|c)*a$ + $a(b|c)*a(a|c)*b$ + $b(a|c)*b$ $(b|a)*c$ + $c(b|a)*c$******$T_2T_3T_1$



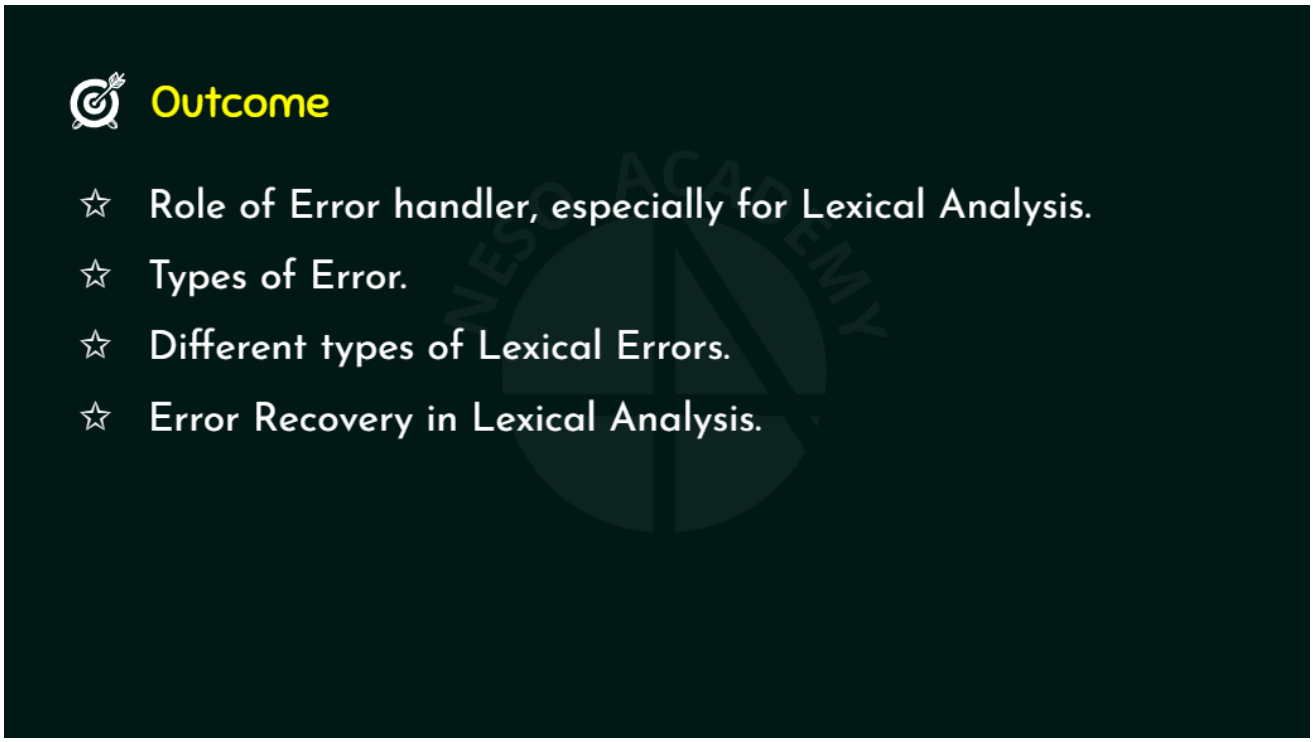
Q2:A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.(D) $T_3T_3T_1$: a? $(b|c)*a$$T_2$: b? $(a|c)*b$$T_3$: c? $(b|a)*c$***bbaacabc$T_3(b|c)*a$ + $a(b|c)*a(a|c)*b$ + $b(a|c)*b$ $(b|a)*c$ + $c(b|a)*c$******T_3



