C Introduction

Kurt Schmidt
Skipjack Solutions
October 18, 2021
Objectives

Intended audience: Student who has working knowledge of Python

Target compiler: I’ll try to center the discussion on C99 using gcc 7
  • Most changes in C11 are outside the scope of this course
  • I’ll discuss relevant ones

Objectives:
  • To gain some experience with a statically-typed language
  • To gain some experience with a compiled language
  • To get practice in using pointers
    • To get practice with dynamically-allocated memory
    • Play around with linked lists
• Code examples might have an accompanying link
  • Follow link to step through example at pythontutor.com
  • Does a nice job of graphically showing variables in memory, the heap, and the stack
• C is the language of kernels
  • Unix and Linux are written in C
  • MacOS is written in Objective-C, a superset of C (mostly)
  • Even the Windoze kernel is written in C (and assembler)
• Other languages, such as Java, Python, Bash, Awk, C#, C itself, and many others, are written in C
• Applications from super computers to embedded microcontrollers are written in C
Overview of the Language

• C is expressed fairly simply
• C is an imperative, procedural language (like Python)
  • C is *not* object-oriented (unlike Python)
• C is compiled to native executable (unlike Python)
• C is statically-typed (also unlike Python)
• C is statically-bound (like Python)
• C supports recursion (like Python)
More Comparisons

- C doesn’t provide a garbage collector
  - The programmer has more work
  - The programmer has much finer control over the program
  - C programs run faster
- C doesn’t allow functions to be defined inside functions
- C requires the `main` function as the entry point
Hello

/* hello.c -- sample program */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
    int age; // years
    age = 13;
    double coord = (1 + sqrt(5)) / 2;
    char *id = getenv("USER");

    printf("Hello. I am %s, age %d, at %f\n", id, age, coord);
    return 0;
}

output:

$ gcc hello.c -o hello
$ ./hello
I am TLorax, age 13, at 1.618
Statements

• Some statements in C:
  • Any expression (including assignment)
  • Branch (if, if-else)
  • Loops (for, while)
  • Jumps (goto\(^1\), break, continue, and return)

• Statements end with a semicolon
• Statements can span lines
  • C largely ignores whitespace and newlines
  • Whitespace just separates tokens

• **Must** occur only inside functions

\(^1\)You will not use this yet
Compound Statements

- Collection of statements and declarations
- Delimited by curly braces `{ }`
- They may be nested
- The body of a function is a compound statement
- Might also appear as the body of a loop, or after if or else
Comments

- Technically, comments are *not* statements
  - They are stripped out by the *precompiler*
- Come in 2 flavors:
  1. Block comments (line #1)
     
     /* Can span multiple lines */

     - They do *not* nest
  2. Line comments (line #8)
     
     // Ignored until next newline

Parsing Comments

Avoid forming comments like this:

    // * Something...
main Function

- Every C/C++ program has one function called `main`
- It is the entry point of the program
- Returns an integer
  - This is the status, the exit code that can be retrieved on the command line
  - Typically, unsigned 7 or 8 bits
  - Generally, 0 indicates success
Declaring Variables

- Lines 8-11 are *variable declarations with initialisers*
- These declare variables *local* to the *main* function
- *age* and *coord* are numbers, an integer and a float type, respectively
- *id* is (points to) an array; it is how we store strings in C
- We’ll be discussing types and arrays further
• Line 10 is an expression, a function call
• `printf` provides formatted output (rather elegantly)
• Takes, at least, a `format string`, and maybe other values
• Each extra value in the argument list replaces a `format specifier` in the string
### printf Format Specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Type/Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%c</code></td>
<td>char</td>
</tr>
<tr>
<td><code>%o</code></td>
<td>unsigned int (octal)</td>
</tr>
<tr>
<td><code>%ld</code></td>
<td>long (decimal)</td>
</tr>
<tr>
<td><code>%f</code></td>
<td>float</td>
</tr>
<tr>
<td><code>%e</code></td>
<td>float (scientific notation)</td>
</tr>
<tr>
<td><code>%d</code></td>
<td>int (decimal)</td>
</tr>
<tr>
<td><code>%h</code></td>
<td>unsigned int (hex)</td>
</tr>
<tr>
<td><code>%%</code></td>
<td>%</td>
</tr>
<tr>
<td><code>%f</code></td>
<td>double</td>
</tr>
<tr>
<td><code>%g</code></td>
<td>Shortest of <code>%f</code> or <code>%e</code></td>
</tr>
</tbody>
</table>

- Line #2

```c
#include <stdio.h>
```

This provides a declaration of the `printf` function, so the compiler is aware of it.

- More specifiers, precision and formatting flags later.
Precompiler Directives
#include

- Any line that starts with an octothorpe (#) is a precompiler directive
- It’s not C code
  - It’s stripped out, before the compiler gets it
- `#include` simply drops contents of named file into your source code
  - Generally, header files contain function, type, and maybe variable declarations
  - So, e.g. the compiler knows what `printf` is, that you called it correctly
  - The compiler knows `stdout` exists, and that its type is `FILE*`

---

1 Might be one pass, but, handy to think of it as 2 separate passes
Commenting Out Block of Code

**Use Precompiler Directives**

To temporarily comment out a block, during development or debugging:

- Avoid block comments (they don’t nest)
- Line comments require a bit of work (even w/a decent editor)

Use a precompiler conditional:

```c
Some solid code
#if 0
Some questionable code
/* might contain comments */
#endif
Some more solid code
```