uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

SYSTEMS ARCHITECTURE

LAST NAME: ....
NAME: ....

LABORATORY SESSION 9: Signals

PROBLEM 1 (3.5 points) – Midterm exam 11.11.2022
Given the following code of the program Q.c:

```
#include <stdio.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

void Manejador(int s);
void Proceso_hijo();

int fin;
pid_t pid;

int main (int ac, char**av)
    {
    int status;

    fin = 0;

    printf("I am father (%d)\n",getpid());

    signal(SIGUSR1,Manejador);
    pid  = fork();
    if (pid < 0)
        {
        fprintf(stderr,"Error en el fork\n");
        exit(EXIT_FAILURE);
        }
    else if (pid == 0) Proceso_hijo();

    while (!fin);

    pid = wait(&status);
    printf("%d has finished\n",pid);
    exit(EXIT_SUCCESS);
    }

void Manejador(int s)
    {
    fin = 1;
    puts ("I have received a signal");
    }
```

Implement the function Proceso_hijo so that the exucution has the following behavior:

```
Examen-C g92>./Q
I am father (920903)
      I am son (920904)
      My father is (920903)
      I finish (920904)
I have received a signal
920904 has finished
```

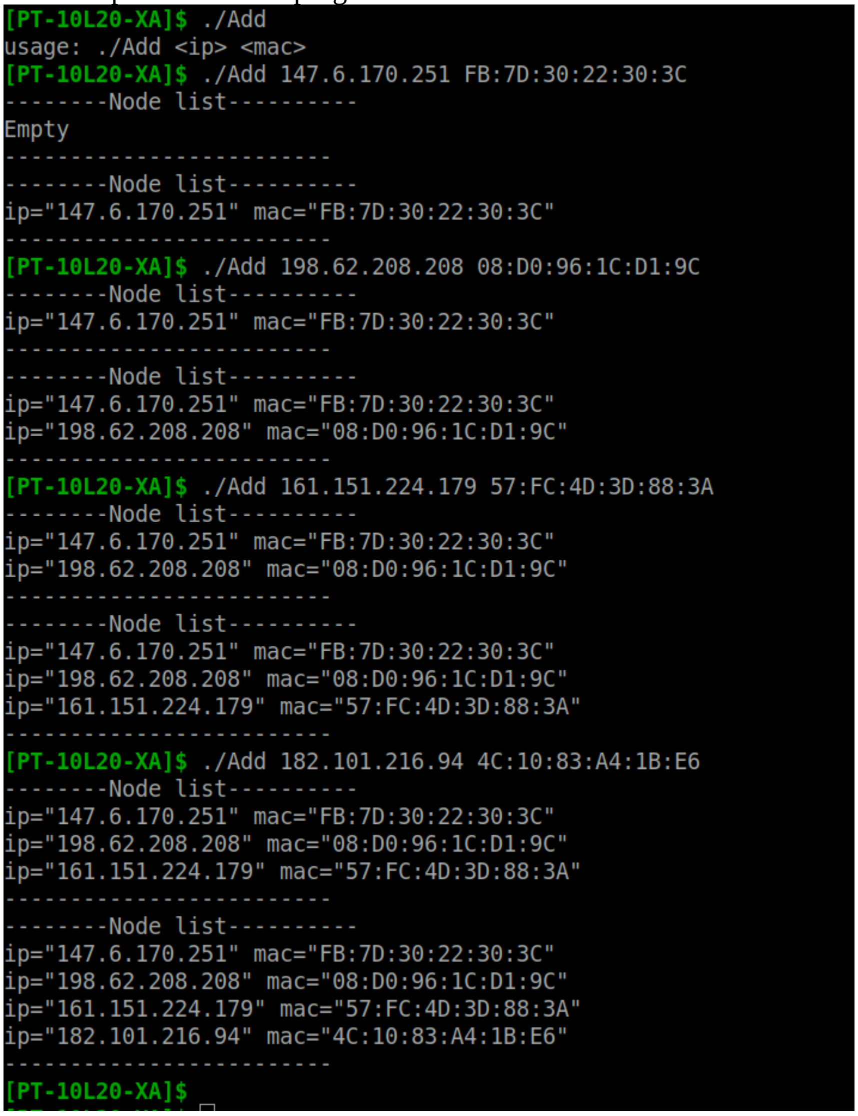
PROBLEM 2 (4 points) – Extraordinary exam 16.06.2023

The PT-10L20-XA router from the company PocoTráfico S.A., for the purpose of this exercise, is simply dedicated to receiving messages and forwarding them to their destination. To do this, it examines the destination IP address of the message and determines which physical device it needs to be transmitted to, for which it needs to know its MAC address.