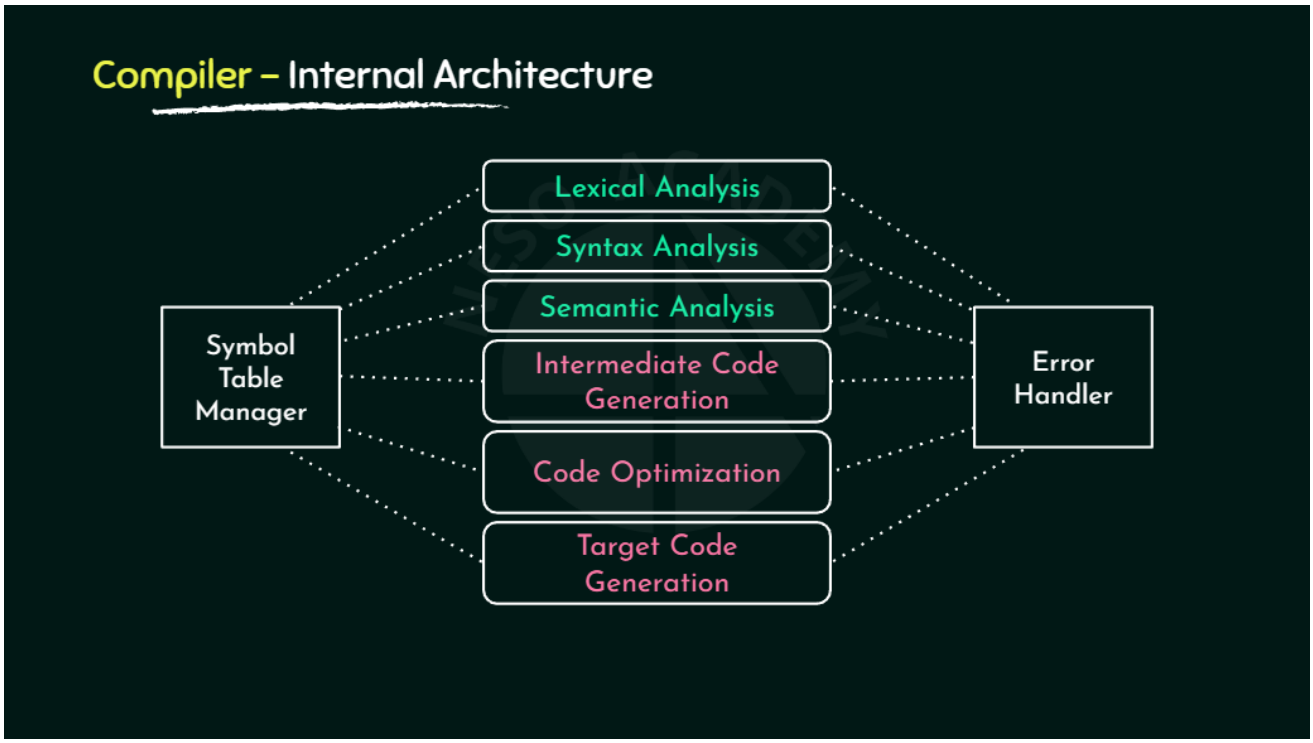
Q2:A lexical analyzer uses the following patterns to recognize three tokens T1, T2, and T3 over the alphabet {a,b,c}.T_1: a? $(b|c)*a$$T_2$: b? $(a|c)*b$$T_3$: c? $(b|a)*c$***Note that 'x?' means 0 or 1 occurrence of the symbol x. Note also that the analyzer outputs the token that matches the longest possible prefix.If the string $bbaacabc$ is processed by the analyzer, which one of the following is the sequence of tokens it outputs?$bbaacabc$$T_3T_3$$bbaacabc$$T_1T_3T_1$$bbaacabc$$T_1T_2T_3$$bbaacabc$$T_2T_3T_1$



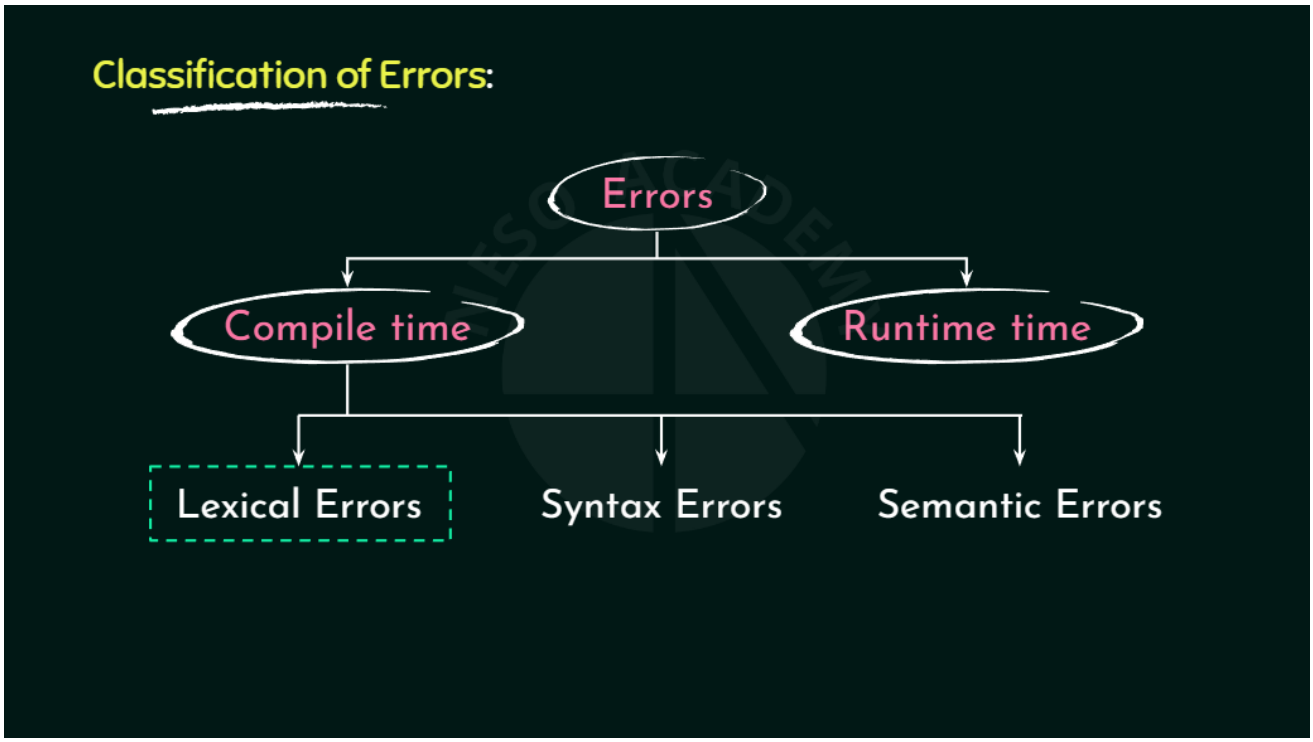
Compiler DesignErrors and Error-Recovery in Lexical Analysis



