TPN°7: Programación Avanzada – Uso de Heap

En esta práctica ninguna función (incluyendo la función main) debe tener más de 20 líneas de código.

Todas las funciones deben validar los datos, indicando los problemas a través de códigos de error.

Ejercicio 1

Resolver los siguientes ejercicios del libro de texto: **5-4 y 5-5**

Ejercicio 2

Escribir un programa que convierta enteros de una base a otra. Los números entre y después de los símbolos '<' y '>' indican la base de entrada y de salida respectivamente. Dichos números serán decimales entre 2 y 10.

Ejemplo: la cadena de entrada ***< 5 > 2*** *indica que leerá números enteros en base 5 y los imprimirá en su equivalente binario. Luego, si se ingresa* ***41*** *se lo deberá pasar a* ***10101.***

Ejercicio 3

Se pretende expresar números reales positivos en notación exponencial normalizada con un único dígito para la parte entera y cantidad variable de dígitos para la mantisa.

Para esto se pide escribir la función entera ***NotacionExp*** que recibe dos parámetros:

- una cadena de caracteres de entrada conteniendo el número original.
- una cadena de caracteres de salida donde dejará el número normalizado, eliminado los ceros no significativos.

En caso de error en la entrada, la función debe devolver el valor cero y dejar en la respuesta el string vacío, caso contrario debe retornar el valor uno.

El número original puede no tener parte fraccionaria, esto implica que "123.0" es válido y "123" también; y en ambos casos la respuesta será "1.23E+02".

Ejemplos:

Texto Original	*Respuesta*
"1"	"1E+00"
"0012"	"1.2E+01"
"145.720"	"1.4572E+02"
"134597"	"1.34597E+05"
"0.34"	"3.4E-01"
"0.00200"	"2E-03"
"987654321011"	"9.87654321011E+11"

Ejercicio 4

Existe un método para encriptar mensajes que consiste en mezclar las letras de un alfabeto y reemplazar cada letra del mensaje por la nueva letra que ocupa su lugar en el alfabeto.

Se pide escribir tres funciones:

- **crearAlfabeto**: recibe como parámetro de salida un vector de caracteres y lo completa con el alfabeto mezclado (letras 'A' a 'Z'). El alfabeto mezclado debe armarse utilizando un ciclo ***for.***
- **codificar**: que recibe dos parámetros de entrada que representan un mensaje a codificar y un alfabeto, y un parámetro de salida en el cual se devuelve el mensaje codificado.
- **decodificar**: recibe dos parámetros de entrada que representan un mensaje a decodificar y un alfabeto, y un parámetro de salida en el cual se devuelve el mensaje decodificado.

Ejemplo:

Alfabeto original = { A, B, C , D, E, F,G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z }
Alfabeto mezclado = { C, I, N, Q, U, X, A, E, H, O, R, T, Z, B, G, K, M, Y, D, L, S, V, F, J, P, W }

Si se codifica el mensaje **"Abeja Reina"** se obtiene el mensaje **"CIUOC YUHBC".**
Si se decodifica el mensaje **"CYAUBLHC"** se obtiene el mensaje **"ARGENTINA"**.

Ejercicio 5

Sean Nombre y Curso dos arreglos que contienen el nombre y el curso('G' o 'H') de cada uno de los alumnos de una materia. Escribir una función que reciba dichos arreglos como parámetros de entrada y retorne en parámetros de salida otros dos arreglos CursoG y CursoH con el nombre de los alumnos separados por curso. El string vacío en el nombre indica fin del arreglo.

Ejercicio 6

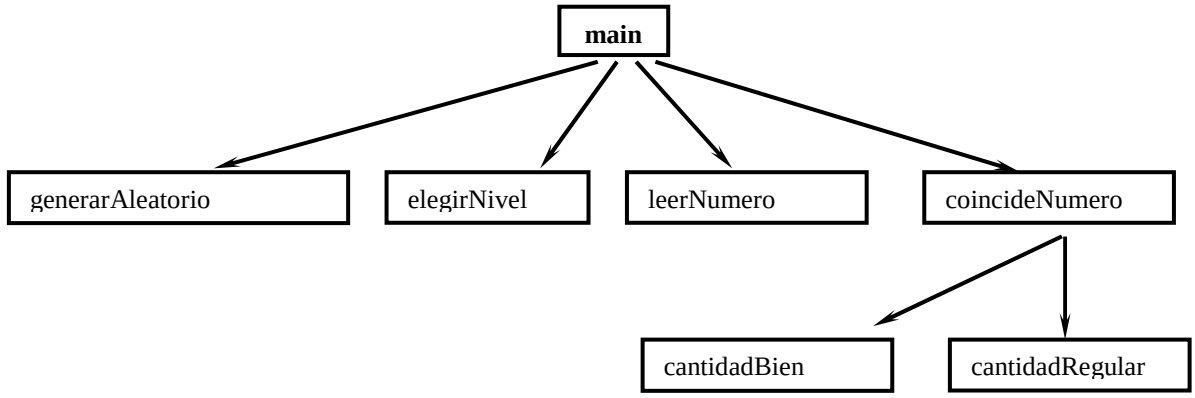
Escribir un programa para desarrollar el juego Mastermind descripto por las siguientes reglas:

