ANALISTA UNIVERSITARIO EN SISTEMAS
TALLER DE PROGRAMACION II

PRÁCTICA 2

Ejercicio 1: Arrays multidimensionales

Construya una calculadora para matrices de dimensión 3x3, la misma deberá ofrecer las siguientes operaciones, las cuales deben estar en un archivo de cabecera llamado funcsmat.h.

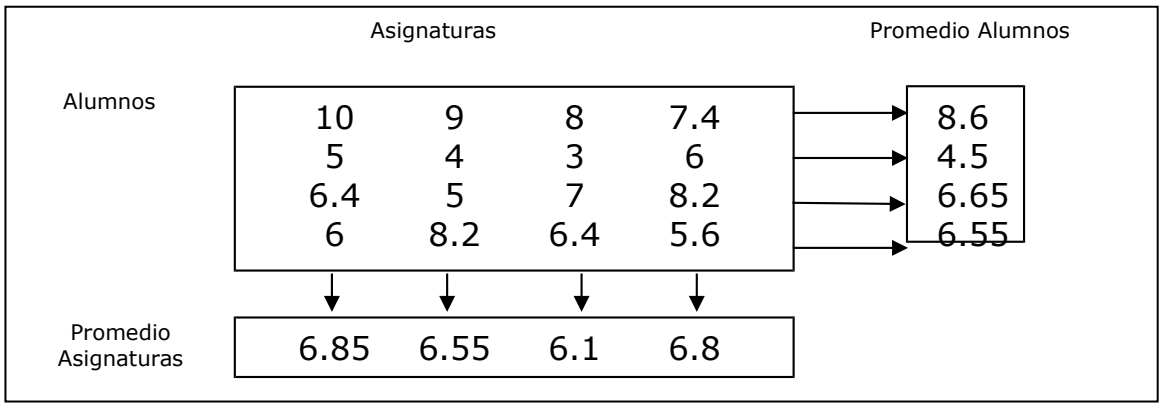
(a) suma de matrices
(b) producto de matrices
(c) producto de una matriz por un escalar
(d) traza de la matriz
(e) matriz traspuesta

Explique que modificaciones haría para que la calculadora maneje matrices cuadradas de dimensión n.

Ejercicio 2:

Un instituto desea controlar los resultados de los alumnos en las diferentes asignaturas de un curso de informática.
El programa debe leer las calificaciones obtenidas en las distintas asignaturas y visualizar en pantalla el número de cada estudiante seguido por su promedio, además se deberá imprimir al lado "*libre*", "*regular*" o "*promovido*", si dicho promedio está en los intervalos [0,6), [6,8) y [8,10] respectivamente.
El programa visualizará también la calificación promedio de todos los estudiantes de cada asignatura.

Ayuda: Organice los datos de la siguiente manera, los recuadros indican arreglos.



Ejercicio 3: Funciones recursivas

(a) Construya funciones recursivas que resuelvan las siguientes funciones:

. suma(a, b) = a + b
. prod(a, b) = a * b
. exp(a, b) = a^b
. factorial(n) = n!
. fib(n) = n-ésimo término de la sucesión de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...)

(b) De no haberlo hecho en el apartado **(a)** re-impleméntelas como *funciones recursivas de cola*

Ejercicio 4: Escriba el siguiente programa y observe que sucede:

```
#include <stdio.h>

int func () {
       static int b = 0;
       return  b++;
}

void main () {
      for (int i=0; i<10; i++)
           printf("%d ", func());
}
```

En base al código anterior implemente una función que retorne en llamadas sucesivas 0, 1, ...,5, 0, 1, ...,5, 0, 1, ..., 5, etc.

