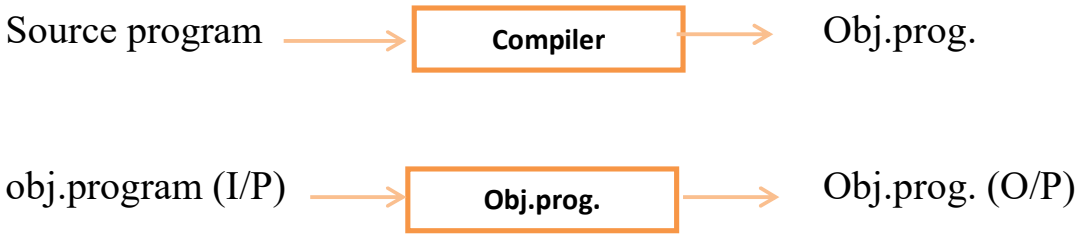
UNIT-I

Introduction to Compiling

Executing a program written in a high level programming language is basically a two-step process.

The source program must first be compiled that is translated into object program then the resulting object program is loaded into memory and executed.

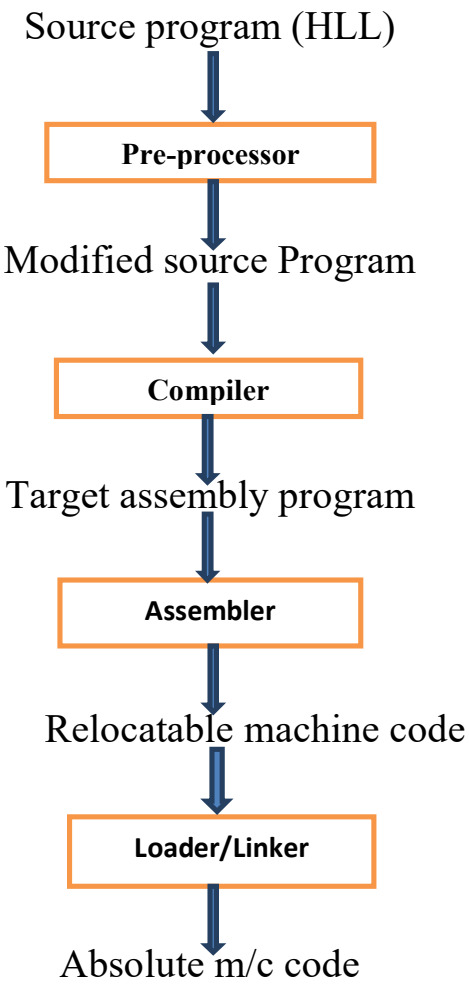


Translators:

There are several other important types of translators, besides compilers of the source language is assembly language and the target language is machine language, then the translator is called as assembler.

The term pre-processor is sometimes used for translators that take programs in one high level language into equivalent programs in another high level language.'

The output of an assembler is called an object file, which contains a combination of machine instruction as well as the data required to place these instructions in memory.



A compiler can broadly be divided into two phases based on the way they compile.

