Complier 1

Lecture 1: Introduction

BY: Assist Prof. Dr. Ielaf O Abdul Majjed

Class code

27kphfp

Programming Languages

Hierarchy of Programming Languages based on increasing machine independence includes the following:

1- Machine languages.

is the lowest form of computer. Each instruction in program is represented by numeric code, and numerical addresses are used throughout the program to refer to memory location in the computers memory.

2- Assembly languages.

is essentially symbolic version of machine level language, each operation code is given a symbolic code such ADD for addition and MULT for multiplication.

3- High level or user oriented languages.

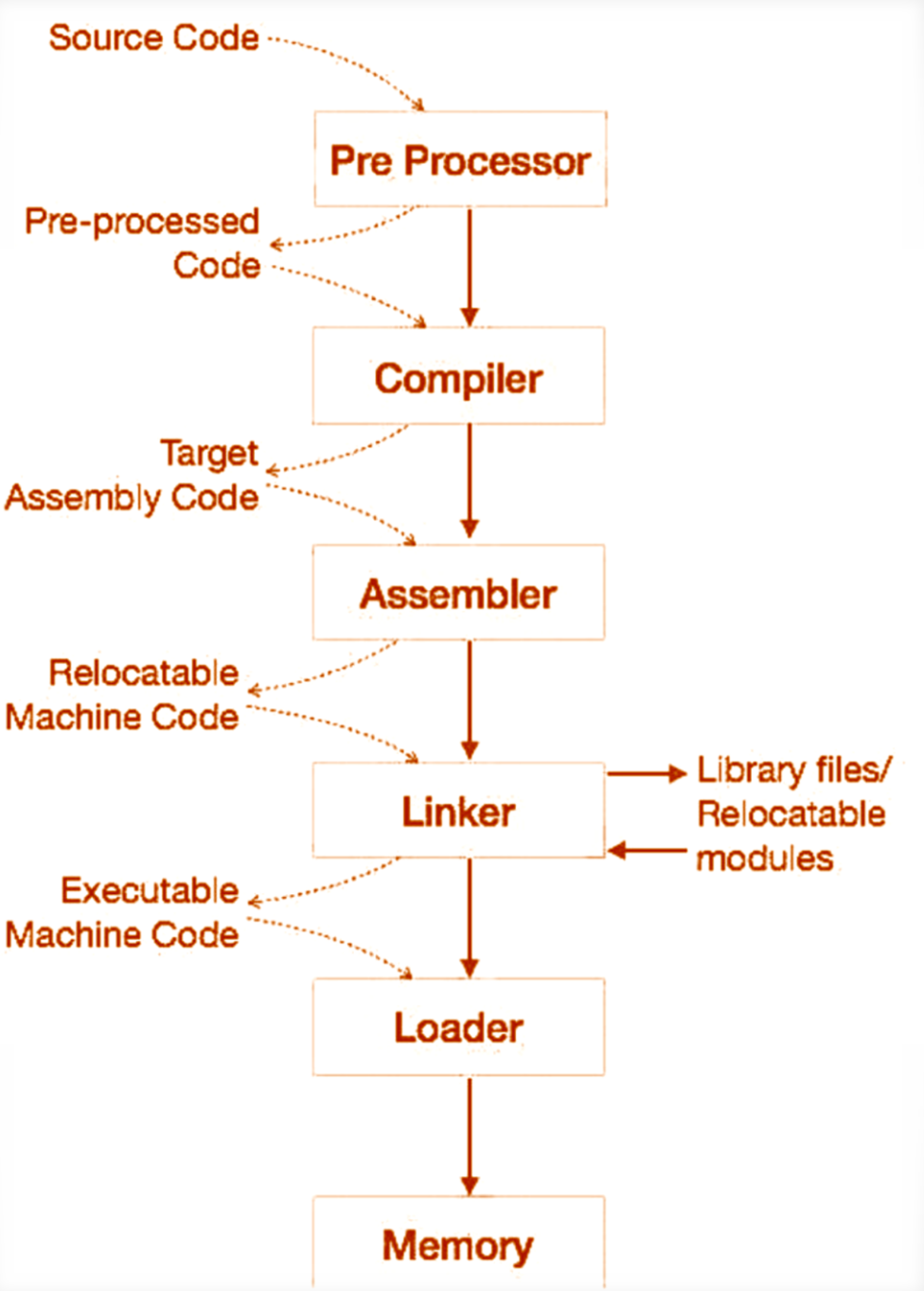
such as Pascal, C.

