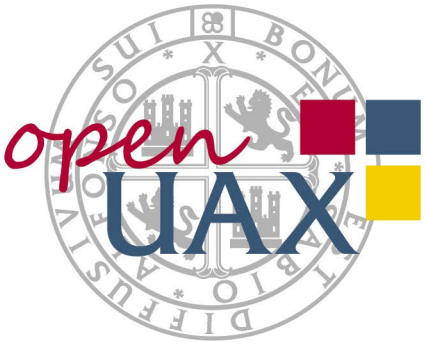

1. Ejercicio FeedBack

ENUNCIADO

Ejercicio 1

Declarar una estructura, de nombre ***arrayLength***, que disponga de tres campos: un array de enteros positivos (*arrInt*) con capacidad para 10 valores y, además, el número de elementos almacenados en el mismo (*arrSize*) y la suma de los elementos contenidos en el array (*arrAdd*). Convertir la estructura en un tipo de datos (arrayLength_t) con la cláusula typedef.

A continuación, se Implementarán varias funciones para la manipulación de este tipo de datos.

Ejercicio 2

Implementar una función de nombre ***initArray*** que reciba por parámetro un puntero a una estructura del tipo *arrayLength_t* y ponga a valor -1 todas sus posiciones. A los campos arrSize y arrAdd se les asignará el valor 0. La función devolverá 0 si no se produce ningún error y -1 en caso contrario.

Un valor -1 en una posición del array arrInt indicará que NO hay un valor a computar en esa posición.

Implementar una función de nombre ***printArr*** que reciba por parámetro un puntero a una estructura del tipo *arrayLength_t*.

La función mostrará por pantalla el contenido de la estructura en el siguiente formato: {[elto0ArrInt, elto1ArrInt, …, elto9ArrInt], arrSize, arrAdd}

Ejemplo: Nada más inicializar el array, la función mostrará:

{[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1], 0, 0}

ArrSize y ArrAdd valen 0 porque se considera que el array está vacío.

Ejercicio 3

Implementar una función de nombre ***addElement*** que reciba por parámetro un puntero a una estructura del tipo arrayLength_t y un nuevo valor que se añadirá a los existentes en el array.

En primer lugar, se comprobará si el valor recibido es positivo. También se determinará si existe hueco en el array para el nuevo elemento y, si es así, se añadirá en la posición siguiente a la última ocupada.

Finalmente, se actualizará el número total de elementos añadidos al array, así como su suma en los correspondientes campos.

La función devolverá 0 si el elemento se ha podido añadir al array sin que se haya producido ningún error y -1 en caso contrario.

Si realizamos las siguientes invocaciones:

```
… //Código anterior
if ( initArray(&arr) != 0){
   printf("Array inicializado");
}else{
   printf("Error en inicialización");
}
printArr(&arr);
if  ( addElement(&arr, 22) == 0) {
   printf("Elemento añadido");
}else{
   printf("Error al añadir elemento");
}
printArr(&arr);
if  ( addElement(&arr, 44) == 0) {
   printf("Elemento añadido");
}else{
   printf("Error al añadir elemento");
}
printArr(&arr);
```

El resultado que se muestre por pantalla será:

{[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1], 0, 0}

{[**22**, -1, -1, -1, -1, -1, -1, -1, -1, -1], **1**, **22**}

{[**22**, **44**, -1, -1, -1, -1, -1, -1, -1, -1], **2**, **66**}

Ejercicio 4

Implementar una función de nombre ***getArrSize*** que reciba por parámetro un puntero a una estructura del tipo arrayLength_t.

La función devolverá el número de elementos en el array y -1 si se produce algún error.

Ejercicio 5

Implementar una función de nombre ***getArrAdd*** que reciba por parámetro un puntero a una estructura del tipo arrayLength_t.

La función devolverá la suma de los elementos almacenados en el array y -1 si se produce algún error.

Ejercicio 6

Implementar una función de nombre ***getElement*** que reciba por parámetro un puntero a una estructura del tipo arrayLength_t y la posición de un elemento dentro del array.

La función comprobará que la posición indicada se encuentra dentro de los límites del array y, además, está ocupada por un valor positivo. En tal caso se devolverá el valor almacenado en la posición indicada o el valor -1 en cualquier otra situación.

Ejercicio 7

Implementar una función de nombre ***setElement*** que reciba por parámetro un puntero a una estructura del tipo arrayLength_t, la posición de un elemento dentro del array y el valor a asignar a dicha posición.

La función comprobará que la posición indicada se encuentra dentro de los límites y que está ocupada por algún valor positivo. También se comprobará que el valor a introducir es positivo.

Si se cumplen las condiciones indicadas se procederá a sustituir el valor que ocupa la posición dada por el nuevo valor especificado. Además, será necesario actualizar el valor del campo *arrAdd*

Si la posición no se encuentra dentro de los límites del array o no almacena un valor positivo se devolverá el valor -1. También se devolverá -1 en caso de que el valor recibido sea negativo.

Ejercicio 8

Implementar una función de nombre ***resetArr*** que reciba por parámetro un puntero a una estructura del tipo *arrayLength_t*.

La función asignará el valor -1 a todas las posiciones del array, también asignará el valor 0 a los campos *arrSize* y *arrAdd*.

Ejercicio 10

Escribir una función ***main()*** para comprobar el funcionamiento de las funciones anteriores. Para ello:

a) Declarar dos estructuras de tipo *arrayLength_t* de nombre ***al1*** y ***al2***.

b) Utilizando las funciones de los apartados anteriores, hacer que el array de ***al1*** almacene los valores 0, 10, 20, …, 90 y se actualicen adecuadamente los valores *arrSize* y *arrAdd*.

c) Haciendo uso de las funciones de los apartados anteriores, mostrar la estructura almacenada en *al1*.

d) Haciendo uso de las funciones de los apartados anteriores, actualizar las posiciones **impares** del array para que almacenen, respectivamente, los valores 1, 3, 5, 7 y 9.

e) Mostrar de nuevo la estructura.

f) Utilizando las funciones desarrolladas, hacer que se añadan al array de al2 las posiciones pares del array al1. Estas posiciones se deben almacenar de forma consecutiva en el array al2.

g) Utilizando las funciones desarrolladas, actualizar las posiciones finales del array de *al2* para que almacenen los valores de 0 a 4

h) Mostrar la estructura al2