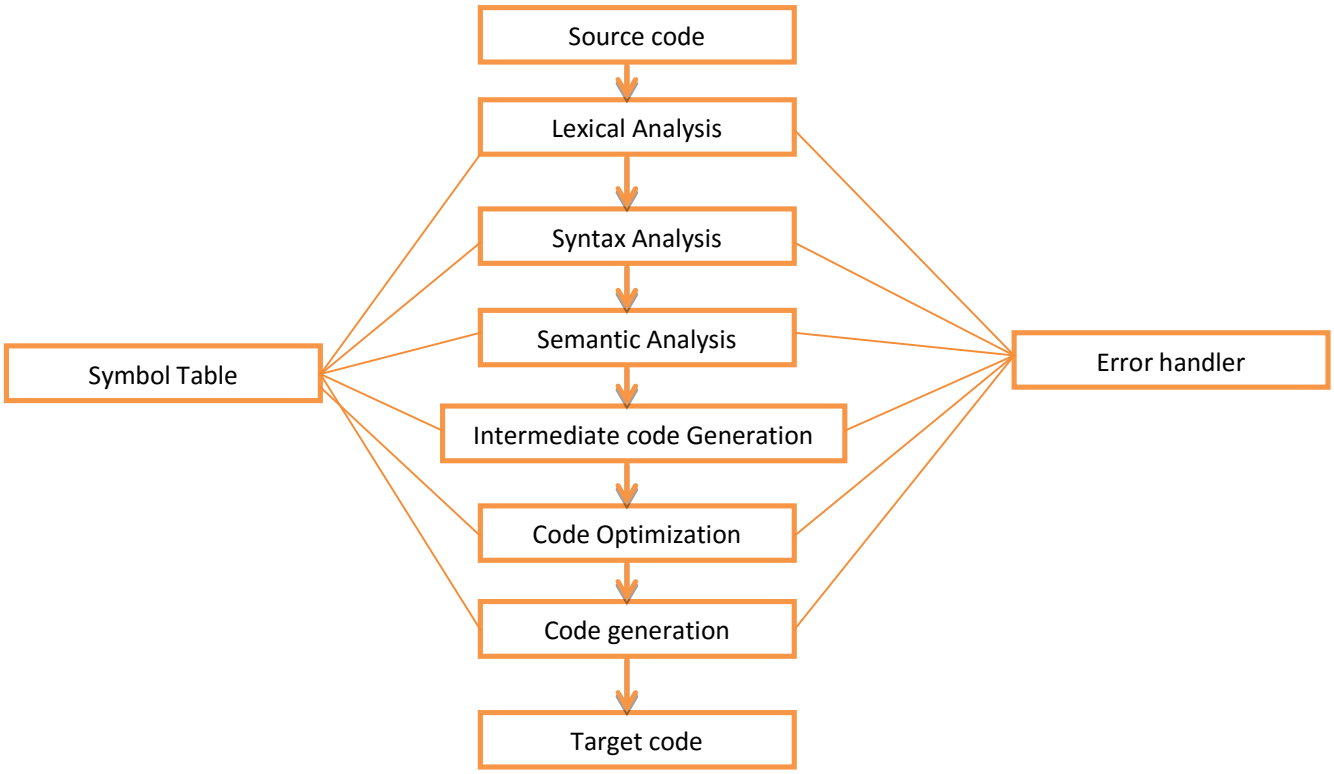
Analysis phase:

Analysis phase known as the front end of the compiler, the analysis phase of the compiler reads the source program, divides it into core parts and then checks for lexical, grammar and syntax errors. The analysis phase generates an intermediate representation of the source program and symbol table, which should be fed up to the synthesis phase as input.

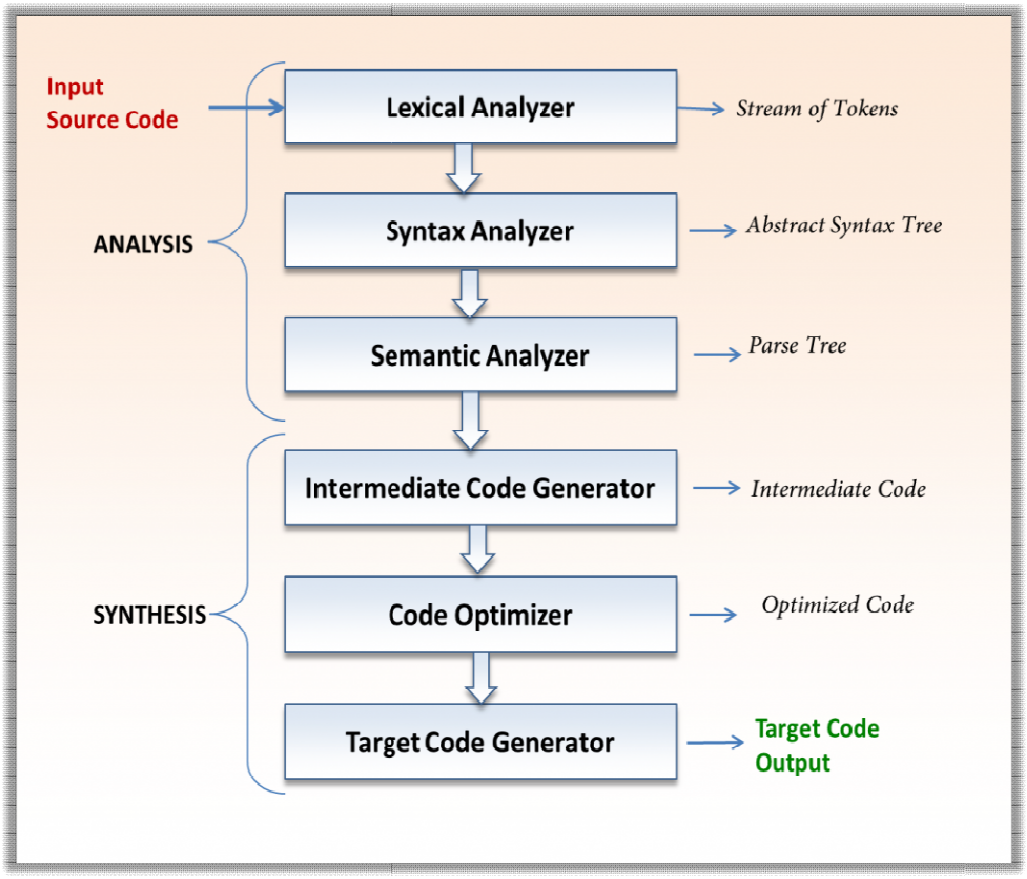
Synthesis phase:

