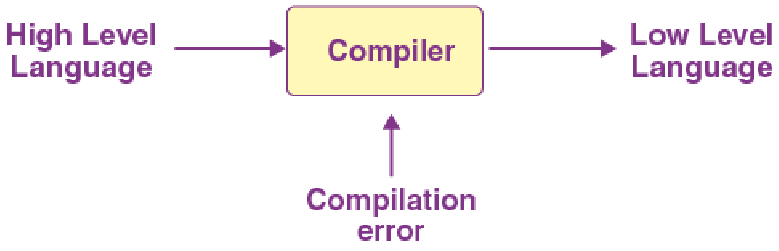
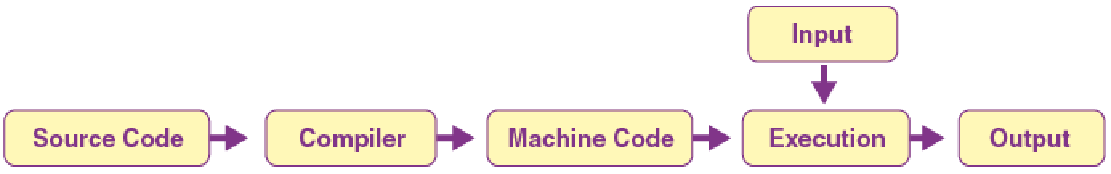
Compiler

What is a Compiler?

A compiler is a computer program that decodes computer code composed in one programming language into another language. Or we can say that the compiler helps in translating the source code composed in a high-level programming language into the machine code. You can also dive deep into the difference between Compiler and Interpreter to get more familiar with this topic.

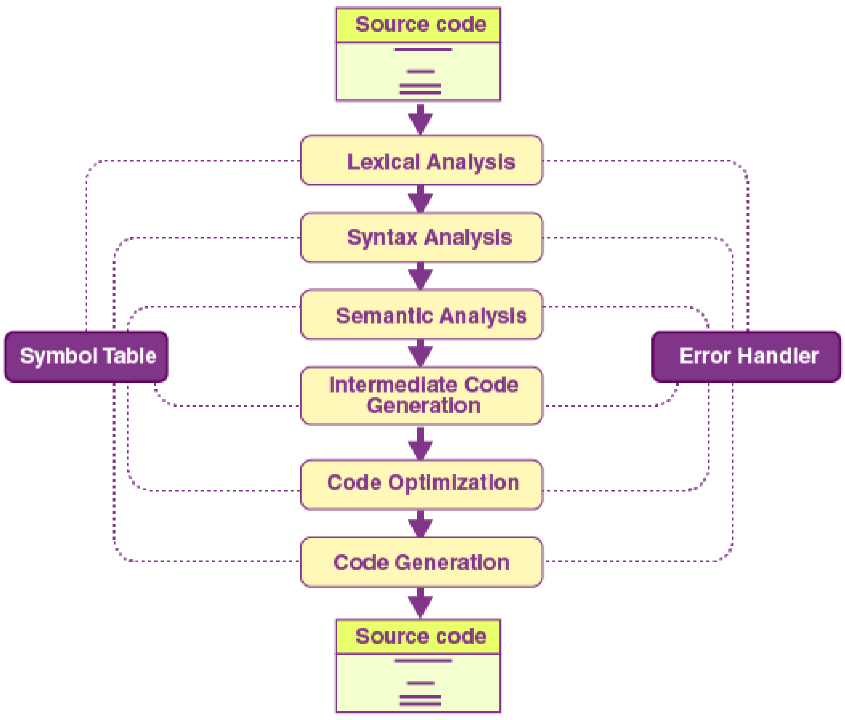




Explore the Phases of Compiler

The 6 phases of a compiler are:

1. Lexical Analysis
2. Syntactic Analysis or Parsing
3. Semantic Analysis
4. Intermediate Code Generation
5. Code Optimization
6. Code Generation

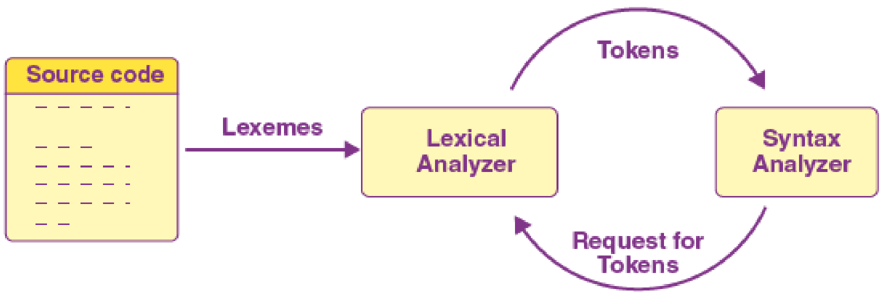


1.**Lexical Analysis:** Lexical analysis or Lexical analyzer is the initial stage or phase of the compiler. This phase scans the source code and transforms the input program into a series of a token.

