Programación II. TAD. Ejercicios 2019/2020

1. Biblioteca

Se desea realizar una determinada aplicación para gestionar los libros que hay en una biblioteca. Supóngase un modelo de biblioteca "simplificado" con las siguientes características:

- Cada libro de la biblioteca se identifica por su título (cadena de caracteres de menos de 128 caracteres) y autor (menos de 64 caracteres).
- El número de ejemplares de cada título que hay en la biblioteca es diferente. Ej.: del libro "El peor viaje del mundo" de Apsley Cherry-Garrad hay 10 ejemplares, pero sólo 5 ejemplares de "Anatomía de un instante" de Javier Cercas.
- La biblioteca puede modificar su inventario comprando (o retirando) libros en lotes de diferente tamaño, por ejemplo en un determinado instante puede añadir 5 ejemplares más a sus fondos del libro "Retrato de un hombre inmaduro".
- La biblioteca nunca tendrá más de 1064 títulos diferentes.
- Los usuarios sacan en préstamo libros de la biblioteca (obviamente siempre que la biblioteca tenga el título en sus fondos y existan ejemplares disponibles) y "devuelven" los libros prestados a la biblioteca.
- El número de ejemplares disponibles para préstamo de un libro no puede ser mayor que su número de ejemplares.
- Los usuarios puede tener por tiempo indefinido los ejemplares sacados en préstamo y tampoco se registran a los usuarios que tienen los libros en préstamo.
- La biblioteca imprime periódicamente un catálogo de los libros que hay en sus fondos.

1) Especifique *informalmente* los TAD que considere necesarios para diseñar la aplicación (datos + primitivas).

Nota: Supóngase que disponemos ya de los TADs : boolean, status, cc (cadena de caracteres) y que no es necesario modelizar a los clientes que acceden a los ejemplares de la biblioteca.

TAD LIBRO: cadena para el título, cadena para el autor, número de ejemplares y número de disponibles

Al menos hacen falta las siguientes primitivas:

Interfaz (funciones publicas) del TaD Book. Pseudocódigo.

```
Book book_ini (CC titulo, CC autor, int ejemplares);
void book_free (Book b);
CC book_getTitulo (Book b);
CC book_getAutor (Book b);
int book_getNumEjemplares (Book b);
int book_getNumDisponibles (Book b);
Status book_setNumEjemplares (Book b, int num);
Status book_setNumDisponibles (Book b, int num);
Status book_print (Device pf, Book b);
Book book_cpy (Book b);
int book_cmp (Book b, Book b);
```

TAD BIBLIO: conjunto de libros y número de libros. Podría guardarse más información pero con esta nos sirve.

Hace falta, al menos, las siguientes primitivas:

Interfaz (funciones publicas) del TaD Library. Abstracto, no C todavía.

```
Library library_ini ();
void library_free (Library l);
Bool library_isEmpty (Library l);
Bool library_isFull (Library l);
Bool library_isBook (Library l, Book b);
Status library_setInventory (Library l, Book b);
Status library_setLoan (Library l, Book b, int num);
int library_print (Device pf, Library p);
```

- **Escriba en código C las estructuras de datos (EdD) adecuadas para los TAD anteriores.**

```
#define MAX_TITULO 128
#define MAX_AUTOR 64

struct _Book {
   char titulo[MAX_TITULO];
   char autor[MAX_AUTOR];
   int nejemplares;
   int ndisponibles;
};
```

```
#define MAX_LIBROS 1064

struct _Library {
   int n;
   Book *libros[MAX_LIBROS];
};
```

- **¿En qué ficheros deben definirse las estructuras anteriores?**
  En book**.c** y library**.c** respectivamente.

