# Complier Design

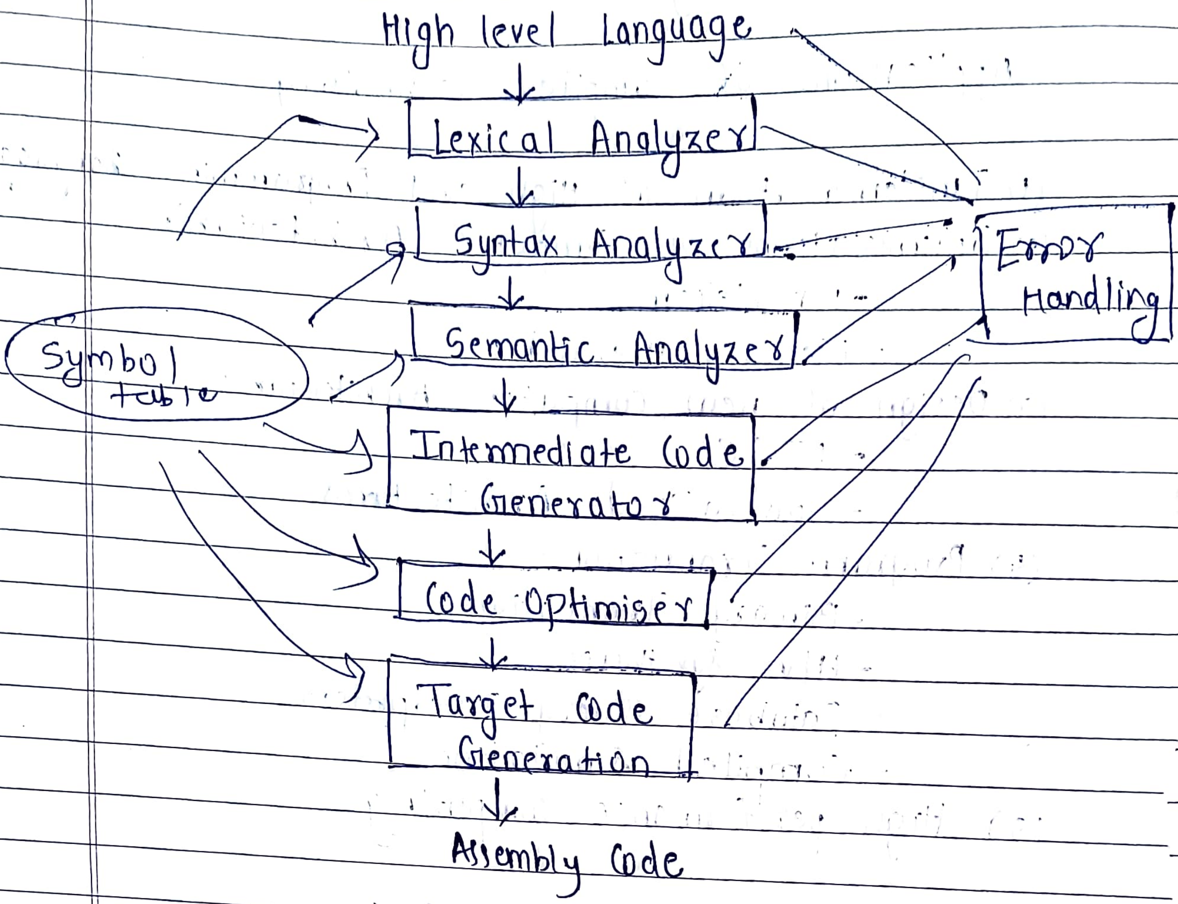
## Applications of Compiler Technology

(1) Implementation of High-level Programming Languages
(2) Optimizations of Computer Architectures
- parallelism
- Memory Hierarchies

(3) Design of new computer Architectures
- RISC
- Specialized Architectures

(4) Program Translations
- Binary Translation
- H/W synthesis
- Database Query Interpreters
- Compiled Simulation

(5) High performance computing



There are two phases of Compilers
(1) Analysis phase
(2) Synthesis phase.

