Clase 9. Ejercicios sobre estructuras, listas, apuntadores y memoria dinámica.

Arreglos de estructuras

Ejercicio 1.

Dado el código del ejercicio 1, requiera memoria dinámica para un arreglo de 10 instancias de la estructura, asigne, en el orden que requiera la memoria, la etiqueta 0 al primero, la 1 al segundo, y así sucesivamente, Conecte el primer nodo con el segundo, el segundo con el tercero, etc. El código ya imprime el valor de la etiqueta y la dirección.

1. Reliace un programa que requiera la memoria dnámica para el arreglo de nodos en un solo bloque.
2. Realice el ejercicio requiriendo un arreglo de apuntadores al nodo y luego memoria para cada estructura en otro bloque.

Código ejercicio 1a


```
1  #include <stdlib.h>
   #include <stdio.h>
3  typedef struct ENODO{
     int etiqueta;
5    struct ENODO *siguiente;
   }NODO;
7  int main(int argc, char *argv[])
   {
9    int nnodos=10;
     //Forma 1, declaro un apuntador para el bloque que
       contiene el arreglo.
11   NODO* arreglo;
     for (int i=0; i<(nnodos-1); i++)
13     printf("El nodo %d con direccion %p esta conectado
       al nodo %d con direccion %p\n",
       i,&arreglo[i],arreglo[i].siguiente->etiqueta,
       arreglo[i].siguiente);
15   printf("El nodo %d con direccion %p esta conectado al
       nodo %d con direccion %p\n",
     9,&arreglo[9],-1,arreglo[9].siguiente);
17   return 0;
   }
```

Código ejercicio 1b


```c
#include <stdlib.h>
#include <stdio.h>
typedef struct ENODO{
  int etiqueta;
  struct ENODO *siguiente;
}NODO;
int main(int argc, char *argv[])
{
  int nnodos=10;
  //Forma 1, declaro un apuntador para un bloque de
    apuntadores
  NODO** arreglo;
  for (int i=0; i<(nnodos-1); i++)
    printf("El nodo %d con direccion %p esta conectado
    al nodo %d con direccion %p\n",
    i, arreglo[i], arreglo[i]->siguiente->etiqueta,
    arreglo[i]->siguiente);
  printf("El nodo %d con direccion %p esta conectado al
     nodo %d con direccion %p\n",
  9, arreglo[9], -1,arreglo[9]->siguiente);
return 0;
}
```

Errores posibles, ejercicio 1


```
1  #include <stdlib.h>
   #include <stdio.h>
3  typedef struct ENODO{
     int etiqueta;
5    struct ENODO *siguiente;
   }NODO;
7  int main(int argc, char *argv[])
   {
9    int nnodos=10;
     NODO* arreglo;
11   arreglo=(NODO*)malloc(nnodos*sizeof(NODO));
     arreglo[0]->siguiente=arreglo[1];
13   return 0;
   }
```

Errores posibles, ejercicio 1

Usar el operador $->$ en lugar del .


```
#include <stdlib.h>
#include <stdio.h>
typedef struct ENODO{
   int etiqueta;
   struct ENODO *siguiente;
}NODO;
int main(int argc, char *argv[])
{
   int nnodos=10;
   NODO* arreglo;
   arreglo=(NODO*)malloc(nnodos*sizeof(NODO));
   arreglo[0]->siguiente=arreglo[1];
   return 0;
}
```

Errores posibles, ejercicio 1

Usar el operador $->$ en lugar del .