- **¿Cómo se definen los nuevos tipos libro y biblioteca? ¿En qué ficheros se definen?**

  En book**.h:** **typedef struct _Book Book;**

  En library**.h:** **typedef struct _Library Library;**

- **Escriba en C el contenido del fichero book.h**

```
#ifndef BOOK_H
#define BOOK_H
#include "tipos.h"
#include <stdio.h>
typedef struct _Book Book;
Book * book_ini (const char * titulo, const char * autor, const int ejemplares);
void book_free (Book *plib);
char * book_getTitulo (const Book *plib);
```

```
char * book_getAutor (const Book *plib);
int book_getNumEjemplares (const Book *plib);
int book_getNumDisponibles (const Book *plib);
Status book_setNumEjemplares(Book *plib, const int num);
Status book_setNumDisponibles (Book *plib, const int num);
Status book_print (FILE * pf, const Book *plib);
Book *book_cpy (const Book *plib);
int book_cmp (const Book *plib1, const Book *plib2); // 0 si son ig
#endif   /* BOOK_H */
```

- **Escriba en C el contenido del fichero library.h**

```
#ifndef LIBRARY_H
#define LIBRARY_H
#include "book.h"
#include <stdio.h>
typedef struct _Library Library;
Library *library_ini();
void library_free(Library *l);
Bool library_isEmpty (const Library *p);
Bool library_isFull (const Library *p);
Bool library_isBook (const Library *l, const Book *b);
Status library_setInventory (Library *l, const Book *b);
Status library_setLoan (Library *l, const Book *b, int num);
int library_print(FILE *pf, const Library *p);
#endif   /* LIBRARY_H */
```

- **Escriba el código C de la primitiva que crea un libro.**

```
Book *book_ini (const char *autor, const char *titulo, const int ejemplares) {
   Book *p=NULL;
   if (autor==NULL || titulo==NULL) return NULL;
   p=(Book *) malloc(sizeof(Book));
   if (p==NULL) return NULL;
   strcpy(p->titulo, titulo);
   strcpy(p->autor,  autor);
   p->nejemplares=ejemplares;
   p->ndisponibles=ejemplares;
   return p;
}
```

- **Escriba el código C de las primitivas que permiten obtener el número de ejemplares de un libro, el número de ejemplares disponibles de dicho libro, el título y el autor, respectivamente.**

  int book_getNumEjemplares (const Book *plib);

  int book_getNumDisponibles (const Book *plib);

```
int book_getNumEjemplares (const Book *plib) {
   if (plib==NULL) return -1;
   return plib->nejemplares;
}
int book_getNumDisponibles (const Book *plib) {
   if (plib==NULL) return -1;
   return plib->ndisponibles;
}
char * book_getTitulo (const Book *plib) {
   if (plib==NULL) return NULL;
   return plib->titulo;
}
char * book_getAutor (const Book *plib) {
   if (plib==NULL) return NULL;
   return plib->autor;
}
```

¿En qué fichero debe estar el código de estas primitivas?

En book.c

- **Escriba el código C de la primitiva para comparar libros:**

  int book_cmp (const Book *plib1, const Book *plib2) ***// 0 si son iguales (como strcmp), y -1 si no.***

```
int book_cmp (const Book *plib1, const Book *plib2){
   if (plib1==NULL || plib2==NULL) return 0;

   if ( strcmp (plib1->autor, plib2->autor)==0 )
      if (strcmp(plib1->titulo, plib2->titulo)==0)
         return 0;
   return -1;
}
```

¿En qué fichero debe estar el código de estas primitivas?

En book.c

- **Escriba el código C de la primitiva para liberar un libro:**

```
void book_free (Book *plib) {
   if (plib!=NULL)
        free(plib);
}
```

- **Escriba el código C de la primitiva que crea e inicializa una biblioteca**

```
Library *library_ini() {
   Library *l;
   int i;

   l =(Library *) malloc(sizeof(Library));
   if (l==NULL) return NULL;

   l->n=0;
   for(i=0; i<MAX_LIBROS; i++)
      l->libros[i]=NULL;

   return l;
}
```

