Lecture_1_C_Basics

February 3, 2025

1 Introduction

This course covers basic concepts of C programming, emphasizing practical problem-solving and system-level programming. By the end of the course, you will:

1. Understand C programming fundamentals, including syntax, control structures, and memory management.
2. Work with file handling and inter-process communication (pipes).
3. Manage signals and understand return values in C programs.
4. Learn processes and multithreading concepts for concurrent programming.

---

2 Cheat Sheet

C quick reference cheat sheet that provides basic syntax and methods. Just press on the image to open the file.

3 Textbooks

We recommend those textbooks for a comprehensive introduction to the field. Just press on the image to open the book.

```
<a href="https://github.com/ai4ci-kpi/Refresh-in-C/blob/main/Books/C_Book_2nd.pdf">
    <img src="https://github.com/ai4ci-kpi/Refresh-in-C/blob/main/img/lec1/C_Book_2nd.png?raw=t
</a>
<a href="https://github.com/ai4ci-kpi/Refresh-in-C/blob/main/Books/linux-system-programming.pdf
    <img src="https://github.com/ai4ci-kpi/Refresh-in-C/blob/main/img/lec1/linux-system-progra
</a>
```

The C Programming Language is the first book on C written by **Brian Kernighan** and **Dennis Ritchie** and published in 1978, quickly became the bible of C programmers. In the absence of an official standard for C, this book — known as K&R or the "White Book" to aficionados — served as a de facto standard.

The development of a U.S. standard for C began in 1983 under the auspices of the **American National Standards Institute (ANSI)**. After many revisions, the standard was completed in 1988 and formally approved in December 1989 as ANSI standard X3.159-1989. In 1990, it was approved by the International Organization for Standardization (ISO) as international standard ISO/IEC 9899:1990.

While ANSI was the initial U.S. standard, ISO now leads the standardization process, which is why later standards are referred to by ISO numbers: C89 (ANSI), C90 (ISO), C99, C11, C17/C18, C23.

4 Edit and compile

We will use Linux-based environment called **Google Colab** as a code editor for writing C programs and compiling them with `gcc` compiler via the shell. Note, Colab is not designed for direct execution of C programs. Colab primarily supports Python, but we can run shell commands using `!`, and then execute it.

In Google Colab, `!` and `%%` serve different purposes, particularly for interacting with the underlying system or running specific types of code.

The `!` allows you to execute shell commands (Linux/Unix commands, such as `ls`, `pwd`, `gcc`, etc.)

The `%%` is used to apply a "cell magic" command to an entire code cell. This affects the whole cell, not just a single line. It is typically used for interacting with external languages, like running shell scripts, or for applying special configurations to the whole cell.

Compilation is the process of translating high-level source code written in programming languages like C into machine code, which is the low-level code that a computer's CPU can execute directly. Machine code consists of binary instructions specific to a computer's architecture.

GCC (GNU Compiler Collection) is a widely used C compiler. To check the version of GCC, run the following command in the terminal:

[ ]:
```
!gcc --version
```

```
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

To use `gcc`, open a terminal, use the command line to navigate to the source file's location and then run:

```
gcc Hello.c -o Hello
```

If no errors are found in the the source code (`Hello.c`), the compiler will create a binary file, the name of which is given by the argument to the `-o` command line option (`Hello`). This is the final executable file.

We can also use the warning options `-Wall -Wextra -Werror`, that help to identify problems that can cause the program to fail or produce unexpected results:

```
gcc -Wall -Wextra -Werror -o Hello Hello.c
```

Once compiled, the binary file may then be executed by typing `./Hello` in the terminal.

5 Hello,

C is a powerful general-purpose programming language it has remained a cornerstone of systems and application development for decades, owing to its unique design principles. Three fundamental traits define its enduring utility and appeal:

1. **C is a low-level language.** To serve as a suitable language for systems programming, C provides access to machine-level concepts (bytes and addresses, for example) that other programming languages try to hide. C also provides operations that correspond closely to a computer's built-in instructions, so that programs can be fast. Since application programs rely on it for input/output, storage management, and numerous other services, an operating system can't afford to be slow.

2. **C is a small language.** C provides a more limited set of features than many languages. (The reference manual in the second edition of K&R covers the entire language in 49 pages.) To keep the number of features small, C relies heavily on a "library" of standard functions. (A "function" is similar to what other programming languages might call a "procedure," "subroutine," or "method.")

3. **C is a permissive language.** C assumes that you know what you're doing, so it allows you a wider degree of latitude than many languages. Moreover, C doesn't mandate the detailed error-checking found in other languages.

The C programming language emerged from the development of the UNIX operating system at Bell Laboratories by **Ken Thompson**, **Dennis Ritchie**, and others. Thompson initially created UNIX operating system for the DEC PDP-7, an early minicomputer with just 8K of memory, using assembly language — an impressive achievement considering the technological constraints of 1969.

Like other operating systems of the time, UNIX was written in assembly language. Programs written in assembly language are usually painful to debug and hard to enhance; UNIX was no exception. Thompson decided that a higher-level language was needed for the further development of UNIX, so he designed a small language named B, inspired by BCPL — a systems language derived from Algol 60.

Ritchie soon joined the UNIX project and began programming in B. In 1970, Bell Labs acquired a more advanced minicomputer a PDP-11 for the UNIX project. Once B was up and running on the PDP-11, Thompson rewrote a portion of UNIX in B. By 1971, it became apparent that B was not well-suited to the PDP-11, so Ritchie began to develop an extended version of B. He called his language NB ("New B") at first, and then, as it began to diverge more from B, he changed the name to C. By 1973, UNIX was rewritten entirely in C, marking a pivotal shift. This transition granted UNIX unprecedented portability: by developing C compilers for other systems, the team could easily port UNIX across machines, cementing C's role as a foundational tool for operating systems and software development.

Dennis Ritchie's design of C combined low-level memory access with high-level language features, making it highly efficient and versatile for system and application development.

C has been used to write everything from operating systems (including major parts of Windows, Linux and many others) to complex programs like the Python interpreter, Git, Oracle database, and more.

Additional uses of C are as follows: 1. C is used to write driver programs for devices like tablets,

printers etc. 1. C language is used to program embedded systems where programs need to run faster in limited memory (UAV, Microwave, Cameras etc.) 1. C is used to develop games, an area where latency is very important, i.e., the computer must react quickly to user input.

The versatility of C is by design. C strikes a balance between low-level machine operations and human-readable syntax. Its close relationship with hardware allows for efficient performance and fine-grained control, making it a popular choice for systems programming, embedded development, and performance-critical applications.

5.0.1 C-Based Languages

C has had a huge influence on modern-day programming languages, many of which borrow heavily from it. Some of the most notable C-based languages include:

1. **C++** includes all the features of C, but xtends C with object-oriented programming (classes, inheritance) and generic programming (templates).
2. **Java** is based on C++ and therefore inherits many C features. It Adopts C's syntax but adds automatic memory management (garbage collection) and platform independence (JVM).
3. **C#** derived from C++ and Java combining C-style syntax with features from Java and .NET framework.
4. **Go (Golang)** simplifies C's syntax, emphasizes concurrency and scalability.
5. **Rust** retains low-level control like C but enforces memory safety.

6 Your first C program

A program written in C is a collection of commands or statements.

Below is a simple code that prints the message **Hello, World!**

[ ]:
```
%%writefile Hello.c

#include <stdio.h>

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

```
Overwriting Hello.c
```

[ ]:
```
%%script bash
gcc Hello.c -o Hello
./Hello
```

```
Hello, World!
```

Let's break down the code `Hello.c` to understand each line:

[ ]:
```
#include <stdio.h>
```

C offers various headers, each of which contains information needed for programs to work properly. This particular program requires the `<stdio.h>` header.

The `#` sign at the beginning of a line is directed to the compiler's preprocessor. In this case, `#include` tells the preprocessor to include the `<stdio.h>` header in our program. So, `#include` is used for adding standard or user-defined header files to the program.

The `<stdio.h>` header defines the standard functions for input and output (I/O) operations. In order to use the `printf` function, we need to first include the header file `<stdio.h>`.

The entry point of every C program is the `main()` function, irrespective of what the program does.

Curly brackets `{ }` indicate the beginning and end of a function, which can also be called the **function's body**. The information inside the brackets indicates what the function does when executed.

The C compiler ignores **empty** lines. In general, empty lines serve to improve readability and code structure.

> Indentations such as spaces, tabs, and newlines are also ignored, although they are used to enhance the visual appeal of the program.

The `\n` escape sequence outputs a newline character. Escape sequences always begin with a backslash `\`.

In C, the semicolon `;` is used to terminate a statement. Each statement must end with a semicolon. It indicates the end of one logical expression.

The last statement in the program is `return 0;`. It terminates the `main()` function and causes it to return `0` to the calling process. The number `0` generally means that our program has been successfully executed. A non-zero value (usually `1`) signals a problem with the code.