COMPSCI 1XC3 - Computer Science Practice and Experience: Development Basics

Topic 3 - Introduction to C

Zheng Zheng

McMaster University

Winter 2023

Adapted from C: How to Program 8th ed., Deitel & Deitel

Thinking in C

The C Programming Language

Other C-Based Languages

Working with C

Hardware Vs Software

- **Hardware** is a collection of physical, electronic components that comprise a computer's physical form.
- **Software** is a series of instructions stored in a computer's memory that may be executed by sometimes arbitrary software systems.
- A processor is a group of circuits that implement operations on memory.
- These operations are known as **instructions** or **hardware instructions**.

Hardware Vs Software (cont.)

Programming languages are more or less abstract, depending on how directly they access a system's underlying hardware.

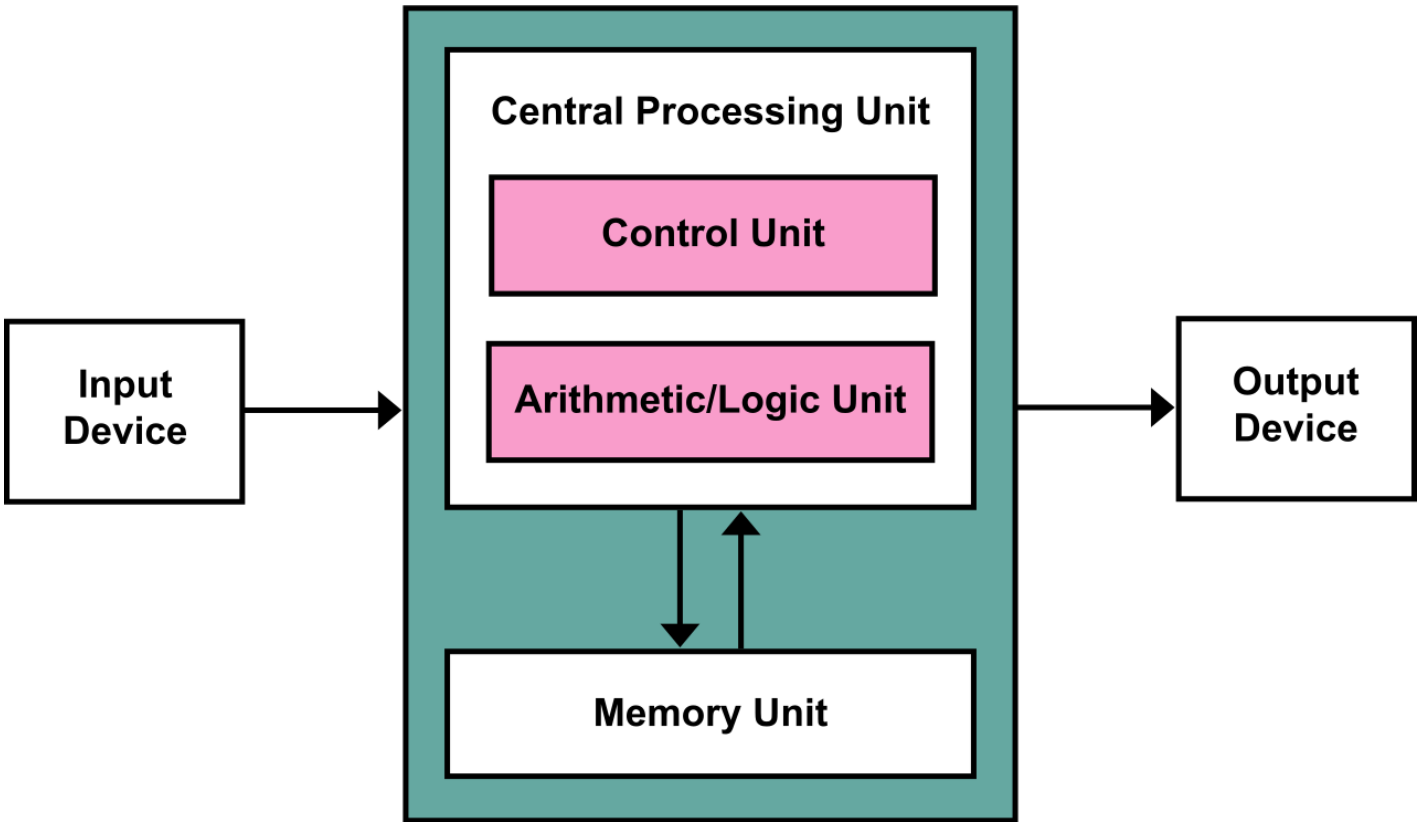
- In **High Level Languages** such as Python and Haskell, an operation may represent many hardware instructions.
- In **Low Level Lanugages** such as C, an operation represents comparatively few hardware instructions.

