First Literate C Program

Video: `https://youtu.be/0q83ZCu8FUI`

This is a short presentation on how to set up Emacs for literate programming. We will then create a literate C file, and execute it inside Emacs. We will further tangle it as a source code file and execute it on the command line.

We begin by opening a terminal and starting Emacs without any configuration. Let's get that configuration file. Open a web browswer with `M-x eww`. Here, `M-x` or "meta-x" is ALT + x on most keyboards. On the Mac, it's OPTION + x.

At the prompt, enter the URL tinyurl.com/EmacsLyon. This is done in the minibuffer at the bottom of the Emacs window.

The file is quickly downloaded. This is Emacs Lisp code.

- Write the text to a new file, .emacs with `C-x C-w .emacs`
- Kill the `eww` buffer with `C-x k`
- Open the file with `C-x C-f .emacs`
- Run the file with `M-x eval-buffer`

The configuration has been loaded, and now we can begin coding. Start by creating an empty file: `C-x C-f firstLit.org`

It's important to get the file names right - `.emacs` will only be recognized by Emacs upon startup with this name, and Emacs will only accept `.org` files as Org-mode files.

We begin by adding some meta data about the file - title and author.

Next, add a headline followed by information - on anything. It's important that the `*` is in the first column to be recognized as markup.

The TAB key on your keyboard will indent any text according to its meaning. In a text field (now), it aligns with the heading. Later, in code, it will support the program indentation.

After the text, we create a code block. The shorthand `<s` will expand into the meta data for the code block if you follow it with a TAB.

There is no syntax highlighing or indentation yet because the code block is still anonymous. You must add a header argument to identify the language.

We're almost done. The code block contains a fully formed C program. To print anything in the buffer, add another header argument, `:results output`.

Finally, to run the code, enter `C-c C-c`. The cursor must be anywhere in the code block.

Behind the scenes, Emacs has compiled and executed the program. So now you're programming C as if it was Python, interactively!

Things get better though: you can add any other language in the same buffer - Emacs-based interactive notebooks are polylingual.

We'll leave it at that for now. One more thing though: literate programs expand into documentation and/or source code. I'll show both of these.

First, let's render the file as a document, e.g. in HTML, for presentation in a browser. This is called "weaving". There we are. It took a little longer because I had not restarted Emacs after adding the `.emacs` file.

I can expand the document into many different formats.

Next, I can expand the code part into a source code file. This is called "tangling". I need to add a header argument to let Emacs now the file name, `:tangle helloEmacs.c` (any name will work).

You can see the confirmation message in the mini buffer at the bottom. Let's go to the file, compile and execute it manually - all without leaving Emacs.

The tangled source code file `helloEmacs.c` is in the top buffer. Now I've opened a terminal (shell) in the lower buffer.

Let's summarize what you just learnt:

1. How to open the Emacs editor (to close: `C-x C-w`).
2. How to create, save, and write a literate file.
3. How to weave a literate file into documentation.
4. How to tangle a literate file into source code.
5. How to execute a code block inside a literate file.
6. How to open a shell inside Emacs and use it to compile and execute source code.