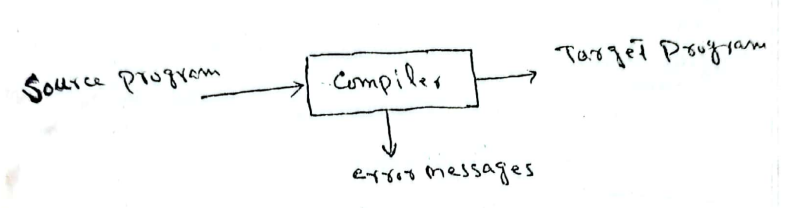
**COMPILER CONSTRUCTION NOTES**

**CS-603**

**Compiler:**



* Target program can be in any language.
* Translator Program

Compiler classified on the basis of purpose:

* Single-pass
* Multi-pass
* Debugging
* Optimizing

**Analysis-Synthesis Model of Compiler:**

Two parts of compilation:

* **Analysis**

Divide source program into pieces and convert into intermediate representation

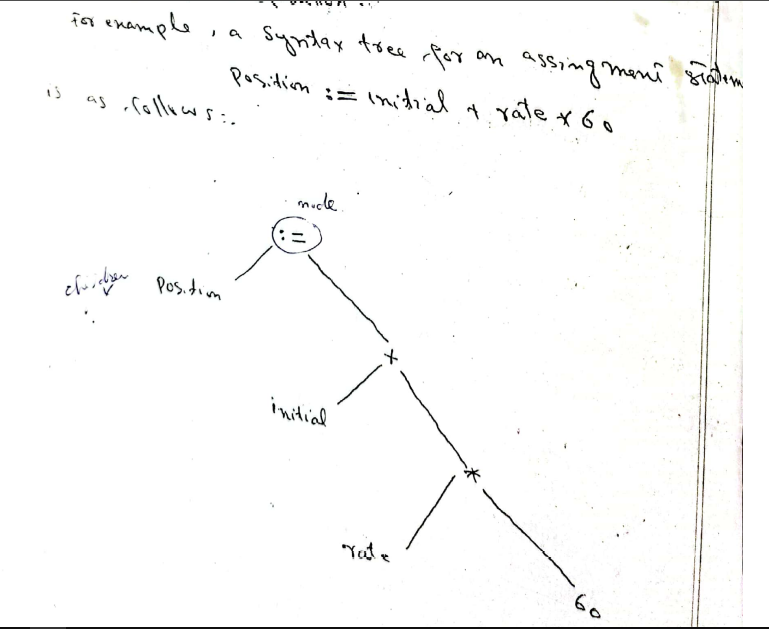
* **Synthesis**

Convert intermediate representation to target program

**Analysis:**

* Converts source program into hierarchical structure called **Syntax Tree** where **node** represents **operations** and **children** of nodes represents **arguments**.

**Example:**

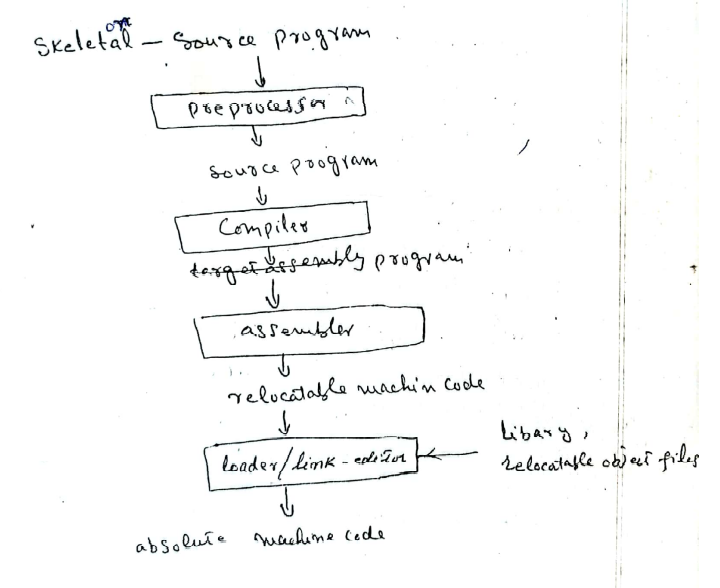


**Tools** which perform the analysis of source program:

* **Structure Editors**
* Text Creation and Manipulation
* Put appropriate hierarchical structure to the source program
* Also supply reminder keywords
* **E.g.,** Provides hints of keywords while writing code
* **Pretty Printers**
* Analyze program and print it in clearly visible structure
* **E.g.,** Comments showed in different colour and code with indentation
* **Static Checkers**
* Detects bugs related to running of program
* **E.g.,** A variable used before defining will result in error
* **Interpreters**
* Instead of translating, performs operations implied by source program
* It calls routines to perform operations

**The Context of a Compiler:**

* In additions to compiler, other programs are also used to create executable target program.



