PHASES OF COMPILER

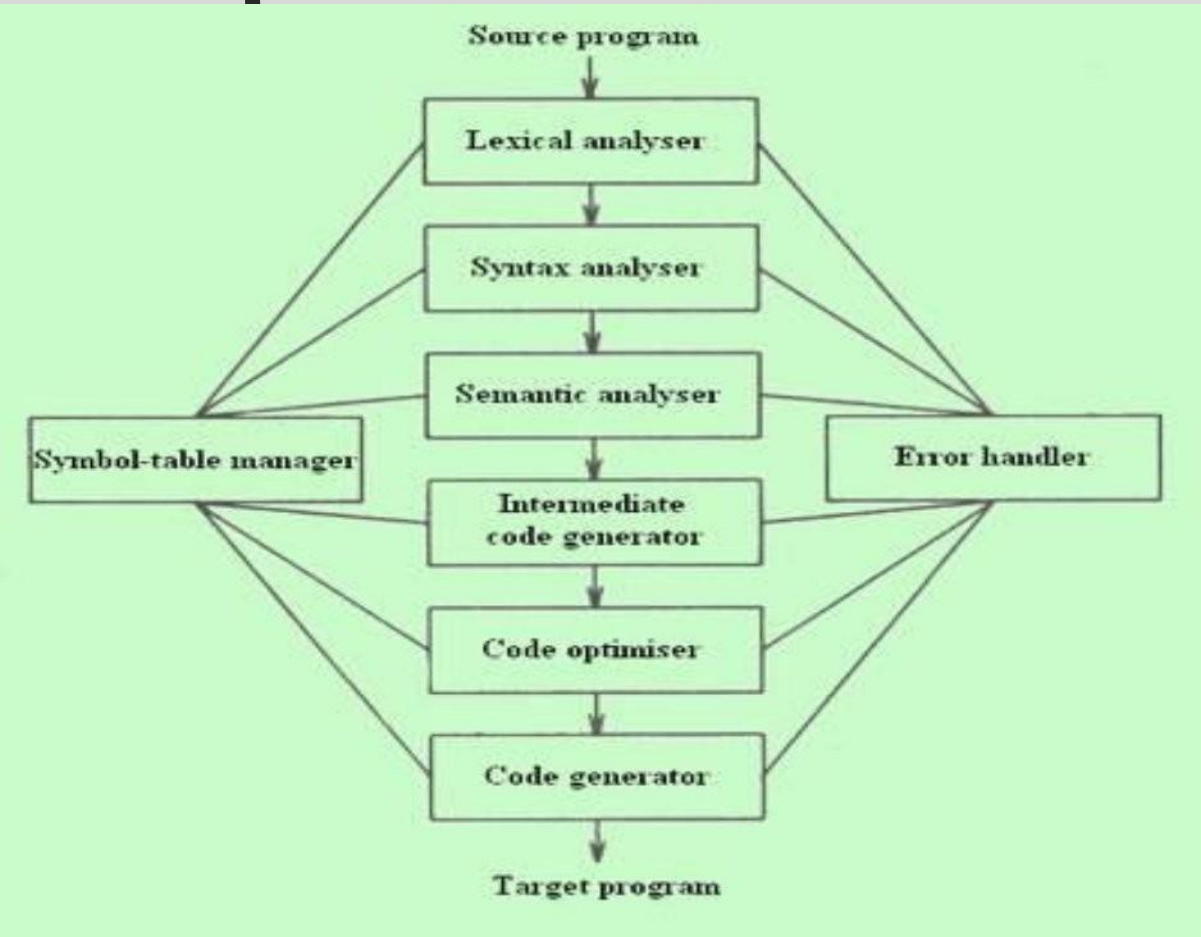
Ms.Nisha K S,CSE

STRUCTURE OF THE COMPILER DESIGN

- ***Phases of a compiler:***
- A compiler operates in phases.
- A phase is a logically interrelated operation that takes source program in one representation and produces output in another representation.

- There are two phases of compilation.
  a. Analysis (Machine Independent/Language Dependent)
  b. Synthesis(Machine Dependent/Language independent)
- Compilation process is partitioned into no-of-sub processes called **'phases'**.

Phases Of Compiler



LEXICAL ANALYZER (SCANNER):

- The Scanner is the first phase that works as interface between the compiler and the Source language program and performs the following functions:
  - Reads the characters in the Source program and groups them into a stream of tokens in which each token specifies a logically cohesive sequence of characters, such as an identifier , a Keyword , a punctuation mark, a multi character operator like := .
  - The character sequence forming a token is called a **lexeme** of the token.
    - The Scanner generates a token-id, and also enters that identifiers name in the Symbol table if it doesn‘t exist.
  - Also removes the Comments, and unnecessary spaces.
    - The format of the token is **< Token name, Attribute value>**

SYNTAX ANALYZER (PARSER):

- The Parser interacts with the Scanner, and its subsequent phase Semantic Analyzer and performs the following functions:
  - Groups the above received, and recorded token stream into syntactic structures, usually into a structure called **Parse Tree** whose leaves are tokens.
  - The interior node of this tree represents the stream of tokens that logically belongs together.
  - It means it checks the syntax of program elements.

SEMANTIC ANALYZER:

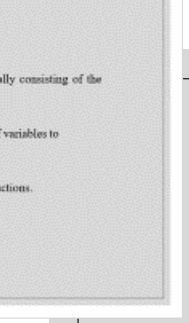
- This phase receives the syntax tree as input, and checks the semantically correctness of the program.
- Though the tokens are valid and syntactically correct, it may happen that they are not correct semantically.
- Therefore the semantic analyzer checks the semantics (meaning) of the statements formed.
    - The Syntactically and Semantically correct structures are produced here in the form of a Syntax tree or DAG or some other sequential representation like matrix.

