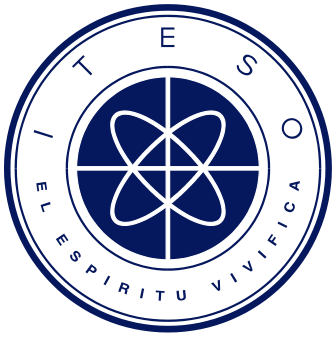
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

INGENIERÍA EN SISTEMAS COMPUTACIONALES



PROGRAMACIÓN CON MEMORIA DINÁMICA

TAREA 1. MANEJO DE APUNTADORES

Autor: Arvizu Barragán, Paulina

24 de mayo de 2018. Tlaquepaque, Jalisco,

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 5 pts
- 60% para la funcionalidad 60 pts
- 30% para las pruebas 20 pts

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 una 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 primera lista correspondía 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 primera 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

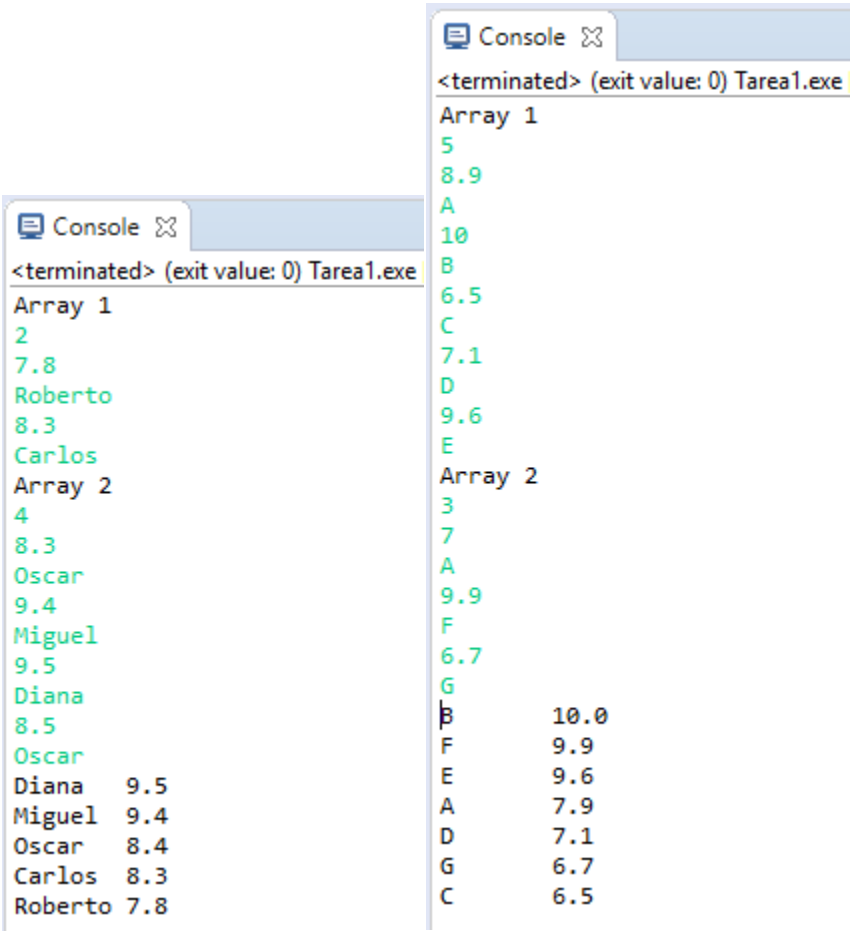
Código fuente:

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <String.h>
4.
5.  typedef struct{
6.      char nombre[15];
7.      float calificacion;
8.  } Profesor;
9.
10. void readArray(Profesor array[], int n);
11. int mergeArrays(Profesor array1[], int n1, Profesor array2[], int n2, Profesor arrayF
    [], int nF);
12. void sortArray(Profesor array[], int n);
13. void printArray(Profesor array[], int n);
14. void eliminarElemento(Profesor array[], int n, int posicion);
15.
16.
17. int main(void){
18.     setbuf(stdin, NULL);
19.     setbuf(stdout, NULL);
20.
21.     Profesor arr1[20];  //Primer arreglo
22.     Profesor arr2[20];  //Segundo arreglo
23.     Profesor arrF[40];  //Arreglo final, con elementos fusionados y ordenados
24.     int n1, n2; //Longitud de los arreglos
25.     puts("Array 1 ");
26.     scanf("%d", &n1);
27.     readArray(arr1, n1); //leer el primer arreglo
28.
29.     puts("Array 2 ");
30.     scanf("%d", &n2);
31.     readArray(arr2, n2); //leer el segundo arreglo
32.
33.     int nF = n1 + n2; //longitud del arreglo final
34.
35.     int nFF = mergeArrays(arr1, n1, arr2, n2, arrF, nF);  //Fusionar los dos arreglos
     en un tercer arreglo
36.
37.     sortArray(arrF, nFF);  //Ordenar los elementos del tercer arreglo, recuerde que p
    ueden existir profesores repetidos
38.
39.     printArray(arrF, nFF);   //Imprimir el resultado final
40.
41. return 0;
42. }
43.
44. void readArray(Profesor array[], int n) {
```

