C: A Beginner's Guide

Joshua Weinstein

2018

Contents

0.1 Preface

This book is intended to serve as a tool to learn how to program in the C programming language. The book is targeted at those who have little to no experience in programming or computer science as a whole. Even if someone hasn't learned any prior programming languages, this book can serve as a first learning experience. This book discusses in depth the low level components of the C language, like memory, addresses, the stack, the heap, and more. The book introduces programming and C from the ground up, simplifying concepts often claimed as difficult or complex.

This books also introduces computer science concepts, like functions, types, data structures and algorithms, in the context of the C language. The purpose of this is to teach the fundamentals of programming from a simple, backbone starting point, and work up to more ”magic” levels of abstraction, such as those seen in the Python language. This approach exposes students to a mechanical and complex understanding of programming. Such insight is exceptionally useful when optimizing and writing high performance software.

The book pays careful attention to the features of C which are often described as confusing, such as the *void* pointer, the functions *malloc* and *free*, and much more.

Chapter 1

Introduction

C is a low-level programming language designed for high portability and fast performance. C first originated in the late 1970's when Brian Kernighan and Dennis Ritchie wrote the book, *The C Programming Language.* C has tremendously influenced the world of programming, leading to the development of C++, an object oriented version of C, and the syntax of many other languages, like Java. C is still widely used today, in many applications such as databases, kernels, drivers, and much more. One of the most popular programming languages, Python, runs on an interpreter written in C.

This chapter will introduce the very fundamental ideas behind programming, and work as a primer for learning actual C concepts. Ideas such as addressing, memory, the stack, the heap, variables, printing, files will be discussed with friendly examples. If you have no or little experience in programming, it is highly recommended you start with this chapter. Otherwise, skip ahead to the next chapter.

1.1 Programs and Computing

In the most basic sense, a computer is a machine which can complete tasks by running instructions commonly known as "code". A computer cannot read or understand human languages, like English. A computer only understands instructions in a format known as *binary* code. Binary code is composed entirely of one's and zeros. 00000001 is an example of binary, which actually translates to the number, 1.

A programming language is a human readable language which is *compiled* into binary code. Such languages serve as a intermediate way of communication between a computer's mechanics and a human mind's mechanics. These languages allow humans to instruct computers to perform almost any task imaginable. A program is a file or series of files of code written in a particular programming language. Web browsers, mobile phone applications, websites, are all programs. All of them are in some way compiled into binary code, which is then executed by a computer as instructions. Programs that need high performance are often written in C or C++, a tightly related language to C.

$$\begin{tikzcd}
Human\,Language \arrow[d] \\
Programming\,Language \arrow[d] \\
Binary\,Language
\end{tikzcd}$$

Figure 1.1: The layers of communication between humans and computers

The format in which humans convey information through language is very different than how computers convey information. Programming languages are not at all similar to learning a foreign language. The *grammar* of programming languages is far more restricted than that of a spoken language. When two people speak to one another, they are conveying information that is often *declarative* in nature, where information is stated in some format as true. "I cook food," is a declarative sentence, it states data or information, it does not detail commands or instructions to follow.

Imperative programming involves the communication and passing of information as commands and instructions. C is an imperative programming language, so is Java, and C++. Such languages focus on grouping commands into executable sets of instructions to compute larger more complex values. In many imperative languages, groupings of commands are called *functions.* In the most basic sense, functions apply commands on some specified set of values.

A function is a special set of commands written in code, with some specified input values and output values. Input values to functions are usually called parameters. Functions are used in programming languages to tell the computer to perform specific instructions onto some set of parameters. They generally take the form of:

$$function : name(input) \rightarrow output$$

The input can be on or more values, so can the output. Functions will be explored more in later chapters.

Definitions

1. **Declarative programming**: A style of programming where data is stated and declared.

2. **Imperative programming**: A style of programming that uses commands and instructions that compute data.

3. **Functions**: A group of commands executed on some specified set of values.