Different languages are good for different things, and a good developer knows which languages are suited to which applications!

So Why Learn a Low Level Language?

- **Applications!** Anywhere you are programming close to the bare metal, you will probably be programming in C. This includes:
  - Operating Systems
  - Kernels
  - Stuff you'll learn about in *COMPSCI 2GA3 - Computer Architecture*
- **Optimization!** Because they use a small number of hardware instructions per operation, programs written for low level languages can be very small, and run very quickly relative to high level languages. Some optimizations are not possible in high level languages!
- **Knowledge!** An appreciation for what our programs are doing "under the hood" will make us better programmers!

Von Neumann Architecture



The C Programming Language

C evolved from two previous languages, BCPL and B.

- **BCPL** ("Basic Combined Programming Language") was developed in 1967 by Martin Richards as a language for writing operating-systems and compilers.
- Ken Thompson modeled many features of his B language after their counterparts in BCPL, and in 1970 he used B to create early versions of the UNIX operating system at Bell Laboratories.

The C Programming Language cont.

- Dennis Ritchie at Bell Laboratories created the C language as an evolution of B in 1972.
- C initially became widely known as the development language of the UNIX operating system.
- Many of today's leading operating systems are written in C and/or C++.
  - The Windows, Linux, OS X, Android and iOS Kernels are all written mostly in C!
- C is (mostly) hardware independent.
- With careful design, it's possible to write C programs that are portable to most computers.

Applications of C

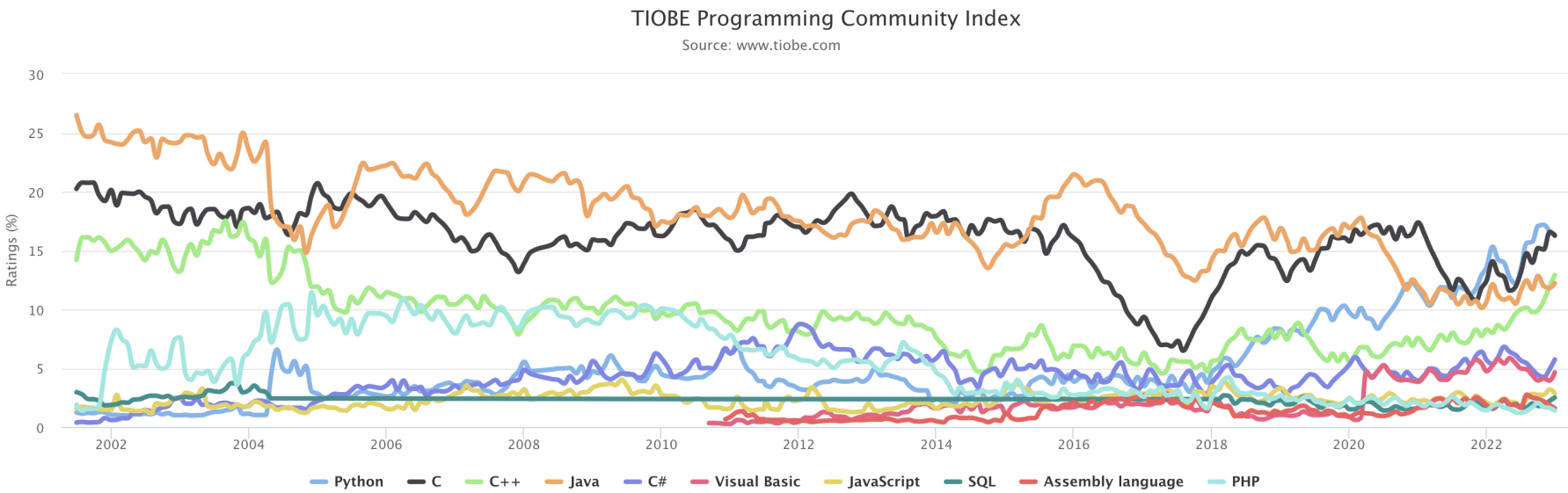
Because of it's high performance characteristics, C is still used a lot, despite being 50 years old!

