CM1604
Computer Systems Fundamentals

Basic Compiler Design

In this lecture..

- Different types of programing languages
- What is compiler
- Phases of a compiler
- Few other tools that work closely with compilers

Computer Languages

- The computers understand / process → binary
- At the beginning, the instructions for the computers were written in binary → **Machine Language**
- They were

  very hard to understand, error prone

```
;File: fig0433.peph
;Computer Systems, Fifth edition
;Figure 4.33

0000  D1000D  ;Load byte accumulator 'H'
0003  F1FC16  ;Store byte accumulator output device
0006  D1000E  ;Load byte accumulator 'i'
0009  F1FC16  ;Store byte accumulator output device
000C  00      ;Stop
000D  4869    ;ASCII "Hi" characters
```

```
;File: fig0433.pepb
;Computer Systems, Fifth edition
;Figure 4.33

0000  1101 0001 0000 0000 0000 1101
0003  1111 0001 1111 1100 0001 0110
0006  1101 0001 0000 0000 0000 1110
0009  1111 0001 1111 1100 0001 0110
000C  0000 0000
000D  0100 1000 0110 1001
```

Computer Languages…

To overcome this …

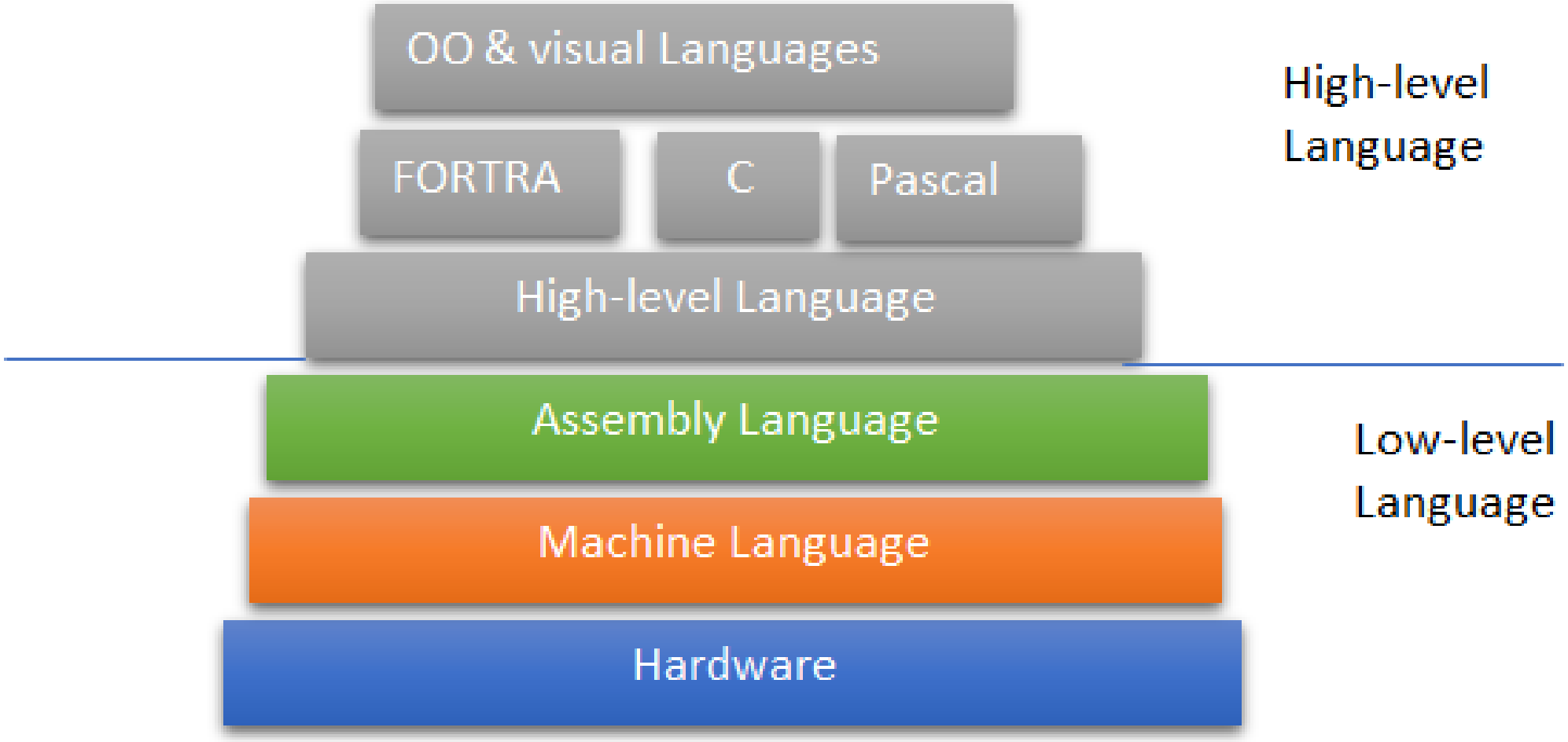
- **Assembly Language** was introduced
- Represents instruction in symbolic codes
- **Assembler** is used to convert it to machine language
- User friendlier than machine language
- But hardware specific
- Still needed skillful individuals to write

```
;Stan Warford
;May 1, 2016
;A program to output "Hi"
;
        LDBA    0x000D,d    ;Load byte accumulator 'H'
        STBA    0xFC16,d    ;Store byte accumulator output device
        LDBA    0x000E,d    ;Load byte accumulator 'i'
        STBA    0xFC16,d    ;Store byte accumulator output device
        STOP                ;Stop
        .ASCII  "Hi"        ;ASCII "Hi" characters
        .END
```

