# Compiler design

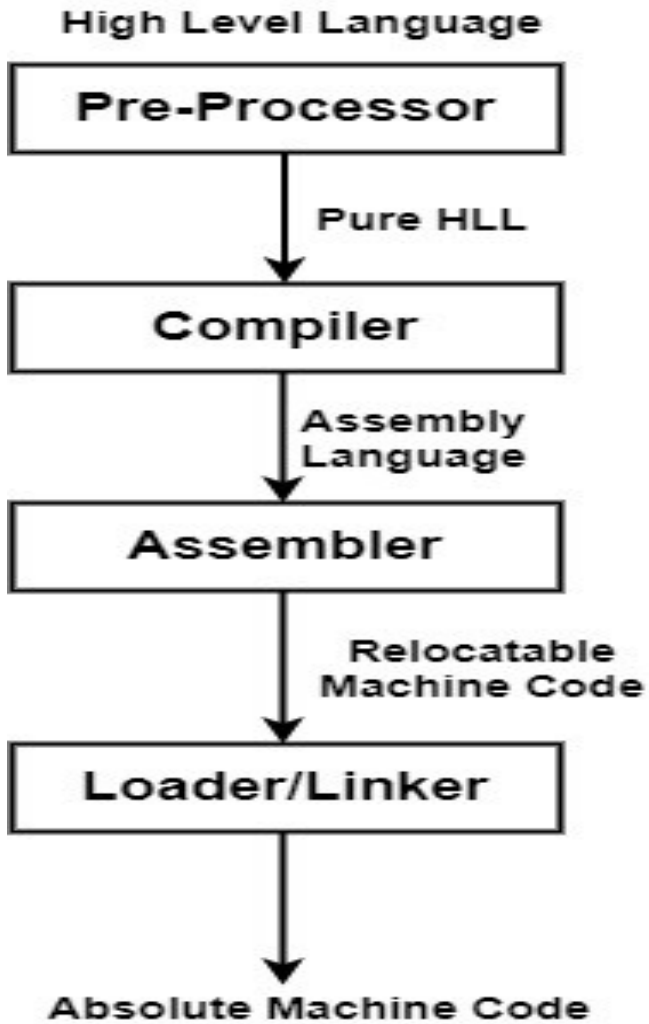
## What is a compiler?

- A compiler is a software that translates or converts a program written in a high level language (source language) into a low-level language.

## Why do we learnt compiler design?

- Computers are a balanced mix of software and hardware. Hardware is just a piece of mechanical device and its functions are being controlled by a compatible software.
- A computer understands a language that is hard for human to understand.
- So we write program in human understandable language. This language can be very different from the machine language that the computer can execute, so some means of bridging the gap is required. This is where the compiler comes in.

# Language Processing System



# Contd…

1. **Pre-Processor:** In this step the HLL or source code will be converted into a pure high level language. The pre-processor terminates all the #include directives by containing the files named file inclusion and all the #define directives using macro expansion.

   For E.g. #include <stdio.h> will be replaced by the original implementation of stdio.h file.

   #define MIN 345 : MIN is a macro that holds the value 345, before compilation,

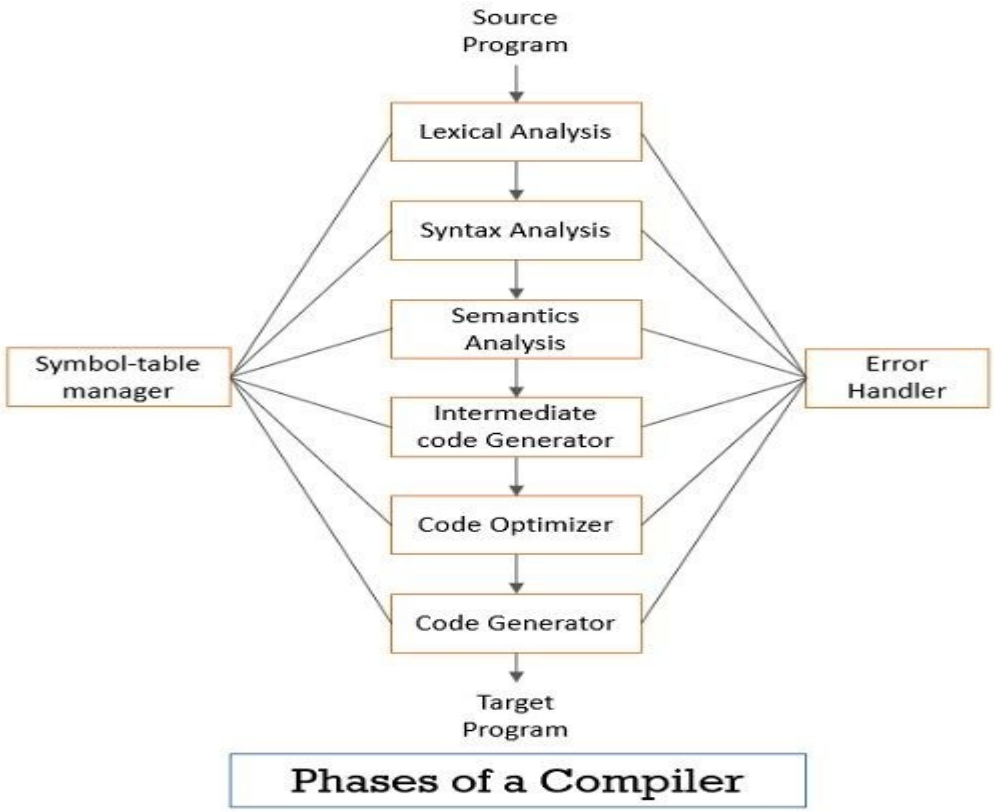
   the preprocessor replaces MIN by 345 across the whole program.