(1) Analysis phase :-
The Analysis phase creates an intermediate representation from the given source code.

(2) Synthesis phase :-
The synthesis phase creates an equivalent target program from the intermediate representation.

Symbol table : It is a data structure being used and maintained by the compiler.

(1) Lexical Analysis :
- Also known as Scanning
- This phase reads the source code and break it into stream of tokens.
- Tokens are basic units of the programming language.

e.g

```
|int| main|(|)|
      {|
      |}
```

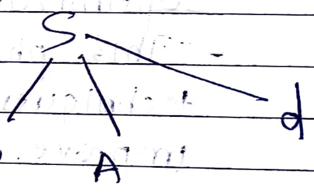
(2) Syntax Analysis :
- The Second phase of a Compiler is syntax analysis.
- This phase takes the stream of tokens generated by the lexical analysis phase and checks whether the grammar correct or not.

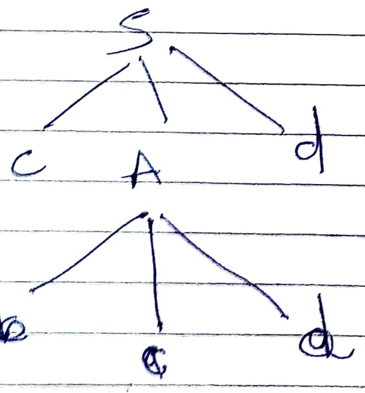
e.g.

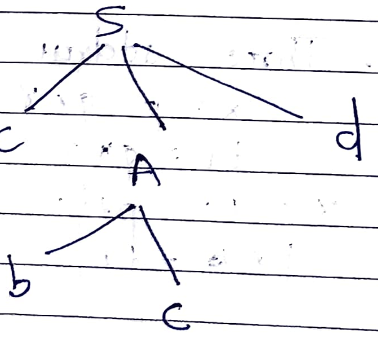
$S \rightarrow cAd$

$A \rightarrow bc \mid a$

$IP \rightarrow cad$







Page No.
Date

3) Semantic Analysis
- The third phase of a compiler is semantic analysis.
- This phase checks whether the code is semantically correct.

e.g.
itn – X
int – ✓

4) Intermediate Code Generation :
- The fourth phase of a compiler is intermediate code generation
- This phase generates an intermedia representation of the src code that can be easily translated into machine code.

(5) Code Optimization
- The fifth phase of a compiler is code optimization
- This phase applies various optimization techniques to the intermediate code to improve the performance of the code.

e.g. Three address code
$x = p + q * r$
$t_1 = q * r$
$x = t_2 = p + t_1$
$\boxed{x = t_2}$

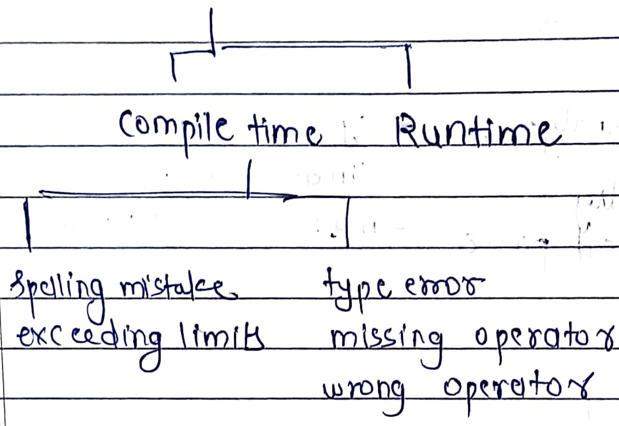
6) Code Generation :-

- The final phase of a compiler is code generation.
- This phase takes the optimized intermediate code and generates the actual 'machine code that can be executed by the target hardware