- **Operating Systems** - Portability across many hardware implementations and overall performance lend C to operating system development.
  - Linux, portions of Windows and Android use C
  - Apple's OS X uses Objective-C, which is derived from C.
- **Embedded Systems** - C is one of the most popular languages for embedded systems development, which are typically highly memory conservative.

Applications of C cont.

- **Real-Time Systems** These "mission critical" applications require very fast response times. A high performance language dramatically increases the feasibility of meeting timing constraints.
- **Communication Systems** Due to the massive quantities of data being routed, optimization becomes crucial.

Popularity of C



As of December 2022, C is the world's second most popular programming language according to the TIOBE index (`https://www.tiobe.com/tiobe-index/`)

Standards and Implementations

The semantics of the C language are set by the International Standards Organisation (ISO) and the International Electrotechnical Commission (IEC), in a series of standard documents

- **C17** - ISO/IEC 9899:2018 (https://www.iso.org/standard/74528.html) is the latest version (June 2018).

There are many C compilers, which are all implementations of the above standard. The following compilers are compliant with the latest version of the standard:

- GCC 12.2 (https://gcc.gnu.org/releases.html)
- LLVM Clang 15.0.7
- IAR EWARM 9.32

The C Standard Library

Because it lacks object oriented structures, the fundamental unit of abstraction in C is the **function**.

- The most commonly used functions are collected into the **C Standard Library**.
- Documentation may be found here: `https://www.gnu.org/software/libc/manual/pdf/libc.pdf`
- Use of library functions is strongly encouraged!
- Library functions (especially from venerable libraries) have had *decades* of optimization and improvement!
- Rule 1: If a library function exists, use it.
- Rule 2: Learn Rule 1 quickly.

C++

C has been extremely influential on the development of many programming languages. Perhaps C++ most obviously.

- C++ was developed by Bjarne Stroustrup at Bell Laboratories.
- It is an iterative improvement on C, crucially adding support for **object-oriented programming** (which C doesn't have!)
- Object Oriented design adds the **object**, a new unit of abstraction that allows the combination of data with functions.
- This increases modularization, and facilitates programming principals which allow very large programms to still be manageable.
- We will not be studying C++ in this course.