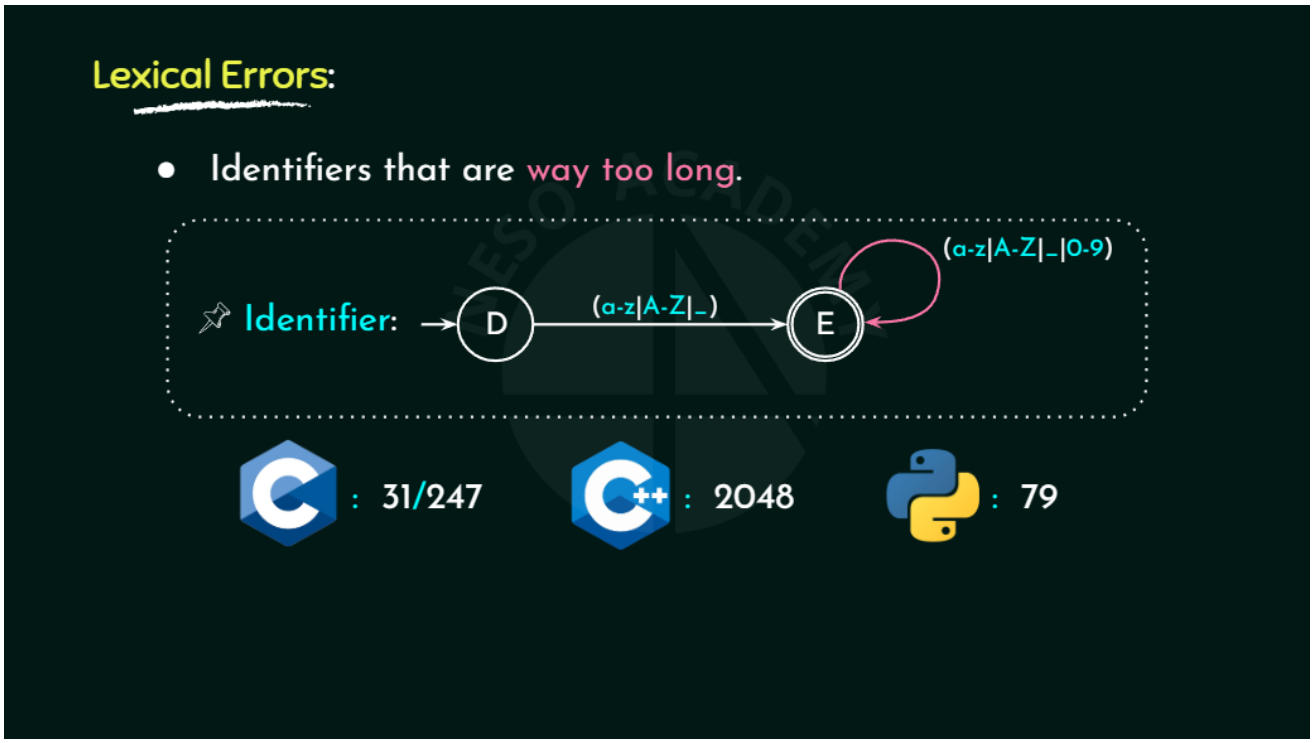
Outcome☆Role of Error handler, especially for Lexical Analysis.☆Types of Error.☆Different types of Lexical Errors.☆Error Recovery in Lexical Analysis.



