Modern C - extended example

Modern C

Jens Gustedt

May 18, 2022

README

- This notebook follows chapter 1 of Modern C (Gustedt, 2020). But the beginnings of other books are also inspiring, including the books by King (2008), Gookin (2022), and Davenport/Vine (2014).
- The purpose is for you to see much of what you're going to learn in more detail later on using another than the usual, simple "hello world" program.
- You will also see Emacs Org-mode, our literate programming environment, in action.
- This lecture is available in Org, PDF, HTML and as YouTube video

Introduction to programming

- What is programming?
- What is programming in C?
- How do you analyze a program?

What is programming?

- Programming means handing a specific **task** to a computer.
- The computer is not like a human at all. It has no **reason**.
- It's also not like a hammer because you can **talk** to it.

- A program is a set of **instructions** for the computer[1].
- This is called **imperative programming**.

What is programming in C?

- C is one of many programming languages
- C is old (1970s) and comes with a lot of jargon
- C allows you to program very close to the machine

First C program

Here is a first program in C[2]:

```
/* This may look like nonsense, but really is -*- mode: C -*- */
#include <stdlib.h>

int main (void) {
  // Declarations
  double A[5] = {
    [0] = 9.0,
    [1] = 2.9,
    [4] = 3.E+25,
    [3] = .00007
  };

  // Doing some work
  for (size_t i = 0; i < 5; ++i) {
    printf("Element %zu is %g, \tits square is %g\n",
   i,
   A[i],
```

[1] You don't necessarily need a program, a human-readable source code document, to program a computer: there is no a growing body of work, and much support for low-code and no-code computing, which don't always work though (Johannessen/Davenport 2021). In business, these solutions are also known as RPA (Robotic Process Automation).

[2] The actual C program is the stuff between `#+begin_src...#+end_src`, a so-called *code block*. The lines that define the code block are part of Emacs Org-mode meta data that tell the editor, Emacs, what to do with the code in the block (e.g. treating it as a C program). You'll learn more on Emacs Org-mode later.

```
   A[i] * A[i]);
  }

  return EXIT_SUCCESS;
}
```

```
Element 0 is 9,  its square is 81
Element 1 is 2.9,  its square is 8.41
Element 2 is 0,  its square is 0
Element 3 is 7e-05,  its square is 4.9e-09
Element 4 is 3e+25,  its square is 9e+50
```

☐ What do you think this program does?

- To answer such a question, don't get bogged down by detail
- Instead, identify elements, blocks and patterns
- Distinguish input and output (if there are any)

Analysis of the program

The analysis consists of a "phenomenological" part (discern meaning by looking closely at the program), and a "semantic" part (use syntax rules of C to discern meaning).

You'll often get into this situation: that you have to read and understand a program someone else has written. More often than not the program will lack documentation and clarity.

Parts of the program

The program has four parts. They start with certain keywords like `#include`, `int main (void)`, `for` and `return`. Some are followed by `{...}`. There are comments that say what some parts are about:

1. Include something.

   ```
   /* This may look like nonsense... */
   #include ...
   ```

2. Declare something.

```
int main (void) ...
   // Declarations
   double A[5] = ...
```

3. Do something.

```
// Doing some work
for {...}
```

4. Return something.

```
return EXIT_SUCCESS;
```

To summarize, these are the things this program does - and this is what all programs do (and a little bit of why they do it):

	WHAT	WHY	EXAMPLE	PURPOSE
1	Include	Standard library functions	`stdlib.h`	Input/Output
2	Declare	Variables (reserve memory)	`double A[5]`	Array declaration
3	Do	Printing function	`main()`, `for()` `printf()`	start, loop, print
4	Return	Signal success	`EXIT_SUCCESS`	Macro insertion

You can see that *functions* are the work horses of C.

Output of the program

The program generates an output: five lines that are generated by the expression `printf(..)`, and that contain integer and non-integer numbers and the results of an arithmetic computation (square).