This is solved by implementing a list of (IP, MAC) pairs in the Nodes.dat file (which is a binary file).

There is a program written in the C language called Add that adds pairs to this file.

An example of how this program works is as follows:



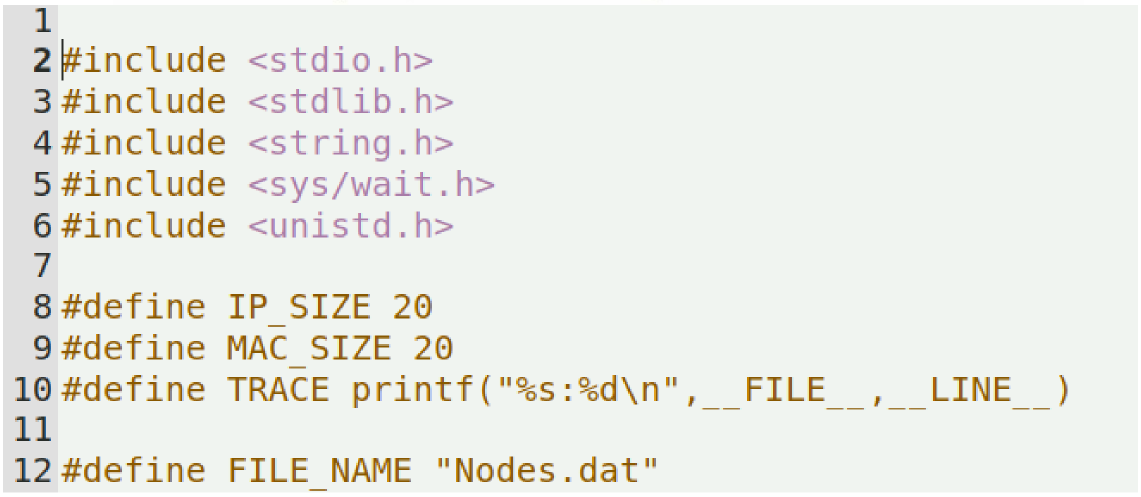
```
[PT-10L20-XA]$ ./Add
usage: ./Add <ip> <mac>
[PT-10L20-XA]$ ./Add 147.6.170.251 FB:7D:30:22:30:3C
--------Node list----------
Empty
-------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
-------------------------
[PT-10L20-XA]$ ./Add 198.62.208.208 08:D0:96:1C:D1:9C
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
-------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
-------------------------
[PT-10L20-XA]$ ./Add 161.151.224.179 57:FC:4D:3D:88:3A
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
-------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
ip="161.151.224.179" mac="57:FC:4D:3D:88:3A"
-------------------------
[PT-10L20-XA]$ ./Add 182.101.216.94 4C:10:83:A4:1B:E6
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
ip="161.151.224.179" mac="57:FC:4D:3D:88:3A"
-------------------------
--------Node list----------
ip="147.6.170.251" mac="FB:7D:30:22:30:3C"
ip="198.62.208.208" mac="08:D0:96:1C:D1:9C"
ip="161.151.224.179" mac="57:FC:4D:3D:88:3A"
ip="182.101.216.94" mac="4C:10:83:A4:1B:E6"
-------------------------
[PT-10L20-XA]$
```

PataPalo S.L. has obtained fragments of the Add program's code and needs an expert like you to complete the program. They have committed to paying you 10 gold doubloons if you succeed.

However, it won't be easy, as there are some mandatory restrictions that must be followed, or else you won't get a single coin.