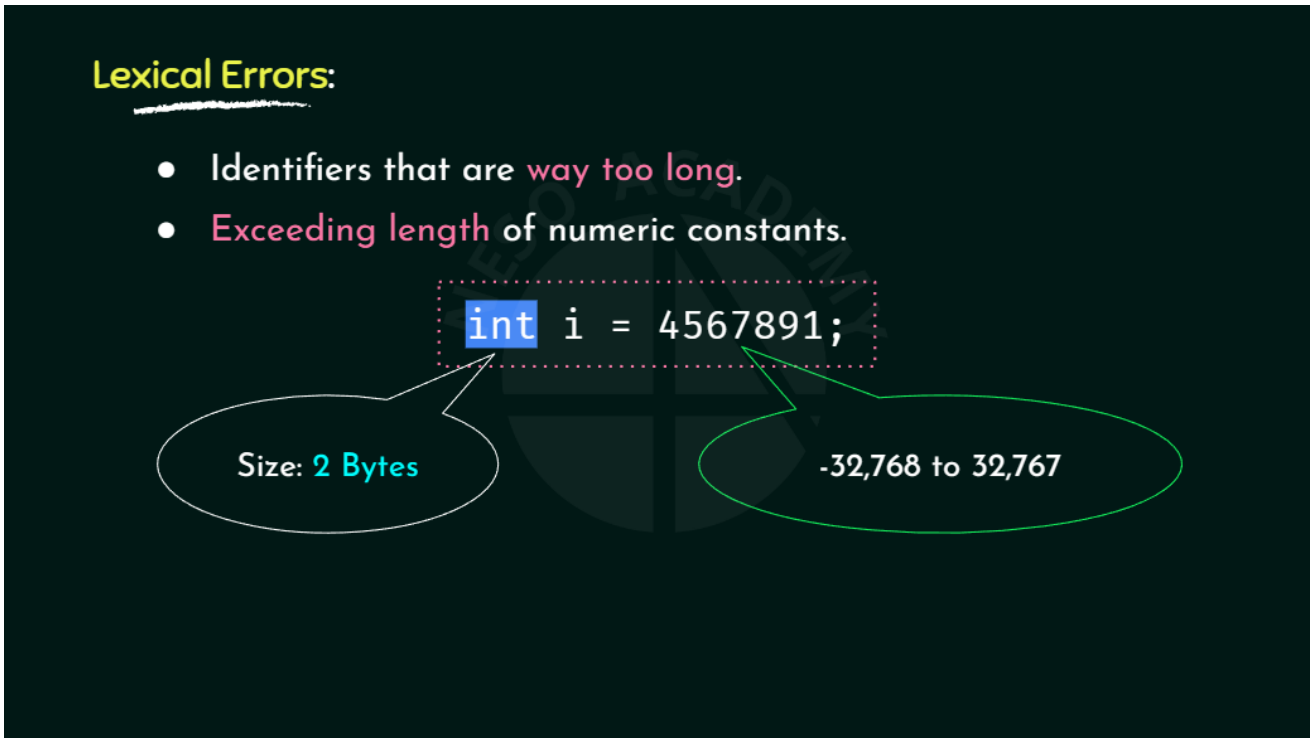
Compiler - Internal ArchitectureLexical AnalysisSyntax AnalysisSemantic AnalysisIntermediate Code GenerationCode OptimizationTarget Code GenerationSymbol Table ManagerError Handler