Computer Languages…

- With the advancement in the computer industry, needed to
  - simplify the process of writing program
  - reduce the cost of writing programs
  - increase usability
  - write machine independent programs
- So, need to make computer program much similar to human languages and less dependent in hardware

  ⇒ **High-level language**


OO & visual Languages
FORTRA
C
Pascal
High-level Language
Assembly Language
Machine Language
Hardware
High-level Language
Low-level Language

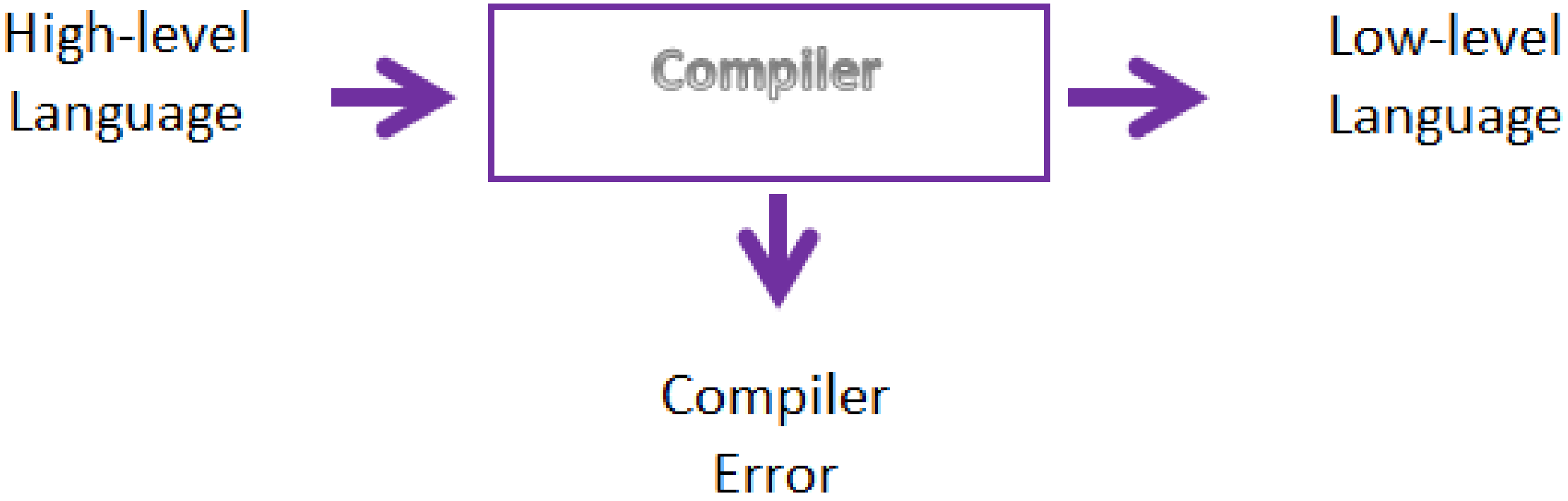
High-level languages are...

- Programmer friendly. They are easy to write, debug and maintain.
- It provides a higher level of abstraction from machine languages.
- It is machine independent language.
- Easy to learn.
- Less error prone, easy to find and debug errors.
- High level programming results in better programming productivity.
- Eg: Java, C, C++, BASIC, Python, Visual Basic, Delphi, Perl, PHP, Ruby

- The programs written in high-level languages need to be converted into computer understandable format.