- **Escriba el código C de las primitivas que indican si una biblioteca está vacía ó llena.**

```
Bool library_isEmpty (const Library *l){
   if (l==NULL) return TRUE;
   if (l->n==0) return TRUE;
   return FALSE;
}
Bool library_isFull (const Library *l){
   if (l==NULL) return TRUE;
   if (l->n==MAX_LIBROS) return TRUE;
   return FALSE;
}
```

- **Escriba el código C de la función que busca un libro en la biblioteca.**

```
int library_indexBook (const Library *l, const Book *b) {
   int i;
   for (i=0; i<l->n && book_cmp (l->libros[i],b)!=0; i++);
   //si ha llegado al final del bucle, no lo ha encontrado
   if (i==l->n)
         return -1;
   return i;
}
```

Alternativa:

```
int library_indexBook (const Library *l, const Book *b) {
   int i;
   for (i=0; i<l->n; i++) {
         if (book_cmp (l->libros[i], b)==0)
               return i;  //si lo encuentra, devuelve su índice
   }
   //si no lo encuentra, devuelve -1
   return -1;
```

- **Escriba el código C de la primitiva que inserta un libro en la biblioteca.**

Idea: Ver si el libro está en la biblioteca
sí → obtener el libro y modificar su nº de ejemplares
no →crear el libro e insertarlo en la biblioteca

```
Status library_setInventory (Library *l, const Book *b){
  int  indice, i, j;

  if (l==NULL || b=NULL) return ERROR;

  // busco si el libro está en la biblioteca
  indice = library_indexBook (l, b);

  //si el libro NO está y la biblioteca no está llena, lo inserto
  if (indice == -1) {
      if (library_isFull(l)==TRUE) return ERROR;

      l->libros[l->n] = book_cpy(b);
      if ( l->libros[l->n]==NULL) return ERROR;

      l->n++;
  }
  //si el libro ya está en la biblioteca, modifico el número de ejemplares y el de disponibles
  else {
      j = book_getNumEjemplares(b);
      i = book_getNumEjemplares (l->libros[indice]);
      book_setNumEjemplares (l->libros[indice], i+j);

      j =book_getNumDisponibles(b);
      i= book_getNumDisponibles (l->libros[indice]);
      book_setNumDisponibles (l->libros[indice], i+j);
  }
  return OK;
}
```

- **Escriba el código C de la primitiva que modifica el número de ejemplares disponibles de un libro determinado (prestar libros).**

```
Status library_setLoan (Library *l, const Book *b, int num) {
  int index =-1;
  int i;

  if (l == NULL || b==NULL) return ERROR;

  // busco si el libro está en la biblioteca (si no, devuelvo error)
  index = library_indexBook (l, b);
  if (index == -1)
      return ERROR;

  i= book_getNumDisponibles (l->libros[index]);
  book_setNumDisponibles (l->libros[index], i+num);

  return OK;
}
```

- **Ejemplo de programa de uso de biblioteca (faltaría añadir control de errores, comprobando lo que devuelven las funciones).**

```
#include <stdio.h>
#include <stdlib.h>
#include "library.h"

int main () {
   Book *p = NULL;
   Library *l = NULL;

   p=book_ini("El ingenioso hidalgo Don Quijote de la Mancha", "Miguel de Cervantes", 5);
   book_print(stdout, p);

   l =library_ini();

   /* Inserto el libro en la biblioteca*/
   library_setInventory (l, p);

   /* hago un prestamo de 4 ejemplares del Quijote*/
   library_setLoan (l, p, -4);
   puts("\n----Imprime biblioteca ---\n");
   library_print (stdout, l);

   /* Aumento el número de ejemplares del Quijote incorporando 3 más, pero me quedo 1 de ellos para su consulta en
sala (no se prestará) */
   book_setNumEjemplares(p, 3);
   book_setNumDisponibles (p, 2);
   library_setInventory (l, p);

   /* Aumento el inventario con un nuevo libro */
   book_free(p);    //Vamos a reutilizar la variable p, los datos a los que hacía referencia ya no sirven, ya se ha creado
                    //copia en la biblioteca al llamar a library_setInventory → liberar p
   p=book_ini("Extraños en un tren", "Patricia Highsmith", 10);      //Usamos el mismo p para crear un nuevo libro
   library_setInventory (l, p);     //Crea copia del libro y lo inserta en la biblioteca

   puts("\n---Imprime biblioteca---\n");
   library_print (stdout, l);

   /* Libera todos los recursos antes de salir */
   book_free(p);
   library_free(l);
   return (EXIT_SUCCESS);
}
```