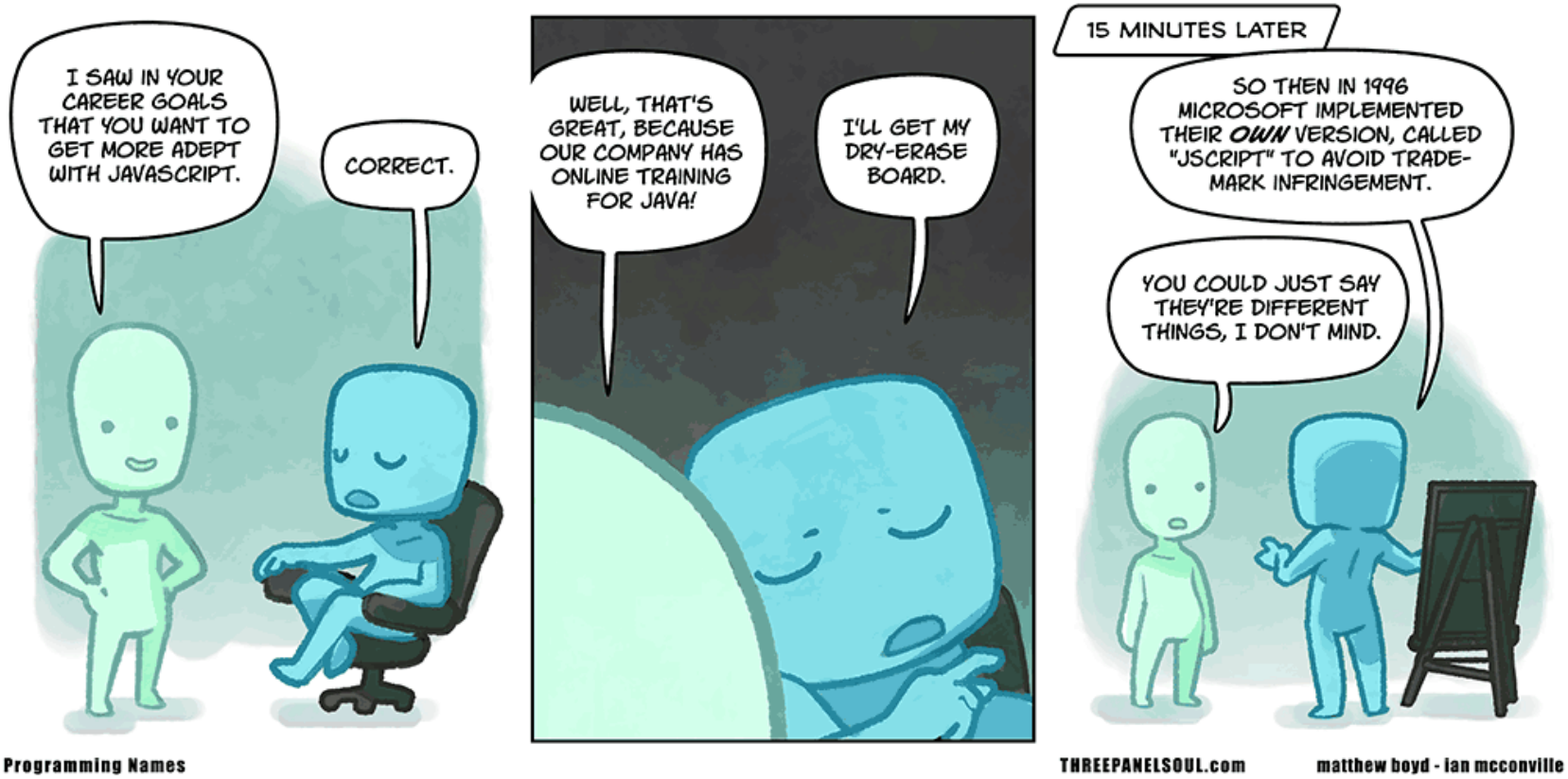
Other C-Based Languages

- **Objective-C** - An object-oriented language developed in the early 80s and eventually acquired by Apple.
- **Java** - A C++ derived language developed by Sun Microsystems in 1991. Uses the "Java Virtual Machine" to extend portability to a massive number of highly diverse systems and architectures. Also, Minecraft.
- **C#** - Microsoft's .net framework integrates internet connectivity into a framework of both Java and C++. Non-Microsoft implementations of C# also exist (such as game object scripting in the Unity game engine).

Other C-Based Languages cont.

- **PHP** - an object-oriented, open source scripting language used primarily in internet, database, and internet database applications.
- **Python** (!) - Released in 1991 and developed by Guido van Rossum, python emphasizes the elimination of superfluous syntactic detail, and has become a very popular language for introductory programming courses.
- **JavaScript** - The most widely used scripting language. Adds dynamic behaviour to web pages.

Java is to JavaScript as Car is to Carpet



So compilers then...

In contrast to the Python we all know and love from 1MD3, C is a **compiled**, rather than an **interpretted** language. The process is as follows:

1. editing
2. preprocessing
3. parsing
4. assembly
5. linking
6. loading
7. executing

Editing a C file

- C files may be edited using any text editor. Common text editors include:
  - Notepad / Notepad++ (Windows)
  - Emacs / Gedit / Vim (Linux)
  - TextEdit (Macintosh)
- Fancier environments (such as Jupyter and VS Code) allow for compilation and execution of C programs within the editor itself.
  - https://www.programiz.com/c-programming/online-compiler/
- C files are given the *.c file extension
- C header files (which we'll get to) have the *.h extension

Invoking the C Compiler

The following process describes compiling a C program from the command line in a Linux-like environment.
Let's examine the following C file:

```
#include<stdio.h>

int main () {
    printf("Hello, World!\n");
    return 0;
}
```

Invoking the C Compiler cont.

To compile this program, we use the following command in bash:

```
[...]$ gcc simple.c -o simple
```

- First, we invoke gcc, the gnu compiler collection. gcc knows we want to interpret the file as a C program because of the file extension.
- Next, we specify the file to be compiled.
- the -o flag allows us to specify the name of the produced executable file.
- The produced file is the original program expressed in machine language (also known as **object code**). Note that this is different from assembly language!