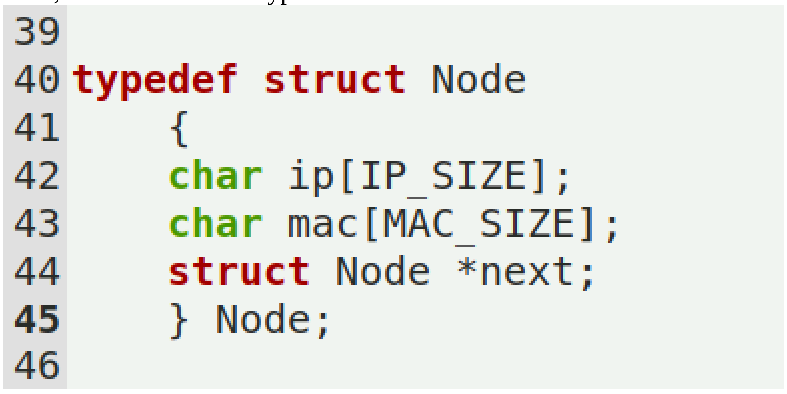
- Respect the original code and use the functions, variables, types, constants, etc. that have been pirated whenever possible. Use the functions, even if you haven't answered the questions where they are implemented.
- Respecting the original program means, among other things, reading the data file into a linked list of Nodes, adding the new node to the end of the list, and saving the list to the file.
- You can use the functions from ASlib.

The first pirated piece is the header of the Add.c file:



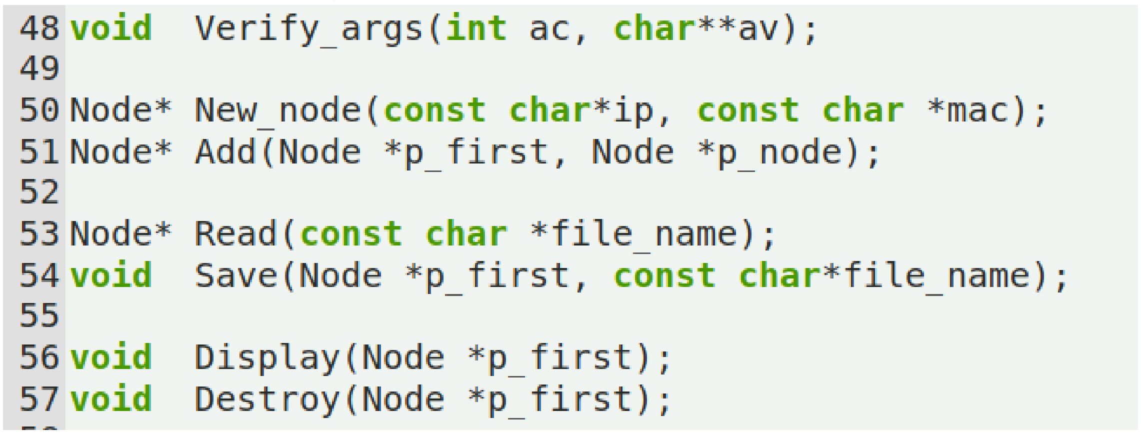
```c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>

#define IP_SIZE 20
#define MAC_SIZE 20
#define TRACE printf("%s:%d\n",__FILE__,__LINE__)

#define FILE_NAME "Nodes.dat"
```

Also, we have the node type:



```c

typedef struct Node
    {
    char ip[IP_SIZE];
    char mac[MAC_SIZE];
    struct Node *next;
    } Node;

```

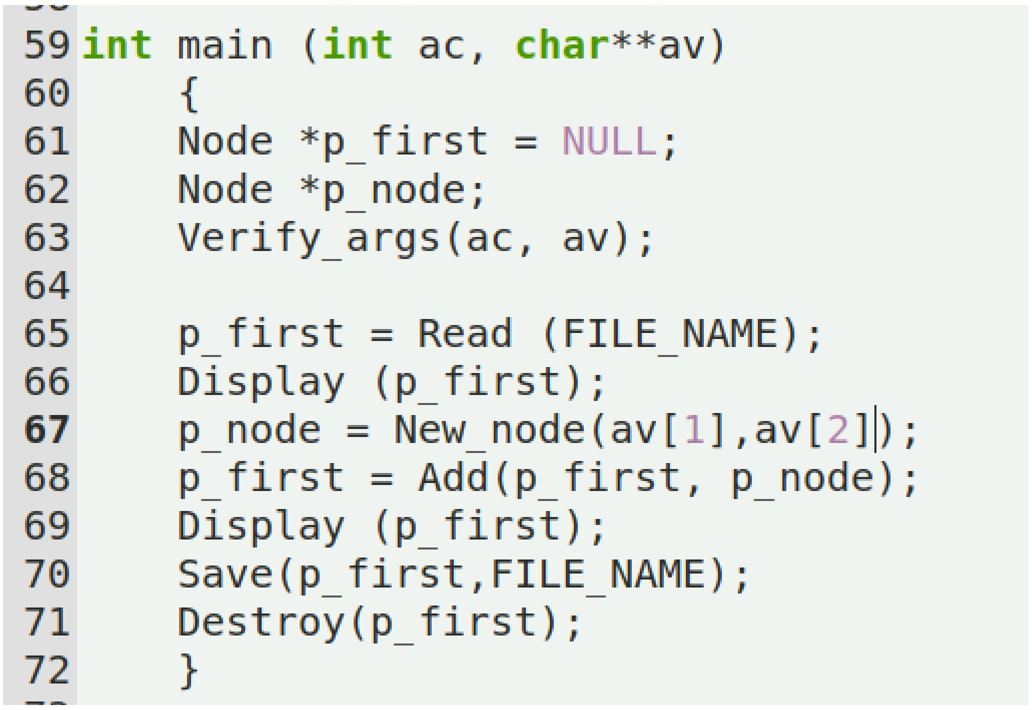
And the functions declarations:



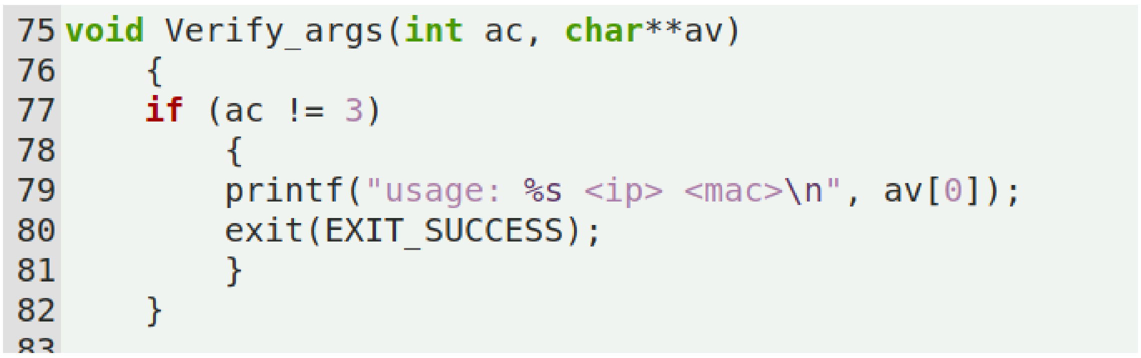
```c
void  Verify_args(int ac, char**av);

Node* New_node(const char*ip, const char *mac);
Node* Add(Node *p_first, Node *p_node);

Node* Read(const char *file_name);
void  Save(Node *p_first, const char*file_name);

void  Display(Node *p_first);
void  Destroy(Node *p_first);
```

- **Verify_args** must check that the program invocation is correct.
- **New_node** should construct a new node from the information in the arguments
- **Add** shall add the node pointed to by p_node to the list starting at p_first
- **Read** shall read the given file constructing the string it returns with the pointer to the first node in the list..
- **Save** must save the list starting at p_first in the given file.
- **Display** shall display the list starting at p_first.
- **Destroy** must free the memory occupied by the list.

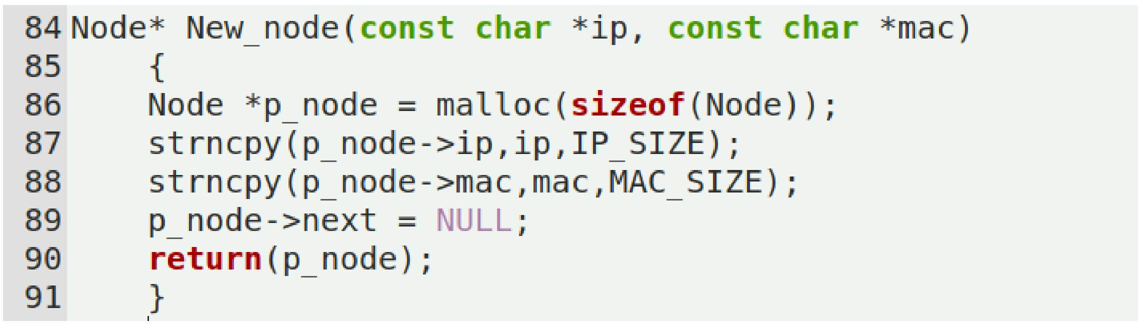
We are provided the following code as well:



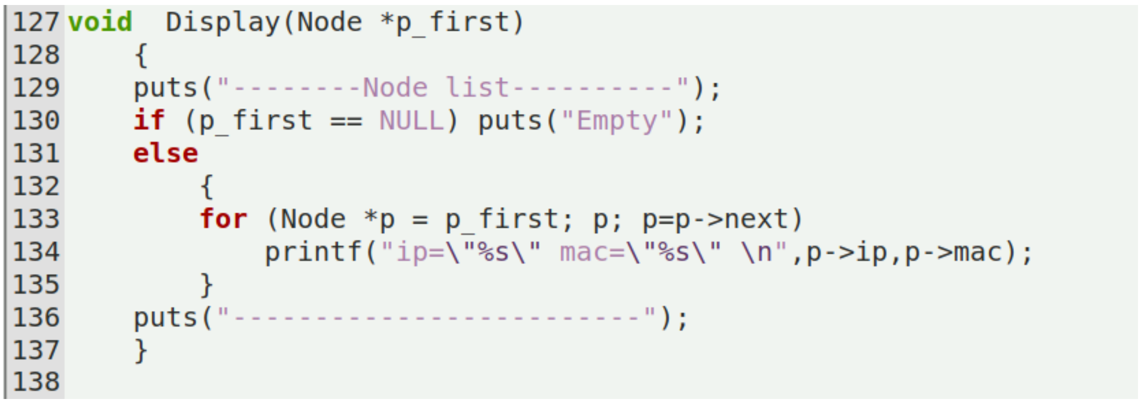
```c
int main (int ac, char**av)
    {
    Node *p_first = NULL;
    Node *p_node;
    Verify_args(ac, av);

    p_first = Read (FILE_NAME);
    Display (p_first);
    p_node = New_node(av[1],av[2]);
    p_first = Add(p_first, p_node);
    Display (p_first);
    Save(p_first,FILE_NAME);
    Destroy(p_first);
    }
```



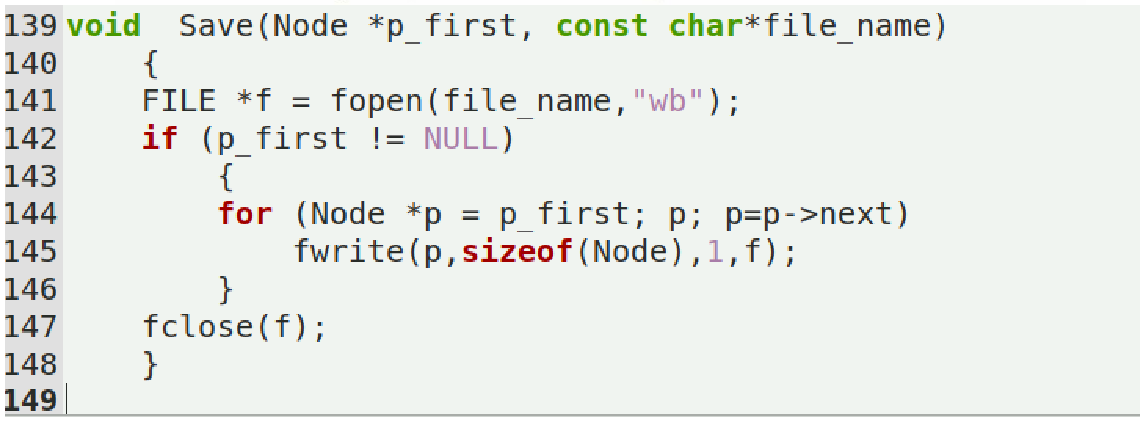
```c
void Verify_args(int ac, char**av)
    {
    if (ac != 3)
        {
        printf("usage: %s <ip> <mac>\n", av[0]);
        exit(EXIT_SUCCESS);
        }
    }
```



```c
Node* New_node(const char *ip, const char *mac)
    {
    Node *p_node = malloc(sizeof(Node));
    strncpy(p_node->ip,ip,IP_SIZE);
    strncpy(p_node->mac,mac,MAC_SIZE);
    p_node->next = NULL;
    return(p_node);
    }
```



```c
void  Display(Node *p_first)
    {
    puts("--------Node list----------");
    if (p_first == NULL) puts("Empty");
    else
        {
        for (Node *p = p_first; p; p=p->next)
            printf("ip=\"%s\" mac=\"%s\" \n",p->ip,p->mac);
        }
    puts("-------------------------");
    }

```



```
void  Save(Node *p_first, const char*file_name)
    {
    FILE *f = fopen(file_name,"wb");
    if (p_first != NULL)
        {
        for (Node *p = p_first; p; p=p->next)
            fwrite(p,sizeof(Node),1,f);
        }
    fclose(f);
    }
```

Section 2.1 (0.7 points)

Node *Read(const char *filename)

Write this function that should read the list of nodes from the binary file.
A hint: As when saving the nodes, it has been made as is, when reading them, the last node will still have next==NULL, and the others will not.