IMPORTANTE: El código de TODAS las funciones de los TAD libro y biblioteca se encuentra en los ficheros .c y .h proporcionados. Aquí no están todas (faltan, por ejemplo, las funciones para imprimir).

2. Ejercicio: GPS

Se desea diseñar e implementar una aplicación para utilizar un GPS. Imagina un modelo muy simplificado, donde SOLO se van a considerar las siguientes características:

- Una localización se representa mediante 2 coordenadas: latitud y longitud (ambas son números enteros).
- Un trayecto se representa mediante una localización origen y una localización destino.
- En el dispositivo GPS se pueden almacenar un máximo de 100 trayectos recientes.

La aplicación principal del GPS debe proporcionar la funcionalidad siguiente:

• Inicializar el dispositivo GPS • Crear una localización, a partir de una latitud y una longitud. • Obtener la latitud de una localización dada. • Obtener la longitud de una localización dada. • Asignar una latitud a una localización dada. • Asignar una longitud a una localización dada. • Crear un trayecto, a partir de su origen y su destino.	• Obtener el origen de un trayecto dado. • Obtener el destino de un trayecto dado • Asignar origen a un trayecto dado. • Asignar destino a un trayecto dado. • Añadir un trayecto al GPS. • Eliminar un trayecto del GPS. • Obtener el nº de trayectos almacenados en el GPS. • Imprimir todos los trayectos almacenados en el GPS: imprimir el nº de trayectos almacenados y, para cada uno de ellos, su longitud y su latitud

a) ¿Cuántos y cuáles serían los Tipos Abstractos de Datos necesarios para representar lo descrito anteriormente? Marca el nº correspondiente y escribe el nombre que darías tú a esos TADs.

□ 1: _____ □ 2: _____ □ 3: _____ □ 4: _____

Los tipos status y boolean se encuentran ya definidos (no se consideran aquí).

Implementa los TADs definidos anteriormente, garantizando al máximo la abstracción de los detalles de la implementación y optimizando el uso de la memoria. Escribe el código en C de dichas Estructuras de Datos (EdD):

```
struct _localiza {
    int longitud;     // tmb vale: int longitud, latitud;
    int latitud;      // y tmb:     int longlat [2];
};
struct _trayecto {
    localiza *porigen;        //penalización si no se pone el puntero (abstracción)
    localiza *pdestino;       //penalización si no se pone el puntero (abstracción)
};
struct _GPS {
    trayecto *tr_recientes [MAX_TRAYECTOS];  //penalización si no se pone el puntero (abstracción)
    int num_trayectos;                //-0,2  si no está este campo: solución ineficiente
 };
```

b) Escribe en C el prototipo de las funciones:

- creaLocalizacion: crea una nueva localización a partir de su latitud y su longitud.
  `localiza * creaLocalizacion (int latitud, int longitud);`
- creaTrayecto: crea un nuevo trayecto (reservando la memoria necesaria para el mismo), y lo inicializa creando copias de las localizaciones origen y destino recibidas como argumentos.
  `trayecto * creaTrayecto (localiza*po, localiza *pd);`
- copiaTrayecto: dado un trayecto, crea otro trayecto que es copia del primero y lo devuelve.
  `trayecto * copiaTrayecto (trayecto *pt);`