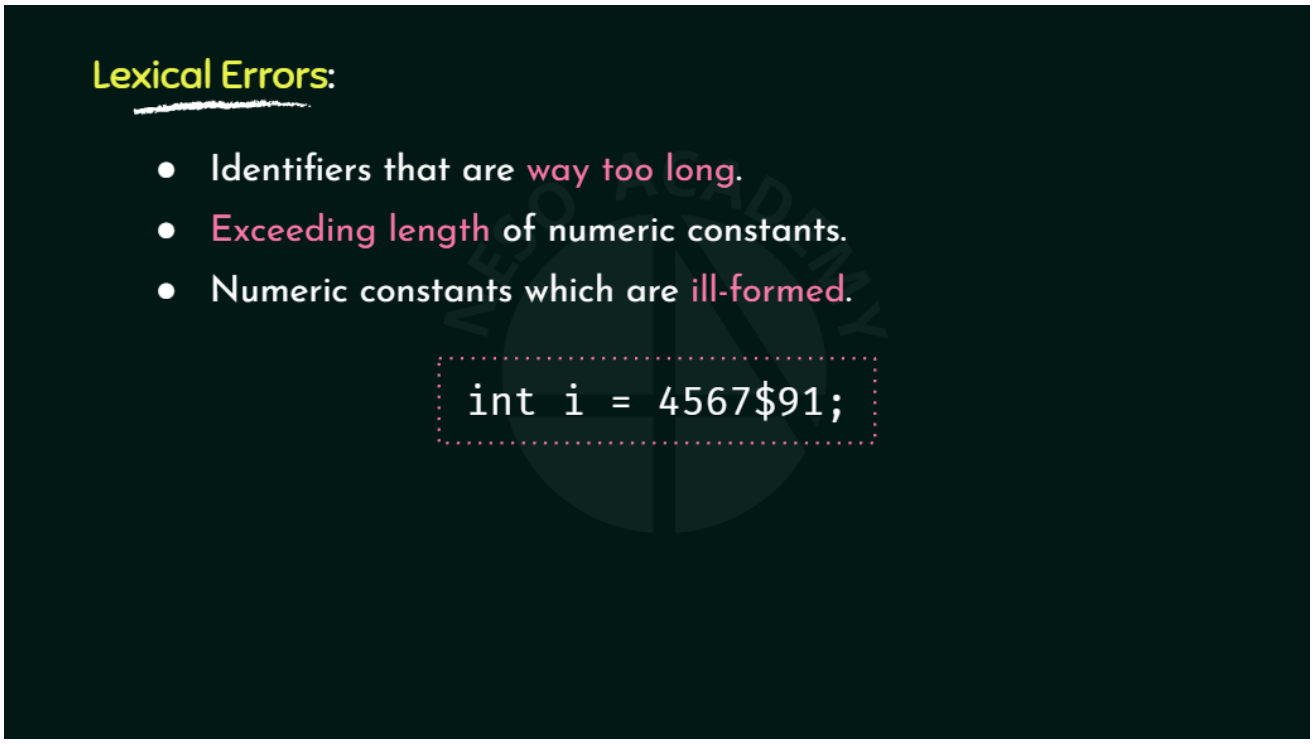
Classification of Errors:ErrorsCompile timeRuntime timeLexical ErrorsSyntax ErrorsSemantic Errors



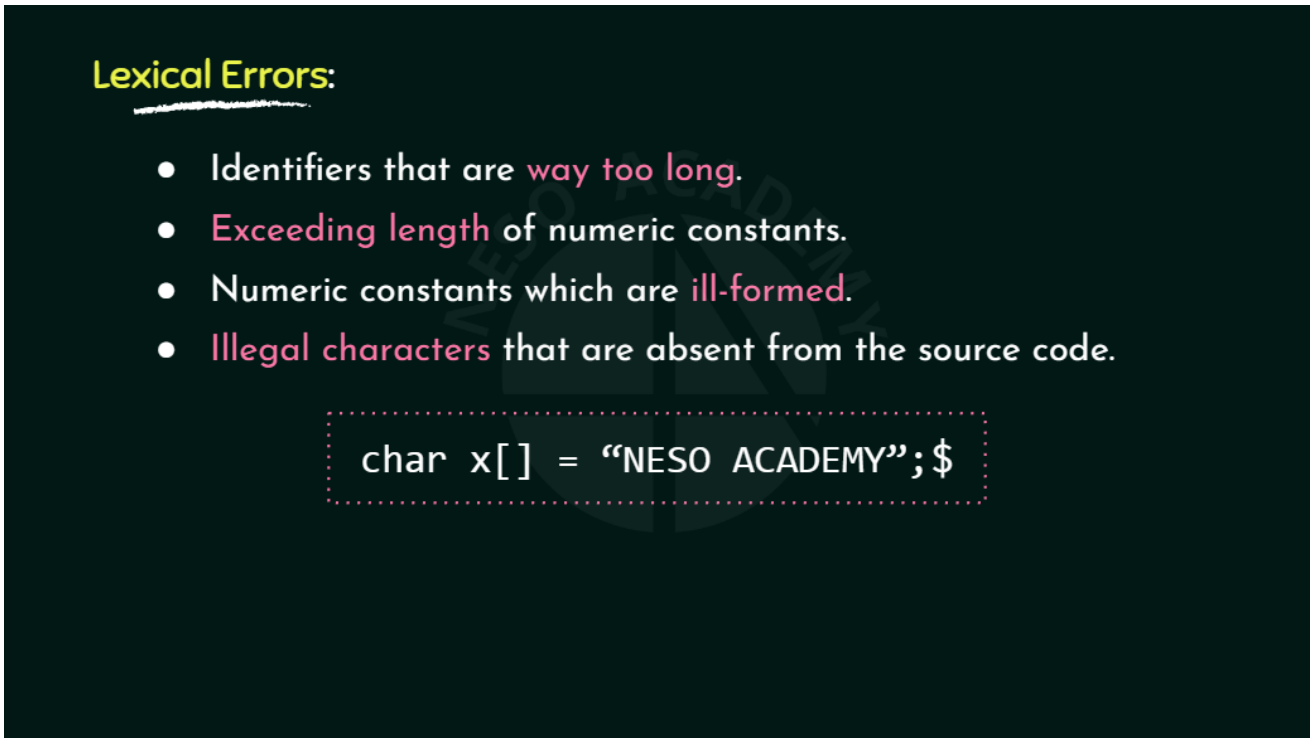
Lexical Errors:●Identifiers that are way too long. D(a-z|A-Z|_)E(a-z|A-Z|_|0-9)Identifier:: 31/247: 2048: 79