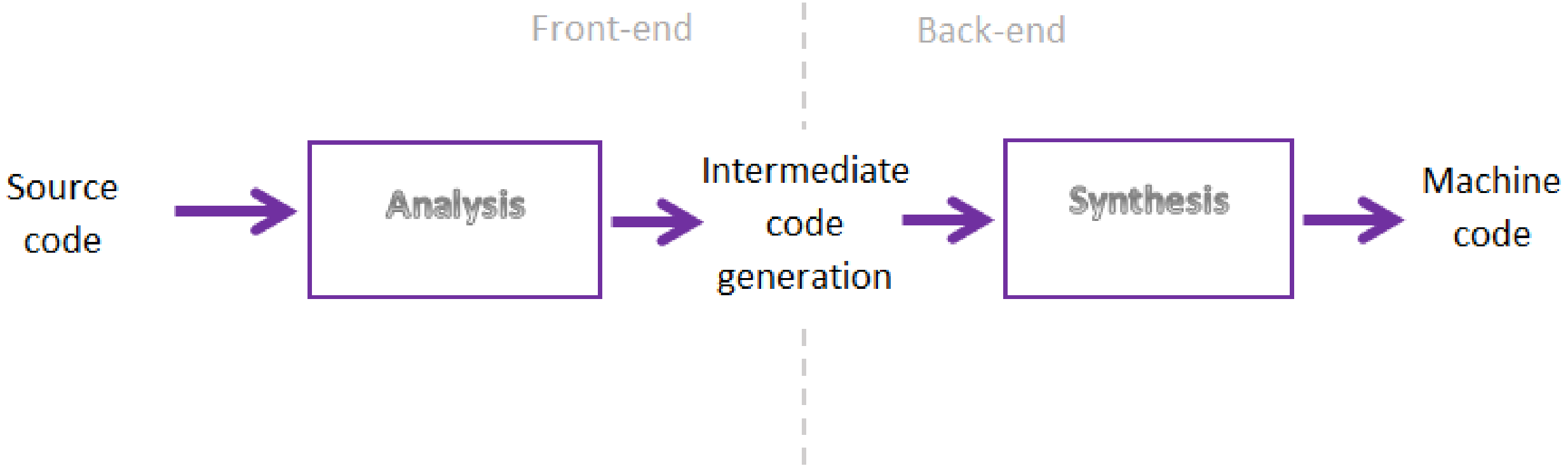
- A computer program that converts programs in a high-level language (source language) into low-level language - (target language either assembly language or machine language)

- Compiler -


High-level
Language
Compiler
Low-level
Language
Compiler
Error

Basically compilers do two things..

- **Analysis**: phase of the compiler that 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
- **Synthesis**: phase that generates the target program with the help of the intermediate source code representation and symbol table that was generated by analysis phase.


Front-end
Back-end
Source code
Analysis
Intermediate code generation
Synthesis
Machine code

Phases of Compiler

Analysis Phase

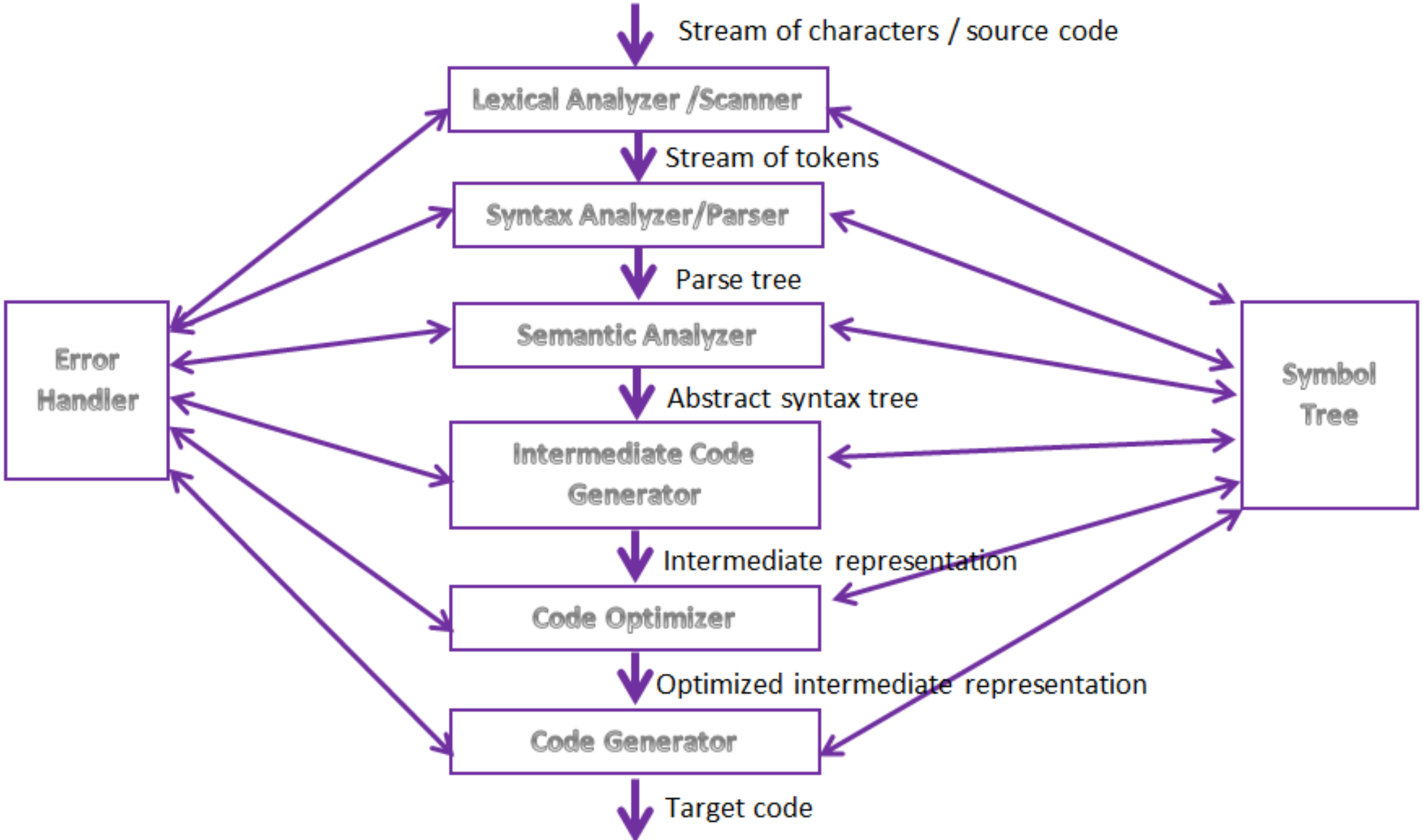
- lexical analyzer / scanner
- syntax analyzer / parser
- semantic analyzer
- intermediate code generator