2. **Compiler:** The compiler takes the modified source code as input and will produced an assembly code/target program as output.

3. **Assembler:** It is also known as the translator, which translate the source program, which is written in assembly language to a target program (relocatable machine code).

4. **Linker/Loader:** Linker links the relocatable machine code of various library files to the main program and combine them to produced an executable file. Loader loads the executable file into the main memory for execution.

# Phases of Compiler



# Phases of compiler

1. **Lexical analysis**: It reads the source program character by character and converts the corresponding character into some meaningful sequences called lexemes. Lexemes means a sequence of character. Each lexeme is converted into the form of token. Token may be a keyword, identifier, operator or a constant.
2. **Syntax Analysis**: It takes token produced by lexical analyzer and generate a parse tree. In this phase it will check whether the corresponding syntax of the source program is correct or not. If the syntax is not correct the error handler will report those errors to the user. If the syntax is correct then it generate a parse tree of the corresponding source program.
3. **Semantic Analysis:** In this phase it will check whether the meaning of parse tree is correct or not. It checks whether the parse tree follows the rules of language. Semantic analyzer keeps track of identifiers, their types and expressions.
4. **Intermediate code generator:** It generates intermediate code, which is a form that can be readily executed by a machine. We have many popular intermediate codes. Example – Three address codes etc. Intermediate code is converted to machine language using the last two phases which are platform dependent.
5. **Code Optimizer**: It takes the intermediate code as the input and optimized the code. It transforms the code so that it consumes fewer resources and produces more speed. The meaning of the code being transformed is not altered.
6. **Code Generator:** This phase takes the optimized intermediate code and generates the actual machine code.

# Working of compiler phases with example

Let’s consider an example. x = a+b*50

1. **Lexical Analyzer :**

In this phase, we will see how you can tokenize the expression.

x -> Identifier- (id, 1)

= -> Operator - Assignment

a -> Identifier- (id, 2)

+ -> Operator - Binary Addition

b -> Identifier- (id, 3)

* -> Operator - Multiplication

50 -> Constant - Integer

Now, the final tokenized expression is given below.

(id, 1) = (id, 2) + (Id, 3)*50

# Working of compiler phases with example

## 2. Syntax Analyzer :

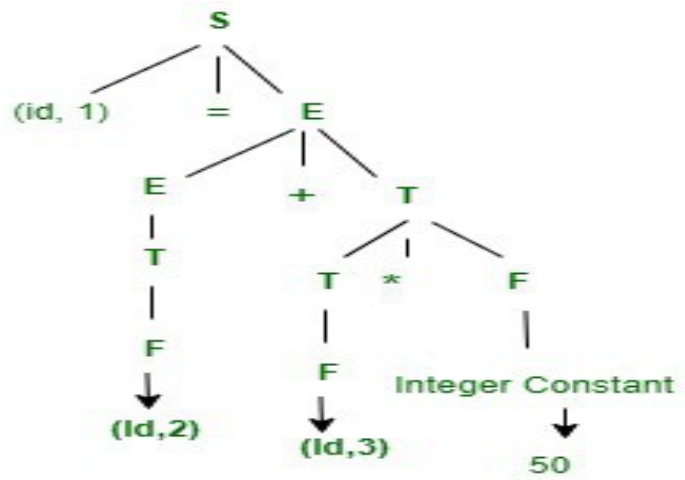
In this phase, we will see how you can check the syntax after tokenized the expression.

S -> Id = E

E -> E+T | T

T -> T*F | F

F -> Id | Integer constant

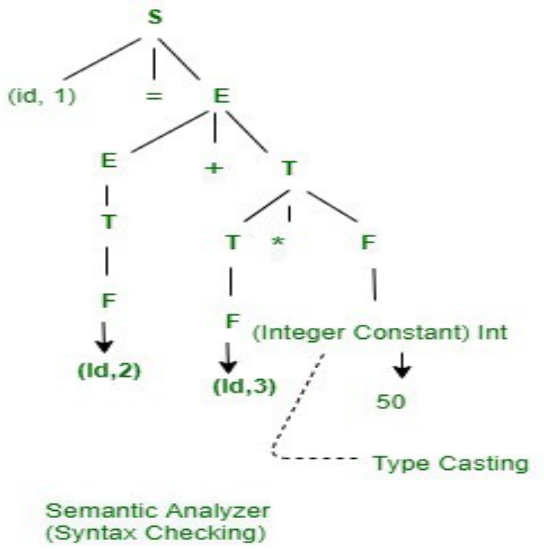


# Working of compiler phases with example

## 3. Semantic Analyzer :

In this phase, we will see how you can check the type and semantic action for the syntax tree.

Given below is the diagram of the semantic analyzer.



# Working of compiler phases with example

**4. Intermediate Code Generator :**

In this phase as an input, we will give a modified parse tree and as output after converting into Intermediate code will generate 3 -Address Code. Given below is an expression of the above-modified parse tree.

3 Address Code –

t1 = b * 50.0

t2 = a+t1

x = t2

**5. Code Optimizer** : It takes intermediate code as an input and generate an optimize code

t1 = b* 50.0

x = a+ t1

**5. Target Code Generator :**

In this phase, the final expression is converted into assembly code.