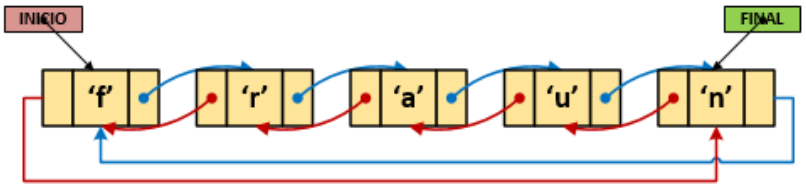
*Tema: Listas Dobles*

Apellido y Nombre: ..... Fecha:...../...../.....

## EJEMPLOS

***Ejemplo 1 – Variante de implementación:*** Dada la siguiente lista circular defina las estructuras que permitan implementarla incluyendo un elemento para registrar la cantidad de nodos almacenados. Además desarrolle las operaciones *iniciar_lista*, *agregar_final*, *quitar_inicio* y *mostrar_lista*. Tenga en cuenta que las operaciones de inserción/extracción deben mantener la lista circular.




***Implementación modificada***

La definición de una lista doble circular es la misma que la definición básica de la lista doble, en realidad, para generar una lista circular sólo deben modificarse las operaciones de inserción/extracción de elementos. En este ejemplo, además debe incluirse un contador (junto a los punteros de la lista) para llevar cuenta de la cantidad de elementos almacenados.

La definición de lista doble correspondiente a esta implementación es la siguiente:

```
TIPOS
pnodo=puntero a tnodo
tnodo=REGISTRO
        dato:CARACTER
        ant:pnodo
        sig:pnodo
      FIN_REGISTRO
tlista=REGISTRO
        inicio:pnodo
        final:pnodo
        contador:ENTERO
      FIN_REGISTRO
```

```
typedef struct tnodo *pnodo;
typedef struct tnodo {
                       char dato;
                       pnodo ant;
                       pnodo sig;
                      };
typedef struct tlista {
                       pnodo inicio;
                       pnodo final;
                       int contador;
                      };
```

La operación *iniciar_lista* se realiza mediante un procedimiento que inicializa la lista. La inicialización crea una lista vacía asignando a *inicio*, *final* y *contador* los valores adecuados. En este caso, a los campos *inicio* y *final* se asigna NULO (NULL en C/C++), mientras que al campo *contador* se asigna cero.

```
PROCEDIMIMENTO iniciar_lista(E/S letras:tlista)
INICIO
   letras.inicio←NULO
   letras.final←NULO
   letras.contador←0
FIN
```

```
void iniciar_lista(tlista &letras)
{
  letras.inicio=NULL;
  letras.final=NULL;
  letras.contador=0;
}
```

La operación *agregar_final* se realiza mediante un procedimiento que añade un nuevo nodo al final de la lista. Esta operación actualiza el puntero *final* (apuntando al nuevo nodo) y conecta el nuevo elemento al primero de la lista. A su vez, el primer nodo actualiza su puntero anterior para hacer referencia al último agregado. Además, debe actualizarse el contador de nodos. Para ello, se consideran 2 casos: una lista vacía y una lista con elementos.

```
PROCEDIMIMENTO agregar_final(E/S letras:tlista,
                             E nuevo:pnodo)
INICIO
   SI letras.contador=0 ENTONCES
      letras.inicio←nuevo
      letras.final←nuevo
   SINO
      letras.final->sig←nuevo
      nuevo->ant←letras.final
      letras.final←nuevo
   FIN_SI
   letras.final->sig←letras.inicio
   letras.inicio->ant←letras.final
   letras.contador←letras.contador+1
FIN
```

```
void agregar_final(tlista &letras, pnodo nuevo)
{
  if (letras.contador==0)
    { letras.inicio=nuevo;
      letras.final=nuevo;
    }
  else
    { letras.final->sig=nuevo;
      nuevo->ant=letras.final;
      letras.final=nuevo;
   }
  letras.final->sig=letras.inicio;
  letras.inicio->ant=letras.final;
  letras.contador++;
}
```