Synthesis phase known as back end of the compiler the synthesis phase generates the target program with the help of intermediate source code representation and symbol table.

Phases of compiler:



A compiler operator in phase'sphases is a logically interrelated operation that takes source program in one representation and produces output in another representation.



There are two phases of compilation:

- Analysis
- Synthesis

Compilation process is partitioned into number of sub process called phases.

Lexical analysis:

Lexical analysis or Scanner reads the source program one character at a time, converts the source program into a sequence of atomic units called tokens. It also called as scanner because it scans the source code character by character. Lexical analyser use lax tool for the construction of tokens. It also removes white spaces, comments and tabs spaces.

E.g. x=a+b*20;

In above example there are 8 tokens.

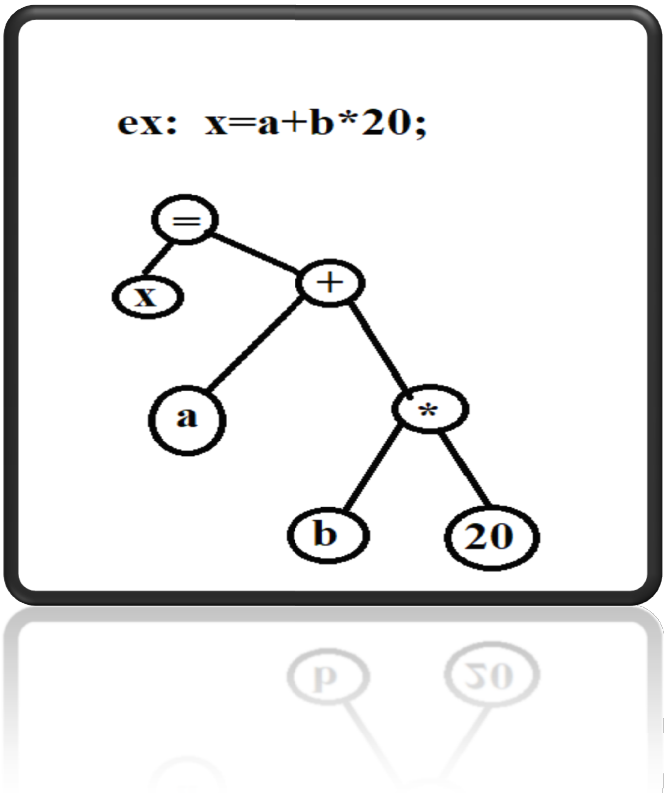
Syntax analysis:

The second phase of compiler is syntax analysis. The syntax analysis uses tokens produces by lexical analysis and creates three like representation called s

In this phase expressions, statements, declaration etc. one identified by using the results of lexical analysis.

