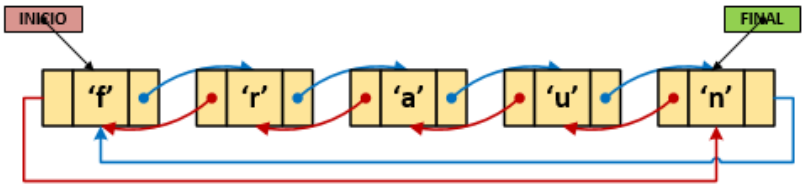
EJEMPLOS

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;
   }
}
```

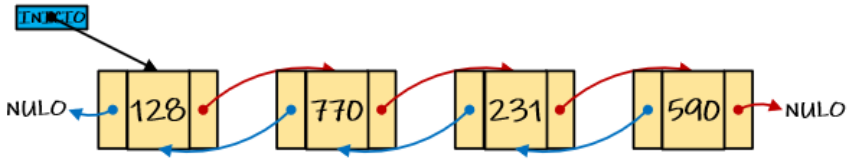
EJERCICIOS

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:

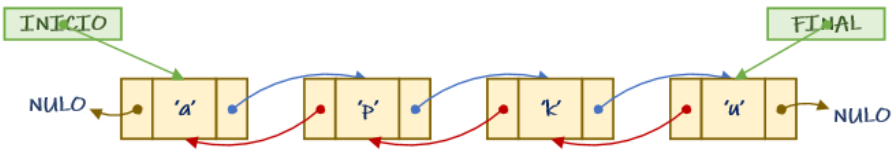
a) implementación con un único puntero al inicio de la lista
b) implementación con punteros al inicio y al final de la lista

2) Dada la siguiente lista doble de enteros:



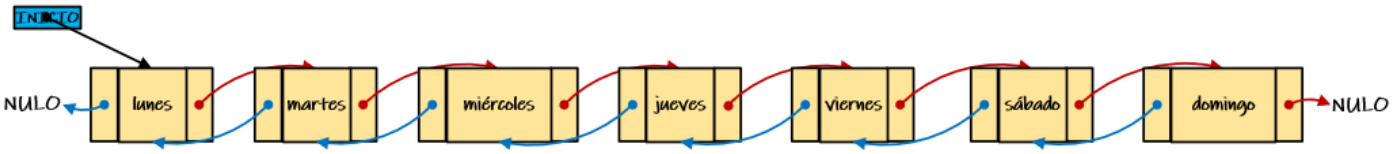
a) Consigne la declaración de tipos y variables de la estructura. Considere que el usuario especificará la cantidad máxima de nodos que podrán almacenarse (tamaño de la lista).
b) Diseñe un procedimiento/función que permita cargar la lista con valores aleatorios de 3 cifras. Considere que el usuario indicará la cantidad de valores que serán agregados, incorporándose éstos por el final de la lista. Si la cantidad solicitada por el usuario supera la capacidad de la lista, debe presentarse un mensaje de advertencia.
c) Diseñe un procedimiento/función que, según un parámetro de opción, ordene el contenido de la lista de menor a mayor o viceversa. Tenga en cuenta que la operación sólo intercambiará los datos almacenados (no moverá los nodos), y que el recorrido de ordenamiento NO debe duplicarse (el mismo recorrido debe permitir trabajar con creciente o decreciente).

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



a) Consigne la declaración de tipos y variables de la estructura. Considere que no deben guardarse datos repetidos
b) Diseñe un procedimiento/función que, según un parámetro de opción, permita agregar un nodo al inicio o final de la lista, liberando aquellos que no puedan agregarse.
c) Diseñe un procedimiento/función que, según un parámetro de opción, muestre el contenido de la lista desde el primer nodo hacia el último o desde el último nodo hacia el primero. Implemente la operación usando ÚNICAMENTE un bucle de recorrido.
d) Modifique el módulo del ítem c) para trabajar sobre una lista con un único puntero de *inicio*.

4) Dada una lista doble de cadenas de caracteres realice lo siguiente:

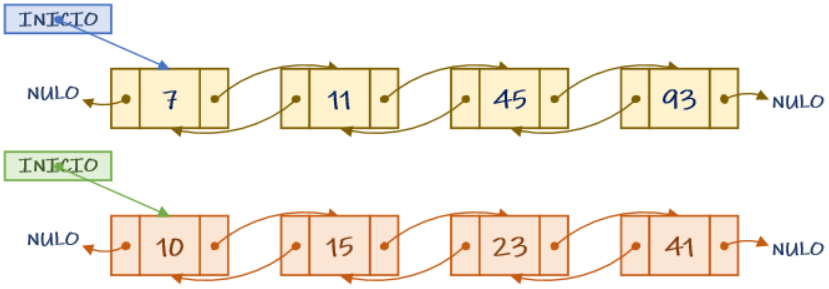


a) Consigne la declaración de tipos y variables de la estructura.

b) Diseñe un procedimiento/función que permita al usuario elegir el primer día de la semana (inicio de la lista). Por ejemplo, si el usuario indica "miércoles" entonces la lista será: miércoles > jueves > viernes > sábado > domingo > lunes > martes. Considere que, por defecto, la lista presentará el orden mostrado en la imagen.

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: domingo > sábado > viernes > jueves > miércoles > martes > lunes.

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



a) Defina las estructuras de datos que permitan implementarlas

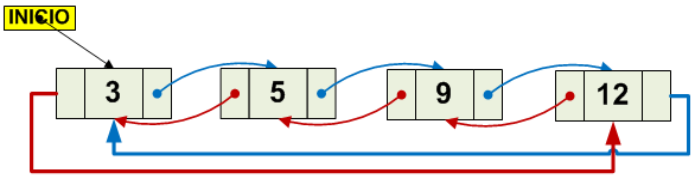
b) Diseñe un procedimiento/función que permita agregar en orden los nodos a las listas. Adicionalmente, diseñe un módulo que realice la carga aleatoria de valores considerando que el usuario especificará la cantidad de datos a cargar.

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.

6) 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) ¿Cómo se modifican las operaciones del ítem b) si la lista tiene punteros de *inicio* y *final*?

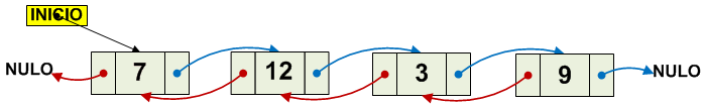
d) Implemente las operaciones **recursivas** *mostrar* (de *inicio* a *final* o viceversa según parámetro de opción), *máximo* (dirección del nodo con el mayor valor) y *contar impares/pares* (según parámetro de opción).

7) Dada la siguiente definición de listas doblemente enlazadas, cuya capacidad se limita a 20 elementos, implemente las operaciones: *iniciar_lista*, *agregar_final*, *quitar_inicio* y *lista_llena*.

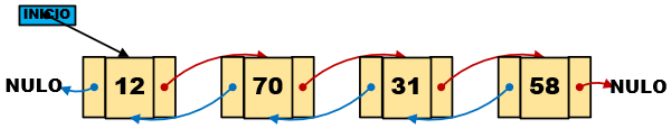
```
typedef struct tnodo *pnodo;
typedef pnodo tpunteros[2];
typedef struct tnodo {
                       char dato;
                       tpunteros vecino;
                      };
typedef struct tlista {
                       tpunteros posicion;
                       int contador;
            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

