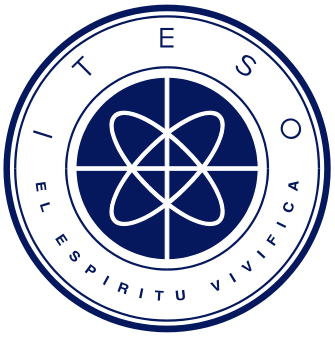
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 1. MANEJO DE APUNTADORES

Autor: Méndez Álvarez, Raúl

Presentación: 5 pts

Pruebas: 20 pts

Funcionalidad: 60 pts

24 de mayo de 2018. Tlaquepaque, Jalisco,

Todas las figuras e imagenes deben tener un título y utiilizar una leyenda que incluya número de la imagen ó figura y una descripción de la misma. Adicionalmente, debe de existir una referencia a la imagen en el texto.

La documentación de pruebas implica:
1) Descripción del escenario de cada prueba
2) Ejecución de la prueba
3) Descripción y análisis de resultados.

Instrucciónes para entrega de tarea

Es ***IMPRESCINDIBLE*** apegarse a los formatos de entrada y salida que se proveen en el ejemplo y en las instrucciones.

Esta tarea, como el resto, se entregará de la siguiente manera:

- **Reporte:** vía *moodle* en **un archivo PDF**.
- **Código:** vía su repositorio **Github**.

La evaluación de la tarea comprende:

- 10% para la presentación
- 60% para la funcionalidad
- 30% para las pruebas

Es necesario responder el apartado de conclusiones, pero no se trata de llenarlo con paja. Si no se aprendió nada al hacer la práctica, es preferible escribir eso.

Objetivo de la actividad

El objetivo de la tarea es que el alumno aplique los conocimientos y habilidades adquiridos en el tema de apuntadores para la resolución de problemas utilizando el lenguaje ANSI C.

Descripción del problema

Denisse estudia una ingeniería en una universidad de excelencia, donde constantemente invitan a sus estudiantes a evaluar el desempeño académico de los profesores. Cuando Denisse esta inscribiendo asignaturas para su próximo semestre, descubre que tiene diversas opciones con profesores que no conoce, entonces, decide crear un aplicación que le ayude a ella, y a sus compañeros a seleccionar grupos acorde a los resultados de las evaluaciones de los profesores.

Para iniciar, Denisse solicitó apoyo a traves de Facebook para que sus compañeros de toda la Universidad le apoyaran en la asignación de calificaciones de los profesores. Esto en base a sus experiencias previas en los diversos cursos. La respuesta que obtuvo fue 2 listas de profesores evaluados, la primer lista correspondia a profesores que imparten clases en Ingenierías y la segunda contenia a todos los profesores que imparten clases en el resto de las carreras.

Debido a que Denisse, le gusta programar, decidio crear una pequeña aplicación que le permitiera capturar los datos de los profesores y posteriormente le imprimiera una sola lista con todos los profesores ordenados acorde a su calificación. Lamentablemente, debido a que Denisse salio de viaje, no pudo terminar el programa. Tu tarea es ayudar a Denisse para completar el código.

Código escrito por Denisse

Importante: no modificar el código escrito por Denisse, solamente terminar de escribir el código e implementar las funciones.

```
typedef struct{
        char nombre[15];
        float calificacion;
} Profesor;

float averageArray(Profesor ______  , int _____);
void readArray(Profesor _________, int _____);
void mergeArrays(Profesor _____ , int _____, Profesor ____, int _____, Profesor _____, int ____);
void sortArray(Profesor _____, int ____);
void printArray(Profesor ____ , int _____);


void main(){
        Profesor arr1[20];  //Primer arreglo
        Profesor arr2[20];  //Segundo arreglo
        Profesor arrF[40];  //Arreglo final, con elementos fusionados y ordenados
```

```
        int n1, n2; //Longitud de los arreglos

        readArray(_____________); //leer el primer arreglo

        readArray(_____________); //leer el segundo arreglo

        mergeArrays(_____________________________); //Fusionar los dos arreglos en un tercer arreglo

        sortArray(_____________); //Ordenar los elementos del tercer arreglo, recuerde que pueden
                                  //existir profesores repetidos

        printArray(___________); //Imprimir el resultado final

        return 0;
}
```

Descripción de la entrada del programa

El usuario ingresara dos listas con máximo 20 elementos (profesores: nombre y calificación). Antes de indicar, uno por uno los datos de los profesores, el usuario debe indicar la cantidad de elementos de la respectiva lista. Así lo primero que introducirá será la cantidad (n1) de elementos de la primer lista (arr1), y en seguida los datos de los profesores de la lista; posteriormente, la cantidad (n2) de elementos de la segunda lista (arr2), seguida por los profesores de los profesores correspondientes.

Ejemplo de entrada:

2
Roberto 7.8
Carlos 8.3

4
Oscar 8.3
Miguel 9.4
Diana 9.5
Oscar 8.5

Descripción de la salida

La salida del programa deberá ser sencillamente la impresión de una lista de profesores y su respectiva calificación (ordenados en orden descendiente, separados por un salto de línea). ¿Qué sucede si tenemos dos o más veces el registro de un profesor? La lista final, deberá mostrar sólo una vez a ese profesor y el promedio de sus calificaciones.