⇒ Machine Independent

Synthesis Phase

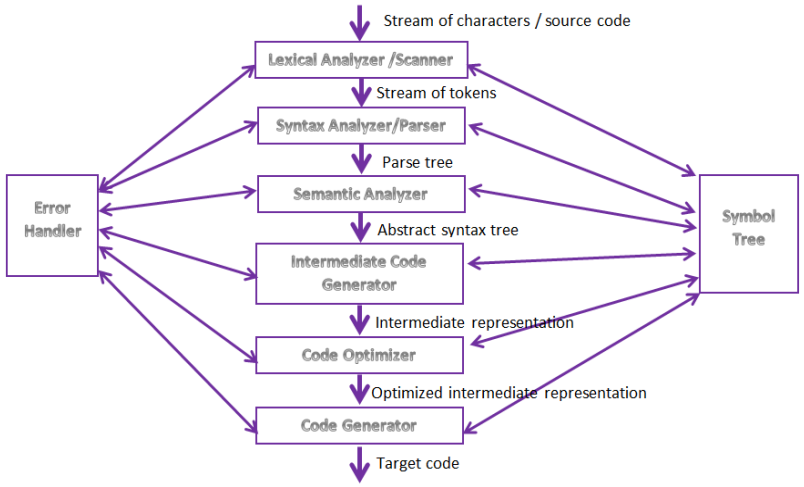
- code optimizer
- code generator

⇒ Machine Dependent


Stream of characters / source code
Lexical Analyzer /Scanner
Stream of tokens
Syntax Analyzer/Parser
Parse tree
Semantic Analyzer
Abstract syntax tree
Intermediate Code Generator
Intermediate representation
Code Optimizer
Optimized intermediate representation
Code Generator
Target code
Error Handler
Symbol Tree

Lexical Analyzer / Scanner

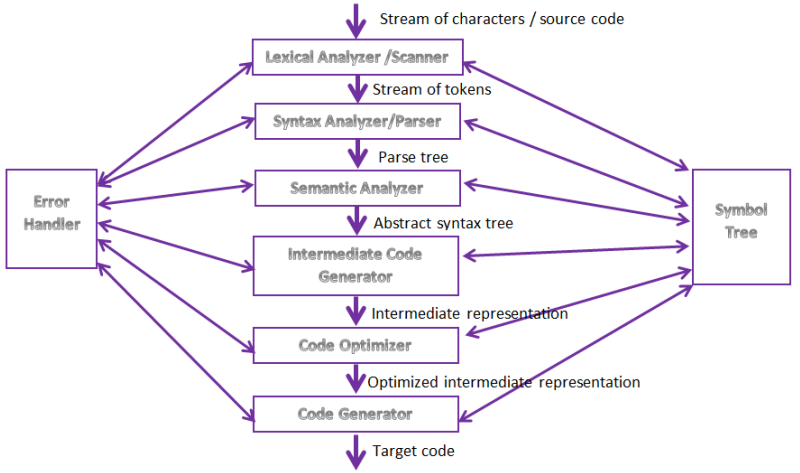
- Initial part of the compiler
- Text is read and divided into tokens into meaningful lexemes (corresponds to a symbol in the programming language - variable, keyword, number)
- These lexemes are represented as **tokens**



