Programming Fundamentals

Muhammad Ateeq

[Updated: 13-02-2023]

Introductory Concepts

Note: The terms “app”, “application”, “software”, “program”, “code”, and “instructions” are used interchangeably and mean the same thing.

What is the difference between simple calculator and computer?

A simple calculator is not programmable, meaning that it can only perform basic arithmetic operations which are directly implemented in hardware, and does not allow the user to write custom programs or algorithms.

On the other hand, a computer is fully programmable and can run software applications and custom programs written in a variety of programming languages. This means the hardware in computers is general purpose and can run different instructions as against the calculators.

In summary, the main difference between a simple calculator and a computer in terms of programmability is that a calculator is not programmable, while a computer is fully programmable and can run custom **programs stored in memory** and software applications.

What is stored-program concept?

The stored-program concept is a key idea in computer science that states that a computer's program and data can be stored in its memory and processed by the central processing unit (CPU). This concept was first proposed by mathematician and computer scientist John **von Neumann** in 1945.

Before the stored-program concept, early computers were built with hard-wired programs (i.e., the operations or features were directly implemented in hardware), meaning that the instructions for each operation were physically wired into the machine.

In summary, the stored-program concept is the idea that a computer's program and data can be stored in its memory and processed by the central processing unit, allowing for greater versatility and flexibility in the use of the computer.

What is von Neumann architecture?

The Von Neumann architecture, also known as the Von Neumann model, is a computer architecture that is based on the **stored-program** concept.

The Von Neumann architecture consists of four main components:

Central Processing Unit (CPU): This is the heart of the computer and performs all the calculations and data processing.

Memory: This is where the computer stores data, instructions, and intermediate results. This memory is commonly known as Random Access Memory (RAM).

Input Devices: These allow the user to enter data into the computer, such as a keyboard, mouse, or touchpad.

Output Devices: These display the results of the computer's processing, such as a screen, speakers, or printer.

In the Von Neumann architecture, the CPU reads **instructions and data** from memory, processes the data, and stores the results back in memory. This architecture is the foundation of most modern computers and has remained largely unchanged since its inception. A view of this architecture is shown in Figure 1.

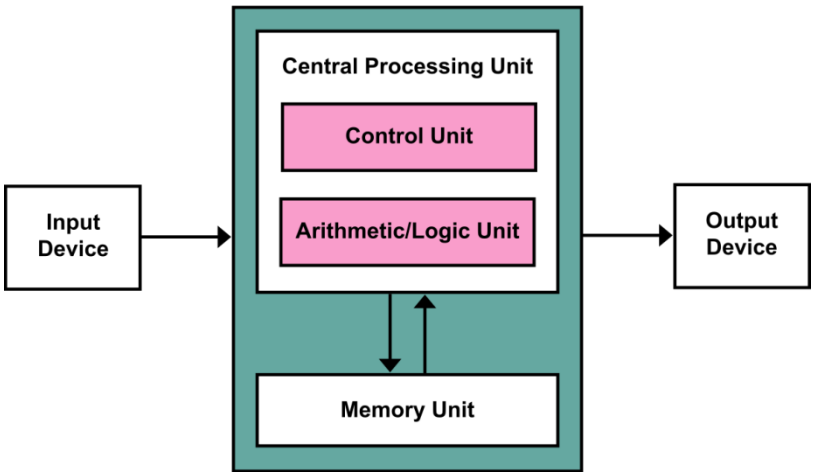


Figure 1. von Neumann architecture.

What is programming?
Programming is the process of designing, writing, testing, debugging, and maintaining the computer **programs**. The goal of programming is to create software that can automate tasks, solve problems, or process data for a wide range of applications.

Who writes the program?
We the computer/data scientists write the programs using some **programming language**.

What is programming language?
Writing programs is as if you are talking to computers. This needs a **language** that we can use to talk to computers. A programming language is a formal language that specifies a set of instructions to create software, applications, and systems that can run on computing devices.