```
45.     Profesor *a = array;
46.     char temp;
47.     int i;
48.     for(i = 0; i < n; i++) {
49.         scanf("%f", &(a+i)->calificacion);
50.         scanf("%c",&temp);
51.         scanf("%[^\n]", (a+i)->nombre);
52.     }
53. }
54.
55. int mergeArrays(Profesor array1[], int n1, Profesor array2[], int n2, Profesor arrayF
    [], int nF) {
56.     Profesor *a1 = array1;
57.     Profesor *a2 = array2;
58.     Profesor *aF = arrayF;
59.     int nFF = nF;
60.     float acum, num;
61.
62.     for(int i = 0; i < n1; i++) { //agrega array1
63.         strcpy((aF+i)->nombre, (a1+i)->nombre);
64.         ((aF+i)->calificacion) = ((a1+i)->calificacion);
65.     }
66.
67.     for(int i = n1; i < nF; i++) { //agrega array2
68.         strcpy((aF+i)->nombre, (a2+(i-n1))->nombre);
69.         ((aF+i)->calificacion) = ((a2+(i-n1))->calificacion);
70.     }
71.
72.     for(int i = 0; i < nF - 1; i++) { //promedia los repetidos
73.         acum = ((aF+i)->calificacion);
74.         num = 1;
75.         for(int cont = i + 1; cont < nF; cont++)
76.             if(0 == strcmp((aF+i)->nombre, (aF+cont)->nombre)) {
77.                 acum += ((aF+cont)->calificacion);
78.                 num++;
79.                 eliminarElemento(arrayF, nF, cont);
80.                 nFF--;
81.             }
82.         ((aF+i)->calificacion) = acum/num;
83.     }
84.     return nFF;
85. }
86.
87. void sortArray(Profesor array[], int n) {
88.     Profesor *a = array;
89.     float x;
90.     char nom[15];
91.
92.     for(int i = 0; i < n - 1; i++) {
93.         for(int cont = i + 1; cont < n; cont++)
94.             if(((a+i)->calificacion) < ((a+cont)->calificacion)) {
95.                 x = ((a+i)->calificacion);
96.                 strcpy(nom, ((a+i)->nombre));
97.
98.                 ((a+i)->calificacion) = ((a+cont)->calificacion);
99.                 strcpy((a+i)->nombre, (a+cont)->nombre);
100.
101.                    ((a+cont)->calificacion) = x;
102.                    strcpy((a+cont)->nombre, nom);
103.                }
104.            }
```

```
105.    }
106.
107.    void printArray(Profesor array[], int n) {
108.        Profesor *a = array;
109.        for(int i = 0; i < n; i++)
110.            printf("%s\t%.1f\n", (a+i)->nombre, (a+i)->calificacion);
111.    }
112.
113.    void eliminarElemento(Profesor array[], int n, int posicion) {
114.        Profesor *a = array;
115.        for(int i = posicion; i < n; i++) {
116.                strcpy((a+i)->nombre, (a+(i+1))->nombre);
117.                ((a+i)->calificacion) = ((a+(i+1))->calificacion);
118.            }
119.        (a+(n-1))->calificacion = -1;
120.    }
```

Ejecución:



Conclusiones:

Esta tarea me ayudó a reforzar los aprendizajes sobre apuntadores a estructuras. Al solucionar el problema, pude resolver las dudas que me habían quedado de este tema.

Tuve problemas al hacer la función de *readArray*, ya que por alguna razón no leía correctamente los valores que le ingresaba, pero lo pude solucionar cambiando el orden de los *scanf* y agregando una variable que guardara el salto de línea. También a la hora de mezclar los dos arreglos, tenia un problema muy sencillo del que no me había dado cuenta, pero al final lo localicé y pude tener un programa que se ejecutara como era de esperarse.