Syntax Analyzer / Parser

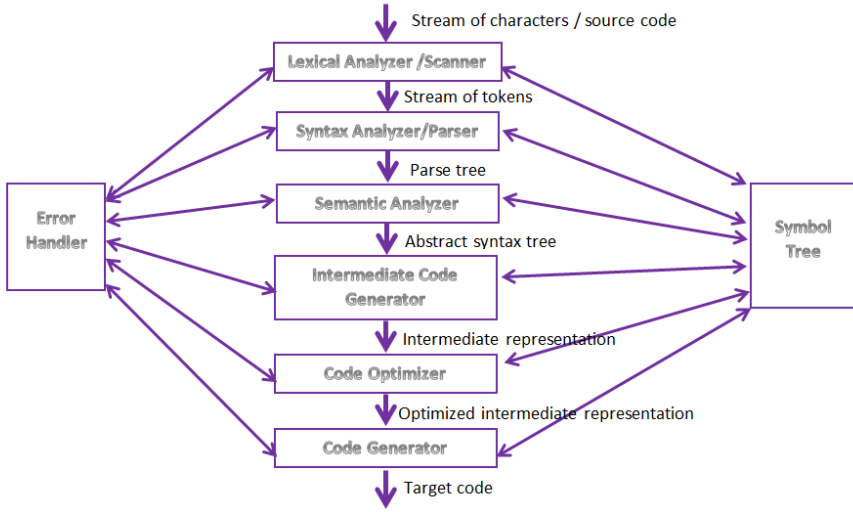
- Takes the list of tokens produced by the lexical analyzer and arranges into a tree structure → parse tree/syntax tree
- Reflects the structure of the program
- Arrangement is checked against the source code grammar

  ⇒ checks if the expressions made by the code is ***syntactically*** correct



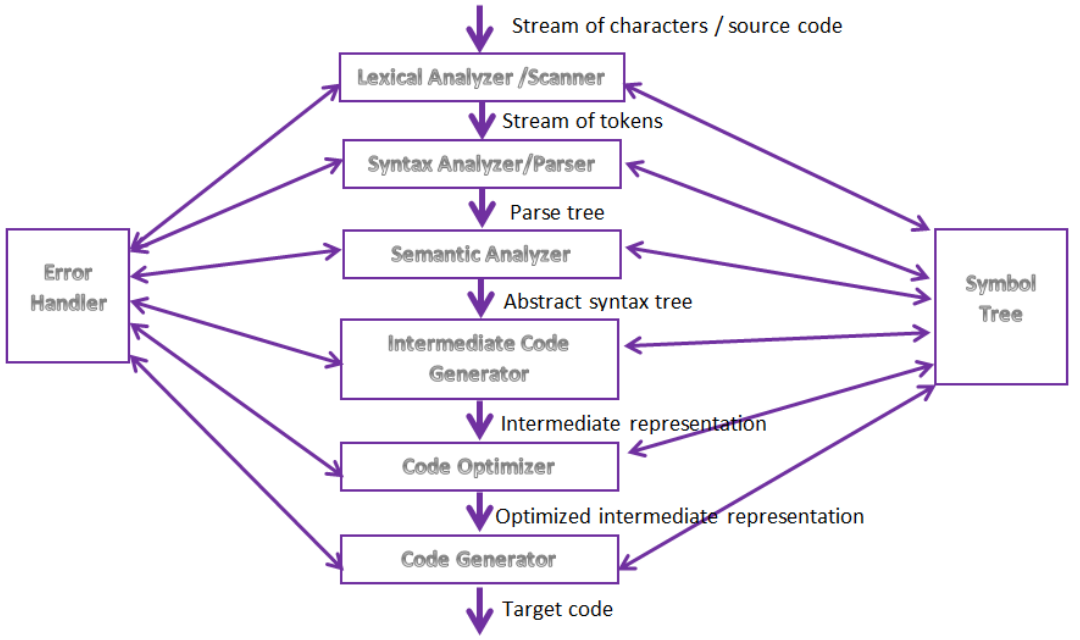
Semantic Analyzer

- Checks if the parse tree constructed follows the rules of the language
- Generates an abstract syntax tree as output
- Checks for:
  - If identifiers are declared beforehand
  - Assignment of values
  - Compatibility of data types



