Guía de Practica N° 1

Listas Simplemente Enlazadas

Código de referencia: eedd_dinamicas_liniales_1.cpp

Referencias:

LSE - Lista Simplemente Enlazada

Recomendaciones:

- Utilizar funciones para el desarrollo.
- Probar los programas, en la máquina o con pruebas de escritorio.
- Utilizar nombres de fácil lectura para las variables.

```
struct nodo_listase
{
  int dato;
  struct nodo_listase* link;
};
typedef struct nodo_listase NListaSE;
```

1. Dada una LSE de nombre `L` del tipo `NListaSE`, vacía, realice una función que la cargue con los valores incluidos en un vector de enteros pasado como argumento. A modo de ejemplo se pide probar esta función pasando como parámetro el vector `int v[] {9,1,3,2,8,7,6};`
Nota: Implemente una función de consulta que muestre por consola la lista de la siguiente manera:

```
L --> 9 --> 1 --> 3 --> 2 --> 8 --> 7 --> 6 --> NULL
```

2. Dada la LSE `L` del tipo `NListaSE`, obtener el promedio de los valores contenidos.

3. Dada la LSE `L` del tipo `NListaSE`, realizar una función que duplique el valor del dato contenido.

Entrada:

```
L --> 4 --> 1 --> 3 --> 5  --> NULL
```

Salida:

```
L --> 8 --> 2 --> 6 --> 10 --> NUL
```

4. Dada la LSE `L` del tipo `NListaSE`, realice funciones para obtener:

a. el máximo usando función iterativa.
b. el mínimo usando función iterativa.
c. el máximo usando una función recursiva exclusivamente.

5. Dada la LSE `L` del tipo `NListaSE`, vacía, realice una función que genere enteros aleatorios entre dos números dados. La cantidad de elementos generados también de ser ingresada como parámetro de la función.
Compruebe los valores generados realizando un análisis visual de la lista, y valide utilizando las funciones mínimo y máximo.

6. Dada la LSE `L` del tipo `NListaSE`, realizar una función que elimine todos sus elementos y además `L` refleje esta situación con el valor correspondiente.

7. Dada la LSE `L` del tipo `NListaSE`, con datos, mostrar todos los valores que superen el promedio.

8. Dada la LSE L del tipo NListaSE, con datos, realizar una función entera que reciba como parámetros un nuevo entero a insertar, y una posición (entero), y realice la inserción en la posición correspondiente. A modo de ejemplo, si se ingresa 1 inserta al principio, 2 antes del segundo elemento, si la lista tiene N elementos, y se ingresa N, debe ingresarse al final.

Ejemplos:

```
L --> 9 --> 1 --> NULL
Nuevo=8 y pos=3   L --> 9 --> 1 --> 8 --> NULL
Nuevo=7 y pos=1   L --> 7 --> 9 --> 1 --> 8 --> NULL
Nuevo=0 y pos=3   L --> 7 --> 9 --> 0 --> 1 --> 8 --> NULL
Nuevo=2 y pos=7   → error
Nuevo=2 y pos=0 o negativo → error
```

9. Dada la LSE `L` del tipo `NListaSE`, determinar si se encuentra ordenada:

a. ascendentemente usando función iterativa.
b. descendentemente usando una función recursiva.
c. Ascendentemente o descendentemente.

10. Dadas 2 LSE, L1 y L2, ambas del tipo `NListaSE`, unir las dos listas de forma tal que la última celda de L1 se enlace con la primera celda de L2. De esta forma, mediante L1 veríamos una larga lista con la concatenación de ambas, mientras que a través de L2 entraríamos a un punto intermedio de la lista L1. Nota: no eliminar ni agregar nodos.

11. Dada la LSE `L` del tipo `NListaSE`, la cual se encuentra ordenada ascendentemente, realice una función que muestre la frecuencia de cada elemento y el numero con mayor frecuencia.

Entrada

```
L --> 1 --> 1 --> 3 --> 7 --> 7 --> 7 --> 9 --> NULL
```

Salida

```
Numero    frecuencia

---------------------

     1             2
     3             1
     7             3
     9             1

Numero con mayor frecuencia: 7
```

12. Dadas 2 LSE, L1 y L2, ambas del tipo `NListaSE` y ordenadas en forma ascendente. Desarrolle una función denominada `combinarListasOrden`, que combine ambas listas en una tercer lista de nombre L, la cual también debe quedar ordenada.

Entrada

```
L1 --> 4 --> 5 --> 9 --> NULL
L2 --> 1 --> 8 --> NULL
```

Entrada

```
L --> 1 --> 4 --> 5 --> 8 -->  9 --> NULL
```

13. Dada la LSE `L` del tipo `NListaSE`, implementar una función que invierta dos elementos cualesquiera de `L`. La misma se encuentra desordenada. No debe intercambiar los datos de los nodos, sino trabajar con la referencia entre los nodos (campo `link`).

14. Dada la LSE `L` del tipo `NListaSE`, implementar una función ordene la lista en forma ascendente.

15. Dada la LSE `L` del tipo `NListaSE`, realizar una función que elimine todas las ocurrencias de un valor `N` recibido como segundo parámetro.

16. Dada la LSE `L` del tipo `NListaSE`, desarrollar una función que efectue su inversión, obteniendo la misma lista con los mismos nodos, pero con el campo enlace invertido. Restricción principal, no generar otra lista ni eliminar ni agregar nodos.