La operación *quitar_inicio* se realiza mediante una función que retorna la dirección de un nodo extraído del inicio de la lista. Para ello se consideran 3 casos: una lista vacía, una lista con un único elemento y una lista con 2 o más elementos. Si la lista está vacía la función retorna el valor NULO (NULL en C/C++) mientras que si ésta contiene elementos entonces se retorna la dirección del nodo extraído. En particular, si la lista contiene sólo un elemento la operación de extracción generará una lista vacía. Se debe tener en cuenta que los punteros anterior (primer nodo) y siguiente (último nodo) deben actualizarse tras la extracción para mantener la lista circular.

```
FUNCIÓN quitar_inicio(E/S letras:tlista):pnodo
VARIABLES
   aux:pnodo
INICIO
   SI letras.contador=0 ENTONCES
      aux←NULO
   SINO
     aux←letras.inicio
     SI letras.contador=1 ENTONCES
        iniciar_lista(letras)
     SINO
        letras.inicio←letras.inicio->sig
        letras.inicio->ant←letras.final
        letras.final->sig←letras.inicio
        letras.contador←letras.contador-1
     FIN_SI
     aux->sig←NULO
     aux->ant←NULO
   FIN_SI
   quitar_inicio←aux
FIN
```

```
pnodo quitar_inicio(tlista &letras)
{ pnodo aux;
  if (letras.contador==0)
    aux=NULL;
  else
   { aux=letras.inicio;
     if (letras.contador==1)
       iniciar_lista(letras);
     else
      { letras.inicio=letras.inicio->sig;
        letras.inicio->ant=letras.final;
        letras.final->sig=letras.inicio;
        letras.contador--;
      }
     aux->sig=NULL;
     aux->ant=NULL;
   }
   return aux;
}
```

Finalmente, la operación *mostrar_lista* se realiza mediante un procedimiento que muestra el contenido de la lista. Para ello se accede, uno a uno, a los nodos de la lista salvo el último que se muestra fuera del bucle de recorrido. Este nodo se trata por separado ya que el bucle finaliza sin mostrar su contenido.

```
PROCEDIMIENTO mostrar_lista(E let:tlista)
VARIABLES  i:pnodo
INICIO
  SI let.contador=0 ENTONCES
    ESCRIBIR "Lista Vacía"
  SINO
    i←let.inicio
    MIENTRAS (i->sig <> let.inicio) HACER
      ESCRIBIR i->dato
      i←i->sig
    FIN_MIENTRAS
    ESCRIBIR i->dato
  FIN_SI
FIN
```

```
void mostrar_lista(tlista let)
{ pnodo i;
  if (let.contador==0)
    cout << "Lista Vacia" << endl;
  else
   { i=let.inicio;
     while (i->sig!=let.inicio)
      { cout << i->dato << endl;
        i=i->sig; }
     cout << i->dato << endl;
   }
}
```

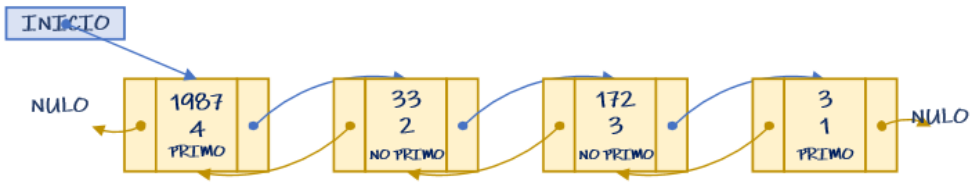
## EJERCICIOS

1) De acuerdo a la definición de *Lista Doble*, implemente sus operaciones fundamentales, considerando:
   - Una lista requiere de elementos, llamados nodos, que almacenen datos y que posean un indicador del próximo elemento de la lista.
   - Una operación de inicialización que permita crear (inicializar) una lista vacía.
   - Una operación que permita crear nodos.
   - Una operación de inserción que permita agregar un nuevo nodo al inicio de la lista.
   - Una operación de inserción que permita agregar un nuevo nodo al final de la lista.
   - Una operación de inserción que permita agregar, en orden, un nuevo nodo a la lista.
   - Una operación que extraiga un nodo del inicio de la lista.
   - Una operación que extraiga un nodo del final de la lista.
   - Una operación que extraiga un nodo específico (según un valor ingresado por el usuario) de la lista.
   - Una operación que permita buscar un nodo (valor) en la lista.
   - Una operación que permita mostrar el contenido de la lista.