The Compilation Process

Once the compiler is invoked, it goes through a couple stages:

- **Preprocessing** - The purpose here is to make the code ready for parsing and generation.
  - Removing comments
  - Expanding any Macros
  - Expanding any included code ("include" in C is equivalent to "import" in Python)
  - A few other things
- **Parsing** - The code is broken down into **tokens** (also known as tokenization). The tokens are arranged into a hierarchical **Abstract Syntax Tree** (AST).
- **Assembly** - The AST is used to create a series of machine code instructions, which are saved as an object file (*.o)
- **Linking** - The object file is linked up with the relevant libraries, and an executable is produced.

The Compilation Process cont.

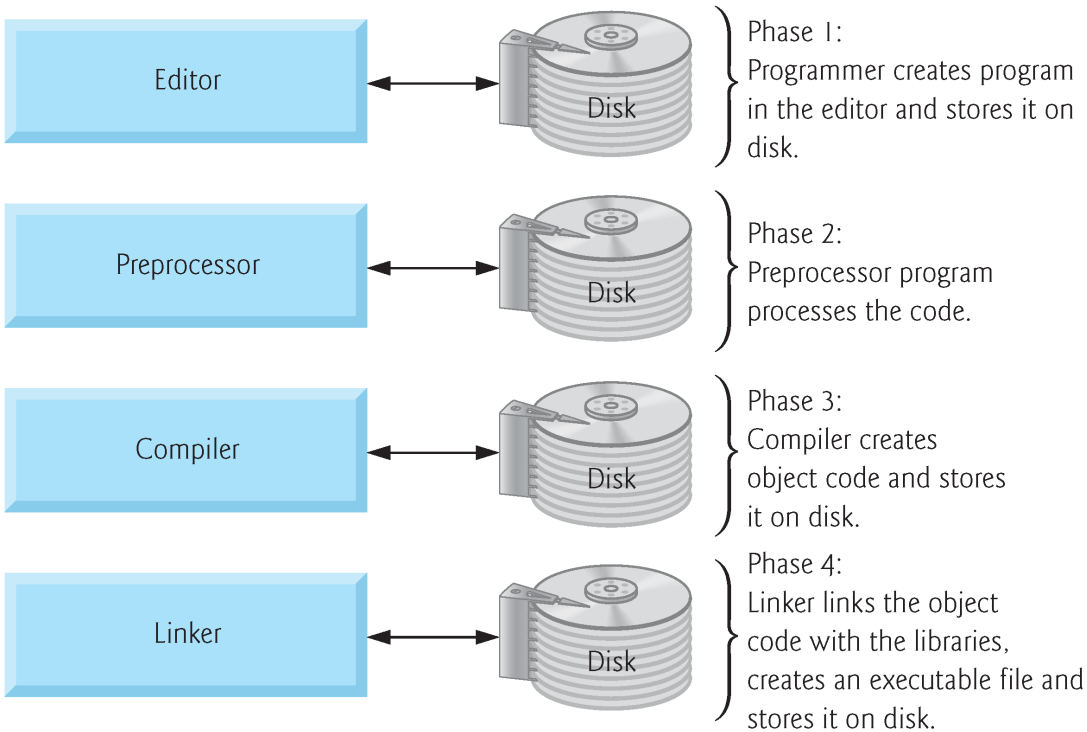


Fig. 1.6 | Typical C development environment. (Part 1 of 2.)

Compiler Complaints!

Your invokation of gcc may not end successfully, if your code has bugs in it!

- **Syntax Errors** occur during parsing, if the code can not be parsed correctly
  - For example, forgetting a semicolon causes a Syntax Error
- **Compiler Warnings** do not halt execution of the compiler, but they can indicate other problems with your code.
  - Some warnings are not shown by default, but the `-Wall` (Warnings: All) flag tells gcc you want to see them.
  - If you don't want to see any warnings (not recommended...), use the `-w` flag.
  - You may be warned about:
    - Using data types and pointers incorrectly
    - Not using variables that have been declared
    - Using = instead of ==

Executing the Executable

In a Linux-like environment, an executable is run using the following command:

```
[...]$ ./simple
```

- **Loading** - The compiled C program is loaded into the system's primary memory (usually the RAM)
- **Execution** - The CPU runs the program, starting with the first instruction, and proceeding until the program terminates.