Syntax analysis is aided by using techniques based on formal grammar of the programming language.

Ex:



Symantic analysis:

Symantic analysis is third phase of compiler. The symantic analysis uses syntax tree and information in the symbol table. It checks whether the parse tree constructed follows the rules of the language.

Intermediate code generation:

After semantic analysis, the compiler generates an intermediate code of the source code for the target machine.

It is in between high level language and the machine language.

This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code

Code optimization:

The next phase does code optimization of the intermediate code. Optimization can be assumed as something that removes unnecessary code line and arranges the sequence of statements in order to speed up the program execution without wasting resources. (CPU, MEMORY)

Code generation:

In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates the intermediate code into sequence of re-locable machine code.

Symbol table management:

Symbol table is a data structure that store variousidentifier and their attributes.

The symbol table makes it easier for the compiler to quickly search the identifier record and retrieve it.

Error handler:

This part of compiler is responsible for handling the error generated by every phase of compiler. The error handling field report the errors to the programmer.

Lexical analysis:

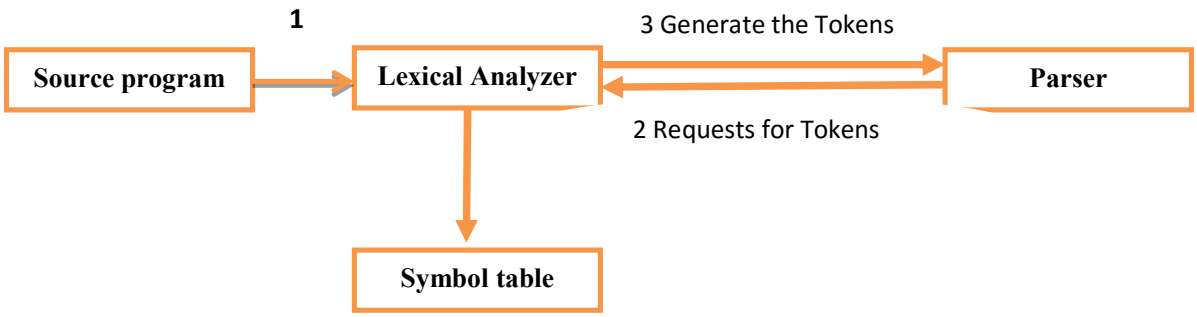
Lexical analysis convers source program into a stream of valid words of the language known as tokens.

Also known as scanning phase.

Basic function:

It reads the input program character by character and produces a stream of tokens which is used by the parse.

If the lexical analyser finds a token invalid, it generates an error. The lexical analyser works closely with syntax analyser.



It reads character streams from the source code, checks for legal tokens, and passes the data to the data to the syntax analyser when it demands.

Q. Explain compiler construction tools

A number of tools have been developed specifically to help construct compiler. These tools have been developed for helping implement various phases of a compiler.

Scanner generator:

Scanner generator generates lexical analyser from a regular expression description of the tokens of language.

Parser generator:

These generatesyntax analysers from the input based on a grammatical description of programming language or a context free grammar.It is useful as the syntax analysis phases is highly complex and consumes more manual and compilation time.One significant advantage of using parser generator is increased reliability.

Syntax directed translation engines:

It generates intermediate code with three address format from the input that consists of a parse tree. These engines have routines to traverse the parse tree and then produce intermediate code. In this each node of the parse tree is associated with one or more translations.

Automatic code generators:

It generates an assembly language/ machine language code for target machine. Each operation of the intermediate language is translated using collection of rules and then is taken as an input by the code generator. Template matching process is used an intermediate language statement is replaced by its equivalent machine language statement by templates.

Data flow analysis engine:

It is used in code optimization. Data flow analysis is a key part of the code optimization that gathers the information that is the values that flow from one part of program to another.

