Introduction to Computer Science
Tutorial 1: Linux command-line interface, vi text editor and a compiler

1. Linux commands

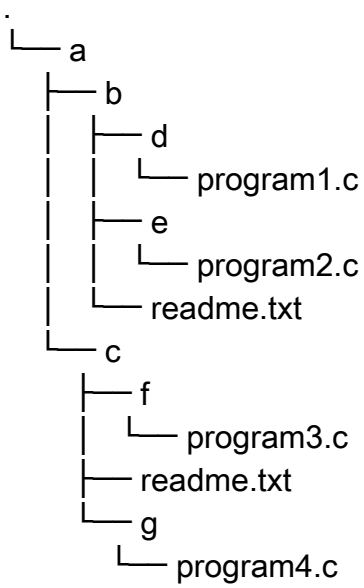
If you have a Linux system on your computer, use it, and open a Terminal (in Ubuntu, the keyboard shortcut is CTRL+ALT+t).

If not, go to https://bellard.org/jslinux/ and select the first option (Alpine Linux 3.12.0, Console, click on startup link), and wait for the system to start.

Exercise 1.

Go to the "home" directory (to be sure, execute the command **cd** alone and you will go to the "home" directory of your user).

Create the following sub-directory structure in the directory:



To create this structure you need to use the commands:

- mkdir DIR : create directory DIR
- cd DIR : enter directory DIR
- cd .. : go to parent directory
- touch FILE : create empty file FILE

2. Editing files with vi

Almost all Linux systems come with this very minimalistic text editor: **vi**.

- **vi** shows a text buffer in which you can see and modify text; then you can save the text buffer to a file on disk.
- **vi** is a *modal* text editor: it reacts to keyboard inputs in a different way according to the mode it is in:
  - in **insertion mode**, it behaves like a text editor
  - in **command mode**, keyboard letters activate commands. There are no on-screen indications so you have to learn the commands.

- Starting `vi`:
  - You can start `vi` from the command line giving to it the name of the file that you want to edit, with the command: `vi filename`. If the file `filename` exists, the editor will show its contents. If it does not exist, it will start without contents, and the file will be created when you save.
  - If you start `vi` without giving a file name, you have to type some name when you save.
- Modes of operation:
  - `vi` is a **modal** editor, that is, the keys you will type are interpreted in different ways according to the mode `vi` is in. It is a very powerful characteristic, but it also requires some learning.
  - Three modes of operation:
    - command mode
    - insertion mode
    - replace mode (the least used)
  - The current mode is indicated by a character in the low left corner of the screen:
    - `-` means command mode,
    - `I` means insertion mode,
    - `R` means replace mode.
  - By default, `vi` starts in command mode.
  - In the command mode, when we type “colon commands”, that is, commands that start with the `:` character, we must validate the command with the `ENTER` key.
- Moving the cursor
  - When you start `vi`, you are in command mode. This means you cannot enter text, but (among other things), we can move the cursor. The simplest way is to just use the arrow keys of the keyboard (arrows may not work on very basic versions of `vi`).

- Basic Editing
  - To type some text, you will need to switch to insertion mode. Use the `i` key. After inserting some text, use `ESC` to come back to command mode. When you are in insertion mode, you can also delete text using the keys `DELETE` and `BACKSPACE`.
- Saving the file and exiting the editor
  - If you are editing a file that already has a known name, you can use `ZZ` (twice the `SHIFT` and `z` key in a row) in command mode. It means "save and exit" (and go to bed, "ZZ…").
  - To save changes without exiting, we can use the `:w` command (type the colon key `:`, then `w`, then `ENTER`).
  - If `vi` does not know the name of the file, you need to type it like this: `:w hello.c` (then type `ENTER`).
  - To exit, use the command `:q`, but `vi` stop you if there are unsaved changes. If you do not want to save these changes and exit anyway, add `!` at the end of the command to force exit: `:q!`. To save and exit with just one command you can use `:wq`.
  - `w` means "write" and `q` means "quit".
- Copy, erase and paste
  - You can copy an entire line using **`yy`**, or erase it completely using **`dd`**.
  - The line gets copied to a temporary memory location, it can be inserted after the current line using **`p`**.
- Moving in a file
  - Search some sequence of characters with the command **`/xyz`** (followed by `ENTER`) to search the sequence of characters **`xyz`**. To repeat the search, use **`n`** to search the next occurrence, or **`N`** to search the previous one.
  - To jump to some line number, say 45, use the command **`:45`**.
  - Finally, the command **`%`** jumps to the symbol that closes or opens the parenthesis or bracket that is below the cursor. Very convenient!
- Undo
  - For a simple "undo" command, use the key **`u`**. You can undo several changes typing **`u`** multiple times.

Exercise 2.

Using the vi text editor, Type the following file contents and save it into a file named helloGT.c:

```
main() {
  printf("Hello GT!\n");
}
```

Then exit vi, and from the command line compile and run this program with gcc:

```
$ gcc -w helloGT.c -o helloGT
$ ./helloGT
Hello GT!
```

Make sure you have gcc installed on your system.

At this point, you may also get some error from gcc if you made a mistake when typing the program. Make sure you did not make mistakes when typing.

Exercise 3.

Notice the -w flag when executing gcc. See what happens when you remove it:

```
$ gcc helloGT.c -o helloGT
```

the compiler outputs some warning messages. A warning is not an error but a recommendation. To make your program **helloGT.c** compile without warning, modify it so that it looks like this:

```
#include <stdio.h>
int main() {
  printf("Hello GT!\n");
  return 0;
}
```

Run again the previous gcc command to check that the warning messages are gone.

"Hello GT" program breakdown

```
#include <stdio.h>
int main() {
  printf("Hello GT!\n");
  return 0;
}
```

The elements of this program are:

- `#include <stdio.h>` : indicates to the compiler where to search for the definition of the printf() function. stdio.h is the "standard input/output" header file.
- `main() { ... }` : The main() function is the entry point of our program. Between the curly brackets we define a sequence of **statements** that will be executed.
- `printf( ... );` and `return 0;` are both statements, they must end with a semicolon symbol. The C language uses curly brackets and semicolon to limit statements.
- `int main()` indicates that the program returns an integer value to the operating system, and `return 0;` indicates that this value should be 0. In this course, we do not care about that but compilers ask these indications. Your program will compile without them, with warnings.
- `"Hello GT!\n"` is a string constant, that gets printed to the screen by the printf() function call. It represents the sequence of characters: H, e, l, l, o, space, G, T, !, newline.

Since a string constant is a sequence of characters, you could also use more than one printf() function call, each one that prints only a part of the message. A program with the same output is:

```
#include <stdio.h>
int main(){
  printf("Hello");
  printf(" GT!\n");
  return 0;
}
```

Or even:

```
#include <stdio.h>
int main(){
  printf("Hell");
  printf("o GT");
  printf("!\n");
  return 0;
}
```

Exercise 4

Write a C program that outputs your name and ID number. The C program must comply with the following conditions:

- it must contain two calls to printf().
- the output format should be, your name on one line, then your ID on the next line, and finally a line jump. Nothing else.
- the file name must be NNNN.c where NNNN is your ID number
- the program must compile and run without errors nor warnings.

To download a program from JSLinux, use the export_file command:

```
$ export_file NNNN.c
```