Córdoba 27 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA A

Ejercicio 1

Escribir en **`pivot.c`** la función

```
bool is_pivot(int array[], unsigned int length, unsigned int piv)
```

que toma como argumentos un arreglo `array`, su longitud `length` y un índice `piv` que se corresponde a una posición en el arreglo `array`. Se debe asumir que `0 ≤ piv < length`. El valor de retorno de la función es `true` si y sólo si para todo `j < piv` se tiene `array[j] ≤ array[piv]`, y para todo `k > piv` se tiene `array[piv] < array[k]`.

Es decir, devuelve `true` si todos los elementos a la izquierda de `piv` son menores o iguales a `array[piv]` y todos los elementos a la derecha de `piv` son mayores estrictos que `array[piv]`. En cualquier otro caso devuelve `false`.

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

Arreglo	*Piv*	*Retorno*
`[1, 2, 6, 5]`	1	**`true`**
`[1, 2, 6, 5]`	0	**`true`**
`[1, 2, 6, 5]`	2	**`false`**
`[1, 1, 6, 5]`	1	**`true`**
`[1, 1, 1, 5]`	1	**`false`**

Córdoba 27 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA A

Ejercicio 2

Escribir en **`odd.c`** la función

```
bool is_odd_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, impar(i)]` está ordenado.

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

Por ejemplo el arreglo `[5, 0, 6, 4]`, si sólo tenemos en cuenta las posiciones `1` y `3`, es decir `[0, 4]`, el arreglo está ordenado. Más ejemplos:

Arreglo	*Retorno*
`[6, 1]`	**`true`**
`[5, 1, 8, 2, 3]`	**`true`**
`[5, 5, 8, 2, 3]`	**`false`**
`[5, 1, 8, 2, 3, 4]`	**`true`**
`[5, 1, 8, 2, 3,-1]`	**`false`**

Córdoba 27 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA A

Ejercicio 3

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

```
typedef struct s_song_t {
  char song_name[MAX_NAME_LENGTH + 1u];
  char artist_name[MAX_ARTIST_LENGTH + 1u];
  unsigned int year;
  unsigned int seconds;
} song_t;
```

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

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

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

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

```
bool array_is_odd_sorted(song_t playlist[], unsigned int length)
```

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

c) Modificar **`main.c`** para que se muestre un mensaje indicando si la *playlist* está ordenada, usando `array_is_sorted()`, y para que muestre si está imparmente ordenada, usando `array_is_odd_sorted()`.

Para verificar pueden usar las *playlist* que les incluimos, debiendo obtener los siguiente resultados:

Playlist	***sorted***	***oddly_sorted***
`unsorted_joplin.lst`	**`false`**	**`false`**
`oddly_arg_rock.lst`	**`false`**	**`true`**
`sorted_queen.lst`	**`true`**	**`true`**

Córdoba 27 de Abril del 2021

Primer Parcial de Laboratorio

Algoritmos y Estructura de Datos II

TEMA A

Ejercicio 4*

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

```
$ ./playlist broken.lst
```

muestre un mensaje

```
Invalid array.
```

y que al ejecutar

```
$ ./playlist all_beattles.lst
```

muestre un mensaje

```
Array is too long!
```

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

El programa debería seguir cargando los archivos de *playlist* del ejercicio anterior **`oddly_arg_rock.lst`**, **`sorted_queen.lst`** y **`unsorted_joplin.lst`** 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 *playlists* sea el año de *copyright* de la canción. Debe considerarse entonces ordenada si las canciones con *copyright* más antiguas están antes.

Para verificar, usando el nuevo criterio la *playlist* **`unsorted_joplin.lst`** debería considerarse ordenada. Lo mismo debería pasar con **`all_beattles.lst`**.