SFWRENG 2MP3 - Programming for Mechatronics

# Topic 1 - Introduction

NCC Moore

McMaster University

Fall 2020

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





Hardware Vs Software

Data Representations

The C Programming Language

The C Standard Library

Other C-Based Languages

C Development





## 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!

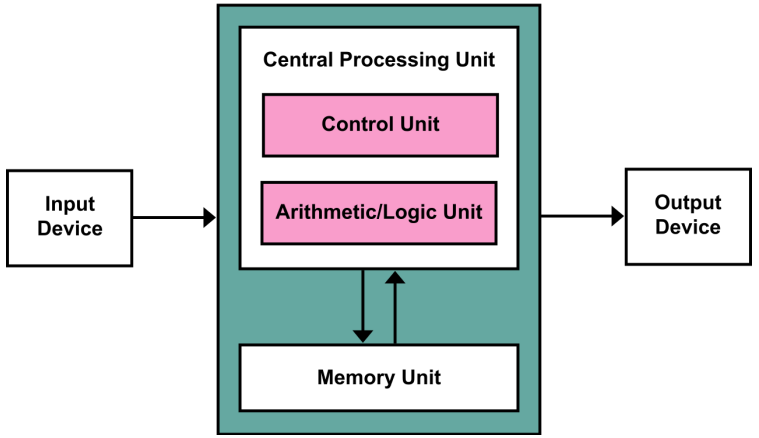
## So Why Learn a Low Level Language?

- **Applications!** Some non-traditional programming environments (such as microcontrollers) do not support high level languages!
- **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








## So Memory Then

Computer memory can be stored in many ways

- **ROM** - "Read-Only Memory" is your system's permanent data storage. The files in your file system are stored in ROM. Physically, this is normally a magnetic hard drive, solid-state drive (SSD) or Flash memory (USB drives).
- **RAM** - "Random Access Memory" is your system's working memory. On execution, programs are loaded from ROM into RAM.
- **Registers** - Memory storage internal to the CPU. Data must be loaded into registers in order for the CPU to perform operations on it.





## Data Collections

The data we access in system memory can be categorized heirarchically.

- **Bit** - Bits are the smallest. A bit is either 1 or zero, and is the atomic unit of memory.
- **Byte** - A byte is made of 8 bits.
- **Word** - The size of a word varies by CPU. It is the number of bits the CPU's instruction set operates on in a single operation.
  - The NES was an 8 bit machine, so words and bytes were the same size.
  - The Nintendo 64 was a 64 bit machine, so a word was 8 bytes.

## Data Collections cont.

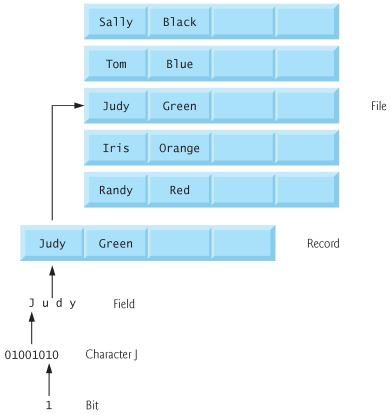



**Fig. 1.2** | Data hierarchy.





## Data Collections cont.

- **Characters**
  - A computer's **character set** is used to write programs and represent data.
  - C supports various character sets, including ASCII, UTF-8, and UTF-16
- **Fields** - Composed of characters, or other data types. Associates data with a context in order to establish meaning.
- **Records** - Several related fields may be composed into a record.





## Data Collections cont.

- **File** - A file contains an arbitrary amount of data in an arbitrary format.
  - In some operating systems, a file is viewed simply as a sequence of bytes—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer.
- **Database** - a collection of data organized for easy access and manipulation.
  - The most popular model is the relational database, in which data is stored in simple **tables**.
  - A table contains a number of records, which contain a number of fields.





## 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 software 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.

- The C language was evolved from B by Dennis Ritchie at Bell Laboratories and was originally implemented 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++.
- 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.