Suponga que la implementación corresponde a una lista de números enteros, realizándose en 2 variantes:

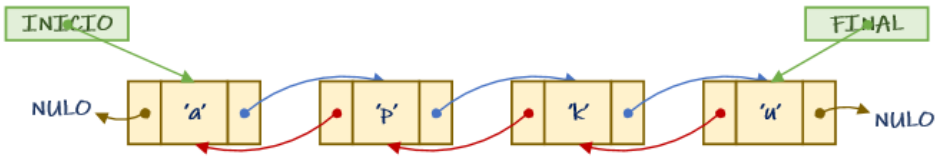
a) implementación con un único puntero al inicio de la lista
b) implementación con punteros al inicio y al final de la lista

2) Dada la siguiente lista doble de enteros:



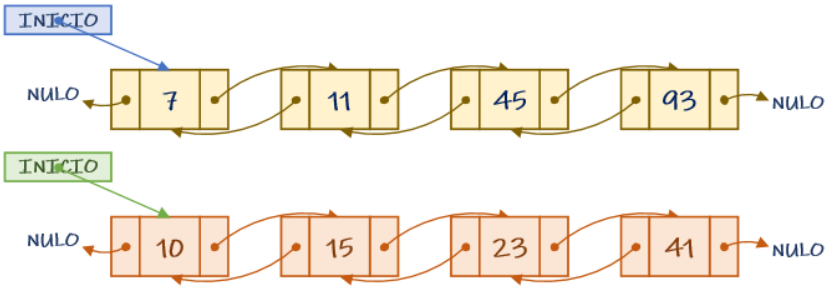

a) Consigne la declaración de tipos y variables de la estructura. Tenga en cuenta que, en cada nodo, debe registrarse la cantidad de dígitos del valor almacenado así como si se trata de un valor primo o no.
b) Diseñe un procedimiento/función que permita agregar un nodo al final de la lista con la siguiente restricción: no podrán almacenarse números primos en nodos consecutivos (como se muestra en la figura). Los nodos que no puedan agregarse deberán ser liberados.
c) Diseñe un procedimiento/función que muestre el mayor valor primo almacenado en la lista.
d) Diseñe un procedimiento/función que el contenido de la lista desde el principio hacia o final o viceversa según un parámetro de opción.

3) Dada una lista doble de caracteres realice lo siguiente:




a) Consigne la declaración de tipos y variables de la estructura.
b) Diseñe un procedimiento/función que agregue un nodo al final de la lista. Considere que la lista está limitada a 10 nodos, debiendo liberarse aquellos que no puedan agregarse.
c) Diseñe un procedimiento/función que invierta el contenido de la lista (por intercambio de datos entre los nodos) sin utilizar estructuras auxiliares (arreglos u otras listas). Por ejemplo, la lista invertida de la figura será 'u' ->' k' -> 'p' -> 'a'.

4) Dadas 2 listas dobles de enteros con único puntero de *inicio*:




a) Defina las estructuras de datos que permitan implementarlas

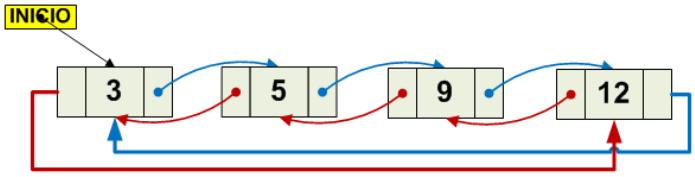
b) Diseñe un procedimiento/función que permita agregar en orden los nodos a las listas. Adicionalmente, diseñe un módulo que realice la carga aleatoria de valores considerando que el usuario especificará la cantidad de datos a cargar.

c) Diseñe un procedimiento/función que, a partir de las 2 listas, genere una lista nueva con los valores ordenados de las originales. Implemente 2 variantes:

- La lista de mezcla se genera con los nodos de las listas originales, dejando éstas vacías.
- La lista de mezcla se genera con nodos nuevos cuyos valores son copiados de las listas originales (no se modifican).


Considere que puede reutilizar las operaciones básicas de listas dobles para diseñar la solución.

