Chapter-6

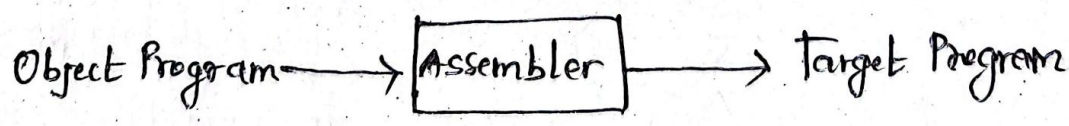
Automata Theory & Compiler

6.1 Basic Concept of compiler, role of lexical analyzer, lexical analysis with deterministic finite automata:-

→ A compiler is a translator software program that takes its input in the form of program written in one particular programming Language and produce the output in the form of program in another Language.

→ A compiler is a translator software program that converts the high level language into the machine language.

→ Compiler executes a program into two parts i.e In first source program is compiled into object program and In second object program is translated into target program.



Chapter-6

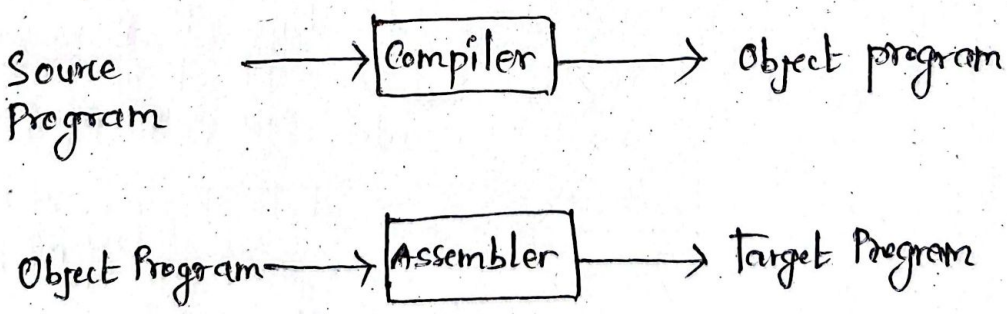
Automata Theory & Compiler

6.1 Basic Concept of compiler, role of lexical analyzer, lexical analysis with deterministic finite automata:-

→ A compiler is a translator software program that takes its input in the form of program written in one particular programming language and produce the output in the form of program in another language.

→ A compiler is a translator software program that converts the high level language into the machine language.

→ Compiler executes a program into two parts i.e In first source program is compiled into object program and In second object program is translated into target program.



Compiler Structure:-

→ A compiler operates in phases. A phases is a logically inter-related operation that takes source program in one representation and produces output in another representation. There are two phases of compilation:

1. Analysis Phase:-

→ In analysis part, an intermediate representation is created from the given source program. This part is called called front end of the compiler. This part consists of mainly four phases:

(i) Lexical Analysis:

Lexical analysis or scanning is the process where the source program is read from left-to-right and grouped into tokens. Tokens are sequences of characters with a collective meaning. In any programming language tokens may be constants, operators, reserved words etc. The lexical Analyzer takes a source program as input, and produces a stream of tokens as output.

Example: Input string:

$$c = a + b * 3$$

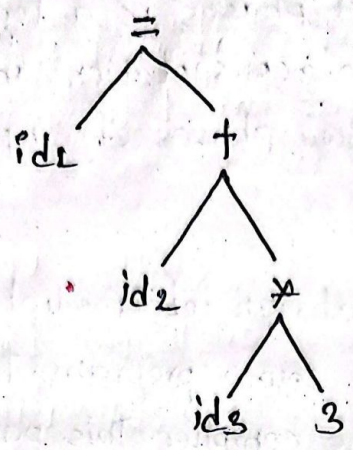
Tokens:

$$id1 = id2 + id3 * 3$$

(ii) Syntax Analysis:

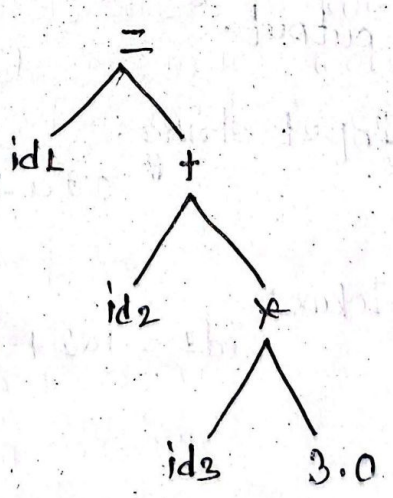
In this phase, the syntax analyzer takes the token produced by lexical analyzer as input and generates a parse tree as output. In syntax analysis phase, the parser checks that the expression made by the token is syntatically correct or not according to the rules that define the syntax of the source language

Example:-



(iii) Semantic Analysis:-

In this phase the semantic analyzer checks the source program for semantic errors and collects the type information for the code generation. Semantic analyzer checks whether they form a sensible set of instructions in the programming language or not. Type checking is an important part of semantic analyzer.



