Universidad Simón Bolívar
Departamento de Computación
y Tecnología de la Información
Laboratorio de Computación II
Ene-Mar 2019

> 308 *El no saber comercial es distinguido.*—El vender su mérito lo más caro posible o negociarle con usura, a título de profesor, funcionario o artista, pone al talento o al genio a la altura de un tendero. No está bien querer ser demasiado *hábil* con la sabiduría.
>
> Aurora, Federico Nietzsche.

Lab. 4—Sem. 5—Listas Enlazadas.

En este laboratorio definiremos una **Lista** como un apuntador a un **Nodo** y definiremos un **Nodo** como un *registro* simple con sólo dos campos a saber: un entero de nombre *val* y un apuntador de nombre *sig* a objetos del tipo de la estructura que se está definiendo—es una definición recursiva— Alerta Rojo: para que una lista esté bien definida hay que inmediatamente después de declararla darle un valor bien NULO (NULL), o bien hacer que apunte a un Nodo que ya existe o bien crea un nuevo nodo y hacer que la Lista apunte a él. Se usara manejo dinámico de la memoria.

Tareas

1. *Declaración de Nodo y Lista.* Copiar **a mano** el siguiente texto en un proyecto nuevo de nombre **Listas** y compile. Si hay algo que no entienda pregúntele al encargado de su clase.

```
#include <stdio.h>
#include <stdlib.h>

///Encabezado: Se define Nodo y Lista.

typedef struct nodo {
   int val;
   struct nodo *sig;
} Nodo;

typedef Nodo * Lista;
```

2. *Añada los siguientes Constructores a su Proyecto.* Agregue los siguientes dos constructores a su proyecto. El primero sirve para garantizar que la Lista que se acaba de declarar tenga un valor correcto y el segundo para agregar nuevos elementos a una Lista. Compile de nuevo su proyecto!

```
Lista newList(){
   return NULL;
}

///Inserta en la cabeza de la Lista
void inserta(int x, Lista *L){
   Nodo *p = malloc(sizeof(Nodo));
   p->val = x;
   p->sig = *L;
   *L = p;
}
```

   La primera se usará al declarar una Lista de la siguiente manera: `Lista L = newList();` que es equivalente, en nuestro caso, a `Lista L = NULL;`. Luego de declarar la lista, se le pueden añadir, por ejemlo, el elemento 5 usando: `inserta(5, &L)`. Note que *inserta* usa la función *malloc* para solicitar al sistema operativo una memoria de tamaño `sizeof(Nodo)`.

3. *Agregue Observadores.* A continuación agreguemos algunas funciones que nos permitirán saber el estado de nuestra Lista. Se suelan llamar observadores. Agregue las siguientes funciones a su proyecto y de nuevo compile.

```
int esVacia(Lista L){
   return L==NULL;
}

// pre: L es no vacía
int firstInList(Lista L){
   return L->val;
}

///Booleano: 1 si x está, 0 si NO
int estaEn(int x, Lista L){
   Nodo *p = L;
   while(p != NULL && p->val != x) p = p->sig;
   return p != NULL;
}

///Escribe la lista L de nombre name entre dos corchetes
void writeList(Lista L, char name[]){
   printf("\n%s = [",name);
   if (L == NULL) printf("].");
   else {
      while (L->sig != NULL) {
         printf("%d, ", L->val);
         L = L->sig;
      }
      printf("%d].",L->val);
   }
}
```

4. *Agregue un* **main()** *a su Proyecto.* Al final del proyecto agregue el siguiente **main()**, compile y ejecútelo. Asegúrese de entender como funciona cada una de las funciones usadas y de haber entendido la prueba.

```
///programa de prueba
int main(){
   printf("\nProbando Lista...:")
   Lista L = newList(), L1 = newList();
   writeList(L, "L");
   inserta(4,&L);
   inserta(9,&L);
   inserta(5,&L);
   writeList(L, "L");
   printf("\nA la cabeza de L esta: %d",firstInList(L));
   if (esVacia(L1)) printf("\nL1 es vacia."); else printf("\nL1 NO es vacia.");
   int x = 7; char s[] = "L";
   if (estaEn(x, L)) printf("\n%d esta en %s.", x, s); else printf("\n%d NO esta en %s.", x,


   return 0;
}
```

