Systems Programming and Computer Architecture

Janis Hutz
https://janishutz.com

January 4, 2026

TITLE PAGE COMING SOON

"*If you are using CMake to solve the exercises... First off, sorry that you like CMake*"

- Timothy Roscoe, 2025

HS2025, ETHZ

Summary of the Lectures and Lecture Slides

Quotes

"An LLM is a lossy index over human statements"

- Professor Buhmann, Date unknown

"If you are using CMake to solve the exercises... First off, sorry that you like CMake"

"You can't have a refrigerator behave like multiple refrigerators"

"Why is C++ called C++ and not ++C? It's like you don't get any value and then it's incremented, which is true"

- Timothy Roscoe, 2025

Contents

1 The C Programming Language

I can clearly C why you'd want to use C. Already sorry in advance for all the bad C jokes that are going to be part of this section

C is a compiled, low-level programming language, lacking many features modern high-level programming languages offer, like Object Oriented programming, true Functional Programming (like Haskell implements), Garbage Collection, complex abstract datatypes and vectors, just to name a few. (It is possible to replicate these using Preprocessor macros, more on this later).

On the other hand, it offers low-level hardware access, the ability to directly integrate assembly code into the `.c` files, as well as bit level data manipulation and extensive memory management options, again just to name a few.

This of course leads to C performing excellently and there are many programming languages whose compiler doesn't directly produce machine code or assembly, but instead optimized C code that is then compiled into machine code using a C compiler. This has a number of benefits, most notably that C compilers can produce very efficient assembly, as lots of effort is put into the C compilers by the hardware manufacturers.

1.1 The Syntax

C uses a very similar syntax as many other programming languages, like Java, JavaScript and many more...to be precise, it is *them* that use the C syntax, not the other way around. So:

File: 00_intro.c

```c
// This is a line comment
/* this is a block comment */
#include "01_func.h" // Relative import

int i = 0; // This allocates an integer on the stack

int main( int argc, char *argv[] ) {
    // This is the function body of a function (here the main function)
    // which serves as the entrypoint to the program in C and has arguments
    printf( "Argc: %d\n", argc );    // Number of arguments passed, always >= 1
                                     // (first argument is the executable name)
    for ( int i = 0; i < argc; i++ ) // For loop just like any other sane programming language
        printf( "Arg %d: %s\n", i, argv[ i ] ); // Outputs the i-th argument from CLI

    get_user_input_int( "Select a number" ); // Function calls as in any other language
}
```

In C we are referring to the implementation of a function as a ***(function) definition*** (correspondingly, *variable definition*, if the variable is initialized) and to the definition of the function signature (or variables, without initializing them) as the ***(function) declaration*** (or, correspondingly, *variable declaration*).

C code is usuallt split into the source files, ending in `.c` (where the local functions and variables are declared, as well as all function definitions) and the header files, ending in `.h`, where the external declarations are defined. Usually, no definition of functions are in the `.h` files

File: 01_func.h

```c
#include <stdio.h> // Import from system path
                   // (like library imports in other languages)

int get_user_input_int( char prompt[] );
```

File: 01_func.c

```c
#include "01_func.h"

int get_user_input_int( char prompt[] ) {
    int input_data;
    printf( "%s", prompt );     // Always wrap strings like this for printf
    scanf( "%d", &input_data ); // Get user input from CLI

    // If statements just like any other language
    if ( input_data )
        printf( "Not 0" );
    else
        printf( "Input is zero" );

    switch ( input_data ) {
        case 5:
            printf( "You win!" );
            break; // Doesn't fall through
        case 6:
            printf( "You were close" ); // Falls through
        default:
            printf( "No win" ); // Case for any not covered input
    }

    int input_data_copy = input_data;

    while ( input_data > 1 ) {
        input_data -= 1;
        printf( "Hello World\n" );
    }

    do {
        input_data -= 1;
        printf( "Bye World\n" );
    } while ( input_data_copy > 1 );

    return 0;
}
```

2 x86 Assembly

3 Hardware