5) Dada la siguiente lista doble **CIRCULAR**:




a) Defina las estructuras de datos que permitan representarla.

b) Implemente las operaciones *iniciar_lista*, *agregar_orden*, *quitar_nodo* y *buscar_valor*. Diseñe la búsqueda de modo que se aproveche el orden de la lista para reducir el recorrido.

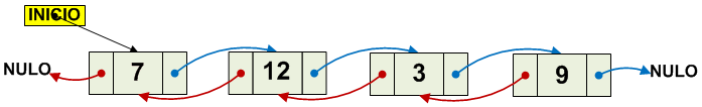
c) Implemente las operaciones **recursivas** mostrar, sumar nodos y contar valores impares. Considere que el módulo mostrar podrá presentar el contenido de la lista desde el inicio hacia el final o viceversa según un parámetro de opción.

6) Dada la siguiente definición de listas doblemente enlazadas, cuya capacidad se limita a 18 elementos, implemente las operaciones: *iniciar_lista*, *agregar_inicio*, *quitar_final y lista_llena.* ¿Cómo se modifican las operaciones si la lista es circular?

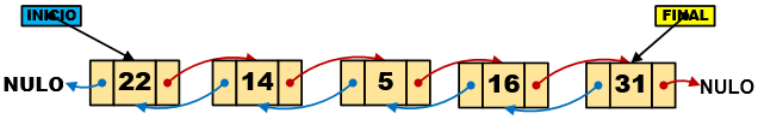
```
typedef struct tnodo *pnodo;
typedef pnodo tpunteros[2];
typedef struct tnodo {
                      char dato;
                      tpunteros vecino;
                     };
typedef struct tlista {
                      pnodo inicio;
                      pnodo final;
                      int contador;
                     };
```

7) Dados los siguientes módulos consigne las **definiciones** de datos usadas, indique los casos evaluados y determine el **propósito** de cada módulo. Además, escriba la versión equivalente recursiva o iterativa según corresponda.

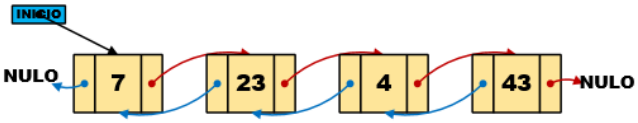
```
void misterio(pnodo &i, pnodo n)
{    if (i==NULL)
       i=n;
     else
      { if (i->sig==NULL)
        { i->sig=n;
          n->ant=i;
        }
        else
         misterio(i->sig,n);
      }
}
```



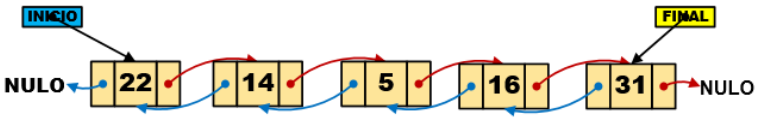

```
void enigma (pnodo i, pnodo f)
{ int aux;
   if (i!=NULL)
    if (i!=f && i->ant!=f)
     { aux=i->dato;
       i->dato=f->dato;
       f->dato=aux;
       enigma(i->sig,f->ant);
     }
}
```



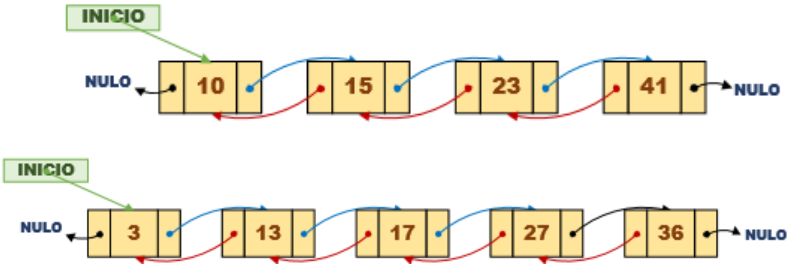

```
pnodo oculto(pnodo i)
{ pnodo m;
  if (i!=NULL)
  { if (i->sig==NULL)
       m=i;
    else
    { m=oculto(i->sig);
      if (i->dato > m->dato)
         m=i;
    }
  }
  else
    m=NULL;
  return m;
}
```




```
bool secreto(tlista lt,int m)
{ pnodo i,j;
  bool b=false;
  i=lt.inicio;
  j=lt.final;
  while (i!=j && i->ant!=j && b==false)
  { if(i->dato==m || j->dato==m)
       b=true;
    i=i->sig;
    j=j->ant;
  }
  if (i==j && i->dato==m)
    b=true;
  return b;
}
```