Ejercicio 5: Estructuras

Dadas las siguientes definiciones de estructura y asignaciones:

```
struct estudiante {
     char *apellido;
     int anyo_ingreso;
     char division;
};

struct estudiante nuevo, *pest = &nuevo;
nuevo.apellido = "Lopez";
nuevo.anyo_ingreso = 1998;
nuevo.division = 'a';
```

Determinar si las siguientes sentencias son correctas.
En caso de ser incorrectas explique porque, en caso de ser correctas indique cual es el valor que representan.

```
(a) nuevo->apellido;
(b) pest->division;
(c) (*pest)->apellido;
(d) *pest->apellido + 2;
(e) *(pest->apellido + 2)
(f) pest->apellido[2]
```

Ejercicio 6: Mediante un arreglo de 20 enteros simule:

- Una **pila** con las funciones: insertar, sacar, tamanyo, mostrar, vaciar, es_vacia
- Una **cola** con idénticas funciones que **pila**
- Una **lista** con las funciones: insertar (al principio, al final, en tal posición), sacar (al principio, al final, en tal posición), primerElemento, ultimoElemento, elementoEnPosicion, largo, mostrar, vaciar, es_vacia
- Una **lista circular** con las mismas operaciones que **lista**

Nota: Trate de aprovechar (reutilizar) al máximo las funciones realizadas en cada ítem.

Ejercicio 7:
Escribir un programa que lea una cadena de caracteres (NO utilice getchar), luego introducir un carácter por vez en una *pila* y en una *cola* simultáneamente. Cuando se encuentre el final de la cadena, utilice las funciones pilas y colas para determinar si la cadena es o no un *palíndromo*.

Ejercicio 8: Listas simplemente enlazadas
Crear un programa para manipular listas **simplemente enlazadas** y realizar las siguientes operaciones sobre las mismas:

. insertar (al principio, al final, en tal posición)
. sacar (al principio, al final, en tal posición)
. primerElemento, ultimoElemento, elementoEnPosicion
. largo
. mostrar
. vaciar
. es_vacia

Ejercicio 9: Listas doblemente enlazadas
Crear un programa para manipular listas **doblemente enlazadas** y realizar las siguientes operaciones sobre las mismas:

. insertar (al principio, al final, en tal posición)
. sacar (al principio, al final, en tal posición)
. primerElemento, ultimoElemento, elementoEnPosicion
. largo
. mostrar
. vaciar
. es_vacia

Ejercicio 10: Función filter
Dada una **lista de enteros L**, y un predicado p:*Z->Bool*, construir una función llamada **filter**, que elimine de L aquellos elementos que no verifiquen el predicado.
Ejemplo:
Si L=[1,2,3,4] y p es el predicado "es un número par", entonces filter(L) provocará que L=[2,4].

Extensión: Crear un arreglo de punteros a funciones (en este caso predicados) y presentar al usuario la posibilidad de elegir que predicado aplicar, ejemplos de predicados son: esPar, esImpar, esPositivo, esPrimo, esMult5, etc.
Ayuda: implemente filter como una función que recibe un puntero a función.

Ejercicio 11: Función merge
Dadas **dos listas de enteros L1 y L2**, generar una tercera, llamada **Merge**, que contenga una única vez los elementos de L1 y L2 ordenados de menor a mayor.
Ejemplo:
Si L1=[1,5,0,3] y L2=[5,2,8,1], deberá ser Merge=[0,1,2,3,5,8].

Ejercicio 12:
Dada **una lista L de n enteros:**
(a) construya una función llamada **cicloUno**, tal que si
L=[a_1, a_2, ..., a_n] produzca L=[a_2, ..., a_n, a_1].
(b) construya una función llamada ciclos, que reciba un entero *i* y realice *i* ciclos en la lista, es decir, si
L=[a_1, a_2, ..., a_{i-1}, a_i, a_{i+1}, ..., a_n], deberá quedar
L=[a_{i+1}, ..., a_n, a_1, a_2, ..., a_{i-1}, a_i].

Ejercicio 13: Función Zip
Dadas **dos listas L1 y L2 de números enteros**, construya una nueva lista de pares ordenados llamada **Zip**, donde las primeras componentes de los pares ordenados provienen de L1 y las segundas componentes de L2.
Ejemplos:
Si L1 = [1,3,2,5] y L2=[7,0,8,6], entonces
Zip = [(1, 7), (3, 0), (2, 8), (5, 6)]
Si L1 = [1,3,2,5] y L2=[9,4], entonces
Zip = [(1, 9), (3, 4)]
Si L1 y/o L2 son vacías, Zip será vacía.