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.

The phases of a compiler are shown in below

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**.

Lexical Analysis: -
LA or Scanners reads the source program one character at a time, carving the source program into a sequence of atomic units called **tokens**

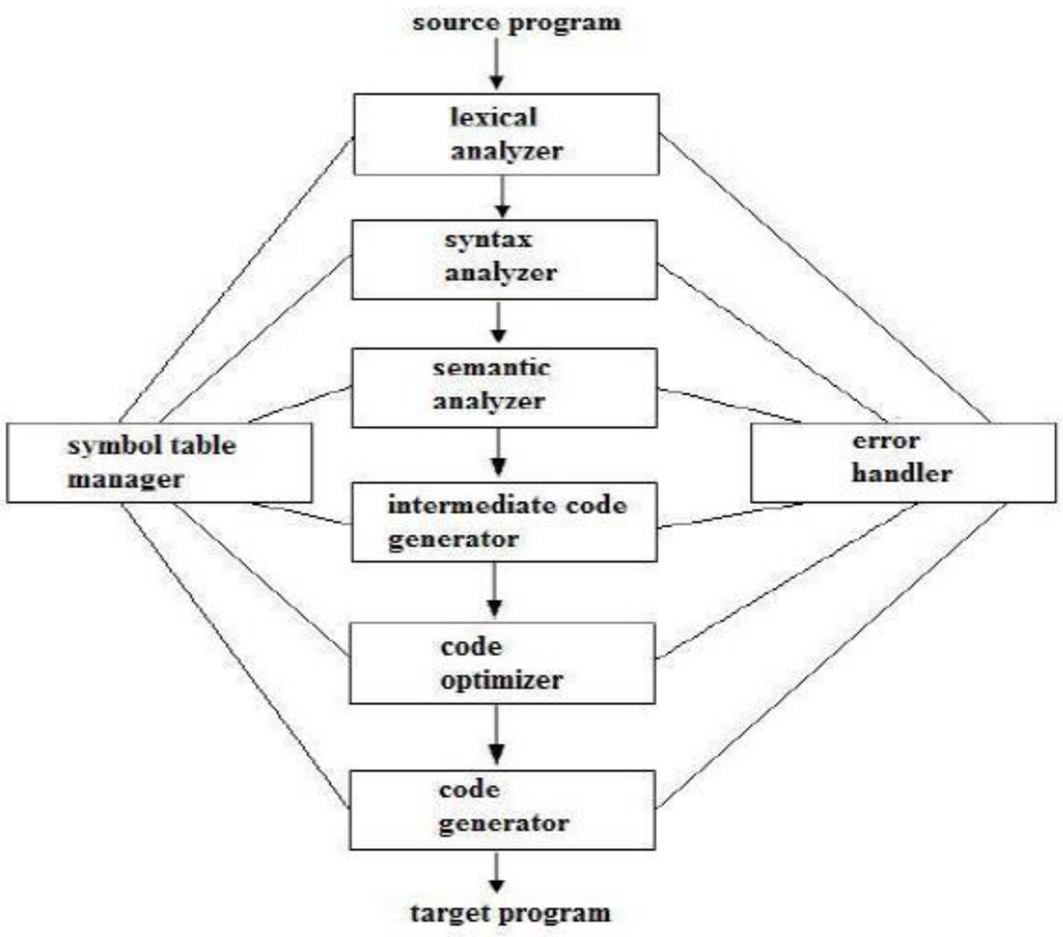

Fig: Phases of Compiler

Syntax Analysis: -
The second stage of translation is called Syntax analysis or parsing. In this phase expressions, statements, declarations etc… are identified by using the results of lexical analysis. Syntax analysis is aided by using

techniques based on formal grammar of the programming language.

Intermediate Code Generations: -
An intermediate representation of the final machine language code is produced. This phase bridges the analysis and synthesis phases of translation.

Code Optimization: -
This is optional phase described to improve the intermediate code so that the output runs faster and takes less space.

Code Generation: -
The last phase of translation is code generation. A number of optimizations to **reduce the length of machine language program** are carried out during this phase. The output of the code generator is the machine language program of the specified computer.

Table Management (or) Book-keeping: - This is the portion to **keep the names** used by the program and records essential information about each. The data structure used to record this information called a ‘Symbol Table’.

Error Handlers: -

It is invoked when a flaw error in the source program is detected. The output of LA is a stream of tokens, which is passed to the next phase, the syntax analyzer or parser. The SA groups the tokens together into syntactic structure called as **expression.** Expression may further be combined to form statements. The syntactic structure can be regarded as a tree whose leaves are the token called a sparse tree.