Output:

```
Element 0 is 9,  its square is 81
Element 1 is 2.9,  its square is 8.41
Element 2 is 0,  its square is 0
Element 3 is 7e-05,  its square is 4.9e-09
Element 4 is 3e+25,  its square is 9e+50
```

The function responsible for this is the `printf` function. Here is the *function call*:

```
printf("Element %zu is %g, \tits square is %g\n",
        i,
        A[i],
        A[i] * A[i]);
```

1. The function name is `printf`, and it takes *arguments* between `(...)`
2. The text between apostrophs `"..."` is a *string literal*
3. The text also contains three markers or *format specifiers* like `%g`
4. The markers indicate positions where numbers are to be inserted
5. The text also contains *escape characters* starting with `\` like `\n`
6. Part two are three *variables* separated by commata
7. There is one marker for each variable
8. The printed value changes with the value of the variable `i`
9. This variable `i` is also called the *loop* variable
10. This *statement* is closed with a semi-colon `;`

Compiling the program

- C is a *compiled* programming language, which means that the *source code* has to be translated into *machine code* to be executed by the computer.
- The source code is readable for humans (and can be edited), the machine code is in *binary* form and is not readable.
- *Binary* is a short form for "made up of 0 and 1", the only two "words" that a digital computer actually understands
- Correct C programs are *portable* between different computers of the same CPU architecture.
- To compile the program, you can *tangle* the code block **??** into a C file `getting_started.c` and execute this command in a terminal (`$` is the terminal prompt):

```
$  c99 -Wall -o first getting_started.c -lm
```

- `c99` is really GCC, the GNU Project C compiler program
- `-Wall` means GCC should warn us about anything it finds unusual
- `-o first` means "give the *output file* the name `first`"
- `getting_started.c` is the compiler's *target* C source code file
- `-lm` means to add standard math functions if necessary

Running the program

- You can now see the executable file:

```
pi@raspberrypi:~/Documents/cc$ ls -l first
-rwxr-xr-x 1 pi pi 8120 May  8 12:39 first
```

- To execute, just type the file name preceded by the precise location of the file[3]:

```
$ ./first
```

- You should now see the output in the terminal.

```
pi@raspberrypi:~/Documents/cc$ ./first
Element 0 is 9,  its square is 81
Element 1 is 2.9,  its square is 8.41
Element 2 is 0,  its square is 0
Element 3 is 7e-05,  its square is 4.9e-09
Element 4 is 3e+25,  its square is 9e+50
```

Debugging a program

- This was an ideal program example because it was flawless. In the wild, your programs may contain errors - then error messages from the compiler are your friend. Here is one that I generated by commenting out the `#include <stdlib.h>` line (so that the file `stdlib.h` was not included):

[3]Curiously, the computer will not recognize the file if you only type its name. This is because any expression without a specific `PATH` is supposed to be a command - like `ls` for `list files`.

```
pi@raspberrypi:~/Documents/cc$ c99 -Wall -o first1 getting_started.c -lm
getting_started.c: In function ‘main’:
getting_started.c:35:14: error: ‘EXIT_SUCCESS’ undeclared (first use in this funct
35 |       return EXIT_SUCCESS;
   |              ^~~~~~~~~~~~
```

- The process of finding and correcting errors in programs is called *debugging*. In this process, the compiler and its warning or error messages, or *diagnostic output*, are your friends.
- There are also more sophisticated tools to aid debugging, like the GNU debugging program, GDB.
- When your program generates errors, the most important thing is your attitude: be patient, be diligent, and celebrate success.

Summary

- Programming means giving a computer something to do (orders)
- C is an old (50 yrs) compiled imperative programming language
- Programs have patterns, input/output and must follow syntax rules
- Programs should compile cleanly without warnings.
- C programs are portable and can be used across different computers

Glossary

TERM	MEANING
Programming	Get a computer to do a job
C	Programming language
Input/Output	What goes in and what comes out
Array	A data set of values of one typex
Loop	An iterated statement
Macro	An expression that's inserted somewhere
Memory	A part of volatile memory (RAM)
Function	A collection of commands
Argument	Values passed to a function
Call	A call to a function with specific arguments
String literal	Text whose value is fixed (cp. to variables)
Variables	Named part of memory that can be used
Format specifier	Marker beginning with `%` for display
Escape character	Character beginning with \ for display
Loop variable	Counter variable for a loop
Closing character	Semi-colon at the end of C statements
Source code	Program written in human-readable form
Compiler	Program to turn source into machine code
Binary	Machine code format
Portability	Programs can run on different computers
Terminal	Command line interface or shell
Prompt	Marker in a terminal (where you enter input)
PATH	Environment variable on your computer
Debugging	Finding and correcting errors ('bugs')

References

- Davenport/Vine (2014). C Programming for Absolute Beginners. Cenpage.
- Gookin (2022). TinyC Projects. Manning. URL: manning.com.
- Gustedt (2020). Modern C. Manning. URL: gustedt.gitlabpages.inria.fr.
- Johannessen/Davenport (June 22, 2021). When Low-Code/No-Code Development Works - and When It Doesn't. In: Harvard Business Review / Technology and Analysis [online]. URL: hbr.org.

• King (2008). C Programming - A Modern Approach. Norton.