Ejercitación – Serie O

Listas simplemente enlazadas

Conceptos que se agregan en esta serie: Manejo de listas simplemente enlazadas.

Como ya se sabe, está prohibido el uso de variables globales.

Implementar las aplicaciones, estructuras de datos y funciones complementarias que permita visualizar el resultado de las funciones realizadas.

Asegurarse en todos los casos de no dejar memoria tomada

Parte 1

Para realizar esta primera parte de la serie, considerar la siguiente estructura de datos

```
#define SZ_ARR 128

typedef struct {
  int x, y;
  char str[SZ_ARR];
} data_t;

typedef struct lista
{
  data_t xy;
  struct lista *sig;
}lista_t;
```

Ejercicio 1: Inversión de lista

Realizar una función que invierta una lista simplemente enlazada, es decir, que los elementos que estaban al principio queden al final y viceversa. Debes hacerlo sin crear una nueva lista ni usar memoria adicional.

La función debe retornar la cantidad de elementos que posee la lista.

Ejercicio 2: duplicados

Realizar una función que determine y retorne la cantidad de los elementos duplicados que posee una lista simplemente enlazada.

Considerar elementos duplicados aquellos que poseen igual valor de **x** e **y**.

Por cantidad de repetidos se debe entender solo una instancia del valor repetido. Es decir que, si el par 5,7 se repite 1 o 20 veces, se debe considerar como un elemento que se encuentra repetido.

Ejercicio 3: Eliminación de duplicados

Realizar una función que elimine los elementos duplicados de una lista simplemente enlazada.

Considerar elementos duplicados aquellos que poseen igual valor de **x** e **y**.

La función debe retornar la cantidad de los elementos eliminados.

Ejercicio 4: traspaso de elementos duplicados

Realizar una función que saque los elementos duplicados de una lista simplemente enlazada y los coloque en una segunda lista, la cual puede estar o no vacía.

Considerar elementos duplicados aquellos que poseen igual valor de **x** e **y**.

La función debe retornar la cantidad de los elementos traspasados.

Ejercicio 5: Intersección de listas

Dadas dos listas enlazadas, desarrolle una función que encuentre e imprima los elementos que poseen el mismo valor en el campo **`str`** en ambas listas. Debes evitar usar estructuras de datos adicionales.

Ejercicio 6: Ordenar una lista (para pensar)

Implementa una función para ordenar una lista simplemente enlazada en orden ascendente o descendente, según se desee.

El criterio de ordenamiento será campo **x**, campo **y**, campo **`str`**. Es decir que en el campo **x** los valores son iguales se define por el campo **y**. Y si **x** e **y** son iguales se define por el campo **`str`**.

Ejercicio 7: Unión de listas

Crea una función que combine dos listas enlazadas en una tercera lista, manteniendo el orden de los elementos. La lista resultante no debe contener elementos duplicados.

Ejercicio 8: Lista palíndroma

Desarrolle una función que determine si una lista enlazada es un palíndromo. Es decir, la lista debe leerse igual de izquierda a derecha que de derecha a izquierda. Por ejemplo, la lista "1 -> 2 -> 3 -> 2 -> 1" es un palíndromo.

La función debe determinar si es un palíndromo en el campo **x**, en el campo **y** y/o en el campo **`str`**, y retornar un valor que indique esta condición (recomendación: usar aritmética de bits).

Ejercicio 9: Partición de lista:

Desarrolle una función que reciba una lista enlazada y la divida en dos listas: una con elementos de **x** pares y otra con elementos **y** impares.

Las nuevas listas deben mantener el orden original que poseían los elementos en la lista original.

Ejercicio 10: Rotación de lista:

Implemente una función que rote los elementos de una lista enlazada hacia la izquierda o hacia la derecha en función de un número dado de posiciones. Por ejemplo, si se rota una lista [1 -> 2 -> 3 -> 4 -> 5] hacia la izquierda 2 posiciones, se convertirá en [3 -> 4 -> 5 -> 1 -> 2].

Parte 2

Ejercicio 11:

Para realizar esta parte de la serie, considerar que la estructura de datos `data_t` , es la siguiente:

```
typedef struct {
  int x, y;
  char *str;
} data_t;
```

Repetir los ejercicios del 1 al 4 de la parte 1. No olvide liberar correctamente los recursos que ya no son necesarios

Parte 3

Ejercicio 12:

En base la siguiente estructura de manejo de listas

```
typedef struct {
  lista_t *pInic; // puntero al inicio de la lista (al primer nodo)
  lista_t *pFni;  // puntero al último nodo de la lista
  lista_t *pAux;  // puntero auxiliar
  int cant;       // cantidad de nodos de la lista
} mng_t;
```

Repetir los ejercicios de la parte 1.

Parte 4 (PaF)

Ejercicio 13: Impresión condicional de una lista

En base a la lista definida en la parte 1, realizar una función que imprima los elementos del array en forma condicional. Se debe poder seleccionar cualquier condición.

Ejercicio 14: para pensar – aunque si quiere, puede implementarlo

¿Podría realizar una serie de funciones para gestionar cualquier tipo de lista?

¿Qué características debería tener esta lista?

¿Si se anima a esbozar las funciones mínimas para manejar una lista de estas características?