Entrada:

```
L --> 9 --> 1 --> 3 --> 5 --> NULL
```

Salida:

```
L --> 5 --> 3 --> 1 --> 9 --> NULL
```

17. Realizar una versión de la función anterior que utilice una estructura auxiliar como pila o cola, realizando el mismo resultado, pero sin reutilizar los nodos.

18. Dada la LSE `L` del tipo `NListaSE`, implementar una función que determine si existen algún valor repetido en el campo dato.

Entrada:		
	L1 --> 9 --> 1 --> 3 --> 5 --> NULL	L2 --> 9 --> 1 --> 9 --> 5 --> NULL
Salida:	false	true

19. Dada la LSE LP del tipo NPersona, desordenada, ordene la lista por fecha de nacimiento en forma descendente.

```
struct sFecha {
    int dia;
    int mes;
    int anio;
};
struct nodo_persona {
    int dni;
    char nombre[60];
    sFecha nacimiento;
};
typedef struct nodo_persona NPersona;
```

20. Se cuenta con una lista LSE de escuelas primarias con los siguientes datos, numero de escuela, nombre, cantidad alumnos, presupuesto anual en miles y provincia, ordenada por provincia.

```
Ejemplo de información de Escuelas
-------------------------------------------------------
1       Escuela 204     950     32      Entre Ríos
10      Escuela Normal  1050    50      Entre Ríos
8       Esc. Las Heras  650     12      Entre Ríos
2       Esc. Lopez      2000    25      Santa Fe
80      Esc. Privada 8  1100    18      Córdoba
```

Se requiere, definir una estructura para alojar esta información y generar una función que genere un reporte de totales de alumnos, presupuesto y escuelas por provincia y totales generales de los mismos datos. A modo de ejemplo se muestra la salida esperada.

```
Provincia       C.Alu   Presupuesto     C.Escuelas
-------------------------------------------------------
Entre Ríos      2650            94              3
Santa Fe        2000            25              1
Córdoba         1100            18              1
-------------------------------------------------------
Total Gral      5750            137             5
```

21. Realizar la implementación de las funciones `listase_agregar_principio`, `listase_agregar_final`,`listase_agregar_ordenado` y `listase_eliminar_ocurrencia`, de manera de poder implementar listas de cualquier tipo de dato a saber, char, int, float, o `NPersona`.

Para las funciones `listase_agregar_ordenado`, `listase_eliminar_ocurrencia`, considere de usar un puntero a función para independizar la función de la implementación de tipo de elemento de la lista.

22. Dado el siguiente archivo de cabecera (header) complete la codificación de las funciones `LSEG_vacia`, `LSEG_agregar_principio`, `LSEG_agregar_final`, `LSEG_agregar_ordenado`, `LSEG_count`, `LSEG_obtener_pos`, `LSEG_eliminar_pos` y `LSEG_eliminar_lista`, correspondientes a la implementación de una lista simplemente enlazada genérica que permita alojar objetos primitivos (char, int, float, string) o objetos definidos por el usuario. .

`LSEG_generica.h`

```
/*****************************************************************/
// Implementación de una  LSE generica de (void*), que permita
//  alojar cualquier tipo de elementos, elementos primitivos:
//  char, int, float, string, e incluso estructuras y objetos
//  definidos por el usuario Personas, objetos de distinto tipo.
// UADER – Algoritmos y Estructuras de Datos
/*****************************************************************/

using namespace std;

struct lse_generica
{  void * pelemento;
   struct lse_generica * link;
};
typedef struct lse_generica LseG;

void LSEG_agregar_principio (LseG * &lista, void * pelemento) {   }

void LSEG_agregar_final (LseG * &lista, void * pelemento) {   }

void LSEG_agregar_ordenado (LseG * &lista,
                            void * pelemento,
                            int (*compar)(const void*,const void*)
                            ) {   }
   Donde comprar debe ser implementada por el usuario y es un puntero a una
   función que compara dos elementos. LSEG_agregar_ordenado llama
   repetidamente a esta función para comparar dos elementos. Deberá seguir el
   siguiente prototipo:   int compar (const void * p1, const void * p2);

bool LSEG_vacia (LseG * &lista) {   }

int LSEG_count (LseG * lista) {   }

void * LSEG_obtener_pos(LseG * &lista, int pos) {   }

void LSEG_eliminar_lista (LseG * &lista) {   }
```

Para uso con primitivo int:

```
main()
   LseG * lista = NULL;
   int * pdato;

   // para hacer agregar principio
   pdato = new(int);
   *pdato = xxxx;
```

```
LSEG_agregar_principio(lista,(void *) pdato);

// para
pdato = (int *) LSEG_obtener_pos(lista,1);
```

Para uso con UDT:

```
struct spersona {
   int dni;
   char nombre[40];
};

main() {
   LseG * lista = NULL;
   spersona *pdato;

   // para hacer push
   pdato = new(spersona);
   (*pdato).dni      = 13241227;
   strcpy((*pdato).nombre, "Maradona Diego");
   LSEG_agregar_principio(lista,(void *) pdato);

   // para hacer pop
   pdato = (spersona *) LSEG_obtener_pos(lista,1);
```

Utilice el archivo de cabecera `LSEG_generica.h`, para producir una nueva versión del ejercicio 20.