13205 C Programming Lecture 1 - 26/Sept/2017

Introduction to C Programming

 Scribe: A. Gomes

This lecture aims to respond to the following questions:

- Why do we need to program computers?
- Why do we use C as the first programming language to learn?
- What is programming after all?

More specifically, the main goals of this lecture are:

- To introduce the concepts of hardware, software, and program.
- To show how a program runs on the computer's von Neumann architecture.
- To make sure students grasp the steps of editing, compiling, and running programs in C.

1 A brief history of C language

C is a general-purpose language, that is, it was designed to be used in a wide variety of application domains, from accountability to finance, from biological sciences to physics, and so forth. It was was written by Ken Thompson in 1970 at Bell Labs, specifically for the first UNIX system on a DEC PDP-7 computer; DEC was the brand of Digital Equipment Corporation. In 1972, Dennis Ritchie writes C at Bell Labs. In 1978, Brian Kernighan and Dennis Ritchie published the famous book entitled "The C Programming Language", which caused a revolution in the computing world. In 1983, the American National Standards Institute (ANSI) established a committee to make C a standard, resulting in "ANSI C" in 1988. Today, most Unix operating systems and their variants Linux are more than 95 percent written in C, and C continues to be one of the most popular languages for application developers and system administrators. In concrete, C ranked second in the *The 2017 Top Programming Languages*, as shown in Fig. 1 (`https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages`).

2 Why we need to program computers?

Could you imagine to calculate the sum of a million of integer numbers manually? It takes a long long time. Using a computer to do the same job takes a small fraction of a second. Therefore, we need computers for efficiency, that is, to save you time because time is money. In fact, time likely is the most important resource in a highly competitive world we live. We live alongside computers since computers are everywhere, including in your pocket in the form of a mobile phone.

3 Why do we need a programming language?

In our daily lives, we need a natural language (e.g., English, Portuguese, and so on) to communicate with other people and simplify things. Likewise, we need a programming language (e.g., C, Python, and so forth) to communicate with computers and simplify things.

To communicate, humans learn the basic elements of a natural language, say letters, words and sentences, which allow them to interact with each other. In practice, such a language work as his/her interface on the

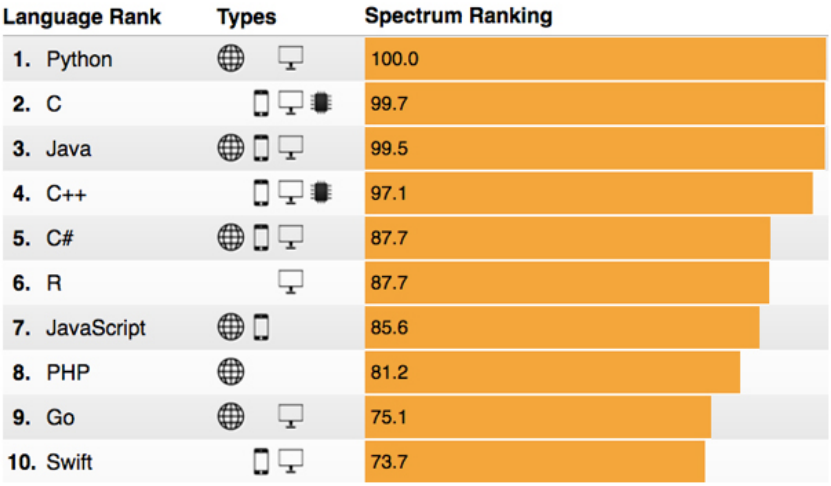

Figure 1: The 2017 top programming languages. Source: IEEE Spectrum, July, 2017.

world. But, the language also is to simplify things, helping us to build increasingly complex things from simple ones.

Analogously, a programming language works as an interface between humans and computers. With such an interface, computers allow us to solve complex problems timely. In practice, computers are for solving problems humans cannot do in a short period of time. But, what is the fun of having the fastest car in the world if you cannot drive it properly?