Pensar que arreglo[i] es una dirección de memoria (cuando es el contenido).


```
#include <stdlib.h>
#include <stdio.h>
typedef struct ENODO{
   int etiqueta;
   struct ENODO *siguiente;
}NODO;
int main(int argc, char *argv[])
{
   int nnodos=10;
   NODO* arreglo;
   arreglo=(NODO*)malloc(nnodos*sizeof(NODO));
   arreglo[0]->siguiente=arreglo[1];
   return 0;
}
```

Errores posibles, ejercicio 1

Lo correcto sería usar el operador . porque arreglo[0] es el contenido de la dirección **arreglo**, y usar el operador & sobre arreglo[1] que tambien ya es un contenido.


```
#include <stdlib.h>
#include <stdio.h>
typedef struct ENODO{
  int etiqueta;
  struct ENODO *siguiente;
}NODO;
int main(int argc, char *argv[])
{
  int nnodos=10;

  NODO* arreglo;
  arreglo=(NODO*)malloc(nnodos*sizeof(NODO));
  arreglo[0].siguiente=&arreglo[1];
  return 0;
}
```

Errores posibles, ejercicio 1


```
1  int main(int argc, char *argv[])
   {
3    int nnodos=10;
     NODO** arreglo;
5    NODO *raiz;
     arreglo=(NODO**)malloc(nnodos*sizeof(NODO*));
7    arreglo[0]=(NODO*)malloc(nnodos*sizeof(NODO));
     arreglo[1]=(NODO*)malloc(nnodos*sizeof(NODO));
9    arreglo[0].siguiente=arreglo[1];
     raiz=arreglo[0];
11   next=raiz;
     while(next!=NULL){
13       printf("Etiqueta %d\n",next->etiqueta);
         next=next.siguiente;
15   }
     return 0;
17 }
```

Errores posibles, ejercicio 1

Usar el operador . en lugar del $->$


```
1  int main(int argc, char *argv[])
   {
3    int nnodos=10;
     NODO** arreglo;
5    NODO *raiz;
     arreglo=(NODO**)malloc(nnodos*sizeof(NODO*));
7    arreglo[0]=(NODO*)malloc(nnodos*sizeof(NODO));
     arreglo[1]=(NODO*)malloc(nnodos*sizeof(NODO));
9    arreglo[0].siguiente=arreglo[1];
     raiz=arreglo[0];
11   next=raiz;
     while(next!=NULL){
13       printf("Etiqueta %d\n",next->etiqueta);
         next=next.siguiente;
15   }
     return 0;
17 }
```

Errores posibles, ejercicio 1

Usar el operador . en lugar del $->$

No inicializar la referencia en NULL


```
1  int main(int argc, char *argv[])
   {
3    int nnodos=10;
     NODO** arreglo;
5    NODO *raiz;
     arreglo=(NODO**)malloc(nnodos*sizeof(NODO*));
7    arreglo[0]=(NODO*)malloc(nnodos*sizeof(NODO));
     arreglo[1]=(NODO*)malloc(nnodos*sizeof(NODO));
9    arreglo[0].siguiente=arreglo[1];
     raiz=arreglo[0];
11   next=raiz;
     while(next!=NULL){
13       printf("Etiqueta %d\n",next->etiqueta);
         next=next.siguiente;
15   }
     return 0;
17 }
```

Errores posibles, ejercicio 1


```
1  int main(int argc, char *argv[])
{
3    int nnodos=10;
   NODO** arreglo;
5    NODO *raiz;
    arreglo=(NODO**)malloc(nnodos*sizeof(NODO*));
7    arreglo[0]=(NODO*)malloc(nnodos*sizeof(NODO))
    arreglo[0]->siguiente=NULL;;
9    arreglo[1]=(NODO*)malloc(nnodos*sizeof(NODO));
    arreglo[1]->siguiente=NULL;;
11   arreglo[0]->siguiente=arreglo[1];
    raiz=arreglo[0];
13   next=raiz;
    while(next!=NULL){
15       printf("Etiqueta %d\n",next->etiqueta);
        next=next->siguiente;
17   }
    return 0;
19 }
```