5. *Agregue Otras Funciones NO RECURSIVAS.* Lo siguiente será agregar algunas otras funciones. Complete y pruebe cada una de estas funciones.

```
///Inserta a x al final de la Lista L---apend
void insertaT(int x, Lista *L){
   Nodo *p = malloc(sizeof(Nodo)), *q = *L;
   p->val = x;
   p->sig = NULL;
   if (*L == NULL) COMPLETAR;
   else {
     COMPLETAR;
   }
}

///Elimina la primera ocurrencia de x de la lista L
void elimina(int x, Lista *L){
   Nodo *p = *L, *q;
   if (p != NULL){
      if (p->val == x){
         COMPLETAR;
      } else {
         while (p->sig != NULL && (p->sig)->val != x) p = p->sig; //BUSQUEDA lINEAL
         if (p->sig != NULL) {
            COMPLETAR;
         }
      }
   }
}

///Produce en *N una copia de L
void clona(Lista L, Lista *N){
   *N = NULL;
   Nodo *q;
   if (L != NULL){
      *N = COMPLETAR;

      COMPLETAR;
      while(L != NULL){
         q->sig = malloc(sizeof(Nodo));
         COMPLETAR;
      }
      q->sig = NULL; //£Por qué?
   }
}

///Concatena las listas L y K. L= L:K, K = NULL
void concat(Lista *L, Lista *K){
   if (*L == NULL) *L = *K;
   else {
      Nodo *p =  COMPLETAR;
      COMPLETAR;

   }
   *K = NULL;
}
```

6. *Agregue algunas Funciones Recursivas.* El siguiente paso será agregar algunas funciones recursivas. Se sugieren:

```
int sizeR(Lista L){
   if(L == NULL) return 0;
   else return 1+ sizeR(L->sig);
}

///Recursiva...
int estaEnR(int x, Lista L){
   if(L == NULL) return 0;
   else if (L->val == x) return 1;
        return estaEnR(x, L->sig);
}

///Inserta a x al final de la Lista L---apend
void insertaTR(int x, Lista *L){
   if(*L == NULL) {
      Nodo *p = malloc(sizeof(Nodo));
      p->val = x;
      p->sig = NULL;
      *L = p;
   }
   else insertaTR(x, &((*L)->sig));
}

///Elimina la primera ocurrencia de x en la Lista *L
void eliminaR(int x, Lista *L){
   if (*L != NULL){
      if ((*L)->val == x) {
         Nodo *p = *L;
         *L = (*L)->sig;
         free(p);
      }
      else eliminaR(x, &((*L)->sig));
   } /// else skip
}
```

Extras:

```
int sizeI(Lista L){
   int r = 0;
   COMPLETAR; es sólo una variante de la búsqueda no acotada!
}
```

Ejercicios

1. *Mezcla de Listas.* Escriba un una función que reciba dos lista ordenadas de forma no decreciente y retorne una nueva lista ordena de forma no decreciente que contenga exactamente todos los elementos de las dos lista que recibe. `Lista merge(const Lista L, cons Lista M);`

2. *Inserción Ordenada.* Escriba una función que reciba un entero x y una lista ordenada L e inserte el elemento x en la liata L de tal forma que la lista final esté ordenada. `void insertOrd(int x, cons Lista L);`

3. Halla el máximo de una lista no vacía. Debe retornar un apuntador al Nodo que contiene la primera ocurrencia del máximo.

4. Hallar la posición de un elemento x dentro de una lista. Debe retornar un apuntador NULO si no se encuentre, de lo contrario debe retornar la dirección de memoria del nodo donde se encuentra.

Nota: Se recomienda seriamente que no se hagan *cut and paste*. Debe copiar su código para que lo entienda mejor!!!