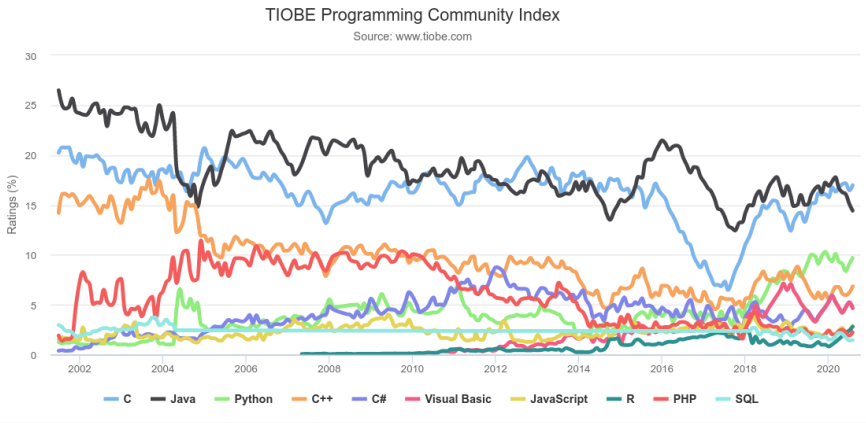
## Applications of C cont.

- **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



## 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

- **C18** - 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 8.1.0
- LLVM Clang 7.0.0
- IAR EWARM v8.40.1





## The C Standard Library

Like many other languages, the 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++ 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.





## 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).

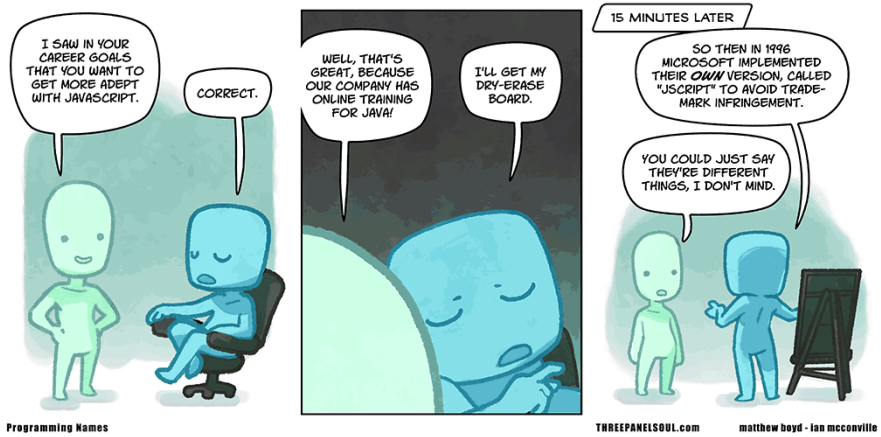
## Other C-Based Languages cont.

- **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 1D04, 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.
- 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" is C for Python's "import")
  - 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.

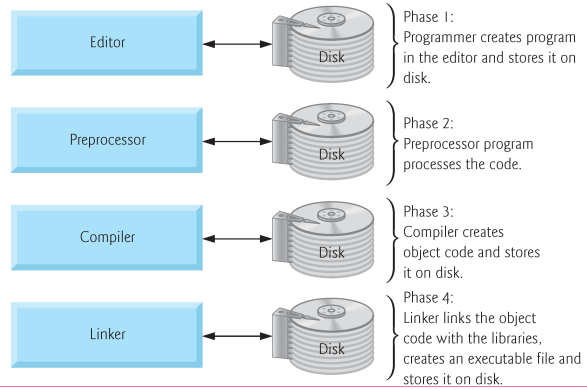



**Fig. 1.6** | Typical C development environment. (Part 1 of 2.)

## 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 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:

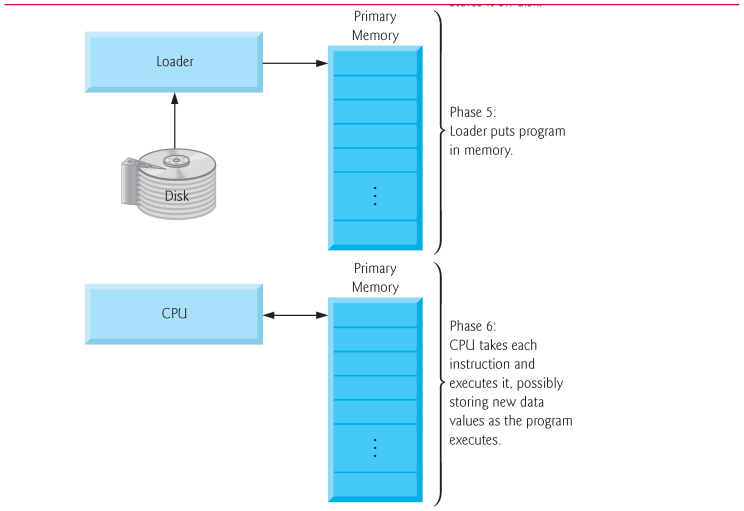
```
[...]$ ./simple
```

- **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.








## 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 Jupyter and VS Code.





## The Last Slide Comic