* **Preprocessor**
* Collection of source program (if consists of modules (multiple files))
* Expand macros (short hands) into source language statements
* **Compiler**
* Creates assembly code of source program
* **Assembler**
* Translates assembly code into machine code and attach some library routines

**Analysis of Source Program:**

In compiling, Analysis consists of **three** phases:

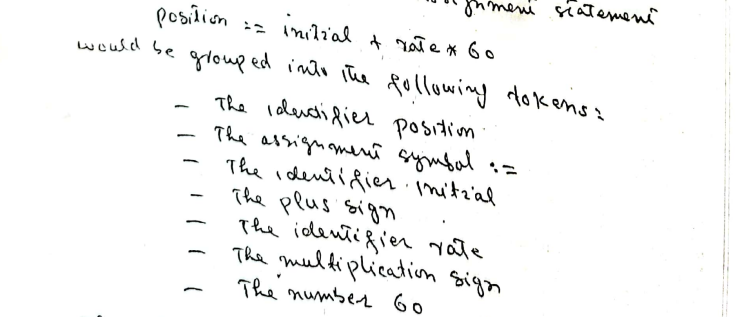
* **Linear Analysis**
* Stream of characters making source program read from left to right and grouped into tokens (sequences of characters that have a collected meaning)
* **Hierarchical Analysis**
* Character / Tokens are grouped into hierarchically into nested collections
* **Semantic Analysis**
* Certain checks to ensure that components of program fits together meaningfully

**Lexical Analysis:**

* In compiler:

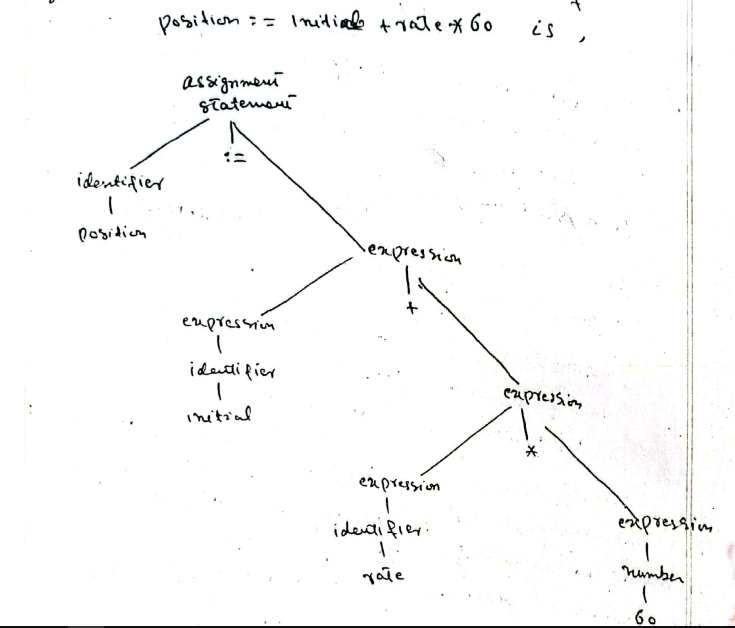
Lexical Analysis = Linear Analysis = Scanning

* **Example of Lexical Analysis:**



**Syntax Analysis:**

* Hierarchical Analysis = Syntax Analysis = Parsing
* Grouping the tokens of source program into grammatical phrases that are used by complier to synthesize the output
* Grammatical Phrases usually represented by parse tree



* Hierarchical structure of program usually represented by **recursive rules**

**Examples:**

**Non-recursive Rules:**

* Any identifier is an expression.

**Recursive Rules:**

* If **identifier1** is an identifier and **expression2** is an expression, then

Identifier1 := Expression2

Is a statement

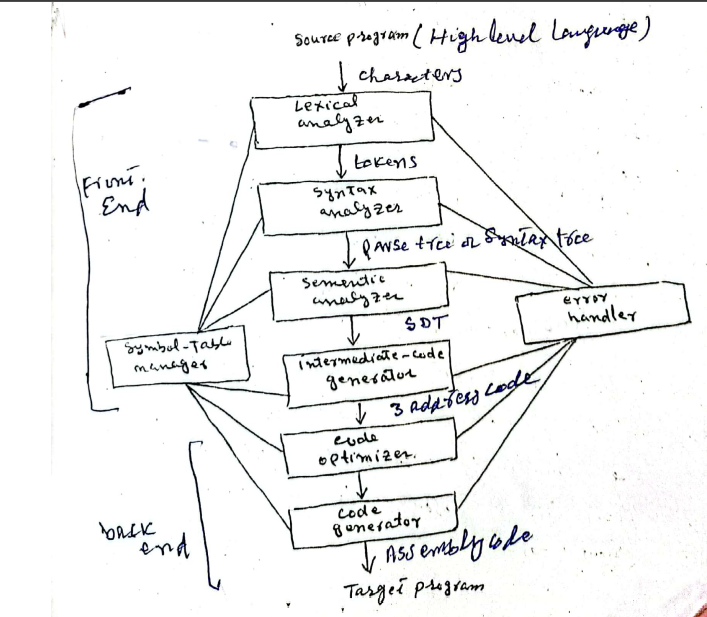
* The division between lexical and syntax analysis is often arbitrary. One is used which simplifies the task of analysis.
* Division is often determined by whether the source language construct is inherently recursive or not.
  + - Lexical Analysis -> non-recursive
    - Syntax Analysis -> recursive

**Semantic Analysis:**

* Checks for semantic (logical) errors
* Gather type information for code-generation phase
* Use parse tree (from syntax analysis) to identify operators and operands of expressions and statements
* Perform type checking
* Complier will report an error if a real number is used to index an array

**The Phases of Compiler:**

* A complier operates in phases where each phase transforms the source program from one representation to another



**Symbol-Table Management:**

* A symbol-table is a data structures which contains the record for each identifier, with fields for attributes.
* Lexical analyzer only put the identifier in the syntax-table not it’s attributes
* Attributes -> provides info about the storage, type and scope of identifier. In case of procedure names, it provides info about number and type of arguments.
* The remaining phases enter explanation about the identifier in symbol table.

**Error Detection & Reporting:**

* Syntax and Semantic Analysis phases usually handle a large fraction of errors detectable by compiler.
* Syntax Analysis handles syntax errors.
* The token which violates the structure rule defined in the language.
* Semantic Analysis handles logical errors.
* Syntax is right but no meaningful operation is involved
* Example:
  + - Addition of an array and a procedure