Symbol Table

| name | type | side | usage |
|---|---|---|---|
| | | | |

Error Handler



LA / DFA / NFA / conversion

float x, y, z

x = y + z * 60

⇓

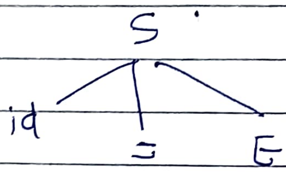
Lexical Analyzer

<id, 1> = <id, 2> + <id, 3> * 60

⇓

Syntax Analyzer

parse Tree



⇓

Semantic Analyzer

- Type checking
- Undeclared variable
- multiple decalaration

Symentically verified parse tree

⇓

ICG

$t_1 = z * 60.0$

$t_2 = y + t_1$

$x = t_2$

⇓

Code Optimization

$t_1 = z * 60.0$

$x = y + t_1$

⇓

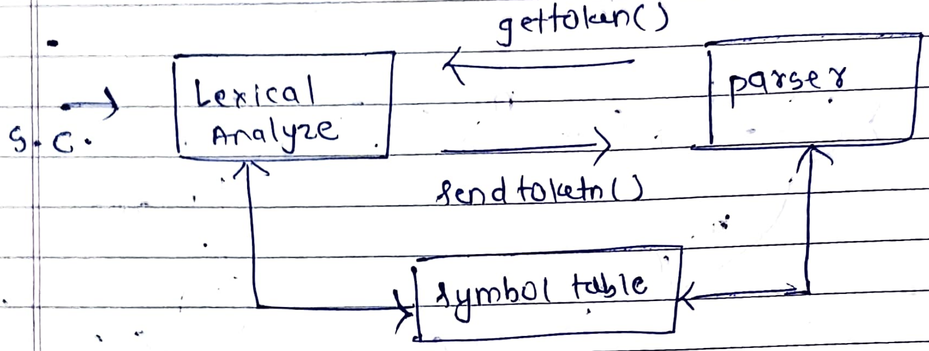
TCG

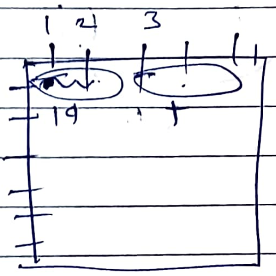
Page No.
Date

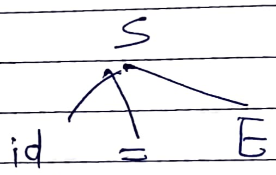
## Lexical Analyzer

- Lexical analyzer divides the given program into meaningful words which is known as tokens

- Tokens are normally identifiers, keywords, Operators, Constant and special symbol







- LA helps in giving error messages providing row no. & column no

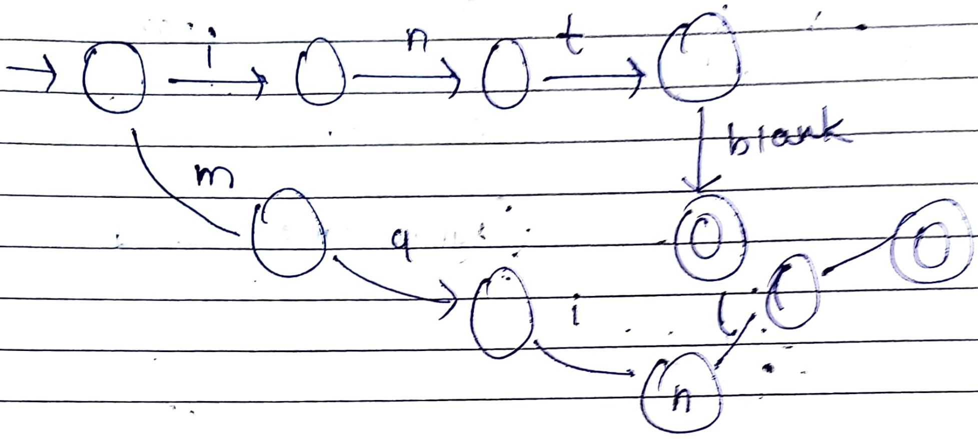
- LA eliminates comment lines from the given program

- LA eliminates white space character (blank, Tab)

- LA uses DFA to do tokenization

- While doing tokenization LA always gives importance to longest matching

int main ( ) — 4
{ = — 5
} — 6



main ) ( ⇒ 3
}
x = a + b * c ;
int x, a, b, c ;
y = x + a
{

$\boxed{28}$

Syntax error / Semantic error