- La computadora construye un conjunto aleatorio de X dígitos distintos (para X usar una constante simbólica).
- Se le pide ingresar al usuario el nivel en que quiere jugar. Según esto se trabajará con una tolerancia de hasta N pasos, dentro de los cuales si no logra ganar el juego se termina automáticamente.
- El jugador entra un conjunto de X dígitos distintos por teclado.
- Si el número propuesto por el jugador coincide con el que generó la máquina o la cantidad de chances excedió el número tope N , el juego finaliza; sino el programa debe informarle cuántos dígitos están BIEN (son correctos y están en su lugar) y cuántos son REGULARES (son correctos pero están fuera de lugar) y se vuelve a ejecutar el paso anterior. Así se va guiando al jugador hasta que adivine el número.

Ejemplo:

*Para **X = 5** y **N = 3**, suponiendo que el número escondido por la computadora fuera 98532, un diálogo posible sería:*

☺ *número introducido por el jugador:* 1**952**4
💻 *respuesta de la computadora:* 1 bien, 2 regular
☺ *número introducido por el jugado:* **9**1**52**7
💻 *respuesta de la computadora:* 2 bien, 1 regular
☺ *número introducido por el jugado:* **9852**6
💻 *respuesta de la computadora:* 3 bien, 1 regular
Lo siento, el número es **98532**



Nombre	*Descripción*	*Parámetros*	
main	Permite jugar al MasterMind	No tiene	
GeneraAleatorio	Arma un arreglo de X elementos no repetidos	**incognita**	Arreglo de enteros De salida V.F.: enteros positivos no repetidos
elegirNivel	Devuelve un número entero positivo entre uno y diez	No tiene	
leerNumero	Solicita al usuario ingresar un número de X dígitos no repetidos	**numero**	Arreglo de enteros De salida V.F.: enteros positivos no repetidos
coincideNumero	Dados dos arreglos de enteros, devuelve un valor booleano indicando si ambos arreglos son exactamente iguales o no. Imprime un cartel indicando cuántos elementos coinciden en valor y posición (BIEN) y cuántos solo en valor (REGULAR).	**incognita**	Arreglo de enteros De entrada V.I.: enteros positivos no repetidos
		numero	Arreglo de enteros De entrada V.I.: enteros positivos no repetidos
cantidadBien	Dados dos arreglos de enteros, devuelve cuántos de sus elementos coinciden en valor y posición	**incognita**	Arreglo de enteros De entrada V.I: enteros positivos no repetidos
		numero	Arreglo de enteros De entrada V.I: enteros positivos no repetidos
cantidadRegular	Dados dos arreglos de enteros, devuelve cuántos de sus elementos coinciden en valor pero no en posición	**incognita**	Arreglo de enteros De entrada V.I.: enteros positivos no repetidos
		numero	Arreglo de enteros De entrada V.I.: enteros positivos no repetidos

Ejercicio 7

Rehacer el problema del ejercicio anterior, pero considerando que la cantidad X de dígitos sea elegida por el usuario. ¿Cambia el árbol de módulos? ¿Cambia la tabla de interfaces?