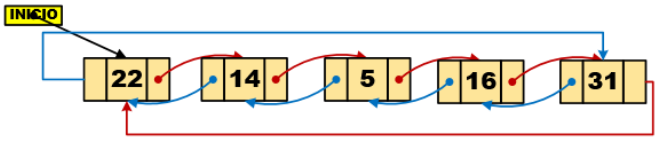
```
void incognita(pnodo a, pnodo b, pnodo &c)
{ pnodo n;
  if (a!=NULL && b!=NULL)
  { if (a->dato < b->dato)
    { crear_nodo(n,a->dato);
      agregar_final(c,n);
      incognita(a->sig,b,c);
    }
    else
    { crear_nodo(n,b->dato);
      agregar_final(c,n);
      incognita(a,b->sig,c);
    }
  }
  else
  { if (a!=NULL)
    { crear_nodo(n,a->dato);
      agregar_final(c,n);
      incognita(a->sig,b,c);
    }
    else
     if (b!=NULL)
      { crear_nodo(n,b->dato);
        agregar_final(c,n);
        incognita(a,b->sig,c);
      }
  }
 }
}
```




8) Utilizando listas dobles circulares, con punteros de inicio y final, diseñe un programa que permita simular una ruleta rusa. Para ello, considere que la ruleta se carga con 37 valores (entre 0 y 36) dispuestos aleatoriamente, asignándose los colores rojo y negro de forma alternada a los nodos de la lista (salvo el número cero, que no tendrá color). El juego consiste en que el jugador elija un número, un color o una paridad, gire la ruleta y una vez que ésta se detenga entonces se verifique si el jugador ganó la apuesta (por número, por color, por paridad). Considere que el movimiento de la ruleta se emulará recorriendo la lista una *N* cantidad de nodos en sentido horario o antihorario. Tanto la cantidad de nodos a recorrer como el sentido del recorrido se determinará de forma aleatoria.

Según el resultado de juego presente los siguientes mensajes:

- "ERES UNA PERSONA CON SUERTE ☺ APROVECHA TU RACHA Y SIGUE JUGANDO ..."
- "HOY NO ES TU DÍA, LA FORTUNA NO ESTÁ DE TU LADO"




9) El director de una empresa desea mantener registrada información acerca de sus empleados y las sucursales en las que éstos trabajan. Por cada empleado debe registrarse: legajo, apellido, nombre, DNI, fecha de nacimiento (día, mes, año), cargo (puesto, horas de trabajo, salario), antigüedad y código de sucursal. Mientras que, por cada sucursal debe

almacenarse: código de sucursal, nombre, ubicación (localidad, provincia) y fecha de apertura (día, mes, año). En virtud de lo enunciado se pide:

a) Consigne la declaración de tipos y variables que represente la situación planteada.
b) Diseñe los procedimientos/funciones que listen, por cada sucursal, los empleados que trabajan en ellas. Además indique cuál es la sucursal con la mayor cantidad de empleados.
c) Diseñe los procedimientos/funciones que identifiquen al empleado con la mayor antigüedad y en qué sucursal trabaja.

10) Dada la siguiente definición de datos:

```
typedef char tcad[30];
typedef int tfecha[3];
typedef struct tdocente *pdocente;
typedef struct tdocente { int legajo;
                          tcad apellido;
                          tcad nombre;
                          int dni;
                          tcad título;
                          tcad cargo;
                          pdocente ant;
                          pdocente sig;
                         };
typedef struct tlista_doc {
                            pdocente inicio;
                            int contador;
                           };
typedef struct tfacu { tcad facultad;
                       tcad universidad;
                       tcad sede;
                       tlista_doc docentes;
                      };
typedef tfacu tfacultades[10];
```

a) Diseñe los procedimientos/funciones que permitan buscar un docente (por apellido y nombre) y mostrar a qué facultad pertenece.
b) Diseñe los procedimientos/funciones que permitan listar todos los docentes (apellido, nombre y título) de una facultad solicitada por el usuario.