In fact, computer programming is all about *problem solving*. If you are not allowed to use a programming language (e.g., C language) close to a natural language like English, your programs must be written using the binary alphabet (i.e., just 0's and 1's) because a computer only "understands" 0's and 1's, words of 0's and 1's, and sentences of 0's and 1's. Obviously, when someone writes a program in C, it is mandatory to convert it into a program of 0's and 1's in order to be recognized by the computer. This process of conversion is called *compilation*.

4 Basic concepts

Let us overview a few important concepts about computers and programs:

Computer hardware. It consists of all *physical* components of a computer, namely: CPU (central processing unit), RAM (random access memory), motherboard, hard disk, Ethernet card, etc.

Computer software. It consists of all programs existing in a computer, usually in the hard disk. Before running a program, it is copied into RAM.

Program. It consists of a set of statements that makes the computer perform a task, usually to solve a problem.

5 The von Neumann architecture

The von Neumann architecture, also known as the von Neumann model and Princeton architecture, is a computer architecture proposed by the mathematician and physicist John von Neumann and others in the *First Draft of a Report on the EDVAC* (see `http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf` for further details).

According to the von Neumann's architecture, a computer is generally organized into three essential parts: CPU, RAM, and I/O (input/output), as illustrated in Fig. 2. The CPU is responsible for processing

all statements of a program, one at a time and sequentially. RAM serves to store programs and their data temporarily, that is, while such programs are running on computer. I/O essentially is an interface that allows for the communication between a computer and I/O (input/output) devices like, for example, mouse, keywboard, hard disk, monitor, etc.

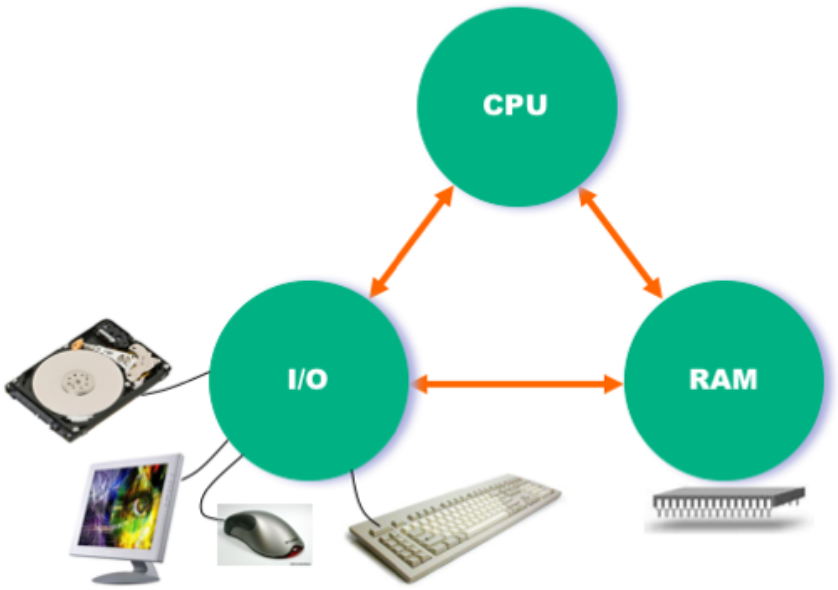

Figure 2: The von Neumann architecture.

6 Running a program

When we intend to run a program on a computer, we only need to type in its name in the command shell of the operating system. What happens next is the following:

- The executable program in hard disk is copied into the RAM (main memory).
- The first statement of the program is transferred (or copied) from the RAM copied into a CPU register (i.e., a memory unit inside the CPU), being then processed by the CPU. This processing on CPU is repeated for the second, third, and every single statement of the program.
- After running the entire program (i.e., all statements), the program terminates and its executable code is released from memory.

