**Operating Systems**

Grado en Informática. Course 2019-2020

**Lab assignment 0:** Introduction to C programming language

To get acquainted with the C programming language we'll start to code a shell, coding of this shell will be continued in next lab assignments.

We'll start with a nearly empty shell, which is basically a loop that

- prints a *prompt*
- reads from the standard input a line of text which includes a command (with its arguments).
- stores this command in a list of commands
- separates the command and its arguments
- processes the comand with its arguments

At this moment this *shell* has to understand only the following commands

**autores [-l|-n]** Prints the names and logins of the program authors. **autores -l** prints only the logins and **autores -n** prints only the names

**pid [-p]** Prints the pid of the process executing the shell **pid -p** prints the pid of its parent process.

**cdir [direct]** Changes the current working directory of the shell to *direct* (using the *chdir* system call). When invoked without auguments it prints the current working directory of the shell (using *getcwd*)

**fecha** Prints the current date

**hora** Prints the current time

**hist [-c]** Shows the *historic* of commands executed by this shell. In order to do this a list to store all the commands input to the shell must be implemented. *hist -c* clears the historic, that's to say, empties the list (See the *NOTES ON LIST IMPLEMENTATIONS* at the end of this document)

**fin** Ends the shell

**end** Ends the shell

**exit** Ends the shell

- This program should compile cleanly (produce no warnings even when compiling with `gcc -Wall`)

- **NO RUNTIME ERROR WILL BE ALLOWED (segmentation, bus error ...)**, unless where explicitly spcified. Programs with runtime errors will yield no score.
- This program can have no memory leaks
- When the program cannot perform its task (for whatever reason, for example, lack of privileges) it should inform the user
- All input and output is done through the standard input and output

**Information on the system calls and library functions needed to code this program is available through** `man`: (*printf, gets, read, write, exit, getpid, getppid, getcwd, chdir, time ...*).

## WORK SUBMISSION

- Work must be done in pairs.
- The source code will be submitted to the subversion repository under a directory named **P0**
- The name of the main program file will be `p0.c`. Program must be able to be compiled with `gcc p0.c` Alternatively a `Makefile` can be supplied so that the program can be compiled with just `make`
- Only one of the members of the workgroup will submit the source code. The names and logins of all the members of the group should be in the source code of the main program (at the top of the file)

DEADLINE: October 4th. This lab assignment will yield no score, neither will it be evaluated. However all the code for this assigmnet can be reutilized for the following assigments. This assigmente will also help get acquainted with the `SVN` repository, needed for the correct submission of all of the following lab assignments (from the next assignment on, work wrongly submitted will no be evaluated)

## CLUES

A shell is basically a loop

```
while (!terminado){
   imprimirPrompt();
   leerEntrada();
   procesarEntrada();
}
```

*imprimirPrompt()* and *leerEntrada()* can be as simple as calls to `printf` y

`gets` (there's a reason why *fgets()* should be used instead fog *gets()*)

The first step whe processing the input string is splitting it into words. For this, the `strtok` library function comes in handy. Please notice that `strtok` nor allocates memory neither does copy strings, it just breaks the input string by inserting end of string ('\0') characters. The following function splits the string pointed by *cadena* (suposedly not null) into a NULL terminated array of pointers (*trozos*). The function returns the number of words that were in *cadena*

```
int TrocearCadena(char * cadena, char * trozos[])
{ int i=1;

  if ((trozos[0]=strtok(cadena," \n\t"))==NULL)
    return 0;
  while ((trozos[i]=strtok(NULL," \n\t"))!=NULL)
    i++;
  return i;
}
```

## NOTES ON LIST IMPLEMETATION

- the implementations of list should consist of the data types and the access funtions. All access to the list should be done used the aforementioned access functions.
- four list implementations are to be considered:
  0) **linked list:** The list is composed of dynamically allocated nodes. Each node has some item of information and a pointer the following node. The list itself is a pointer to the first node, when the list is empty this pointer is NULL, so creating the list is asigning NULL to the list pointer, thus the functions `CreateList`, `InsertElement` and `RemoveElement` must receive the list by reference as they may have (case of inserting or removing the first element) to modify the list.
  1) **linked list with head node:** Similar to the linked list except that the list itself is a pointer to a *empty* (with no information) first node. Creating the list is allocating this first element (head node). `CreateList` must receive the list by reference whereas `InsertElement` and `RemoveElement` can receive the list by value.
  2) **array:** Elements in the list are stored in a statically allocated array of nodes, so the list type is a pointer to a structure containing the array of nodes and optionally one or more integers (depending

on the implementation: nextin and nextout indexes, counter ...). For the purpose of this lab assigment, we can assume the array dimension to be 4096 (which should be declared a named constant, and thus easily modifiable).

3) **array of pointers;** The list is an array (statically allocated) of pointers. Each pointer points to one element in the list which is allocated dynamically. For the purpose of this lab assigment we can assume this statically allocated array dimension to be 4096, which should be declared a named constant, and thus easily modifiable. To implement the list with this array we can use either a NULL terminated array or we can use aditional integers.

- **EACH WORKGROUP MUST DO THE LIST IMPLEMENTATION RESULTING FROM THE FOLLOWING PROCEDURE**: Add the two last digits of the D.N.I. of the workgoup components and calculate its module 4, that will yield the implementation to use. Should one (or more) component of the workgroup have no D.N.I., then the ascci code of its capitalized family name initial should be used instead.
  - **example 1:** workgroup components D.N.I.s are 55555581 and 55555507, so the implementation to use will be given by $(81 + 07)\%4$, so this group will have to use implementatio 0, `linked list`
  - **example 2:** workgroup componets are D.N.I. 55555581 and Donald Trump (who, as of now, does not have a valid D.N.I., to the best of our knowledge) so, as the ascii code for the **T** is 84, this group would have to use implementation $(81+84)\%4 = 1$. `linked list with head node`