What language do computers understand?
Computers are digital machines and understand only **0's and 1's**. The language based on 0's and 1's is known as **binary language or machine language**.

Can we, humans write big and complex programs using binary/machine language?
May be for very small or few specific programs, the answer can be yes. Writing programs using machine language is a **difficult and time-consuming** process, as it requires a deep understanding of the underlying computer architecture. It is also very **error-prone**, as even a small mistake in the code can cause the program to fail or produce incorrect results. Therefore, it is almost impossible for humans to use binary/machine language to write complex programs. We prefer to use the symbols and characters that we commonly use like **alphabets, and numbers**, etc.

How do we represent alphabets and numbers in computers while they understand only 0's and 1's?
Computers only understand **binary code**, or sequences of 0s and 1s. To represent alphabets and numbers in binary code, standard coding systems are used, such as **ASCII** (American Standard Code for Information Interchange) and Unicode.

ASCII is a **7-bit** coding system that assigns a unique binary code to each of the 128 (2^7) characters in the basic ASCII character set, which includes the letters of the alphabet, numbers, punctuation marks, and control characters.
Some examples are:

a is represented by 01100001 → **8 bits not 7?**
b is represented by 01100010
c is represented by 01100011
A is represented by 01000001

Unicode is a larger, **16-bit** or 32-bit coding system that assigns unique binary codes to over 100,000 characters from many different scripts and symbol sets, including not just the basic ASCII character set, but also characters from different alphabets and scripts used around the world.

In summary, to represent alphabets and numbers in binary code, standard coding systems such as ASCII and Unicode are used, which assign unique binary codes to each character, allowing computers to store and process this data.

What is a bit? What is a byte?
A **bit** (short for "**binary digit**") is the smallest unit of information that a computer can store, process, and transmit. It is a binary value that can represent either a 0 or a 1, which are typically used to represent two possible states, such as "on" or "off," "true" or "false," or "yes" or "no."

In a computer, bits are grouped together into larger units of information, such as a **byte** (which consist of **8 bits**). The combination of bits make up a computer's memory and storage can represent any type of digital data, including text, images, audio, and video.

You talked about 7 bit ASCII, but the examples for different letters given above have 8 bit codes?
This is because the memory in computers is **byte-addressable**. This means that the computer cannot access less than a byte of memory (RAM) at a time. That's why a whole byte (1 extra bit) is allocated for even 7-bit ASCII. Using ASCII all alphabets, digits, and symbols can be represented making a mapping/translation between human understandable characters/symbols and computer understandable characters/symbols.

Ok, so we have a coding scheme to deal with mismatch caused by binary language. What next?
Using the ASCII coding scheme, we can use the characters that we understand easily like alphabets, numbers, and symbols. With 7-bit ASCII the total number of characters that can be represented are 128 (i.e., 2^7, we use 2^7 because 2 is the base and 7 is the number of bits.)

Right! How about the language then using the characters that we understand?
Using the characters we humans understand, a new language comes to being, known as **assembly language**. Machine language and assembly language are the same in terms of number of instructions with 1-1 correspondence with the only difference being the use of alphabets, numbers, and symbols instead of combinations of 0's, and 1's.

Ok. I understand that computers are digital and use 0's and 1's to represent information. As we humans find it difficult to use this binary language, we developed coding schemes like ASCII and Unicode. So how do we communicate with computers then? How do we write programs? Can computers understand our language?

First, the languages in computers are divided into two parts: low-level and high-level languages. By low-level languages we mean binary/machine languages and its equivalent using the human understandable symbols (also called mnemonics) with the help of coding schemes (like ASCII), is knows as assembly language.

Good. So, we are going to use assembly language instead to write the programs?

Yes, and also no. Although, assembly language makes it easier for use to write the programs compared to the machine language. It still is not that expressive. We, as humans are comfortable with languages that are much more **expressive** and powerful with diverse expressions. Therefore, assembly language can be used in specific unavoidable situations. For writing more powerful software, we need more expressive language(s) that are closer to our usual style of communication.