A token is basically the arrangement of characters that defines a unit of information in the source code.

NOTE: In computer science, a program that executes the process of lexical analysis is called a scanner, tokenizer, or lexer.

You can gain in-depth knowledge of lexical analysis to get a better understanding.

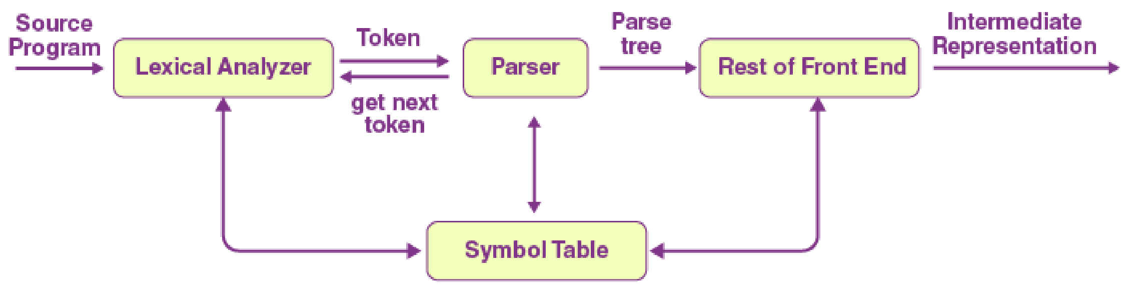


Roles and Responsibilities of Lexical Analyzer

- It is accountable for terminating the comments and white spaces from the source program.
- It helps in identifying the tokens.
- Categorization of lexical units.

2. **Syntax Analysis:**

In the compilation procedure, the Syntax analysis is the second stage. Here the provided input string is scanned for the validation of the structure of the standard grammar. Basically, in the second phase, it analyses the syntactical structure and inspects if the given input is correct or not in terms of programming syntax.



It accepts tokens as input and provides a parse tree as output. It is also known as parsing in a compiler.

Roles and Responsibilities of Syntax Analyzer

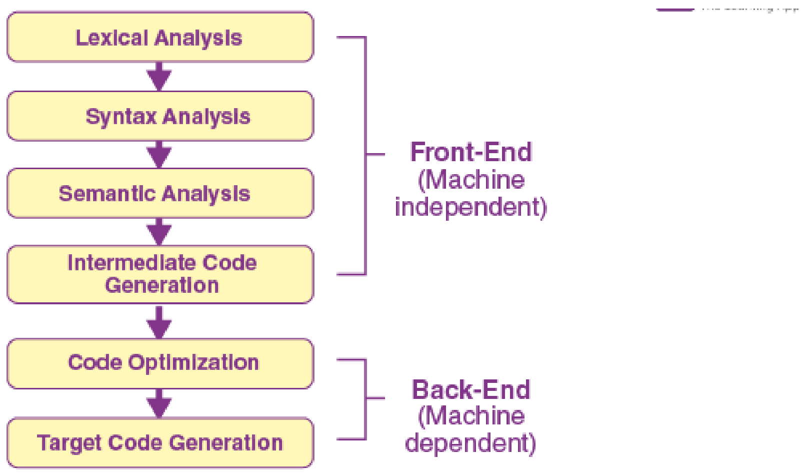
- Note syntax errors.
- Helps in building a parse tree.
- Acquire tokens from the lexical analyzer.
- Scan the syntax errors, if any.

3. **Semantic Analysis:** In the process of compilation, semantic analysis is the third phase. It scans whether the parse tree follows the guidelines of language. It also helps in keeping track of identifiers and expressions. In simple words, we can say that a semantic analyzer defines the validity of the parse tree, and the annotated syntax tree comes as an output.

Roles and Responsibilities of Semantic Analyzer:

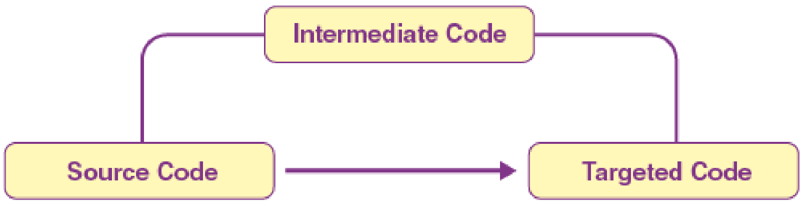
- Saving collected data to symbol tables or syntax trees.
- It notifies semantic errors.
- Scanning for semantic errors.

4. **Intermediate Code Generation:** The parse tree is semantically confirmed; now, an intermediate code generator develops three address codes. A middle-level language code generated by a compiler at the time of the translation of a source program into the object code is known as intermediate code or text.


