Memoria dinámica y Listas enlazadas

Realice los ejercicios utilizando memoria dinámica.

Listas simplemente enlazadas

1. Escriba una función que devuelva la cantidad de elementos de una lista.
2. Dada una lista enlazada, formada por números enteros (creela usted), hacer un programa que la divida en dos listas: pares e impares.
3. Escribir una función que busque un elemento de la lista, por comparación con una clave e indique si se encuentra o no. Si lo encuentra, imprima su posición en la lista, y la suma de los datos de los nodos por los que paso hasta llegar a él.
4. Usando las funciones vistas en clase (initList, insFirst / insLast) y un loop cree una lista que cuyos valores esten ordenados de menor a mayor. Luego, cree una función que le pida un valor al usuario y lo agregue a la lista, manteniendola ordenada.
5. Escribir un programa que cree una lista tipo LIFO (pila) con datos ingresados por el usuario. Su estructura ListaNodos tendrá los siguientes parametros:

   ```
   Nodo nodos[MAX_CANTIDAD];
   int tamanio;
   int finalQueue;
   int cabezaQueue;
   ```

   Las funciones que debe crear son:

   - enqueue() : inserte un elemento al final de la cola
   - dequeue() : quite y devuelva el elemento en la cabeza de la cola
   - size() : devuelve la cantidad de elementos en la cola.

6. Cree una lista circular (el último elemento apunta al primero). Imprima todos los valores de la lista una única vez (será necesario darme cuenta, de alguna manera, cuando termina la lista)

Listas Doblemente enlazadas

7. Escriba una función que inserte un nuevo nodo después de un nodo específico.
   Si no encuentra el nodo, la función no debe realizar ninguna operación y debe retornar un mensaje adecuado. Si lo encuentra, el nuevo nodo debe estar correctamente enlazado tanto al nodo anterior como al siguiente.
8. Escriba una función que tome un valor y :
   a. imprima el valor anterior, el valor dado, y el siguiente junto con sus posiciones en la lista
   b. elimine el nodo que contiene dicho valor
9. Dada una lista doblemente enlazada que ya está ordenada, escriba una función que inserte un nuevo nodo manteniendo el orden de la lista. Para lograrlo, la función debe recorrer la lista hasta encontrar la posición adecuada para insertar el nuevo nodo.
   Aproveche que la lista está ordenada para optimizar el recorrido: comience a recorrer la lista desde el extremo en el que el nuevo valor esté más cerca. Si el valor a insertar está más cerca del primer valor, comience desde el principio de la lista; si está más cerca del valor final, comience desde el último nodo.
   Ejemplo: Si la lista tiene un 1 en el primer nodo y un 50 en el último nodo, y desea insertar un 49, será más eficiente empezar a recorrer desde el final. En cambio, si desea insertar un 11, es mejor comenzar desde el principio
10. Escriba una función que:
    a. tome como argumento 2 posiciones e intercambie los datos de los nodos que se encuentren en ellas.
    b. tome como argumentos 2 datos e intercambie de posición los nodos que los continenen.

Ejercicios generales

11. Escriba una función que tome dos listas enlazadas distintas y las concatene. La lista final debe ser doblemente enlazada aunque las originales no lo sean.
12. Escriba una función que tome una lista y elimine todos los nodos cuyos valores esten duplicados. Por ejemplo, si tengo 3 nodos cuyo dato es 2, al finalizar la función debería tener un único nodo con ese dato.
13. Escriba una función que tome dos listas enlazadas y las fusione, pero intercalando sus valores.
14. Escriba una funsión que detecte y elimine ciclos en una lista enlazada.

Ejercicio Integrador

15. Debe crear un arbol binario. El nodo a utilizar puede ser el siguiente:

```
typedef struct{
   int data;
   struct treeNode_t padre;
   struct treeNode_t hijo1;
   struct treeNode_t hijo2;
} treeNode_t;
```

Debe crear, además, las siguientes funciones (los valores que toma como argumento y devuelve quedan a su discreción):

a. `create_tree_node()` : esta función crea un nodo del tipo treeNode_t y asigna algún valor (random, o no) a su data.

b. `insertar_hijo()` : esta función debe buscar un nodo en el arbol y añadirle un hijo.

c. `show_tree()` : esta función imprime el arbol completo.

d. `print_branch()` : esta funcion imprime solamente una rama. Por ejemplo, desde el root (primer nodo del arbol) hasta el final, moviendome siempre a través de la cadena 'hijo1'.

e. `search_node(int data)` : esta funcion debe buscar un nodo en particular. Su argumento sera la data contenida en el nodo buscado.

f. `delete_node(int data)` : esta función busca un nodo y lo elimina (junto con todos sus hijos)

g. `print_node_data()` : busca un nodo e imprime toda la data que contiene - padre, data, e hijos

g. `print_leafs()` : esta funcion imprime todas las hojas del arbol.

h. `free_tree()` : esta funció elimina (libera la memoria) de todo el arbol
i. `delete_branch()` : esta función elimina una rama completa del arbol.

Este último ejercicio les propone hacer una árbol binario junto con algunas de las funciones que trabajan con el mismo. La consigna no hace muchas aclaraciones porque la idea es que trabajen de la manera que mejor les resulte, proponiendo los métodos y el flow de trabajo que quieran.