

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	4
1.1	The Syntax	4
2	x86 Assembly	5
3	Hardware	6

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 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 who's 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

```

1 // This is a line comment
2 /* this is a block comment */
3 #include "01_func.h" // Relative import
4
5 int i = 0; // This allocates an integer on the stack
6
7 int main( int argc, char *argv[] ) {
8     // This is the function body of a function (here the main function)
9     // which serves as the entrypoint to the program in C and has arguments
10    printf( "Argc: %d\n", argc );    // Number of arguments passed, always >= 1
11                                    // (first argument is the executable name)
12    for ( int i = 0; i < argc; i++ ) // For loop just like any other sane programming language
13        printf( "Arg %d: %s\n", i, argv[ i ] ); // Outputs the i-th argument from CLI
14
15    get_user_input_int( "Select a number" );
16 }
```

File: 01_func.h

```

1 #include <stdio.h> // Import from system path
2                 // (like library imports in other languages)
3
4 int get_user_input_int( char prompt[] );
```

File: 01_func.c

```
1  #include "01_func.h"
2
3  int get_user_input_int( char prompt[] ) {
4      int input_data;
5      printf( "%s", prompt );    // Always wrap strings like this for printf
6      scanf( "%d", &input_data ); // Get user input from CLI
7
8      // If statements just like any other language
9      if ( input_data )
10         printf( "Not 0" );
11     else
12         printf( "Input is zero" );
13
14     switch ( input_data ) {
15         case 5:
16             printf( "You win!" );
17             break; // Doesn't fall through
18         case 6:
19             printf( "You were close" ); // Falls through
20         default:
21             printf( "No win" ); // Case for any not covered input
22     }
23
24     int input_data_copy = input_data;
25
26     while ( input_data > 1 ) {
27         input_data -= 1;
28         printf( "Hello World" );
29     }
30
31     do {
32         input_data -= 1;
33         printf( "Bye World" );
34     } while ( input_data_copy > 1 );
35
36     return 0;
37 }
```

2 x86 Assembly

3 Hardware