4- Problem oriented language.

provides for the expression of problems in specific application or problem area .examples of such as languages are SQL for database retrieval application problem oriented language.

Language Processing System

Any computer system is made of hardware and software. The hardware understands a language, which humans cannot understand. So we write programs in high-level language, which is easier for us to understand and remember. These programs are then fed into a series of tools and OS components to get the desired code that can be used by the machine. This is known as Language Processing System.


Source Code
Pre Processor
Pre-processed
Code
Compiler
Target
Assembly Code
Assembler
Relocatable
Machine Code
Linker
Library files/
Relocatable
modules
Executable
Machine Code
Loader
Memory

Let us first understand how a program, using C compiler, is executed on a host machine.

1. User writes a program in C language (High-Level Language).

2. The C ***compiler*** compiles the program and translates it to assembly program (Low-Level Language).

3. An ***Assembler*** then translates the assembly program into machine code (object).

4. A ***Linker*** tool is used to link all the parts of the program together for execution (Executable Machine Code).

5. A ***Loader*** loads all of them into memory and then the program is executed.

Before getting into compilers, we should know a few additional tools.

Interpreter

Similar to a compiler, an interpreter converts high-level language into machine language. The method by which they read the input or source code differs. A **compiler** reads the entire source code at once, generates intermediate code, checks semantics, runs the entire program, and may entail multiple passes. A statement is received from the input by an **interpreter**, which then translates it to intermediate code, executes it, and then reads the next statement in the sequence. An interpreter halts execution and logs errors if they happen. A compiler, however, reads the entire program even if it comes across multiple errors.

Assembler

An assembler converts programs written in assembly language into machine code. The product of an assembler is referred to as an object file, which combines machine instructions with the information needed to store these instructions in memory.

Linker

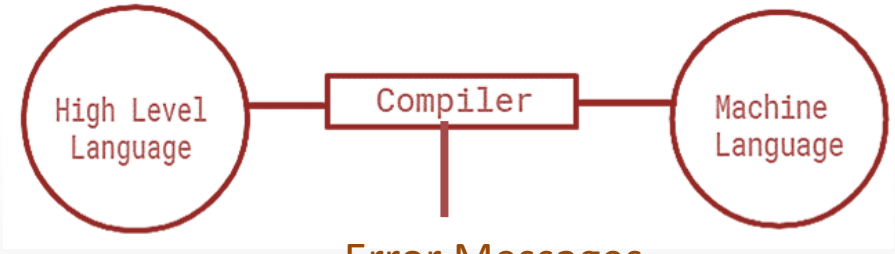
linker is a computer program that links and merges various object files together in order to make an executable file. The major task of a linker is to determine the memory location where these files will be loaded.

Loader

Loader is a part of operating system and is responsible for loading executable files into memory and executes them. It calculates the size of a program (instructions and data) and creates memory space for it.

A ***Compiler*** is a software that typically takes a high level language (Like C++ and Java) code as input and converts the input to a lower level language at once. It lists all the errors if the input code does not follow the rules of its language.

A ***compiler*** translates the code written in one language to some other language without changing the meaning of the program. It is also expected that a compiler should make the target code efficient and optimized in terms of time and space. A program which is input to the compiler is called a **Source program**. This program is now converted to a machine level language by a compiler is known as the **Object code**.



Types of Compilers

Compilers come in different forms, these include

- ***Cross-Compiler***: The compiled program can run on a computer whose CPU or Operating System is different from the one on which the compiler runs.
- ***Bootstrap Compiler*** – The compiler written in the language that it intends to compile.
- ***Decompiler*** : The compiler that translates from a low-level language to a higher level one.
- ***Transcompiler*** : The compiler that translates high level languages.

Phases of a Compiler:

There are two major phases of compilation, which in turn have many parts. Each of them takes input from the output of the previous level and works in a coordinated way.

***1- Analysis Phase*:** An intermediate representation is created from the given source code :

- Lexical Analyzer
- Syntax Analyzer
- Semantic Analyzer
- Intermediate Code Generator

Lexical analyzer divides the program into “tokens”, the Syntax analyzer recognizes “sentences” in the program using the syntax of the language and the Semantic analyzer checks the static semantics of each construct. Intermediate Code Generator generates “abstract” code.

Synthesis Phase: Equivalent target program is created from the intermediate representation. It has two parts :

- Code Optimizer
- Code Generator

Code Optimizer optimizes the abstract code, and the final Code Generator translates abstract intermediate code into specific machine instructions.