Lexical Errors:●Identifiers that are way too long. ●Exceeding length of numeric constants. int i = 4567891;Size: 2 Bytes-32,768 to 32,767



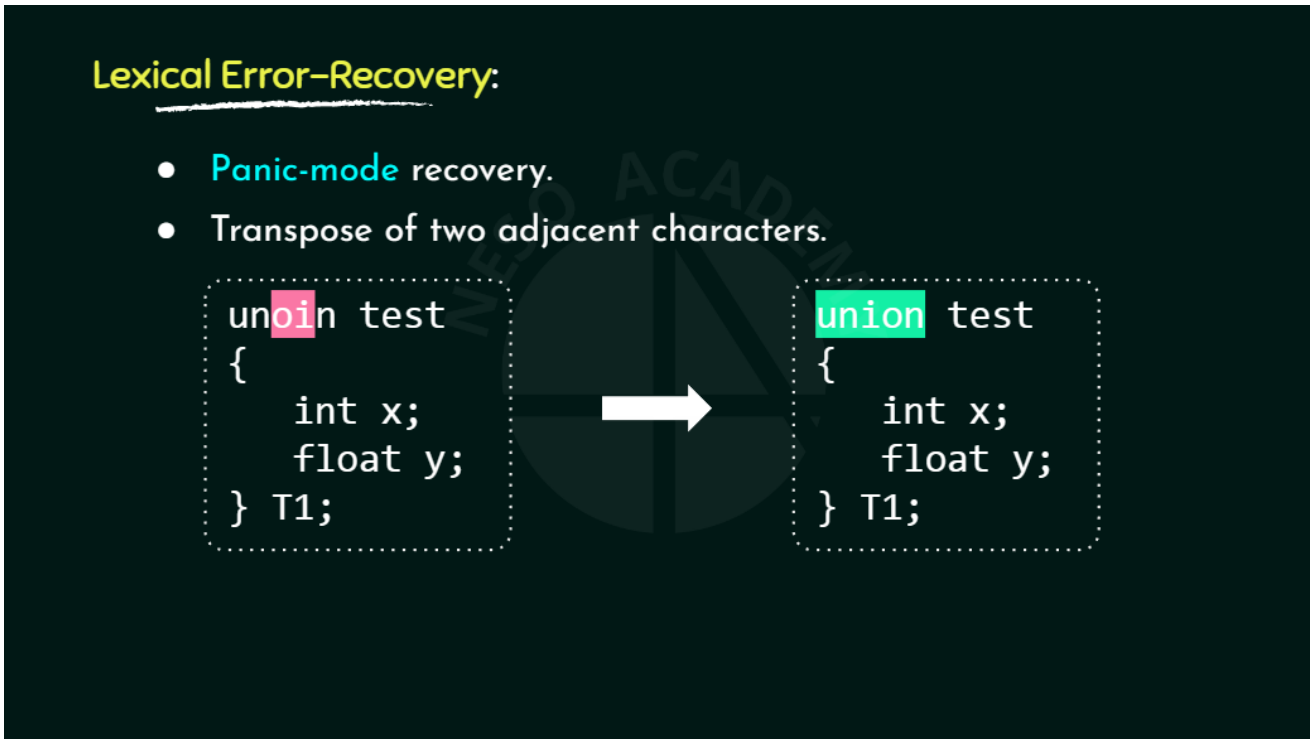
Lexical Errors:●Identifiers that are way too long. ●Exceeding length of numeric constants. ●Numeric constants which are ill-formed. int i = 4567$91;



