Systems Programming and Concurrence
Dpto. de Lenguajes y Ciencias de la Computación

April 19th, 2018

SURNAME____NAME_____________

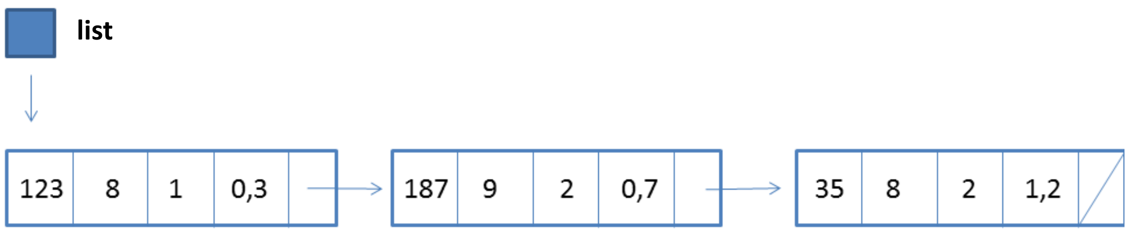
DNI/PASSPORT ____________________ COMPUTER________________GROUP________

Mid-term exam: Systems Programming

We have a binary file COMPONENTS.BIN with data about some of the components that are sold in an electronics store. The data for each component is represented using the following structure:

```
struct Component
{
 //code to identify a component
 int id;
 //code to identify the manufacturer
 int manufacturer;
 //code to identify the type of component:
 //    1: resistor
 //    2: inductor
 //    3: capacitor
 //    4: diode
 //    5: transistor.
 //    new codes may be added as needed
 int type;
 //price of the component
 float price;
};
```

To manage the data, we will use dynamic memory allocation. In our application, when components are loaded on memory, their data is stored in a linked list: in each node we have the data for just one component, and a pointer to the next list node. The following diagram illustrates how the data is stored in the linked list:



You are given:

- a source file main.c (this file is already complete, you do not need to change it)
- a header file componentList.h
- an empty source file componentList.c
- and a binary file COMPONENTS.BIN

You must:

- download these files
- in Eclipse, create a C project named **firstexam**
- copy the files to the folder of the project **firstexam**, then in eclipse refresh the workspace (shortcut key F5)
- add a comment at the beginning in files componentList.c and componentList.h stating your name and DNI/Passport number
- implement the following functions in componentList.c, and the type TList in componentList.h
- when you are done, you must upload ***ONLY*** the files componentList.h and componentList.c

---

void createList(TList *list);

//Initialize the list of components as an empty list (0.5 points, ***INLCUDING DECLARATION OF TYPE TList IN componentList.h***)

---

void insert_component(Tlist *list, struct Component comp, int *ok);

//Inserts the component at the end of the list given as the first parameter, if there is no component with the same id code in that list. In the parameter ok, the function will return 1 if the insertion was successful, and 0 otherwise. (2 points)

---

void new_ordered_list(TList unordered, TList *ordered);

//Given the list in the first argument, this function returns in the second argument a new list with the same elements, but ordered by ascending id code. (3 points)

---

int read_file_to_list (TList *list);

//Adds to the list the contents of the file COMPONENTS.BIN, making sure that no new component is inserted if it has the same id as any element already present in the list. The return value will be the number of components actually inserted in the list. (1 point)

---

void show_list(TList list);

//prints the list on screen (1 point)

void add_to_file (struct Component comp, int *ok);

//Adds the component to the file COMPONENTS.BIN. The component will be added to end of the file, after any contents the file may previously have. The argument ok will be set to 1 if the addition was succesful, or 0 if it could not be added because of some I/O error. (1 Point)

---

void remove_elements(TList *list, int manufacturer);

//Removes from the list all components of the specified manufacturer. If the manufacturer is negative, removes all elements. (1.5 points)

ADDENDUM. Here you have the prototypes of the I/O functions from library <`stdio`.h> you will need to use (You are expected to know the prototypes of the functions you will need for dynamic memory allocation from library <`stdlib`.h>):

```
FILE *fopen(const char *filepath, const char *mode);
```
Open the specified file in the specified mode ("rb" for binary read, "wb" for binary write). If successful, returns a pointer to the file handle; returns NULL if any error prevented the file from being opened.

```
unsigned fread(void *ptr, unsigned membsize, unsigned nmemb, FILE
*stream);
```
Tries to read `nmemb` data blocks, each one of size `membsize`, from file `stream`, copying them to the memory region pointed by ptr. Returns the actual number of read data blocks.

```
unsigned fwrite(const void *ptr, unsigned membsize, unsigned
nmemb, FILE *stream);
```
Tries to write `nmemb` data blocks, each one of size `membsize`, to file `stream`, reading them from the memory region pointed by `ptr`. Returns the actual number of written data blocks.

```
int fclose(FILE *fp);
```
Flushes all buffers of the specified file and closes it. Returns 0 if the operation was successful; returns -1 if there was any error.