Executing the Executable cont.

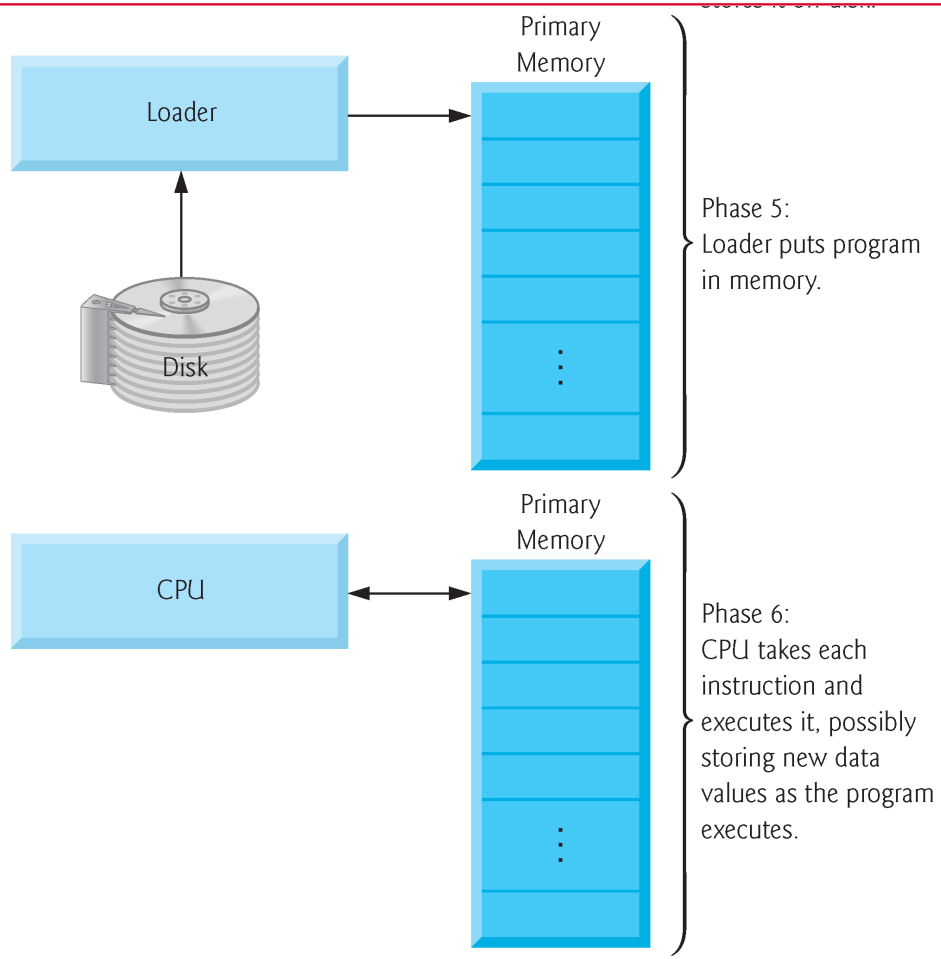


Fig. 1.6 | Typical C development environment. (Part 2 of 2.)

When Runtime isn't Funtime

Often, your code will contain bugs, despite being compiled and linked successfully. These are known as **semantic** errors.

- Some semantic errors will just straight up crash your program. These are known as **fatal errors**.
  - Dividing by Zero!
  - Trying to access memory that doesn't belong to you! (The dreaded Segfault!)
- Others just cause a mismatch between the expected output of a program and it's actual output.
  - It is important to know what the expected result of a program is for specific inputs.
  - Running a program with specific inputs and looking for a known "correct" output is known as **testing**.

Runtime Interactions

If we wish to interact with a C program using a monitor and keyboard, that C program needs to interact with the following:

- `stdin` - (standard input stream) a place in your computer where keystrokes are logged for retrieval by programs.
- `stdout` - (standard output stream) a place which collects things programs wish to print to the screen.
- `stderr` - (standard error stream) similar to stdout, but reserved specifically for error messages.

All three of these streams are either **emulated** or are connected in a more complex manner in environments such as VS Code, but are used directly in bash-style environments.

Acknowledge

The contents of these slides were liberally borrowed (with permission) from slides from the Winter 2021 offering of 1XC3 (by Dr. Nicholas Moore).