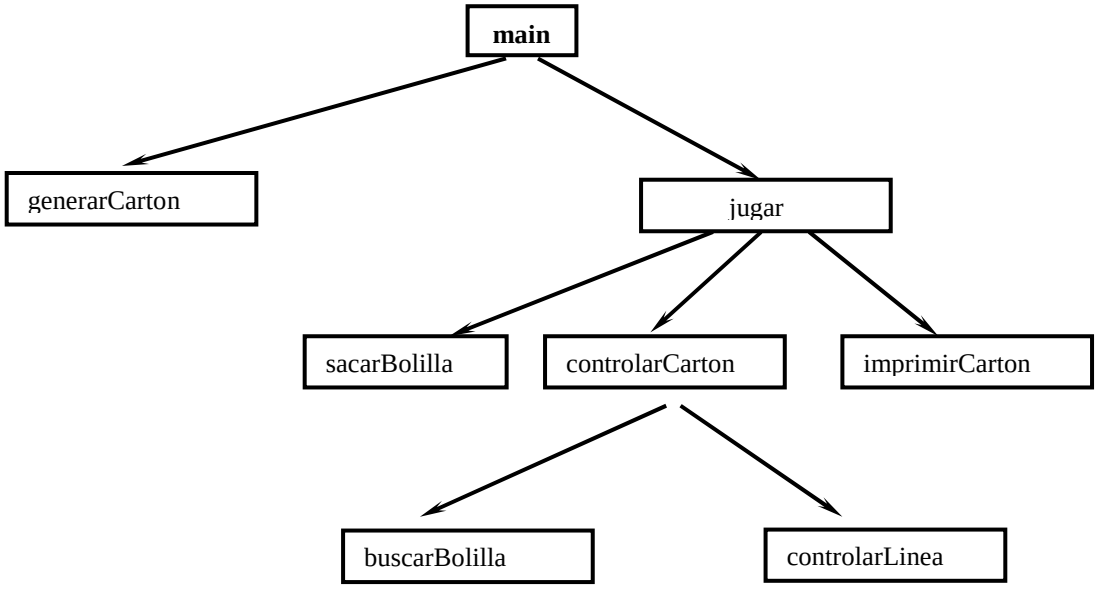
Ejercicio 8

Escribir un programa que realice la simulación de un bingo para dos jugadores, el cual imprimirá paso a paso la bolilla extraída y los números faltantes (que aún no salieron)de cada cartón. Las bolillas están representadas por los números del 1 al 90 y cada jugador tiene un cartón con 15 números distintos. Los cartones son generados por el programa en forma aleatoria. No es necesario conservar los números que se van marcando en cada cartón (cuando hay coincidencia con la bolilla extraída)

El tipo de datos a utilizar para el cartón será :

```
typedef  int TipoLinea[5];
typedef  TipoLinea  TipoCarton [3];
```

Se da a continuación un esquema del diseño del mismo.



Nombre	*Descripción*	*Parámetros*	
Main	Simula un juego de bingo. Genera dos cartones, uno para cada jugador y va extrayendo bolillas hasta que alguno de los jugadores hace bingo. Imprime el o los ganadores.	No tiene	
GenerarCarton	Genera un cartón de bingo en forma aleatoria.	**carton**	***TipoCarton*** De salida V.F.: 15 enteros distintos entre 1 y 90

Jugar	Va extrayendo bolillas hasta que alguno de los dos jugadores hace bingo. Con cada extracción se muestra el estado de los cartones e imprime el ganador de la primera línea. Devuelve un número entero con un bit en uno por cada jugador que hizo bingo.	**bolillero**	Arreglo de enteros De entrada / salida V.I: vector de enteros con el bolillero inicial.
		jugador1, jugador2	***TipoCarton*** De entrada / salida V.I: 15 números distintos entre 1 y 90, que representan el cartón de cada jugador.
SacarBolilla	Devuelve un número de bolilla aún no extraída y actualiza el bolillero y la cantidad de bolillas que aún quedan en el mismo.	**bolillero**	Arreglo de enteros De entrada / salida V.I: vector de enteros que representa las bolillas que aún no han sido extraídas. V.F: ídem V.I. pero sin la bolilla extraída.
		cantBolillas	Entero positivo De entrada / salida V.I: cantidad de bolillas aún no extraídas. V.F: cantidad de bolillas que quedan en el bolillero.
controlarCarton	Dado un cartón y una bolilla extraída, verifica si pertenece al mismo, en cuyo caso lo marca y controla si se formó línea o bingo. Devuelve la cantidad de líneas completadas del cartón con esa bolilla.	**carton**	***TipoCarton*** De entrada / salida V.I: cartón con los números faltantes.
		bolilla	Entero De entrada V.I: número de bolilla extraída.
imprimirCarton	Imprime el estado de un cartón. Se imprimen por cada línea los números que no han sido marcados aún.	**carton**	***TipoCarton*** De entrada V.I: cartón con los números faltantes.
buscarBolilla	Dado un cartón y la bolilla extraída la busca entre los números restantes y si la encuentra lo marca y devuelve 1 (verdadero), en caso contrario devuelve 0 (falso).	**carton**	***TipoCarton*** De entrada / salida V.I: cartón con los números faltantes.
		bolilla	Entero De entrada V.I: número de bolilla extraída
controlarLineas	Recibe una línea del cartón y devuelve 1 si la misma está completa (tiene todos sus números ya marcados) y 0 en caso contrario.	**linea**	***TipoLinea*** De entrada V.I.: línea, con nros marcados