8) 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.

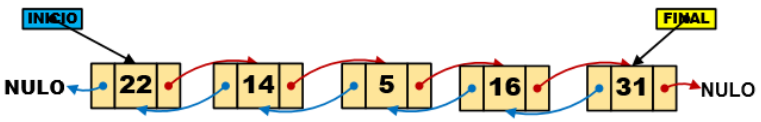
```
void misterio(pnodo &i, pnodo n)
{   if (i==NULL)
     i=n;
   else
    { if (i->sig==NULL)
      { i->sig=n;
        n->ant=i;
      }
      else
       misterio(i->sig,n);
    }
}
```



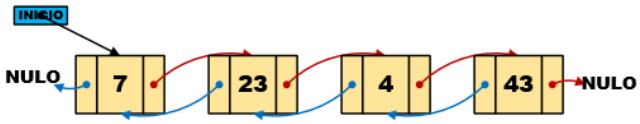

```
pnodo enigma(pnodo &inicio)
{ pnodo aux;
  if (inicio==NULL)
     aux=NULL;
  else
  { if (inicio->sig==NULL)
    { aux=inicio;
      aux->ant=NULL;
      inicio=NULL;
    }
    else
      aux=enigma(inicio->sig);
  }
  return aux;
}
```



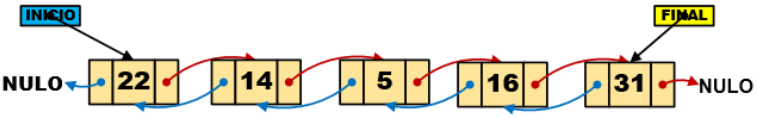

```
bool secreto(pnodo i, pnodo f,int n)
{ bool aux;
   if (i==NULL)
      aux=false;
   else
   { if (i->dato==n || f->dato==n)
       aux=true;
     else
     { if (i->sig==f || i==f)
         aux=false;
       else
         aux=enigma(i->sig,f->ant);
     }
   }
   return aux;
}
```



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



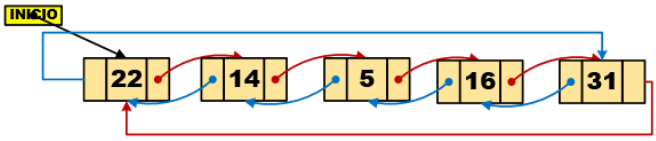

```
bool incognita(tlista lis)
{ pnodo i,j;
  bool b=true;
  if (lis.inicio==NULL)
     b=false;
  else
  { i=lis.inicio;
    j=lis.final;
    while (i!=j && i->ant!=j && b==true)
    { if (i->dato!=j->dato)
        b=false;
      i=i->sig;
      j=j->ant;
    }
  return b;
}
```



9) 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 apostador elija un número (premio $10.000), un color (premio $5.000) y/o una paridad (premio $3.000), gire la ruleta y se verifique si el jugador ganó o no la apuesta. 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 elegirá de forma aleatoria.

Según el jugador haya acertado el número, color y/o paridad presente los siguientes mensajes:

- "LA BUENA FORTUNA TE SONRÍE ☺, HOY ES TU DÍA, SIGUE JUGANDO!!! – PREMIO $XX.XXX"
- "NO TE RINDAS, SIGUE INTENTANDO, AÚN PUEDES GANAR!!!"



10) El secretario académico de la universidad desea mantener registrada información acerca de los docentes y facultades en las que éstos cumplen funciones. Por cada docente debe registrarse: legajo, apellido, nombre, DNI, fecha de ingreso (día, mes, año), cargo (nombre, dedicación, materia), antigüedad y código de facultad. Mientras que, por cada facultad debe almacenarse: código de facultad, nombre, autoridades (decano, vicedecano) y domicilio (calle, número, barrio, localidad, provincia). 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 facultad, los docentes que cumplen funciones en ellas. Además, indique cuántos docentes se desempeñan en cada unidad académica.
c) Diseñe los procedimientos/funciones que muestren, por cada facultad (nombre y decano), los docentes de mayor antigüedad (apellido, nombre, año de ingreso).