(iv) Intermediate Code Generation:-

If the program syntatically and semantically correct then intermediate code generator generates a simple machine independent intermediate language The intermediate code should be generated in such a way that it can easily be translated into the target machine.

Example:-

```
t1 = 3.0;
t2 = id3 * t1;
t3 = id2 + t2;
id1 = t3;
```

(2) Synthesis phase:-

In synthesis part, the equivalent target progam is created from intermediate representation of the program created by analysis part. This part is also called the backend of the compiler. This part consists of mainly two phases:

(i) Code Optimization:-

It is used to improve the intermediate code so that the output of the program could run faster and take less space. It removes the unnecessary lines of the code and arranges the sequence of statements in order to speed up the program execution without coasting resours.

Example:-

```
t2 = id3 * 3.0;
id1 = id2 + t2;
```

(ii) Code Generation! -

Code generation is the final stage of the compilation process. It takes the optimized intermediate code as input and maps it to the target machine language.

Example! -

```
MOV   R1, id3
MUL   R1, #3.0
MOV   R2, id2
ADD   R1, R2
MOV   id1, R1
```

(v) Symbol Table! -

Symbol tables are data structures that are used by compilers to hold information about source-program constructs. The information is collected incrementally by the analysis phase of compiler and used by the synthesis phases to generate the target code. Entries in the symbol table contain information about an identifier such as its type, its position in storage, and any other relevant information.

(vi) Error Handling! -

Whenever an error is encountered during the compilation of the source program, an error handler invoked. Error handler generates a suitable error reporting message regarding the error encountered. Errors can be encountered at any phase as:

Lexical analysis phase! due to misspelled tokens unrecognized character etc.
Syntax analysis phase! due to syntatic voilation of language.
Intermediate Code Generation! due to incompatibility of operands type for operator.

Code Optimization phase:-

due to some unreachable statements.