Ok. Are we going to write programs in English language then?

While it would be convenient to write computer programs directly in English or another natural language, it is not practical because natural languages are too **ambiguous** and variable to be processed by a computer in a reliable way. Natural language processing (**NLP**) is an active field of research, but current technology is still far from allowing computers to understand and execute instructions written in a natural language in the same way that humans do.

So what language(s) are we going to use after all?

Instead, computer programs are typically written in programming languages, which are **specifically designed** to be read by computers. These languages have a **defined syntax** and a **limited vocabulary**, which makes them easier for computers to parse and interpret. Additionally, programming languages have **well-defined semantics**, which specify exactly what each piece of code does, so there is less room for interpretation or ambiguity.

So, while it may be tempting to try to write computer programs in English or another natural language, it is much more effective and efficient to use a programming language, which has been specifically designed to be understood by computers.

What is the difference between computer language and human language?

Computer languages and human languages have several differences:

Syntax: Computer languages have a strict syntax and specific rules for writing code. Human languages have more flexible grammar rules and can have multiple ways of expressing the same idea.

Purpose: Computer languages are designed to provide instructions to computers, while human languages are used for communication between humans.

Vocabulary: Computer languages have a limited vocabulary of keywords and functions. Human languages have a much larger vocabulary and can create new words and expressions.

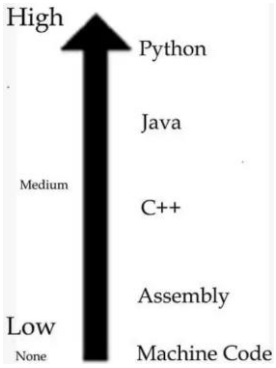
Ambiguity: Computer languages are less ambiguous, as the computer requires explicit instructions. Human languages often have multiple interpretations for the same sentence or phrase.

Emotion: Human language is capable of expressing emotions and sentiments, while computer languages are not designed to convey emotions.

Overall, computer languages are highly structured, whereas human languages are more flexible and context-dependent.

So which language are we going to use for this course? And why?

We are going to use C++ for this course. Let's first see where C++ fits in the computer language hierarchy in the following figure 2.



High Level	*Middle Level*	*Low Level*
High level languages provide almost everything that the programmer might need to do as already built into the language	Middle level languages don't provide all the built-in functions found in high level languages, but provides all building blocks that we need to produce the result we want	Low level languages provides nothing other than access to the machines basic instruction set
Examples: Java, Python	C, C++	Assembler

Figure 2. Language hierarchy and C++

We are beginning with C++ to enjoy the advantage of easier syntax as well as better understanding of the underlying operations. C++ is helpful in understanding the system level working compared to high level languages like Python and Java. It becomes easy to learn high level languages later having a solid foundation of problem solving. C++ is also faster than high level languages because of less translation requirements.

Am I right in thinking that C++ needs to be translated into machine/binary language for executing on computer?

Yes, translation will be required. The process of translation is called compilation. The program (known as source code) is translated into low level language by a special program called compiler. The exact steps are listed below for information. These steps might appear abstract to you but you will better understand these as you spend more time.

The steps in C++ translation are:

Preprocessing: This step involves preprocessing of source code, including macro expansion and inclusion of header files.

Compilation: In this step, the preprocessed code is translated into assembly code for a specific platform.

Assembly: The assembly code generated in the previous step is then assembled into machine code, which is a sequence of binary instructions that can be executed by the computer's CPU.
Linking: This step involves combining the machine code generated from multiple source files and libraries into a single executable program.
Loading: Finally, the linked executable program is loaded into memory and executed by the operating system.

Just to better understand and revise, what exactly is a compiler?

A compiler is a computer program that transforms code written in one programming language (the source language) into another programming language (the target language), usually machine code for a specific computer architecture. Compilers are used to translate the high-level, human-readable source code into machine code that can be executed by a computer's processor. The process of compiling is called compilation.