Ejercicio 9

Rehacer el problema del ejercicio anterior para N jugadores, siendo N:

a) una constante simbólica.
b) Un número ingresado por el usuario al iniciar el juego.

Realice la menor cantidad de cambios posible en la interfase de las funciones.

Ejercicio 10

Escribir un programa interactivo para jugar al juego del ahorcado. Las palabras que almacena la máquina deben estar guardadas en un arreglo de strings constantes y cada vez que se elija jugar, utilizar aleatoriamente alguna de dichas palabras.

Antes de escribir el programa, realizar el diseño y la modularización, documentando cada módulo interviniente, tal como se mostró en los ejercicios 8 y 10.

Ejercicio 11

Indicar -de ser posible- cual es la salida estándar en cada uno de los siguientes fragmentos de código, suponiendo que previamente se definió la función ***aMayusculas*** de la siguiente forma:

```
char *
aMayusculas( char * s)
{
      char * t;
      for ( t = s; *t = toupper(*t); t++)
            ;

      return s;
}
```

a)

```
{
      int v[] = {1,2,3,0};

      printf( "%s\n", aMayusculas(v + 1));
      printf( "%s\n", v);
}
```

b)

```
{
      char * p = "Hola mundo";

      printf ("%s\n", aMayusculas(p));
      printf ("%s\n", p);
}
```

c)

```
{
      char p[] = "Hola mundo";

      printf ("%s\n", aMayusculas(p) + 2);
      printf ("%s\n", p);
}
```

Ejercicio 12

Dada la declaración:

```
char *p[2][3] =  {{"abc","defg","hi"},{"jklmn","pqrstuvw","xyz"}};
```

a) ¿Cuántos bytes de memoria ocupa la declaración?
b) Completar el siguiente cuadro:

Expresión	*Expresión Equivalente*	*Valor*
`***p`	`p[0][0][0]`	`'a'`
`**p[1]`		
`**(p[1]+2)`		
`*(*(p+1)+1)[7]`		
`(*(*(p+1)+1))[7]`		
`*(p[1][2]+2)`		

Ejercicio 13

¿ Qué es lo que realiza este programa ?

```
#include <stdio.h>
void misterio1 ( char *s1, const char *s2);

int
main(void)
{
      char string1[80], string2[80];
      printf ("Ingrese dos cadenas: ");
      scanf("%s%s", string1, string2);
      misterio1(string1,string2);
      return 0;
}

void
misterio1 ( char *s1, const char *s2)
{
      while (*s1 != '\0')
            ++s1;
      for ( ; *s1 = *s2; s1++, s2++)
            ;
}
```

Ejercicio 14

¿ Qué es lo que realiza este programa ?

```
#include <stdio.h>
int misterio2 ( const char *s);

int
main ( void )
{
      char string[80];
      printf ("Ingrese una cadena: ");
      scanf("%s", string);
      printf("%d\n",misterio2(string));
      return 0;
}

int
misterio2 ( const char *s)
{
      int x=0;
      for ( ;  *s != '\0'; s++)
            ++x;
      return x;
}
```

Ejercicio 15

Encontrar el error en cada uno de los segmentos de programa. De ser posible, explicar cómo corregirlo:

a)
```
int *numero;
printf ("%d\n", *numero);
```

b)
```
float *realPtr;
long *integerPtr;
integerPtr = realPtr;
```

c)
```
int *x, y;
x = y ;
```

d)
```
char s[] = "esta es una cadena de caracteres";
for ( ; *s != '\0'; s++)
      printf("%c ", *s);
```

e)
```
short *numPtr, result, auxiliar=0;
void *genericoPtr;
numPtr = &auxiliar;
genericoPtr = numPtr;
result =  * genericoPtr + 7;
```

f)
```
char *s;
printf("%s\n", s);
```

Ejercicio 16

Se tiene una función con la siguiente declaración:

static int x[8] = { 10, 20, 30, 40, 50, 60, 70, 80 };

a) ¿Qué se referencia con **x** ?
b) ¿Qué se referencia con **(x+2**)?
c) ¿Qué se referencia con ***x** ?
d) ¿Qué se referencia con ***x+2** ?
e) ¿Qué se referencia con ***(x+2)** ?
f) Suponiendo que x[0] esté almacenado en la dirección de memoria $10F0, en qué dirección de memoria se encuentra almacenado **x[4]** ?