* Block Diagram:-

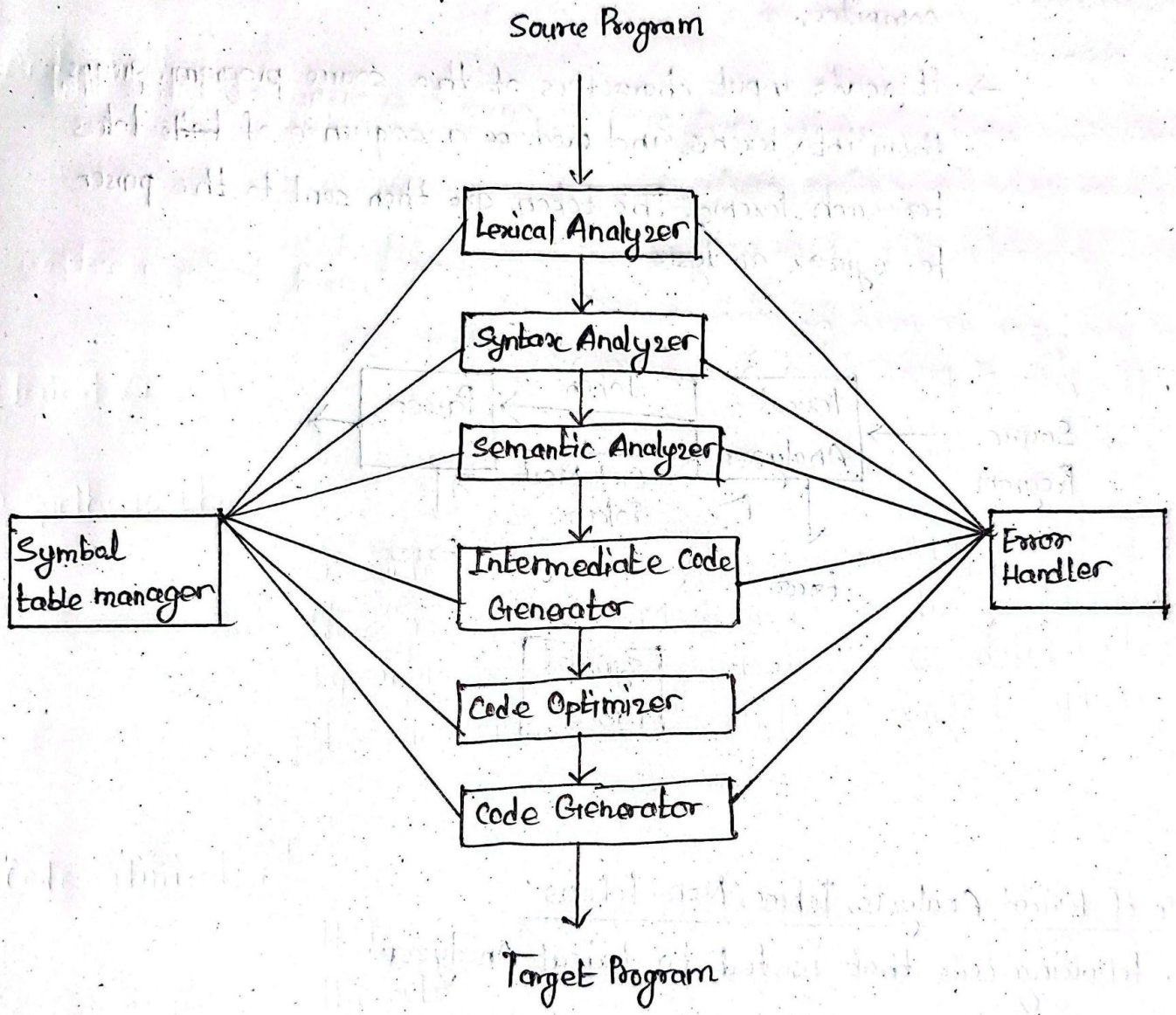
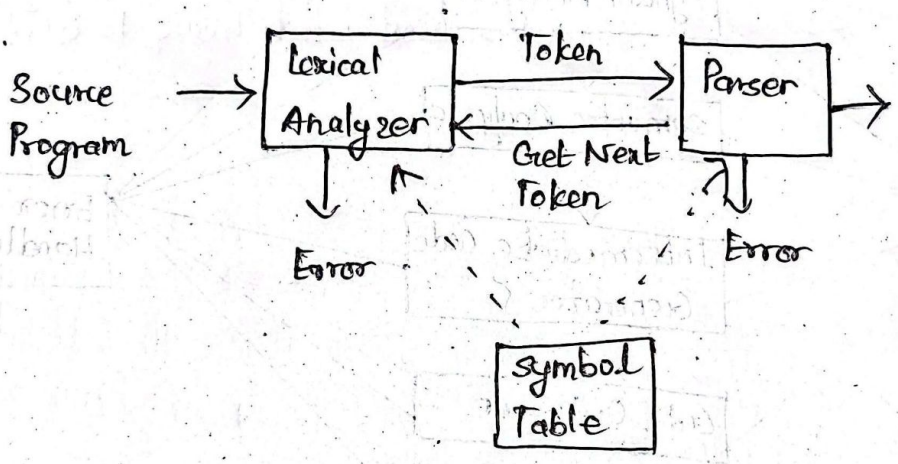


fig:- Phases of Compiler

⊛ Lexical Analysis :-

→ The lexical analysis is the first phase of a compiler where a lexical analyzer acts as an interface between the source program and the rest of the phases of the compiler.

→ It reads input characters of the source program, groups them into lexems, and produces a sequence of ~~tokens~~ tokens for each lexeme. The token are then sent to the parser for syntax analysis.



Example of lexical Analysis, Tokens, Non-Tokens

Consider following code that is fed to lexical Analyzer.

```
#include <stdio.h>
int Largest(int x, int y)
  {
  if(x>y)
   return x;
  else
  return y;
   }
```

Examples of Tokens created:-

Lexeme	Token
int	Keyword
Largest	Identifier
(	Operator
int	keyword
x	Identifier
,	Operator
int	keyword
y	Identifier
)	Operator
{	Operator
If	keyword

Examples of non-tokens:-

Type	Examples
Comment	// This will compare 2 numbers
Pre-processor Directives	# include<stdio.h>

⊛ Role of Lexical Analyzer:-

→ Lexical analyzer help to identify token into the symbol table.

→ It can either work as a seperate module or as a sub-module.

→ It is responsible for eliminating comments and white spaces from the source program.

→ It reads the input character and produces output sequence of tokens that the parser user for syntax analysis.

→ It also generates lexical errors

→ Lexical analyzer is used by web browsers to format found and display a webpage with the help of parsed data from Javascript, HTML, CSS.