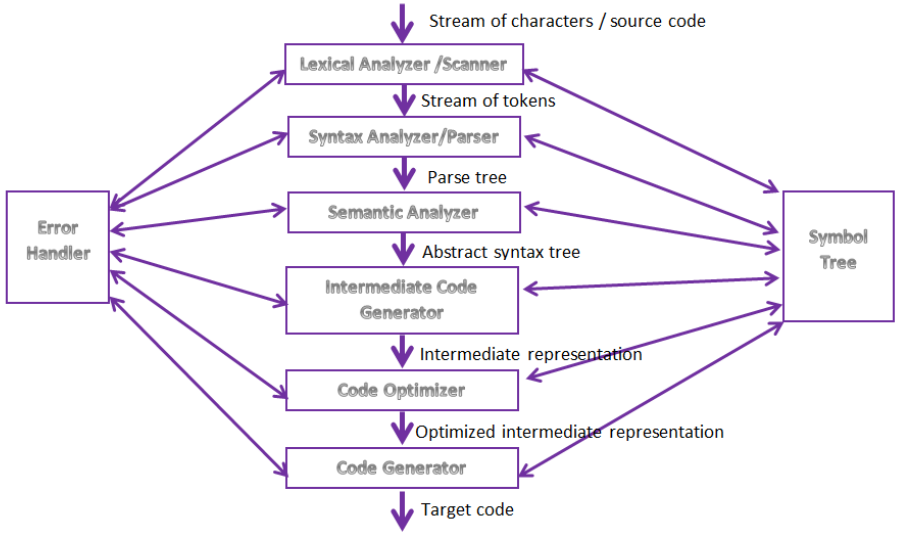
Intermediate Code Generator

- Program is translated to a simple machine independent intermediate language



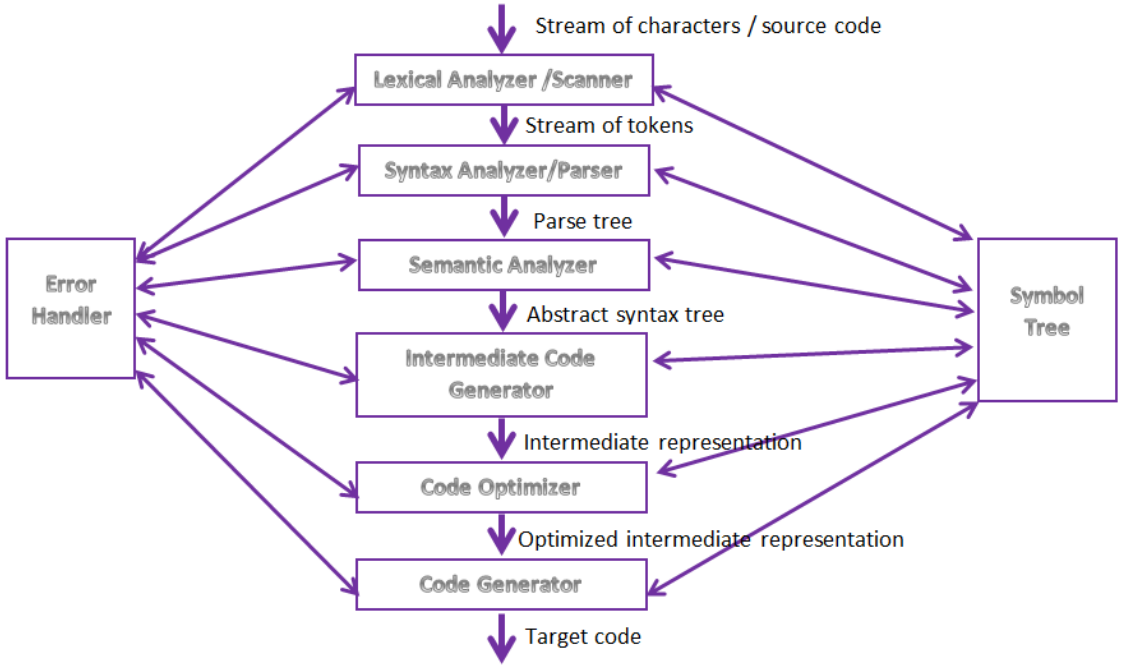
Code Optimizer

- Process that removes unnecessary code lines, and arranges the sequence of statements in order to speed up the program execution without wasting resources (CPU, memory)
  - Remove unused variables, unreachable code
- Optional