Q. Write a short note on one pass compiler.

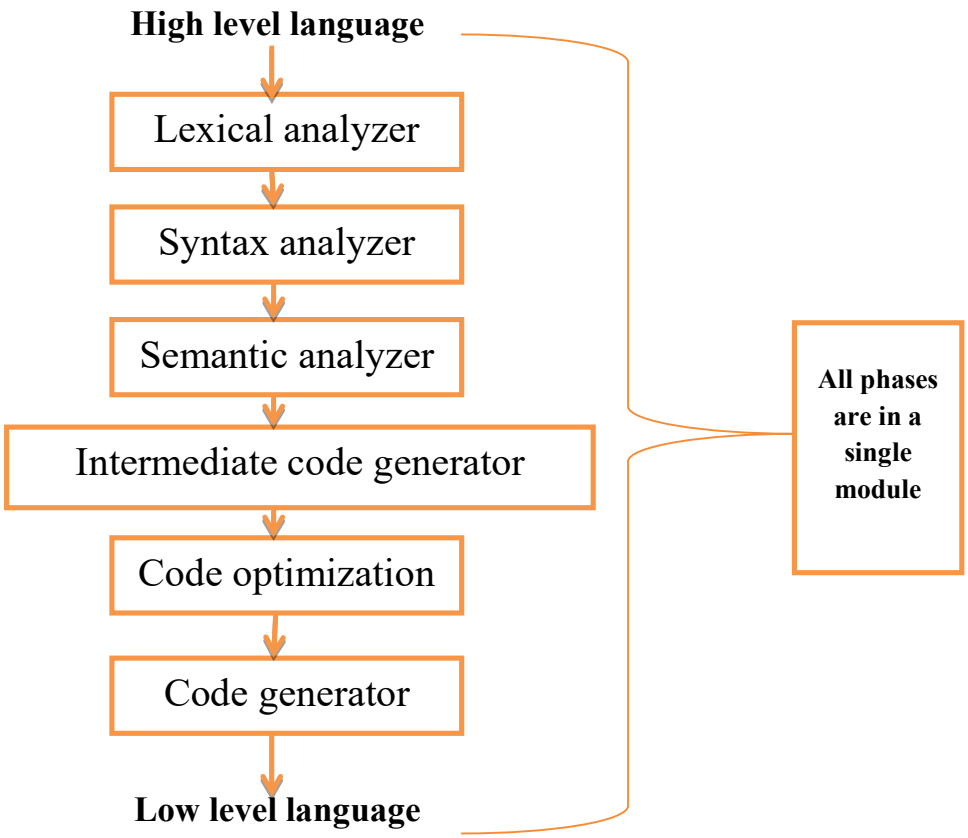
Compiler pass are two types:

1) Single pass compiler / one pass compiler

2) Two pass compiler / multipass compiler

1) Single pass compiler / one pass compiler:

If we combine or group all the phases of compiler design in a single module known as single pass compiler. In below diagram there are all six phases are grouped into in a single module. A one pass /single pass compiler is that type of compiler that passes through the part of each compilation unit exactly one. Single pass compiler is faster and smaller than the multipass compiler. As a disadvantage of single pass compiler is that it is less efficient in comparison with multipass compiler. Single pass compiler is one that processes the input exactly once, so going directly from lexical analysis to code generator, and then going back for the next read.



 # Problems with single pass compiler:

1) We can't optimize very well due to the context of expression are limited.
2) As we can't back up and process it again so grammar should be limited or simplified.
3) Command interpreters such as bash/sh/tesh can be considered as single pass compiler, but they also execute entry as soon as they are processed.
   NOTE: single pass compiler almost never done early pass compiler did this as an introduction.
4) A one pass compiler read on into files and writes output files.
5) The front end phases like lexical analysis, syntax analysis, semantic analysis and intermediate code generation might be grouped into one pass.
6) The code generation is in the back end phase for target language.
7) The code optimization might be as optional phases or pass.

8) The compiler design intermediate representation which allow frontend back end for target language with the combination of front end and back end we can produce compilers for different source language.

Q.1.Explain need of translators.

Q.2.Descibe phases of compiler.

Q.3.Explain simple one pass compiler.

Q.4.Write any three differences between compiler and interpreters.

Q.5.Explain optimization.

Q.6.Discuss the code generation.

Q.7.Discribe the lexical analysis.

Q.8.Explain the intermediate code generation.

Q.9. Explain the syntax analysis.

Q.10.Discuss the compiler construction tools.