Errores posibles, ejercicio 1

Lo correcto. En este caso se requirió primero un arreglo de apuntadores, no de estructuras, y despues cada posición del arreglo de apuntadores se apunto hacia memoria requeriada para la estructura, entonces en este caso arreglo[0] es un apuntador y hay qu utilizar el operador $->$ para acceder al contenido.


```
1  int main(int argc, char *argv[])
{
3    int nnodos=10;
    NODO** arreglo;
5    NODO *raiz;
     arreglo=(NODO**)malloc(nnodos*sizeof(NODO*));
7    arreglo[0]=(NODO*)malloc(nnodos*sizeof(NODO))
     arreglo[0]->siguiente=NULL;;
9    arreglo[1]=(NODO*)malloc(nnodos*sizeof(NODO));
     arreglo[1]->siguiente=NULL;;
11   arreglo[0]->siguiente=arreglo[1];
     raiz=arreglo[0];
13   next=raiz;
     while(next!=NULL){
15       printf("Etiqueta %d\n",next->etiqueta);
         next=next->siguiente;
17   }
     return 0;
19 }
```

Ejercicio 2.

Obtenga mediante una función recursiva los datos del último nodo de la lista del ejercicio 1b.

Nota: No se debe de usar el arrelo de punteros sola la dirección del elemento 0, los demás elemento se pueden usar solo para fines didacticos, es decir, saber que dirección quiero apuntar y si se está apuntando bien.

Código ejercicio 2


```
1  #include <stdlib.h>
   #include <stdio.h>
3  typedef struct ENODO{
     int etiqueta;
5    struct ENODO *siguiente;
   }NODO;
7  NODO obtieneDatos(NODO *raiz)
   {
9    NODO datos;
     datos.etiqueta=-1;
11   datos.siguiente=NULL;
     if(raiz==NULL)
13     return datos;
     else if(raiz->siguiente==NULL)
15           return datos=*raiz;
     else datos=obtieneDatos(raiz->siguiente);
17 return datos;
   }
```

Código ejercicio 2. Continuación

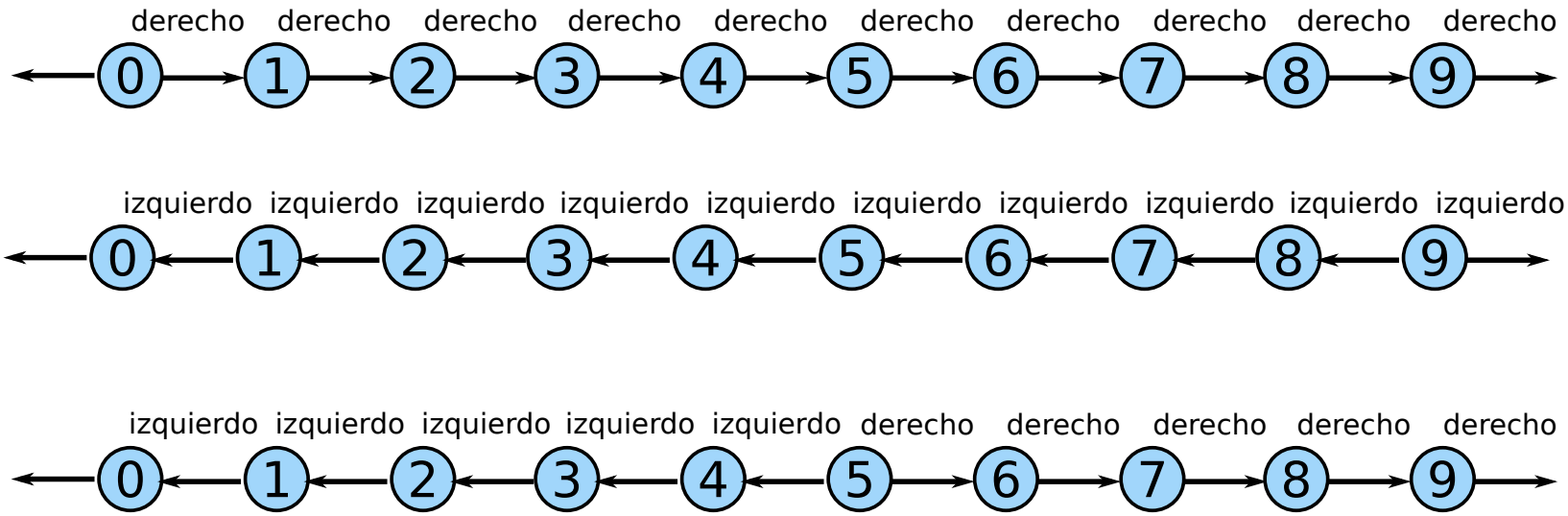
```
1  NODO* obtieneDatosLiberaMemoria(NODO *raiz, NODO *datos
       )
   {
3    datos->etiqueta=-1;
     datos->siguiente=NULL;
5  return raiz;
   }
7
   void printLista(NODO *raiz)
9  {
     if (raiz!=NULL)
11   {
       printLista(raiz->siguiente);
13     printf("Etiqueta= %d\n",raiz->etiqueta);
     }
15 }
```