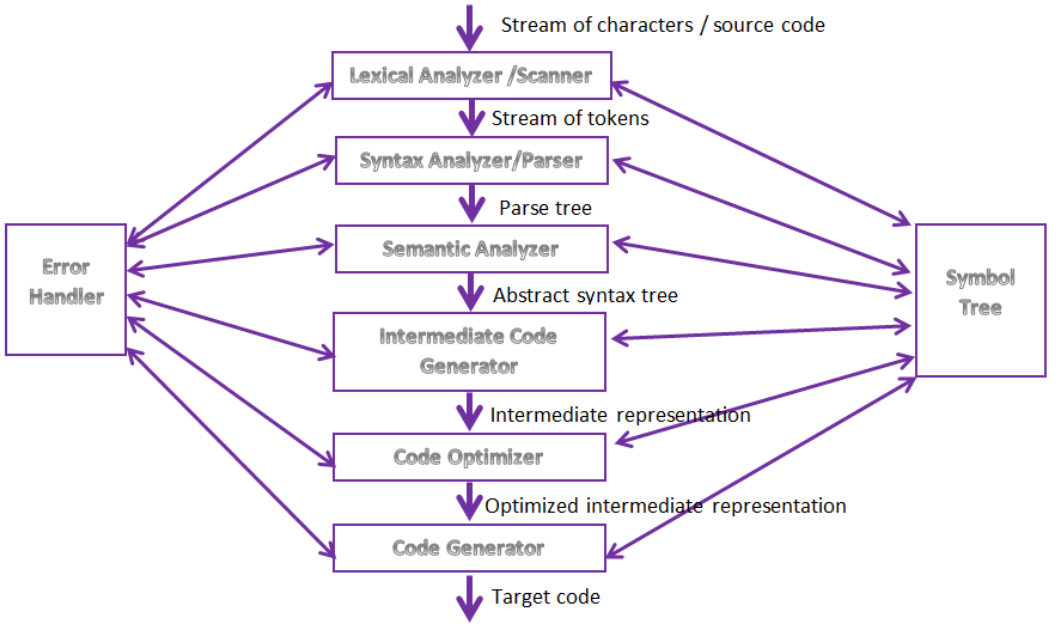
Code Generator

- Maps the optimized intermediate representation to the target language



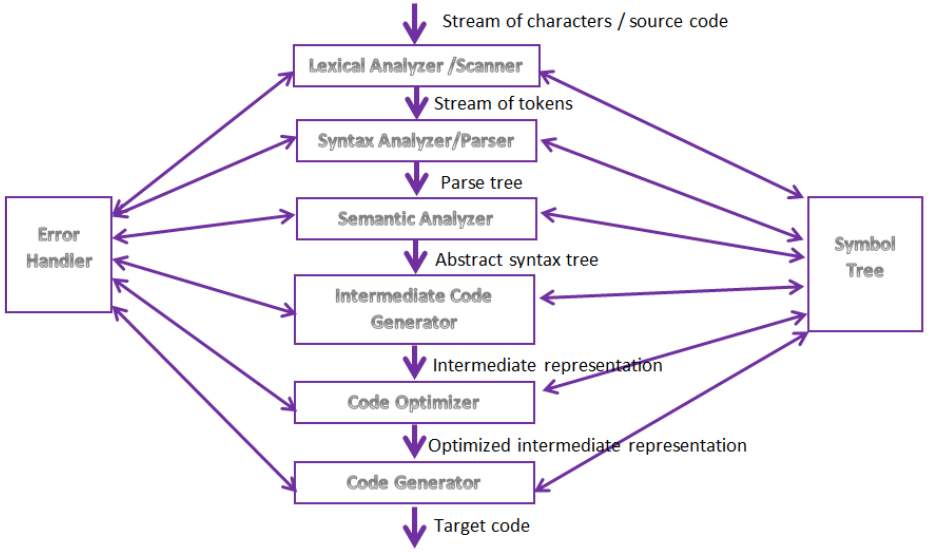
Symbol Tree

- A data structure that stores all the identifier's along with the data types (identified at lexical analysis)
- Used by all the phases of compiler



Error Handler

- Handles error handling and error reporting at each phases
  - invalid character sequence - lexical analysis
  - invalid token sequence - syntax analysis
  - type & scope errors - semantic analysis

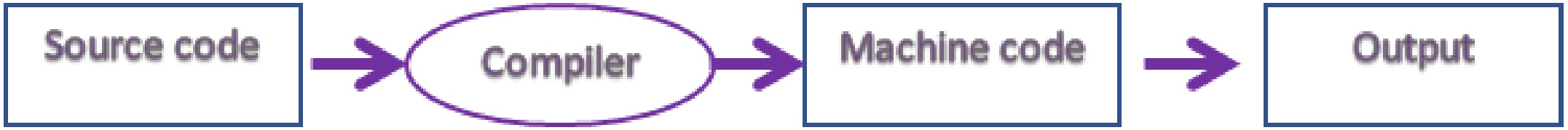


Few other tools work closely with compilers..

- Interpreter
- Decompiler
- Language Translator
- Cross compiler
- Pre-processor
- Assembler
- Linker
- Loader

Interpreter

- Software that converts the high-level language into low-level language line by line



Interpreter vs Compiler..

Compiler	**Interpreter**
Source need to be compiled before execution	The code is interpreted at the time of execution line by line
Execution is comparatively faster	Slower since need to be interpreted while running
Since sees the entire code- better optimization	Since see only line by line - not better optimization
Stores machine code for execution	No machine code is stored
C, C++, C#, CLEO, COBOL,	JavaScript, Perl, Python, BASIC

Decompiler

- A software that converts the program in low-level language to a high-level language
- Reverse operation of a compiler

Language Translator

- A software that converts a program written in one high-level language to another high-level language

  Eg: converts C# to Java

Cross compiler

- A compiler that runs on one platform (CPU, OS) and capable of generating executable code for another platform

Pre-Processor

- A software tool that prepares the input for the compilers
- Normally considered as a part of compiler
- Performs file inclusion, augmentation, macro-processing etc.
  #include, #define