Section 2.2 (0.7 points)

Node * Add(Node * p_first, Node*p_node)

Write this function that must add the node given by p_node to the list given by p_first.
Note that Add must return the pointer to the first node in the resulting list.

Section 2.3 (0.3 points)

void Destroy(Node *p_first)

Write this function that should free the memory occupied by the list given in the argument.

ADDENDA

The chief engineering officer at PocoTráfico is a cunning security engineer and has left a bug embedded in the code, which has cleverly been pirated.
Now it's time to undo it.
The problem is that, due to hardware limitations, it's not possible for the process triggered when executing Add to perform both the insertion operation in the file and displaying the status of the list.
The solution is to create a subprocess that will be responsible for listing the information on the screen. Communication between both processes will be done using signals (SIGUSR1 and SIGINT).
When the child process receives a SIGUSR1, it reads and prints the file, and when it finishes, it sends SIGUSR1 to the parent.
When the child process receives SIGINT, it terminates.
Let's make this modification by answering the following questions.
In each question, add the necessary variables as needed.
You can assume the existence of the remaining functions.
Remember that if a signal is sent and the receiver is not running or has not programmed that signal, it will be lost. A 1-second pause will be sufficient in this case.

Section 2.4 (0.3 points)

Child_handler()

Write this function that should capture the signals received by the child.

Section 2.5 (0.7 points)

Parent_handler()

Write this function that should capture the signals received by the parent

Section 2.6 (0.3 points)

Child_process()

Write here the code of the child process, to cover the described needs.

Section 2.7 (1 point)

Rewrite the main method according to the new needs, including orderly completion.
In case you want to copy-paste it the original code is:

```
int main (int ac, char**av)
   {
   Node *p_first = NULL;
   Node *p_node;
   Verify_args(ac, av);

   p_first = Read (FILE_NAME);
   Display (p_first);
   p_node = New_node(av[1],av[2]);
   p_first = Add(p_first, p_node);
   Display (p_first);
   Save(p_first,FILE_NAME);
   Destroy(p_first);
   }
```

PROBLEM 3 (5 points) – Extraordinary exam 24.06.2022

The 7 dwarfs from the story being: Doc, Grumpy, Happy, Sleepy, Bashful, Sneezy and Dopey.
#define N_DWARFS 7
const char *dwarfs[N_DWARFS] = {"Doc", "Grumpy", "Happy", "Sleepy", "Bashful", "Sneezy", "Dopey"};
Each dwarf is a process written in C language. The parent process of all of them is, of course, Snow White. The 7 dwarf processes must be created. After the fork: the child process will invoke the function Process_dwarf with arguments (pid of Snow White and the name of the dwarf). Once all the children have created, the parent process will invoke the Process_Snowwhite.
Once all the children have been created, the parent process will invoke the Process_Snowwhite.

Process_Snowwhite
Waits for one second.
For each of the dwarfs:

- Write "HI,HO!! <dwarf's name>"
- Write "HIHO" to the dwarf, which will be implemented by sending the SIGUSR1 signal.
- Wait for the reception of the HIHO from the dwarf, also implemented with the SIGUSR1 signal.

When the loop ends and all the processes have finished, it also ends its own.
Process_dwarf

- Write "HELLO I am <dwarf's name> my pid is <pid> and my father is <pidp>".
- Wait for the reception of the HIHO
- Type "<dwarf's name> HI,HO!!!"
- Sends HIHO to Snow White
- Finish

Example of execution
```
HELLO I am Doc my pid is 1891 and my father is 1890
HELLO I am Grumpy my pid is 1892 and my father is 1890
```

```
.....
HI,HO!! Doc
Doc HI,HO!!
HI,HO!! Grumpy
Grumpy HI,HO!!
```

Section 3.1
Write the main function with all the features that are necessary for the described system to work.

Section 3.2
Write the function Process_Snowwhite and other functions if they are needed.

Section 3.3
Write the function Process_dwarf and the functions it needs.