Código ejercicio 2. Continuación 2


```
1  int main(int argc, char *argv[])
{
3    int nnodos=10;
    NODO** arreglo, *raiz,datos;
5    arreglo= (NODO**)malloc(nnodos*sizeof(NODO*));
    for (int i=0; i<(nnodos); i++){
7      arreglo[i]=(NODO*)malloc(sizeof(NODO));
      arreglo[i]->etiqueta=i;
9    }
    for (int i=0; i<(nnodos-1); i++)
11       arreglo[i]->siguiente=arreglo[i+1];
    arreglo[9]->siguiente=NULL;
13   raiz=arreglo[0];
    datos=obtieneDatos(raiz);
15   printf("Etiqueta del ultimo nodo %d\n", datos.
      etiqueta);
//     while(raiz!=NULL){
17 //       printLista(raiz);
//       raiz=obtieneDatosLiberaMemoria(raiz, &datos);
19 //       printf("Ultimo nodo %d\n", datos.etiqueta);
//     }
21   return 0;
}
```

Ejercicio 3

Utilizando nodos con 2 apuntadores y una función recursiva pase de la lista en el primer dibujo y a la segunda y tercera.



Puede comenzar del 9 hacia el 0 en lugar del 0 hacia el 9.

Código ejercicio 3. Crear Lista


```
typedef struct ENODO{
  int etiqueta;
  struct ENODO *izq,*der;
}NODO;
//Crea lista de nnodos
NODO *creaLista(int nnodos, int der)
{
  NODO *raiz;
  if (nnodos>0){
     raiz=(NODO*)malloc(sizeof(NODO));
     raiz->etiqueta=nnodos-1;
     if (der){
       raiz->izq=NULL;
       raiz->der=creaLista(nnodos-1,der);
     }else{
       raiz->der=NULL;
       raiz->izq=creaLista(nnodos-1,der);
     }
  }else
     return NULL;
return raiz;
}
```

Código ejercicio 3. Imprimir y eliminar lista


```
void printLista(NODO* raiz, int der){
    if (raiz!=NULL){
      printf("nodo= %d ligado con:\n",raiz->etiqueta);
      if (der)
         printLista(raiz->der,der);
      else
         printLista(raiz->izq,der);

    }else{
      printf("NULL\n");
    }
}

NODO *eliminarLista(NODO *raiz){
  if (raiz!=NULL){
      eliminarLista(raiz->der);
      eliminarLista(raiz->izq);
      free(raiz);
      return NULL;
  }
return NULL;
}
```

Código ejercicio 3. Main


```
int main(int argc, char *argv[])
{
  int nnodos=10;
  NODO *raiz=creaLista(nnodos,1);
  printLista(raiz,1);

//    raiz=voltear(raiz, NULL, 1);

  printLista(raiz,0);
  raiz=eliminarLista(raiz);
  return 0;
}
```

Tarea

prog5.1 Dada una lista con un dato flotante realice funciones recursivas que:

(a) Calcule la suma de los datos de la lista.
(b) Calcule el promedio.
(c) Calcule la varianza.

prog5.2 Relice un programa lea desde un archivo las dimensiones de una matríz nxm y los n x m elementos, y almacene cada renglón en un nodo de una lista. Realice la lectura de la matriz con una función recursiva (primero lea las dimensiones y después los elementos recursivamente). Imprima la matríz de forma recursiva.