Assembler

- The software that converts the assembly language into machine code

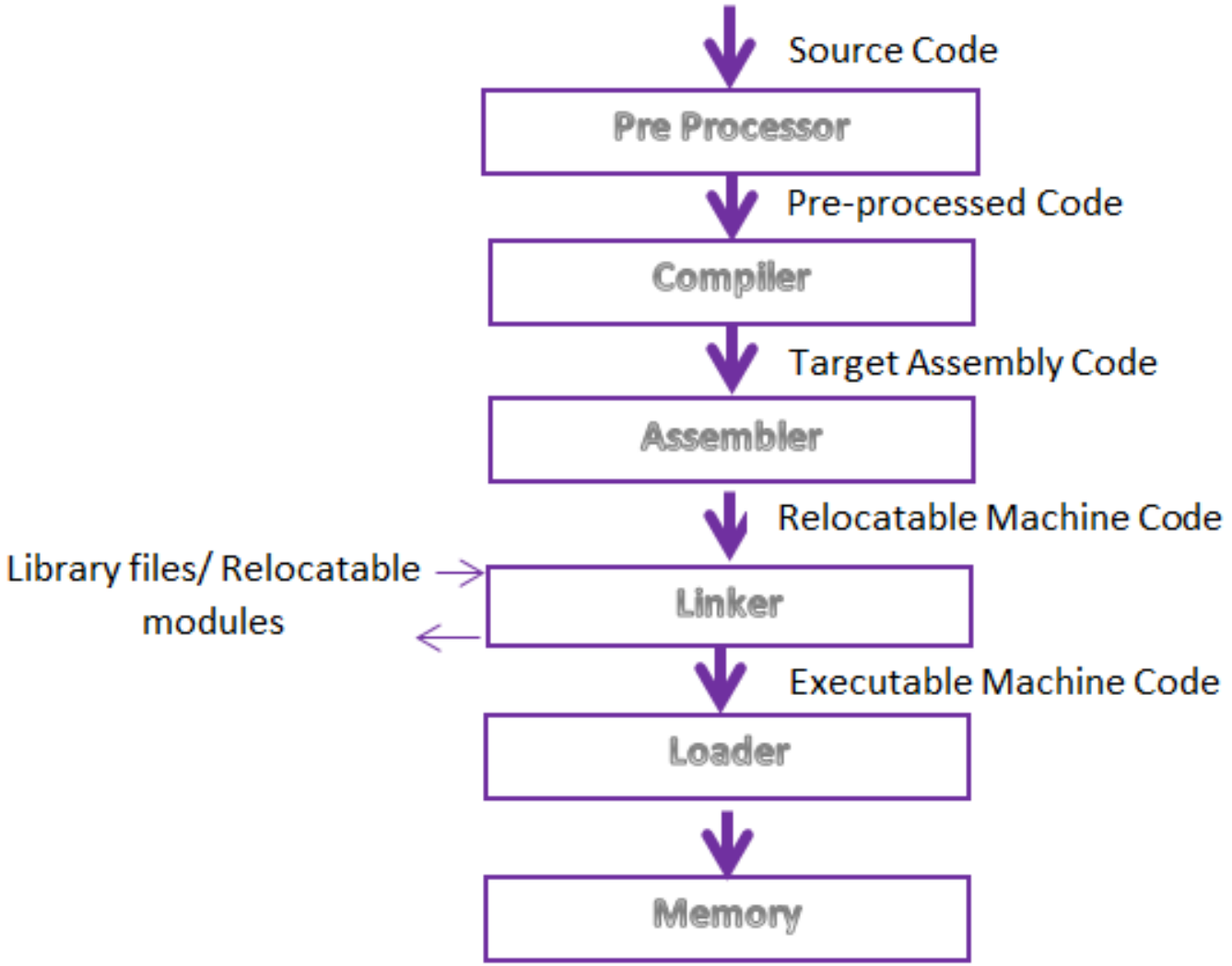
Linker

- A program that combine a variety of object files into a single file to make it a executable program

Loader

- A program that accepts the linked modules as inputs and loads them in the main memory
- Copies modules from secondary memory to the main memory

All in one picture



Summary

- Low-level language
  - Machine code, Assembly language
- High-level language
  - Characteristics, examples
- Compilers
- Phases of compilers
  - Analysis Phase (lexical analyzer / scanner, syntax analyzer / parser, semantic analyzer, intermediate code generator)
  - Synthesis Phase (code optimizer, code generator)
- Tools related to compilers
  - Interpreter, Decompiler, Language Translator, Cross compiler, Pre-processor, Assembler, Linker, Loader

REFERENCE

- J. Stanley Warford, 2016. Computer Systems, Fifth edition. Jones & Bartlett Publishers
- Compiler Design Tutorial https://www.tutorialspoint.com/compiler_design/index.htm
- Compiler Design Basics https://www.slideshare.net/akmrinal/compiler-design-basics

READING

- Compiler Design Tutorial

  https://www.tutorialspoint.com/compiler_design/index.htm

- Compiler Design Basics

  https://www.slideshare.net/akmrinal/compiler-design-basics