Ejemplo de la salida:

Diana	9.5
Miguel	9.4
Oscar	8.4
Carlos	8.3
Roberto	7.8

SOLUCIÓN DEL ALUMNO, PRUEBAS Y CONCLUSIONES

Código fuente:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Profesor
{
  char nombre[15];
  float calificacion;
};

typedef struct Profesor profesor; //sinónimo

//Prototipos:
void readArray (profesor arr[], int n)
{
  profesor * parr = arr;
  for (int i=0; i<n; i++)
    {
      scanf("%s", (parr+i)->nombre);
      scanf ("%f", &(parr+i)->calificacion);
    }
};

void mergeArrays(profesor arr1[], int n1, profesor arr2[], int n2, profesor arrF[])
{
   for(int i=0; i<n1; i++)
   {
      arrF[i] = arr1[i];
   }

   for(int k=0; k<n2; k++)
   {
      arrF[k+n1] = arr2[k];
   }
}

void sortArray (profesor parr[], int n) // ordena arrF (salida de merge)
{
   profesor * arr = parr;
   profesor aux;
   for(int i=0; i<n; i++)
   {
      for(int k=0; k<n-1; k++)
      {
         if((arr+k)->calificacion<(arr+k+1)->calificacion)
         {
            aux.calificacion=(arr+k)->calificacion;
            strcpy(aux.nombre, (arr+k)->nombre);

            (arr+k)->calificacion=(arr+k+1)->calificacion;
            strcpy((arr+k)->nombre, (arr+k+1)->nombre);

            (arr+k+1)->calificacion=aux.calificacion;
```

```
            strcpy((arr+k+1)->nombre, aux.nombre);
          }
        }
    }
};

void printArray (profesor arr[], int n)
{
    for(int i=0; i<n; i++)
    {
    printf("%s\t", arr[i].nombre);
    printf("%.1f\n", arr[i].calificacion);
    }
};

void main ()
{
  profesor arr1[20];              //Primer arreglo (de estructuras tipo prof)
  profesor arr2[20];              //Segundo arreglo
  profesor arrF[40];              //Arreglo final, con elementos fusionados y ordenados
  int n1, n2;              //Longitud de los arreglos

scanf("%d", &n1);
   readArray(arr1, n1);          //leer el primer arreglo
scanf("%d", &n2);
   readArray (arr2, n2);         //leer el segundo arreglo

  int nF=n1+n2; //longitud final

mergeArrays(arr1, n1, arr2, n2, arrF); //Fusionar los dos arreglos en un tercer arreglo - arrF

sortArray(arrF, nF);     //Ordenar los elementos del tercer arreglo arrF, recuerde que pueden
  //existir profesores repetidos

printArray(arrF, nF); //Imprimir arreglo final (arrF)

  return 0;
}


//-------------------------------------------------------------------------------------------
/*
profesor p;
profesor * ap = p;
ap->calificacion=7;
scanf("%f", &ap->calificacion); //& porque es variable, no lleva si fuese array

int v[]={1,2,3,4};
int *p=v; //no lleva & porque nombre del array representa dirección inicial
*p++:
*p=*p+1;

p++
*/
```

Ejecución:

```
2
ola 10
adios 20
4
uno 30
dos 40
tres 50
cuatro 60
cuatro  60.0
tres    50.0
dos     40.0
uno     30.0
adios   20.0
ola     10.0
```

```
4
ola 15.8
we 17.4
kestas 9.7
asiendo 18.2
3
pos 2.5
mi 6.1
tarea 1.8
asiendo 18.2
we      17.4
ola     15.8
kestas  9.7
mi      6.1
pos     2.5
tarea   1.8
```

Conclusiones (obligatorio):

Si bien técnicamente conocía ya los temas involucrados en la tarea, lo que aprendí y lo que siempre termino desarrollando más en cualquier tarea sigue siendo el aspecto de algoritmos como tal: considero que, desde que empecé esta carrera, lo que más se me ha dificultado en términos de programación, y más que "codear" en sí, ha sido el planteamiento o la ideación de lo necesario para resolver determinado problema en términos del algoritmo en sí – es decir, lo complicado en mi caso ha sido más el -por ejemplo- manejo de ciclos anidados o algoritmos de ordenamiento de distintos tipos antes incluso que la comprensión o programación con determinado lenguaje. Considero que cae dentro de lo que habría de ser pensamiento lógico más que lo que es codear como tal, pues no tengo tanto problema plasmando la solución, sino que lo tengo llegando a ella.

Si es que existiera algo que me agrada del hecho de re-cursar esta materia -siendo que técnicamente la he aprobado ya- es el pensar que, seguramente, al finalizar (otra vez…) el curso habría ahora de haber desarrollado un tanto más el pensamiento lógico para el planteamiento de algoritmos. Y, claro, de más está decir que me interesa llevar a otro nivel (mayor al cual el pasado curso me llevó) el dominio de punteros.