Programming in C

Klaus-Peter Zauner

ELEC1201: Programming
Programming plays a dual role

1. Communicate instructions to a machine
2. Communicate an algorithm to another person

Both are equally important—you need to learn how to do both well.
C

- created in the early 70’s
- on PDP11 with 24 KB of RAM (early UNIX: 12 KB)
- for the development of UNIX

Outcome:

- minimalist → easy to learn and to implement
- pragmatic → trumps aesthetics
- portable and close to the hardware
- core language has no I/O and no dynamic memory management

... in many ways ideal for embedded systems.
Programming in C

C will put you in full control and not get in your way—no matter how big the mistake is you are about to make.

- For historical reasons there are some subtle pitfalls
- For efficiency reasons there are no safety nets at runtime
- It has a terse syntax that is not very restrictive—but has little redundancy!

Very likely the compiler will find an interpretation for your code.
C will put you in full control and not get in your way—no matter how big the mistake is you are about to make.

- For historical reasons there are some subtle pitfalls
- For efficiency reasons there are no safety nets at runtime
- It has a terse syntax that is not very restrictive—but has little redundancy!

Very likely the compiler will find an interpretation for your code.
The International Obfuscated C Code Contest
Adrian Cable, 2013
What does this do?

▶ Recognizes every printable ASCII character of handwritten text in an image

Comments from the author:

▶ “Newcomers to C find it hard to learn all those different ways to control flow: for, while, if, do, goto, continue, break and heaven knows what else! So, in this program we only use for, so absolute beginners can get into the code straight away.”

▶ “main() is the most useful function in all of C—so it is a mystery to the author why most programs use it only once. Here we use it over and over for maximum benefit.”

http://www.ioccc.org/2013/cable2/hint.html
Looks like A but does B

Ideal to smuggle malicious code past a review! → backdoors
If bugs make it past the review... why would it stop some carefully crafted code?

The Underhanded C Contest:

"C is an ideal language for this... C lets you overwrite stack entries, screw up function pointers, and poison all data at the bit level. C nods encouragingly as you attempt to execute a floating point array. In terms of enforcing program correctness, your typical C compiler is basically the two guards from Swamp Castle in Monty Python and the Holy Grail."

http://www.underhanded-c.org/
Is something like this still useful?

IEEE Spectrum: The 2017 Top Programming Languages

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Python</td>
<td>🌐💻</td>
<td>100.0</td>
</tr>
<tr>
<td>2. C</td>
<td>📱💻📱</td>
<td>99.7</td>
</tr>
<tr>
<td>3. Java</td>
<td>🌐💻</td>
<td>99.5</td>
</tr>
<tr>
<td>4. C++</td>
<td>📱💻📱</td>
<td>97.1</td>
</tr>
<tr>
<td>5. C#</td>
<td>🌐💻</td>
<td>87.7</td>
</tr>
</tbody>
</table>

“...the languages currently in demand by recruiting companies, C comes out ahead of Python by a good margin.”

IEEE: Top Languages
Is something like this still useful?

January 2018

C Language of Year 2017

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>14.215%</td>
<td>-3.06%</td>
</tr>
<tr>
<td>C</td>
<td>11.037%</td>
<td>+1.69%</td>
</tr>
<tr>
<td>C++</td>
<td>5.603%</td>
<td>-0.70%</td>
</tr>
<tr>
<td>Python</td>
<td>4.678%</td>
<td>+1.21%</td>
</tr>
<tr>
<td>C#</td>
<td>3.754%</td>
<td>-0.29%</td>
</tr>
</tbody>
</table>

https://www.tiobe.com/tiobe-index
When to use C

Where C is used, there is typically no alternative:

▶ if low-level hardware access is required
▶ when runtime resources are critical
▶ when realtime behaviour is critical
▶ software for new hardware
  ▶ C is often the first language available
C in Practice
Later on we will cross-compile for the IlMatto.

Source code → Compiler → Executable

Host Target
Later on we will cross-compile for the IlMatto:

Cross-compilation
Tools we will use:

- **Editor** to write and modify the source code file
- **Compiler** to translate the source code into machine instructions
  - sometime a single C instruction translates into a single machine instruction, but often it translates into a small programme of machine instructions
Debugging

- Most of the time of programmers is not spent writing code, but on debugging code.

- If your code is not well written, debugging can become very difficult
  - any change to a working program can introduce new bugs

  ...you need to make a lot of mistakes until you can debug problems fast.
Build Process

To understand the error messages from the “compiler” we need to have a more detailed view of the process of building an executable:

- **Preprocessor**: Text substitutions
- **Compiler**: Translation of source code into assembly language
- **Assembler**: Translation into machine instructions
- **Linker**: Combination of translation units into executable code
Reading Error messages

An incorrect program w

- Pay most attention to the first message
- The Error may be before the location indicated

Do not be tempted to randomly change things until the errors go away...
Hello World

```c
/* myprog.c */
/* A famous test for the C build process. */
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
```

The editor understands the C-syntax and colours it.
Hello World dissection

/* myprog.c */
/* A famous test... */
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
Hello World dissection

/* myprog.c */
/* A famous test... */

#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}

- Header comments
- Preprocessor command to load a library
- Entry point for the program
- Call to a library function
- Returning a value to the caller of the program
Hello World dissection

/* myprog.c */
/* A famous test... */
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
Hello World dissection

- Header comments
- Preprocessor command to load a library
- Entry point for the program
- Call to a library function
- Returning a value to the caller of the program

/* myprog.c */
/* A famous test... */
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
Hello World dissection

/* myprog.c */
/* A famous test... */
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
Translation Units

- **Scopes:**
  - program, file, function/block
- Programs are organised in files
- Files can be compiled independently (.c ➔ .o)

Linux kernel ≈ 8 M lines of C in over 16 k files.
main.c  p1.h  p1.c  p2.h  p2.c
Compiler  Compiler  Compiler
lib1.h ... libn.h
lib1.o ... libn.o main.o
p1.o  p2.o
Linker  Linker
main.elf
Header files: *.h

- Contains declarations for exposed functions and variables
- Included with preprocessor commands
  ```c
  #include <stdio.h>, #include "my.h"
  ```
  - in source file that holds corresponding definitions
  - in source files that make use of the declared functions
- ...serves also as documentation.