Let us see a simple program in C:

```c
#include <stdio.h>

int main()
{
  int x;
  float y;

  printf("Escreva um valor inteiro: ");
  scanf("%d",&x);
  y=x+5.4;
```

```
  printf("O valor de y=%f\n",y);
  return 0;
}
```

Every single program in C has a main function called "main", which has one or more statements wrapped inside a block. A block starts with a left curly bracket '{' and ends with a right curly bracket '}'. Every statement ends with a semi-colon ';'. So, in the previous program, we have seven statements. When the program starts running (don't forget it is already in RAM!), its flow is as follows:

- The first statement `int x;` is copied into the CPU, which executes it straight away. In this case, what is done is to allocate memory space for the integer variable `x`. Note that no integer value is stored in `x` yet. This sort of statement is known as *variable declaration*.
- The second statement `float y;` declares the floating-point variable `y`, that is, a variable that will hold a real number value. When the CPU executes this statement, it also makes room in memory for the variable `y`, but it does hold any real number yet.
- Then, the statement `printf("Escreva um valor inteiro: ");` is taken from the memory into the CPU to be executed. Its execution on CPU results in writing the string "Escreva um valor inteiro: " on the monitor, which is an output device. Hence, the function `printf` is called an *output function* for data.
- The fourth statement `scanf("%d",&x);` performs an input data operation. Specifically, it reads an integer number typed in keyboard by the user. Let us assume that the typed value for `x` is 5. This number is stored in the variable `x` in memory.
- When the fifth statement `y=x+5.4;` is executed on CPU, the value of `x` is read from memory, and is added to the value `5.4`. The resulting floating-point value 10.4 is then stored in the variable `y` in memory.
- The sixth statement `printf("O valor de y=%f \n",y);` first reads the value 10.4 stored in `y` and outputs the string `O valor de y = 10.4` on the monitor.
- Finally, the program terminates after executing the seventh instruction `return 0;`, so the program is released from memory, as well as its associated variables `x` and `y`.

7 Development of software applications

Generically speaking, a software application is just a program. The development of a program involves three main steps: editing, compilation (and linking), and execution (see Fig. 3).

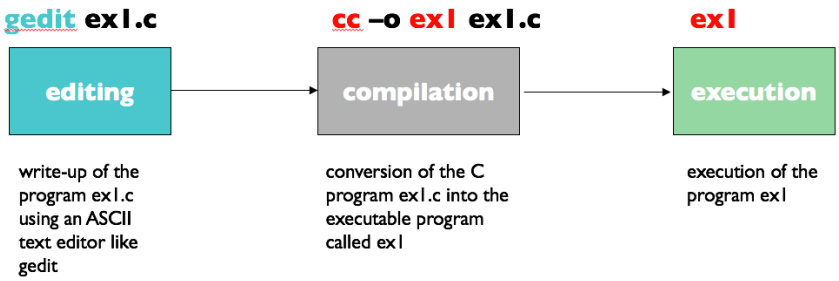

Figure 3: Development steps of a program.

Editing a program means writing a program (in C) using an ASCII text editor; for example, gedit is a suited text editor for programming, but others like vim or pico may be also used. *Compiling* the program `ex1.c` through the command `cc -o` allows us to generate the executable program `ex1`, as illustrated in Fig. 3. Finally, the program `ex1` is *executed* from the command shell (or terminal), being enough to type its name followed by pressing the key `<return>`. We are here assuming you use a UNIX-like command shell, as usual on Linux and Mac OS X. In fact, using basic programming tools is the right way to learn programming.

However, the process of editing, compiling, running (and debugging) programs can be done using a single IDE (Integrated Development Environment). The typical IDE for Mac OS X is the Xcode. The most popular IDE for Windows is the Microsoft Visual Studio. Under Linux, Kylix is the most used IDE. The advantage in using an IDE is that it allows for several different programming languages in addition to C, such as C# and C++, but at the cost of more complexity in setting up the environment, including libraries and the like.