Córdoba 29 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA B

Ejercicio 1

Escribir en **`sorted.c`** la función

```
unsigned int sorted_until(int array[], unsigned int size)
```

que toma como argumentos un arreglo `array` y su longitud `size`. La función debe devolver el índice hasta el cual el arreglo está ordenado (en forma ascendente).

Es decir, si el arreglo es no vacío, devuelve:

```
{Max i : 0 ≤ i < size : array[0...i] está ordenado}
```

En la siguiente tabla se ven ejemplos de cómo debe funcionar `sorted_until()`:

Arreglo	*Size*	*Retorno*	*Razón*
`[1, 3, 0, 4, 5]`	**5**	**1**	ya que [1, 3] está ordenado pero [1, 3, 0] no)
`[1, 3, 4, 0, 0]`	**5**	**2**	ya que [1, 3, 4] está ordenado pero [1, 3, 4, 0] no
`[1, 0, 4, 0, 0]`	**5**	**0**	ya que [1] está ordenado pero [1, 0] no
`[1, 3, 4, 5, 5]`	**5**	**4**	ya que está ordenado completamente
`[3]`	**1**	**0**	ya que [3] está ordenado

Córdoba 29 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA B

Ejercicio 2

Escribir en **`even.c`** la función

```
bool is_even_sorted(int array[], unsigned int length)
```

que toma como argumentos un arreglo `array` y su longitud `length`. La función devuelve `true` si y sólo si el arreglo `[array[i] | i < length, par(i)]` está ordenado.

Es decir, devuelve `true` si teniendo en cuenta sólo las posiciones pares del arreglo, el mismo se encuentra ordenado. En cualquier otro caso devuelve `false`.

Por ejemplo el arreglo `[5, 1, 6, 0]`, si sólo tenemos en cuenta las posiciones `0` y `2`, es decir `[5, 6]`, el arreglo está ordenado. Más ejemplos:

Arreglo	*Retorno*
`[6, 1]`	`true`
`[1, 6, 2, 0]`	`true`
`[1, 6, 0, 0]`	`false`
`[1, 0, 0, 0]`	`false`
`[1, 6, 2, 0, 3]`	`true`
`[1, 6, 2, 0, -1]`	`false`

Córdoba 29 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA B

Ejercicio 3

Ahora se trabajará sobre una estructura **`movie_t`** definida como

```
typedef struct s_movie_t {
  char name[MAX_NAME_LENGTH + 1u];          // Nombre de la peli
  char director[MAX_DIRECTOR_LENGTH + 1u];  // Nombre del director
  unsigned int runtime;                     // Duración en minutos
  float avg_rating;                         // Rating promedio
  float n_votes;                            // Cantidad de votos
} movie_t;
```

que guarda información sobre películas. Particularmente trabajaremos sobre datos recopilados por el sitio IMDB de películas que se estrenaron este año.

a) Completar en **`sort.c`** la definición de la funcion

```
bool goes_before(movie_t s1, movie_t s2)
```

de tal manera que devuelva `true` si y sólo si la duración en minutos de `s1` es menor o igual a la duración de `s2`.

b) Hacer una implementación de `sorted_until()` que trabaje sobre un arreglo con elementos del tipo `movie_t`, es decir que tenga el siguiente prototipo:

```
unsigned int array_sorted_until(movie_t movielist[], unsigned int size)
```

Debe basarse en el criterio de orden impuesto por `goes_before()`

c) Modificar **`main.c`** para que se muestre un mensaje indicando si la *movielist* está completamente ordenada y, de lo contrario, que indique hasta qué índice lo están.

Para verificar pueden usar las *movielist* que les incluimos siguiente resultados:

Movielist	*sorted*	*sorted_unitl*
parcial-animation.mvl	**false**	**7**
zero-action.mvl	**false**	**0**
sorted-scifi.mvl	**true**	**9**

Las salidas de la ejecución deben ser:

```
$ ./movielist parcial-animation.mvl
(:)
Movielist is sorted until 7
```

```
$ ./movielist sorted-scifi.mvl
(:)
Movielist is sorted
```

Córdoba 29 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA B

Ejercicio 4*

a) Modificar **`helpers.c`** para que la lectura del archivo contemple los casos de error y lograr que al ejecutar

```
$ ./movielist broken-comedy.mvl
```

muestre un mensaje

```
Invalid array.
```

y que al ejecutar

```
$ ./movielist huge-horror.mvl
```

muestre un mensaje

```
Array is too long!
```

y no ocurra una ***violación de segmento*.**

El programa debería seguir cargando los archivos de *movielist* del ejercicio anterior **`parcial-animation.mvl`**, **`zero-action.mvl`** y **`sorted-scifi.mvl`** sin problemas.

IMPORTANTE: No pueden modificar ningún otro archivo distinto a **`helpers.c`**

b) Modificar **`sort.c`** de manera tal que el criterio de orden aplicado a la *movielist* ya no sea según la duración sino que sea según el producto entre la cantidad de votos de la película y su rating promedio. Debe considerarse entonces ordenada si las películas que obtienen un valor **más grande** en ese producto están primero.

Para verificar, usando el nuevo criterio la *playlist* **`zero-action.lst`** debería considerarse ordenada.