Ejercicio 17

Se tiene una función con la siguiente declaración:

static float tabla[2][3] = { { 1.1, 1.2, 1.3 }, { 2.1, 2.2, 2.3 } };

Graficar el almacenamiento en memoria del arreglo **tabla** (a partir de la dirección $1000) y decir qué tipo de dato es y qué se referencia con:

a) tabla
b) tabla+1
c) *(tabla + 1)
d) *(tabla + 1) + 1
e) *tabla + 1
f) *(*(tabla + 1) + 1)
g) *(*tabla + 1)
h) *(*(tabla + 1))
i) *(*tabla + 1) + 1
j) tabla[1]
k) tabla[0] + 1
l) *(tabla[1]) + 1
m) *tabla[1]
n) tabla[0][1]

Ejercicio 18

Se tiene un programa con la siguiente declaración:

static char *color[6] = { “rojo”, “verde”, “azul”, “blanco”, “negro”, “amarillo” };

a) Graficar el almacenamiento en memoria del arreglo **color** (a partir de $2004).
b) ¿Qué se referencia con **color**?
c) ¿Qué se referencia con **(color+2**)?
d) ¿Qué se referencia con ***color**?
e) ¿Qué se referencia con ***(color + 2)**?
f) ¿En qué se diferencian **color[5]** y ***(color + 5)**?
g) ¿Qué se referencia con ****color**?
h) ¿Qué se referencia con ***color[0]**?
i) ¿Qué se referencia con ****color + 2**?

Ejercicio 19

Resolver los ejercicios **5.3**, **5.4** y **5.5** del Libro de Kernighan & Ritchie.

Ejercicio 20

Indicar y corregir los errores en cada una de las siguientes funciones.

a)

```
/* Copiar el string a uno nuevo pasado a mayusculas */
char *
pasaMayusculas(const char * s)
{
      char auxiliar[5000] ;
      int i;
      char * resp;

      for(i = 0; s[i] != 0; i++)
            auxiliar[i] = toupper(s[i]);

      auxiliar[i] = 0;
      resp = auxiliar ;
      return resp;
}
```

b)

```
/* Copiar un vector a otro pero en forma invertida.
** Se recibe el vector original y la dimensión del mismo
*/
int
copiaVector (int v[], int dim)
{
      int resp[], i;

      resp = malloc(dim);
      for (i = dim; i > 0; i -- )
            resp[i] = v[dim - i];

      return resp;
}
```

Ejercicio 21

Escribir una función que dado un string ***s*** y un número entero ***n*** retorne en su nombre un nuevo string con los primeros ***n*** caracteres del string ***s***. Si la longitud de ***s*** es menor a ***n*** debe retornar una copia de ***s***. En caso de que no se pueda copiar debe retornar NULL. ¿Qué cuidado debe tener el usuario de esta función?

Ejercicio 22

Escribir una función que reciba dos vectores, uno con los nombres y apellidos de los alumnos de un curso y un vector de enteros donde se almacene el promedio de cada alumno en el curso. La función debe retornar en su nombre un nuevo vector con los alumnos que hayan aprobado el curso, esto es, que tengan un promedio mayor o igual a 4.

Como indicador de final del arreglo de alumnos se tiene el string vacío.

El código de la función debe ser independiente de la forma en que se defina el vector de alumnos.

Ejemplos de invocación:

```
....
char alumnos [][15] = {"Ana", "Juan Pablo", "Mariana", "Fernando",
                       "Carolina", "" };
int notas[] = {8, 3, 10, 7, 5};
char * alumnos2 [] = {"Ana", "Juan Pablo", "Mariana", "Fernando", "Carolina",
                       "" };
....  alumAprob ...
....  alumAprob2 ...

alumnAprob = aprobados( alumnos, notas );
alumnAprob2 = aprobados( alumnos2, notas );

......
```

Ejercicio 23

Escribir una función que permita liberar la memoria reservada por la función del ejercicio anterior.