	Universidad Tecnológica Nacional **Facultad Regional Buenos Aires** **Departamento de Ingeniería Electrónica**	**Informática I** 13 de Julio de 2024 1° Parcial

Preguntas Teóricas

1. Explique la diferencia entre la dirección y el contenido de una variable.

2. Explique qué es un vector en C ¿Y una matriz?

3. ¿Por que el uso de punteros a función puede ser mejor que usar un switch/case?

4. Dado la siguiente declaración de variables:

```
unsigned char c1,c2,c3,c4;
int v[6] = {12,45,32,11,5,28};
float f1,f2,f3;
```

Asumiendo que el sistema operativo nos asigna la dirección 0xcde42520 para el comienzo del mismo, números de punto flotante IEEE754 de precisión simple y enteros de 4bytes responda:

a) ¿Qué dirección de memoria contendrá el valor de v[2] ?

b) ¿Qué dirección de memoria resultará de resolver v+3 ?

c) Recibiríamos algún error durante la compilación o el linkeo si tratáramos de acceder al contenido de v+10? Justifique su respuesta.

d) Indique en que posición de memoria comienza la variable f1.

5. Exprese los siguientes números binarios en código hexadecimal, octal, y decimal (CA2):

a) 10010101 01100110

b) 01010110 10110111

Parte Practica

Se desea desarrollar un programa que administre la información de códigos de patentes que son recibidos por línea de comandos.

El programa debe leer la información y crear un listado con las patentes válidas, y luego ordenarlas. El programa se invoca de la siguiente manera:

```
./patentes <PAT1> <PAT2> <PAT3> … <PAT_N> [-o <orden>]
```

Las patentes válidas son aquellas que cumplen el siguiente formato:

```
AB123CD
[Letra][Letra][Num][Num][Num][Letra][Letra]
```

No se sabe de antemano la cantidad de parámetros con los que se ejecutará el programa. El parámetro <orden> podrá ser mayor a 0 para orden ascendente, o menor a 0 para descendente. Este parámetro puede encontrarse en cualquier posición cuando se ejecuta el programa. Es un parámetro opcional, de modo que en caso de no ser entregado el parámetro, se asumirá orden ascendente.

Ejemplo #1 de invocación del programa (3 Patentes Válidas, Orden Ascendente):

```
[IN]:  ./patentes AB123CD AA356FD 1A2B3C4 AA34ABC BA765AA -o 4
[OUT]: datos[0]=AA356CD
       datos[1]=AB123CD
       datos[2]=BA765AA
```

Ejemplo #2 de invocación del programa (3 Patentes Válidas, Orden Descendente):

```
[IN]:  ./patentes AB123CD -o -5 BA765AA HolaMundo AA111BB 123456
[OUT]: datos[0]=BA765AA
       datos[1]=AB123CD
       datos[2]=AA111BB
```

Se pide desarrollar las siguientes funciones:

1) main()

Desarrollar la función main para que resuelva el problema enunciado, invocando las funciones adicionales que se detallan en los puntos 2, 3, 6 y 7. Puede agregar otras funciones si lo cree conveniente.

2) int validar_orden(char** argv, int argc, int* orden)

Función que analiza si existe parámetro de ordenamiento en alguna posición dentro de los datos con los que se invocó al programa principal. El valor apuntado por <orden> es modificado por referencia según corresponda. En caso de éxito retorna [0], y [-1] en caso de error.

3) char** cargar_datos(char** argv, int argc, int* cant)

Función que analiza todos los argumentos ingresados y va creando un arreglo con los datos de las patentes válidas. Recibe la lista completa de argumentos <argv> y su cantidad <argc>. El valor apuntado por <cant> es modificado por referencia con la cantidad efectiva de parámetros válidos. Al terminar su ejecución, en caso de éxito, retorna el valor del puntero al inicio de esta lista de datos válidos. En caso de error, retorna [NULL] con una leyenda informativa correspondiente. Tener en cuenta que esta función en una sola ejecución debe cumplir su objetivo.

4) int validar_patente(char* pat)

Función que analiza si el código de patente ingresado es válido. Devuelve [0] en caso de patente válida y [-1] en caso de error.

5) int ordenar_datos(char** datos, int cant, int orden)

Función que ordena los datos válidos. Recibe el vector de datos a ordenar, la cantidad de elementos y el parámetro de orden seleccionado ([1]:ASC [-1]:DESC). Devuelve [0] en caso de éxito y [-1] en caso de error. El criterio de ordenamiento queda a criterio del alumno.

6) ingresar_vehiculos(char** datos, int cant)

Una vez cargado el listado de patentes y ya ordenado, el programa deberá interactuar con el usuario para recibir los datos de los autos asociados a cada patente. Para esto le pedirá una patente al usuario , la buscará en el vector de pantentes y si existe crea una estructura con datos que consigue llamando a la función char* dato_pantente(char* pat). Esta función devuelve un string separado por puntos y comas que se debe separar y alojar en una estructura del siguiente tipo:

```
struct auto {
      char marca[20];
      char tipo[10];
      char modelo;
      int num_motor;
      float peso;
};
```

Por ejemplo si recibe el string `ford;sedan;2022;1345677;1020.2` lo deberá separar y alojar adecuadamente en cada una de los campos de la estructura. Puede usar funciones de librería para realizar las conversiones de tipos.

Para estos datos deberá crear un vector de al menos 10 estructuras que debe poder crecer en caso de que se necesiten más datos. Alojar este vector de una manera que pueda cumplir con estos requerimientos.

Esta función termina cuando el usuario ingresa una patente sin datos (string vacío).

7) Realizar una función que una vez terminado el ingreso, muestre por pantalla los datos ingresados en el vector de estructuras. Elija los datos que debe pasarle a esta función para que funcione adecuadamente.