INTERMEDIATE CODE GENERATOR(ICG):

- This phase takes the syntactically and semantically correct structure as input, and produces its equivalent intermediate notation of the source program. The Intermediate Code should have two important properties specified below:
  - It should be easy to produce,and Easy to translate into the target program. Example intermediate code forms are:
- Three address codes,
- Polish notations, etc.

CODE OPTIMIZER:

- This phase is optional in some Compilers, but so useful and beneficial in terms of saving development time, effort, and cost. This phase performs the following specific functions:
  - Attempts to improve the IC so as to have a faster machine code. Typical functions include –Loop Optimization, Removal of redundant computations, Strength reduction, Frequency reductions etc.
  - Sometimes the data structures used in representing the intermediate forms may also be changed.



CODE GENERATOR:

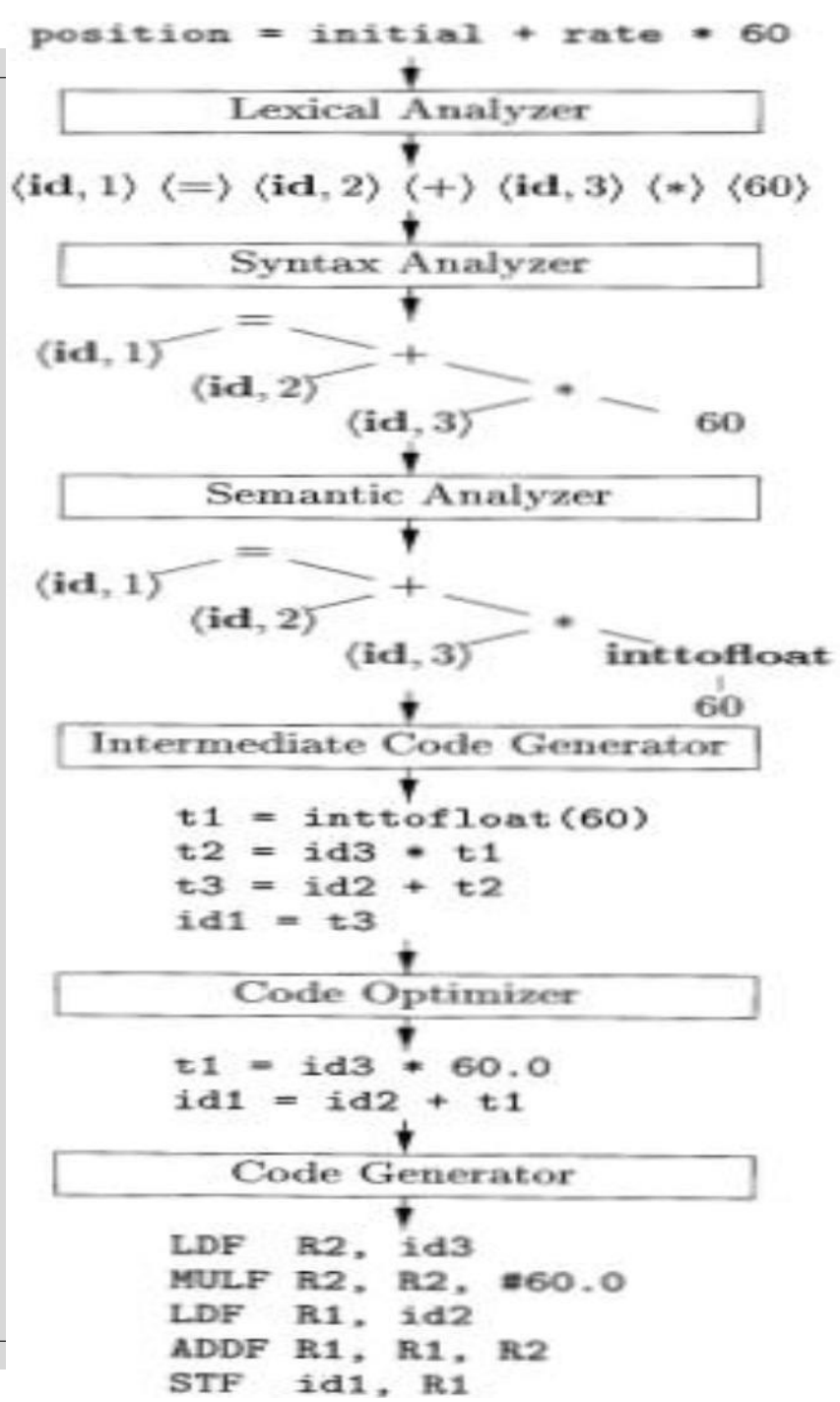
- This is the final phase of the compiler and generates the target code, normally consisting of the relocatable machine code or Assembly code or absolute machine code.
  - Memory locations are selected for each variable used, and assignment of variables to registers is done.
  - Intermediate instructions are translated into a sequence of machine instructions.

Symbol table management and **Error handling**

- The Compiler also performs the **Symbol table management** and **Error handling** throughout the compilation process.
- Symbol table is nothing but a data structure that stores different source language constructs, and tokens generated during the compilation.
- These two interact with all phases of the Compiler.

Example

- The input source program is **Position=initial+rate*60**
-


position = initial + rate * 60
Lexical Analyzer
⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩
Syntax Analyzer
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
60
Semantic Analyzer
=
⟨id, 1⟩
+
⟨id, 2⟩
*
⟨id, 3⟩
inttofloat
60
Intermediate Code Generator
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
Code Optimizer
t1 = id3 * 60.0
id1 = id2 + t1
Code Generator
LDF R2, id3
MULF R2, R2, #60.0
LDF R1, id2
ADDF R1, R1, R2
STF id1, R1