**Compiler Design**

Table of Contents

[Introduction 2](#_Toc65536441)

[Compiler 4](#_Toc65536442)

[Assembler 6](#_Toc65536443)

[Linker 7](#_Toc65536444)

## Introduction

The hardware of our computers understand a language which humans cannot. Programs are thus written in high-level language, which is easier for humans to understand and remember. These programs are then fed into a series of tools and OS components that are designed to generate the desired code that can be used by the computer. This is known as the Language Processing System.

* Pre-processing - The pre-processor will run first as it scans the document it will first run into the hash include directives at the top. This directive signals to the pre-processor to go out and find the file in the computer which is known as header file, copy the contents of that file and paste then right directly into the current file, so hash include gets replaced by whatever code found in the header file. After that the pre-processor will continue scan the file for anymore directives. So, pre-processor is responsible for preparing a code for compilation by including and substituting code in the places programmers define.
* Compilation - It is performed by compiler program. This stage is responsible for generating optimized assembly code from source code. The compiler reads through the program source checks for correctness of code and if there are errors, this will cease compilation and spit out the errors. If there are no errors then the compiler proceeds to generate what is known as intermediate representation code, now the IR code is ready to converted into assemble code. In this stage variables are mapped to registers, assignments are translated into corresponding instruction code.
* Machine code generation – Assembly language is translated to machine code.
* Linking stage - The linker will scan through and match up each function declaration and definition so that the program knows what to do at runtime when it runs into a certain function during execution. At the end everything is placed and linked into one executable file which has .exe extension.

## Compiler

TLDR: Compiler compiles

The high-level language is compiled and converted to assembly language (low-level language) using a compiler. A compiler that does the opposite is called a decompiler. Compilation takes place in 4 major steps:

* Scanning – A scanner reads one character at a time from the source code and keeps track of which character is present in which line.
* Lexical Analysis – The compiler converts the sequence of characters that appear in the source code into a series of strings of characters, known as tokens, which are associated by a specific rule by a program called a lexical analyser. A symbol table is used by the lexical analyser to store the words in the source code that correspond to the token generated.
* Syntactic Analysis – Syntax analysis is performed, which involves pre-processing to determine whether the tokens created during lexical analysis are in proper order as per their usage. The correct order of a set of keywords, which can yield a desired result, is called syntax. The compiler has to check the source code to ensure syntactic accuracy.
* Semantic analysis – This step is comprised of several intermediate steps. First, the structure of tokens is checked, along with their order with respect to the grammar in a given language. The meaning of the token structure is interpreted by the parser and analyser to finally generate an intermediate code, called object code. The object code includes instructions that represent the processor action for a corresponding token when encountered in the program. Finally, the entire code is parsed and interpreted to check if any optimizations are possible. Once optimizations can be performed, the appropriate modified tokens are inserted in the object code to generate the final object code, which is saved inside a file.

A compiler reads the whole source code at once and executes the whole program, even if it encounters several errors. By contrast and interpreter, which is similar to a compiler in many ways, reads and executes the source code line by line. If it encounters an error, it stops execution and reports the error.

A compiler that takes the source code of another programming language is called a source-to-source compiler.

## Assembler

TLDR: Assembler assembles

The assembly language is converted into machine-level language using an assembler. An assembler translates assembly language programs into machine code. The output is an object file, which contains machine instructions as well as data required to place these instructions in memory.

An assembler enables software and application developers to access, operate and manage a computer's hardware architecture and components.

An assembler is sometimes referred to as the compiler of assembly language. It also provides the services of an interpreter.

An assembler primarily serves as the bridge between symbolically coded instructions written in assembly language and the computer processor, memory and other computational components. An assembler works by assembling and converting the source code of assembly language into object code or an object file that constitutes a stream of zeros and ones of machine code, which are directly executable by the processor.

Assemblers are classified based on the number of times it takes them to read the source code before translating it; there are both single-pass and multi-pass assemblers. Moreover, some high-end assemblers provide enhanced functionality by enabling the use of control statements, data abstraction services and providing support for object-oriented programming structures.

## Linker

TLDR: Linker links

A linker links and merges various object files, that may even have been created by different assemblers, in order to create an executable file. Linkers are also called link editors. Linking is process of collecting and maintaining piece of code and data into a single file. Linker also links a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for loader.

Linking is performed at both compile time, when the source code is translated into machine code and load time, when the program is loaded into memory by the loader. Linking is performed at the last step in compiling a program.

Linking is of two types:

1. Static Linking – It is performed during the compilation of source program. Linking is performed before execution in static linking. It takes collection of relocatable object file and command-line argument and generate fully linked object file that can be loaded and run.
2. Dynamic linking – Dynamic linking is performed during the run time. This linking is accomplished by placing the name of a shareable library in the executable image. Thus, the libraries don’t have to be inside the .exe file, making it smaller. Load-time linking can be used to load the libraries when they are needed, making the .exe file bigger when it runs. This is called a process. Dynamic libraries can also be used (.dll files). The .exe files are allowed to have instructions that are not executable. These are replaced by links to the .dll files as needed. Since they are not added to the .exe file, size does not change. A hazardous part of using dynamic libraries is that variables that are changed by one program may affect all programs. So dynamic libraries should be designed carefully to make sure core functions are not manipulated.