Lexical Analysis
Syntax Analysis
Semantic Analysis
Intermediate Code Generation
Code Optimization
Target Code Generation
Front-End (Machine independent)
Back-End (Machine dependent)

Few Important Pointers:

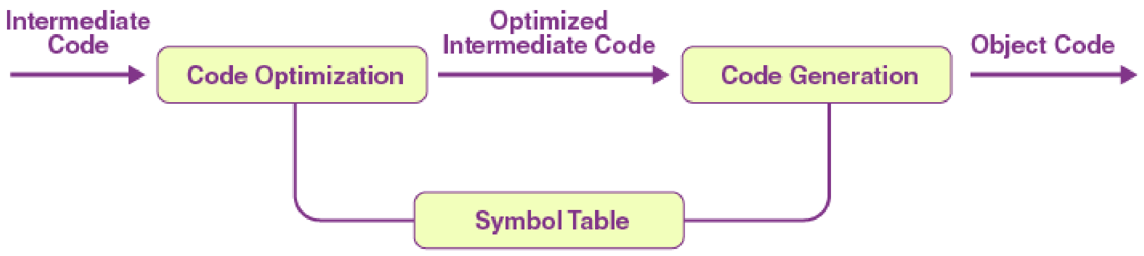
- A code that is neither high-level nor machine code, but a middle-level code is an intermediate code.
- We can translate this code to machine code later.
- This stage serves as a bridge or way from analysis to synthesis.



Roles and Responsibilities:

- Helps in maintaining the priority ordering of the source language.
- Translate the intermediate code into the machine code.
- Having operands of instructions.

5. **Code optimizer:** Now coming to a phase that is totally optional, and it is code optimization. It is used to enhance the intermediate code. This way, the output of the program is able to run fast and consume less space. To improve the speed of the program, it eliminates the unnecessary strings of the code and organises the sequence of statements.



Roles and Responsibilities:

- Remove the unused variables and unreachable code.
- Enhance runtime and execution of the program.
- Produce streamlined code from the intermediate expression.

6. **Code Generator:** The final stage of the compilation process is the code generation process. In this final phase, it tries to acquire the intermediate code as input which is fully optimised and map it to the machine code or language. Later, the code generator helps in translating the intermediate code into the machine code.

Roles and Responsibilities:

- Translate the intermediate code to target machine code.
- Select and allocate memory spots and registers.

What is a Symbol Table?

The symbol table is mainly known as the data structure of the compiler. It helps in storing the identifiers with their name and types. It makes it very easy to operate the searching and fetching process.

The symbol table connects or interacts with all phases of the compiler and error handler for updates. It is also accountable for scope management.

It stores:

- It stores the literal constants and strings.
- It helps in storing the function names.
- It also prefers to store variable names and constants.
- It stores labels in source languages.

Practice Problem – Phases of Compiler

Q. Consider the grammar given below:

S → Aa

A → BD

B → b | ∈

D → d | ∈

Let a, b, d, and $ be indexed as follows:

a	**b**	**d**	**$**
3	2	1	0

Compute the FOLLOW set of the nonterminal B and write the index values for the symbols in the FOLLOW set in descending order. (For example, if the FOLLOW set is {a, b, d, $}, then the answer should be 3210)

Q. A lexical analyzer uses the following patterns to recognize three tokens *T*1, *T*2, and *T*3, over the alphabet *{a,b,c}.*

T1: a? (b|c)∗a

T2: b? (a|c)∗b

T3: 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?

Q. Consider the following grammar and the semantic actions to support the inherited type declaration attributes. Let *X*1, X2, X3, *X4*, *X*5, and *X*6 be the placeholders for the nonterminals D, T, L or L1 in the following table:

Production rule	**Semantic action**
D → T L	*X*1.type = *X*2.type
T → int	T.type = int
T → float	T.type = float
L → Ll , id	*X*3.type = *X*4.type addType(id.entry, *X*5.type)
L → id	addType(id.entry, *X*6.type)

Which one of the following is the appropriate choices for *X*1, *X*2, *X*3 and *X*4?

(A) X1=L,X2=T,X3=L1,X4=L

(B) X1=T,X2=L,X3=L1,X4=T

(C) X1=L,X2=L,X3=L1,X4=T

(D) X1=T,X2=L,X3=T,X4=L1

Q.Consider the augmented grammar given below:

S' → S

S → ⟨L⟩ | id

L → L,S | S

Let I0 = CLOSURE ({[S' → •S]}). The number of items in the set GOTO (I0 ,

⟨

⟨) is: ________.

Q.Consider the following grammar:

stmt → if expr then expr else expr; stmt | 0

expr → term relop term | term

term → id |number

id → a | b | c

number → [0-9]

Where relop is a rational operator (e.g.,<,>,…), O refers to the empty statement, and if, then, else are terminals.

Consider a program P following the above grammar containing ten if terminals. The number of control flow paths in P is ________, in the program:

if e1 and e2 else e3

has 2 control flow paths, e1→e2 and e1→e3.

Frequently Asked Questions Related to Phase of Compiler

Q1

Why do we use parsing in compilers?

The parser in the compilation process is utilised to split the data into smaller components reaching from the lexical analysis phase (first phase). It takes input in the form of a series of tokens and creates output as the parse tree.

Q2

What is the output of Lexical analysis?

The Lexical analysis creates a stream of tokens as output.