- creaGPS: crea un nuevo GPS y lo devuelve.
  GPS * creaGPS ( );
- getNumTrayectosDeGPS: devuelve el nº de trayectos almacenados en un GPS.
  int numTrayectosDeGPS (GPS *g);

c) Escribe el pseudocódigo, sin control de errores, de la primitiva de prototipo
   status imprimeTrayectosDeGPS (fichero f, GPS g);
que imprime todos los trayectos almacenados en el GPS g, imprimiendo primero el nº de trayectos almacenados y, para cada uno de ellos, su longitud y su latitud.
Asume que existen las primitivas de prototipos:
   status imprimeTrayecto (fichero f, trayecto t); //Imprime en f un trayecto t (su longitud y latitud)
   status imprimeEntero (fichero f, entero e): //Imprime en f un entero

```
status imprimeTrayectosDeGPS (fichero f, GPS g)      // Versión más C
    n= num_trayectos(g);                                  // n= g-> num_trayectos;
        imprimeEntero (f, n);                                 // imprimeEntero (f, n);
    para i desde 1 hasta n:                               // para i de 0 a n-1
        imprimeTrayecto (f, trayecto(g, i));              //      imprimeTrayecto (f, g->tr_recientes[i]);
    dev OK;
```

d) Dada la implementación en C de los TADs que has propuesto, proporciona el código C, sin control de errores, de la siguiente función, que añade un trayecto a la lista de trayectos almacenada en el GPS, insertando en el GPS una copia del trayecto que recibe como argumento:

```
status añadeTrayectoAlGPS (GPS *pGPS, const trayecto *pTray){
    pGPS -> tr_recientes [pGPS->n_trayectos] = copiaTrayecto ( pTray);  //copia el trayecto en la última posición
    pGPS->n_trayectos ++;                                               //incrementa nº trayectos
        return OK;
```

e) Implementa en C y sin control de errores una función principal main que realice las siguientes acciones:
   - Crear un GPS.
   - Crear un trayecto con origen en la localización (34, 58) y destino en la localización (41,71), y almacenar ese trayecto en el GPS.
   - Imprimir por pantalla la información almacenada en el GPS.

   Después de imprimir, ya no se va a trabajar más con el GPS.

   Gestiona adecuadamente la memoria utilizada.

```
void main ( ) {
        GPS *g =NULL;
        trayecto *t=NULL;
        localiza *origen =NULL, *destino=NULL;
        g = creaGPS ();
        origen=creaLocalizacion (34, 58);
        destino=creaLocalizacion (47, 71);
        t=creaTrayecto (origen, destino);
        libera_localizacion (origen);  //Libera las localizaciones. La función creaTrayecto copia las localizaciones
        libera_localizacion (destino);  //origen y destino → liberar estas variables, ya no hacen falta.
        añadeTrayectoAlGPS (g, t);
        libera_trayecto (t);   //Libera t. añadeTrayectoAlGPS hace una copia del trayecto y la guarda en el GPS
        imprimeTrayectosDeGPS (stdout, g);
        libera_GPS(g);    //Libera el GPS (esta función llama a liberar trayectos, esta a liberar localizaciones, etc.)
}
```

3. App

Una empresa de telefonía desea reformar su aplicación (App) que gestiona los contactos que se almacenan en sus terminales móviles. Nuestro cometido es diseñar los TADs que serán utilizados en la nueva propuesta, la cual tiene los siguientes requisitos:

- La nueva App podrá gestionar, para el usuario de la misma, sus diferentes agendas (se permitirá un máximo de 100 agendas). Por ejemplo: agenda con los contactos personales, agenda con los contactos del trabajo, agenda con los contactos del club de ajedrez, etc. Por lo tanto, la App tendrá los datos del propietario del móvil y el conjunto de sus agendas.
- Cada agenda tiene un nombre ("mi trabajo", "personal", "club de ajedrez", etc.) y está compuesta por contactos (con un máximo de 10.000 contactos por agenda). De cada contacto se desea almacenar su nombre, sus apellidos y los diferentes medios disponibles para que sea contactado.
- Los posibles medios para contactar a un contacto son: direcciones postales, números de teléfonos y direcciones de correo electrónico. Para cada uno de estos medios, además de almacenar su valor ("911234567", "nombre.apellido@uam.es", etc.), se indicará si este medio es personal o de trabajo. Se permitirá guardar un máximo de 20 medios por cada contacto.