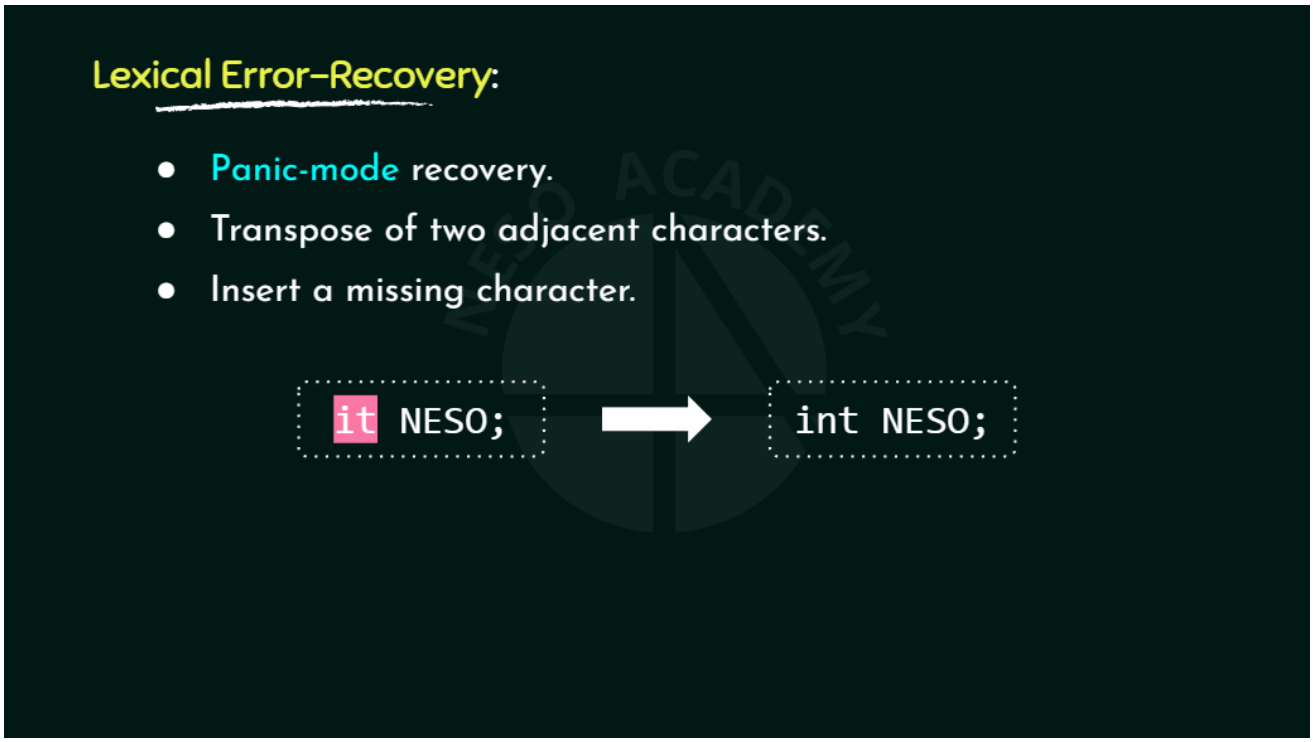
Lexical Errors:●Identifiers that are way too long. ●Exceeding length of numeric constants. ●Numeric constants which are ill-formed. ●Illegal characters that are absent from the source code. char x[] = "NESO ACADEMY";$

int 4NESO;Lexical Error-Recovery:●Panic-mode recovery.

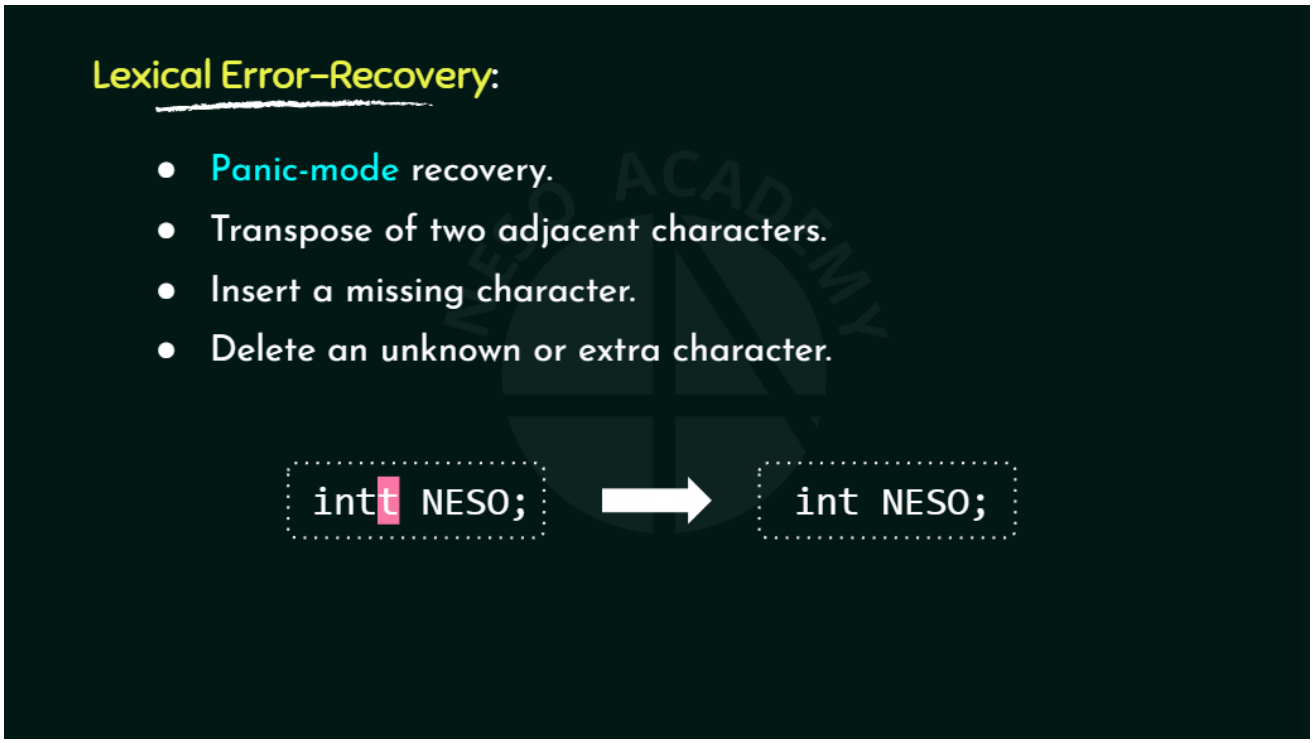
Lexical Error-Recovery:●Panic-mode recovery. while(condition){ _____ _____ _____}



