Phases of Compiler

- A compiler is used to convert high-level language to low-level language.
- Phases: **Lexical Analysis** -> **Syntax Analysis** -> **Semantic Analysis** -> **Intermediate Code Generator** -> **Code Optimization** -> **Target Code Generation**
- Phases **Lexical Analysis** -> **Syntax Analysis** -> **Semantic Analysis** -> **Intermediate Code Generator** are called front-end ie they are **machine independent**.
- Phases: **Code Optimization** -> **Target Code Generation** are called back-end ie they are **machine dependent**.
- Symbol Table & Error Handler is used in all phases.

Lexical Analysis

It performs tokenization, outputs error messages and eliminates comments, white spaces, etc.

- Input: `Stream of characters` (like `printf "Hello, World!"`) | Output: `Tokens`
- It performs these steps in-order:

1. Tokenization: Converts and separates tokens according to type.

```
int sum() {
    int x=20;
    int y=30;
    int z=x+y;
    printf("x=%d, y=%d");
    printf("z=%x",&z);
}
```

    - Types of tokens (`int x=10+y+&z;` contains all types of tokens):
        - Identifier: Used to identify elements, like `x,y,z` .
        - Punctuations / Separators: Used to separate code, like `;` {}, ()
        - Keyword: reserved words, like `int`
        - Operator: Used to operate on elements, like `+,=` , -,/,*,%
        - Constants (literals): Values, like `10`
        - Special character: Any character with a special meaning. like `&` ,%,$.
    - Stores the tokens in a Symbol Table (alongwith type, size, etc.)
2. Output error messages, like length exceeded, illegal character, etc.
    - **Lexical error**: When the token pattern does not match the prefix of the remaining input, the lexical analyzer gets stuck and has to recover from this state to analyze the remaining input. In simple words, a lexical error occurs when a sequence of characters does not match the pattern of any token. It typically happens during the execution of a program.
    - The Error Handler doesn't report logic, syntax errors, etc. It only reports lexical errors.
    - Examples:
        - Exceeding length: String name too long
        - Unmatched string: Did not end a comment properly (no `*/` after `/*`)
        - Illegal character: Uexpected character, out of order.
3. Eliminate comments, white space (Tab, Blank Space, new line)
    - Removes comments, tab, blank space, new line, etc.

- Example of conversion:

```
int sum() {
/* Add x and y */
    int x=20;
    int y=30;
    int z=x+y;
}
```

    - Tokens: int sum ( ) { int x = 20 ; int y = 30 ; int z = x + y ; | Total: 22
- Example: `printf("i=%d,&i=%x",i,&i);`

    - Tokens: printf (“i=%d,&i=%x” , i , & i) ; | Total: 10
    - Token Type:
        - Keywords: printf
        - Punctuations: ( ) , ;
        - Constant: “i=%d,&i=%x”
        - Identifiers: i
        - Special characters: &
- The entire string within inverted comma is taken as a single token, assuming " " (space) is the separator. Also, the inverted commas themselves are not counted as tokens.
- (DFA or NFA) Finite Automata (DFA, NFA) is used here.

Parser (Syntax Analyzer) [VVI]

Semantic Analysis

Intermediate Code Generation [VI]

Code Optimization