a) ¿Cuántos Tipos Abstractos de Datos necesarios son necesarios para representar lo descrito anteriormente y qué nombre darías a cada uno de ellos? Los tipos Status y Bool se encuentran ya definidos (no se consideran aquí)

b) Escribe el código en C de las Estructuras de Datos (EdD) necesarias para implementar los TADs que has respondido en la pregunta anterior, y di en qué ficheros se escribirían esos códigos.

Se cuenta con las siguientes definiciones:

```
typedef enum {ERROR=0, OK=1} Status;

typedef enum {FALSE=0, TRUE=1} Bool;

typedef enum {PERSONAL=1, TRABAJO=2} Tipo;

#define MAX_AGENDAS 100

#define MAX_CONTACTOS 10000

#define MAX_MEDIOS 20

#define MAX_TXT 124
```

c) Dados los siguientes prototipos, escribe en C el código, sin control de errores, de estas funciones:

- Medio *medio_ini(char *dato, Tipo tipo); /* crea un medio para contactar a un contacto a partir de un dato y su tipología/tipo */
- Status contacto_add_medio (Contacto *pContacto, Medio *pMedio); /*agrega un medio de contactar a un contacto dado*/
- void app_free (App *pApp); /*libera toda la memoria ocupada por una App, liberando todos sus componentes*/

d) Implementa en C y sin control de errores una función principal main que realice las siguientes acciones:

- Crea un contacto con estos datos:
  i. Ana Perez Lopez
  ii. Teléfono trabajo: 916788761
  iii. Email personal: ana.pl@person.es
- Introduce dicho contacto en una agenda personal del propietario de la App del móvil que tiene estos datos:
  i. Juan Gil Mayor
  ii. Telefono personal: 676767676
  iii. Email trabajo: juan.gm@empresa.es
- Imprime por pantalla los contactos de las agendas del propietario de la App del móvil antes indicado.

  Notas: Después de imprimir, ya no se va a trabajar más con las agendas de la App del móvil.
  Gestiona adecuadamente la memoria utilizada.

Para la realización de este código se cuenta con los prototipos de las siguientes funciones:

```
Medio *medio_ini(char *dato, Tipo tipo);
Contacto *contacto_ini(char *nombre, char *apellidos);
Status contacto_add_medio (Contacto *pContacto, Medio *pMedio);
Agenda *agenda_ini (char *tipo);
Status agenda_add_contacto (Agenda *pAgenda, Contacto *pContacto);
App *app_ini (Contacto *pContacto);
Status app_add_agenda (App *pApp, Agenda *pAgenda);
int medio_print (FILE *pf, Medio *pMedio);
int contacto_print (FILE *pf, Contacto *pContacto);
int agenda_print (FILE *pf, Agenda *pAgenda);
int app_print (FILE *pf, App *pApp);
void medio_free (Medio *pMedio);
void contacto_free (Contacto *pContacto);
void agenda_free (Agenda *pAgenda);
void app_free (App *pApp);
```

SOLUCION

a)

App Agenda Contacto Medio

b)

```
//medio.c
struct _Medio {
   char dato[MAX_TXT];
   Tipo tipo;
};
//contacto.c
struct _Contacto {
   char nombre[MAX_TXT];
   char apellidos[MAX_TXT];
   Medio *medios[MAX_MEDIOS];
   int numMedios;
};
//agenda.c
struct _Agenda {
   char tipo[MAX_TXT];
   Contacto *contactos[MAX_CONTACTOS];
   int numContactos;
};
//app.c
struct _App {
   Contacto *propietario;
   Agenda *agendas[MAX_AGENDAS];
   int numAgendas;
};
```