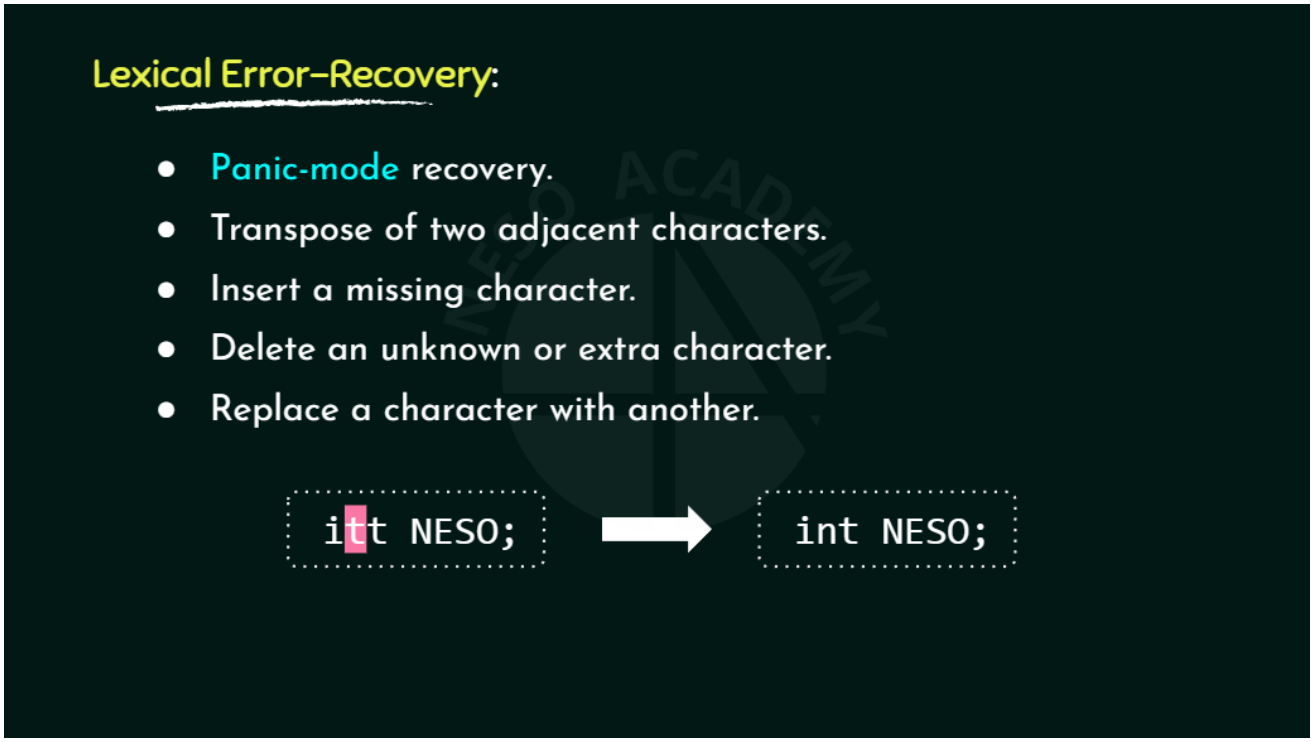
Lexical Error-Recovery:●Panic-mode recovery. ●Transpose of two adjacent characters. unoin test{int x;float y;} T1;union test{int x;float y;} T1;



Lexical Error-Recovery:●Panic-mode recovery. ●Transpose of two adjacent characters. ●Insert a missing character. it NESO;int NESO;



Lexical Error-Recovery:●Panic-mode recovery. ●Transpose of two adjacent characters. ●Insert a missing character. ●Delete an unknown or extra character. intt NESO;int NESO;



Lexical Error-Recovery:●Panic-mode recovery. ●Transpose of two adjacent characters. ●Insert a missing character. ●Delete an unknown or extra character. itt NESO;int NESO;●Replace a character with another.