1.2 Memory

When a person performs day to day tasks, like driving to work, making meals, or writing letters, they use their memory of past events. Even though the brain is not a computer, it still records data from events we experience which we think about or remember at some later time. When a computer executes binary code or compiled code from a programming language, it needs to use memory. If humans had no way of remembering or recalling anything in the past, they could never solve any problem, mathematical or otherwise. Yet, a computer's memory is not truly chronological. Consider the following example:

Tom needs to buy groceries to make a soup. He knows he needs to buy tomatoes and corn. He makes a list, specifying one tomato and two cobs of corn. He goes tot he grocery store and checks his list, to buy the items he wanted.

In this example, *Tom* needs to buy some amount of vegetables, so he uses a list to check which items he should buy. If a computer were to purchase or order groceries, it would also need a list of what items to purchase. In the previous section, we discussed functions, an arrangement of instructions that are executed on some input values. Lets look at an abstract example.

$$function : BuyGroceries(list) \rightarrow PurchasedItems$$

Here, a function is taking in an input of a list, and outputting the items the list stated to be purchased. The output is related to what the list holds in terms of data. More specifically:

$$\begin{tikzcd}
List \arrow[r] \arrow[d, dash] & PurchasedItems \arrow[d, dash] \\
input & output
\end{tikzcd}$$

To keep track of the list, when a computer is executing commands, it needs to store that list. Secondly, it needs a place to output those purchased items to. To store such information, for a simple function as shown or many other tasks, a computer uses memory.

1.2.1 What is memory?

Memory is a physical component inside a computer that stores temporary data and information necessary for a computer to complete tasks. When a program receives an input, such as a mouse click on a button, or message of text, it needs to have a place to store that input, and any other values it needs to compute the desired output. To start with, we can think of memory as a finite number of boxes, each which can store a value. Such boxes allow us to reference that value in order to change it or create new values. Here is an example:

Example:

$$A : [0]\ B : [0]\ C : [0]$$

Let A, B, and C be boxes of memory. Initially, they all hold the value 0. A memory box must always hold some value. Next, let us inset or load numbers into boxes B and C.

$$\begin{tikzcd}
 & 2 \arrow[d, "load"] & 3 \arrow[d, "load"] \\
A : [0] & B : [2] & C : [3]
\end{tikzcd}$$

Memory box B and C can be added together, with the result of the mathematical operation being stored in box A.

$$\begin{tikzcd}
B \arrow[dr] & & C \arrow[dl] \\
 & A : [5] &
\end{tikzcd}$$

The final state of each memory box is :

$$A : [5]\ B : [2]\ C : [3]$$

In this example, at every step, the memory of what value A, B, or C holds is persisted. Those values are stored and changed as the commands continue to be executed. This is an example of how memory in registers also works, a topic that will be covered later on.

Types of memory

In a computer, there are a few different types of memory. They are used for different tasks, and different roles. Some are also slower than others.

1. **Registers**: This is the fastest, most highly accessible form of memory, it's commonly use in executing instructions.

2. **RAM**: Also known as simply memory or local memory, this is memory used to store temporary values during the time a program is running. In most programming languages assignment statements, such as $a = 3$ are using local memory.

3. **Disk**: This is where a computer stores files, the least accessible, yet longest lasting form of memory.

Memory is measured in units. The smallest, accessible unit of memory is called a byte. Computers and files often measure data in *kilobytes* or *megabytes*. These mean a thousand bytes and a million bytes respectively. The number system behind memory and data will be discussed in the next section.

1.3 Numbers

A computer and it's programs use numbers, just as humans encounter and use numbers in their daily lives. Phone numbers, street addresses, freeway exits, numbers exist all around us. Numbers are the core component of any data inside a computer. Computers can only understand sequences of numbers, they don't see text or images like living beings can. In a previous section, we discussed *binary* language in reference to communicating with a computer. Binary language is just a composed sequence of ones and zeros. This section will provide insight into the system and types of numbers used in programming, such as binary numbers.

Numbers are quantifiable, distinct values that exist in different sizes, composed of digits. The size of a number is determined by it's radix.