c)

```
Medio *medio_ini(char *dato, Tipo tipo){
      Medio *pMedio;

      pMedio = (Medio *) malloc (sizeof(Medio));
      pMedio->tipo = tipo;
      strcpy (pMedio->dato, dato);
      return pMedio;
}
Status contacto_add_medio (Contacto *pContacto, Medio *pMedio) {
   pContacto->medios[pContacto->numMedios] = pMedio;
   pContacto->numMedios++;
   return OK;
}
void app_free (App *pApp) {
   int i;
   for (i=0; i < pApp->numAgendas;i++)
        agenda_free (pApp->agendas[i]);
   contacto_free (pApp->propietario);
   free (pApp);
}
```

d)

```
int main(int argc, char** argv) {
   App *pApp;
   Agenda *pAgenda;
   Contacto *pContacto1, *pContacto2;
   Medio *pMedio1,*pMedio2,*pMedio3, *pMedio4;
   Status temp;

   pMedio1 = medio_ini ("916788761", TRABAJO);
   pMedio2 = medio_ini ("ana.pl@person.es", PERSONAL);
   pContacto1 = contacto_ini("Ana", "Perez Lopez");
   contacto_add_medio (pContacto1, pMedio1);
   contacto_add_medio (pContacto1, pMedio2);

   pMedio3 = medio_ini ("676767676", PERSONAL);
   pMedio4 = medio_ini ("juan.gm@empresa.es ", TRABAJO);
   pContacto2 = contacto_ini("Juan", "Gil Mayor");
   contacto_add_medio (pContacto2, pMedio3);
   contacto_add_medio (pContacto2, pMedio4);

   pAgenda = agenda_ini ("agenda_personal");
   temp =  agenda_add_contacto (pAgenda, pContacto1)    ;

   pApp = app_ini (pContacto2);
   app_add_agenda (pApp, pAgenda);

   app_print (stdout,pApp);

   app_free (pApp);
   return 1;
}
```

4. Listas de la compra

Se desea desarrollar una aplicación móvil que permita a un usuario crear y gestionar sus listas de la compra (pueden ser varias), cada una de ellas con sus artículos correspondientes.

El usuario se identificará con un nombre y una contraseña, y podrá dar de alta un máximo de 50 listas de la compra.

Una lista de la compra (identificada mediante una cadena de caracteres cualquiera) está compuesta por un conjunto de productos, incluyendo para cada uno de ellos, además de su nombre, la cantidad y la unidad de medida deseadas. La lista también contendrá el nombre del establecimiento en el que el usuario desea comprar esos productos. Por ejemplo, para el usuario de nombre Ángel y con clave TodoRico se tendría:

Ángel	
Varios	Fruta
Carrefive	Mentadona
□ 1 docena de huevos	□ 1 kg. de fresas
□ 1 bote de crema de cacao	□ 3 kg. de naranjas
□ 1 brick de nata para montar	□ 1 kg. de kiwis
□ 1 botella de leche	

Finalmente, en cada lista la aplicación permitirá marcar (o desmarcar) cada producto para indicar si ya se ha comprado o no. Con estas especificaciones, y asumiendo la existencia de tipos de datos genéricos (p.e. status, boolean) responde a las siguientes cuestiones:

a) (1 punto) Enumera los TADs que habría que definir para almacenar y gestionar la información descrita anteriormente, indicando su nombre y qué información se guardaría en cada uno de ellos.

- Usuario: nombre, clave y conjunto de listas
- Lista: nombre, establecimiento y conjunto de productos
- Producto: nombre, tipo de medida, cantidad y marca (comprado ya o no).

Escribe en C las estructuras de datos que utilizarías para implementar los TADs anteriores, garantizando al máximo la abstracción de los detalles de la implementación y optimizando el uso de la memoria. Puedes utilizar, si lo deseas, TADs estudiados durante el curso.

```
#define MAXN 20
#define MAXLISTAS 50
#define MAXPRODS 50

struct _USUARIO{
  char nombre [MAXN];
  char clave [MAXN];
  Lista * plistas [MAXLISTAS];
  int nlistas;
};

struct _LISTAC{
  char nombre [MAXN];
  char establec [MAXN];
  Producto * pProds [MAXPRODS];
  int nProds;
};
```

```
struct _PRODUCTO{
  char nombre [MAXN];
  char medida [MAXN];
  int cantidad;
  bool comprado;
};

typedef struct _USUARIO  Usuario;
typedef struct _LISTA ListaC;
typedef struct _PRODUCTO Producto;
```

Nota (no se preguntaba, no hace falta):

Las estructuras se escriben en los ficheros .h (usuario.h, listac.h, producto.h) y la definición de los TADs en los correspondientes ficheros .c.

b) (0,25 puntos) Escribe el prototipo de una función C que imprima en un flujo de salida dado (ej: fichero abierto) todos los datos de un producto de la lista, incluyendo si se ha comprado ya o no, y devuelva el nº de caracteres impresos.

```
int producto_imprimir (FILE *pf, Producto *pprod);   //también vale añadir como arg la lista, si se
                                                     //desea comprobar si pprod está en la lista
```

¿En qué fichero estará este prototipo? __producto.h_____

c) (0,5 puntos) Escribe, sin control de errores, el código C de una función que, dado un usuario, imprima su nombre y las listas de la compra que tenga, incluyendo toda la información de cada una de ellas (ver ejemplo en el enunciado). La función debe devolver el nº de caracteres impresos.

Si las necesitas, puedes utilizar las funciones de prototipos:

```
int entero_imprimir (FILE * pf, int entero);
int cadena_imprimir (FILE * pf, char * cadena);
int lista_imprimir ((FILE *pf, ListaCompra *pl);
```

```
int usuario_imprimir (FILE *pf, Usuario *pu) {
    int i, numcars;
    numcars = cadena_imprimir (pf, pu->nombre);
    for (i=0; i<pu->numListas; i++)
        numcars += lista_imprimir (pf, pu->pListas [i]);
    return numcars;
}
```

¿En qué fichero estará esta función? _usuario.c_____

d) (0,5 puntos) Escribe, con control de errores, el código C de una función que cree una lista de la compra nueva y la devuelva inicializada (vacía).

```
ListaC * creaLista (){
        ListaC * pl=NULL;
        pl = (Lista *) malloc (sizeof (Lista));
            if (pl == NULL)
                return NULL;
            pl->nProds = 0;
            for (i=0; i< pl->nProds; i++)
                pl->pProds[i] = NULL;   //opcional si se tiene el campo nProds, obligatorio si no.
            strcpy (pl->nombre, "");  //opcional
            strcpy (pl->establec, ""); //opcional
            return pl;
}
```

¿En qué fichero estará esta función? _lista.c_____

e) (0,25 puntos) ¿De qué otra forma podrías haber implementado las listas de la compra? ¿Esa forma sería más/menos sencilla de implementar que la que has escogido tú? ¿Su gestión sería más/menos eficiente? Justifica tu respuesta.

Podría haber utilizado listas enlazadas. Esa forma sería muy sencilla de implementar si tengo la biblioteca de listas disponible (solo tengo que llamar a las funciones). Si no, podría ser más complicado de implementar que el array por el manejo de punteros y memoria dinámica. No obstante, la gestión de los productos sería más eficiente, ya que en el array de punteros a producto cada vez que quiero eliminar un producto de la lista tengo que mover todos los demás (recolocar el array para que los huecos queden al final), mientras que con listas solo hay que reasignar los punteros. La inserción de un nuevo producto en ambos casos puede ser inmediata: en el array se inserta directamente en la última posición, sin recorrerlo (si tengo el campo nProds) y en la lista puedo insertar por el principio. Por otra parte (esto no se pregunta, pero ya que estamos…) con listas puedo tener justo la memoria que necesito, ni más ni menos. Con el array tengo que realizar una estimación.