Tercera Unidad

Estructura de datos lineales dinámicos

Sumario

Comprende y sintetiza los conceptos asociados a las listas enlazadas. Se abstrae Identifica y clasifica las características y acciones básicas (primitivas) relacionadas a las listas enlazadas.

- Listas lineales simples
- Otras estructuras dinámicas

Lección 5

Listas lineales simples

"La lista enlazada consta de una serie de registros que no necesariamente son adyacentes en memoria. Cada registro contiene el elemento y un apuntador a un registro que contiene su sucesor. A este se le llama el apuntador siguiente. El apuntador siguiente de la ultima celda apunta a nil; este valor esta definido en pascal y no puede confundirse con algún otro apuntador. Un valor característico usado para nil es 0."(Allen, 1995, p. 47).

1.1. Características

- La lista forma parte de lo que se denominan estructuras lineales.
- Intuitivamente son estructuras en los que los distintos elementos se sitúan en línea, de ahí su nombre. Dicho de otro modo, cada elemento de una estructura lineal, salvo el primero y el último, tan sólo tienen un anterior y un siguiente. En una lista cada elemento apunta al siguiente excepto el último que no tiene sucesor y el valor del enlace es la dirección vacia o nulo (NULL en C++).
- Se denomina nodo a cada uno de los elementos de la lista, los que pueden contener datos simples o TAD's. Cada nodo tiene dos partes:

  a) **valor**.- Representa el dato a almacenar. Puede ser de cualquier tipo.
  b) **enlace**.- Es la conexión o lo que en C++ se conoce como puntero o apuntador, con este puntero enlazamos con el sucesor (siguiente), de forma que podemos construir la lista. El enlace se representa gráficamente como una flecha (→)

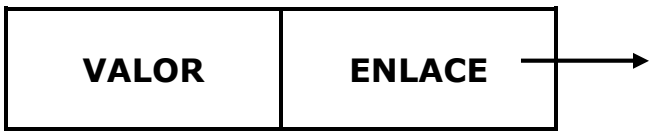


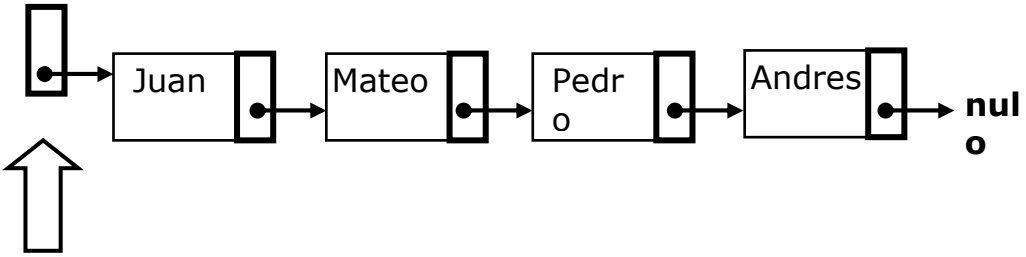
Figura 5.1. Estructura de un nodo

"Reconocemos algunas propiedades comunes de la colección en cada una de las listas:

- *Es homogénea, es decir todos los elementos son del mismo tipo*
- *Tiene longitud finita(el numero de elementos).*
- *Los elementos están colocados de manera secuencial." (Larry, 2006,254)*

El primer elemento de la lista es direccionado por un puntero conocido como raíz o "Cabecera de Lista", que contiene la dirección del primer elemento de la lista y el último elemento elemento de la

lista tiene un apuntador o puntero que apunta a una dirección vacia, lo que en C++ se expresa por NULL (nulo).



raiz o cabecera de la lista

Figura 5.2. Representación de una lista

No se puede acceder a los nodos de una lista enlazada directamente. Hemos de acceder al primer nodo de la lista mediante el puntero raiz o "cabecera de lista", al segundo nodo mediante el puntero "siguiente" del primer nodo, al tercer nodo mediante el puntero "siguiente" del segundo nodo y así sucesivamente hasta el final de la lista.

"Una lista es una colección de elementos de información dispuestos en un cierto orden. A diferencia de las matrices y registros, el numero de elementos de la lista no suele estar fijado, ni suele estar limitado por anticipado." (Brassard, 1998, 171).

1.2. Operaciones

- **Insertar**: inserta un nodo en la lista, pudiendo realizarse esta inserción al principio, al medio o al final de la lista, dándose por posición o por valor.
- **Eliminar**: elimina un nodo de la lista.
- **Buscar**: busca un elemento en la lista.
- **Vaciar**: borra todos los elementos de la lista

1.3. Tipos de Listas

- **Lista lineal simple:** el último nodo apunta a la dirección vacia o nulo.

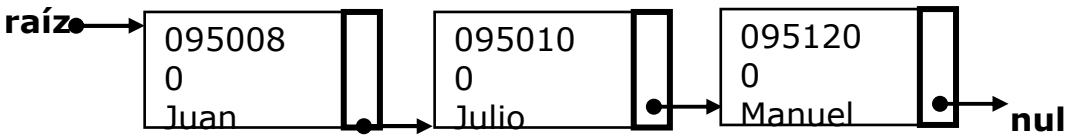


Figura 5.3. Lista lineal simple

- **Lista circular:** El último nodo apunta al primer nodo.

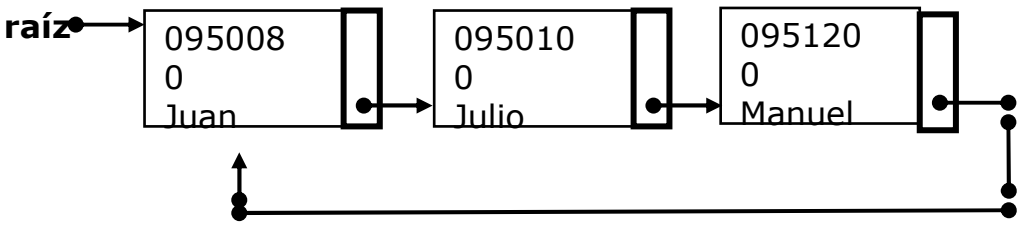


Figura 5.4. Lista circular

- **Lista lineal doblemente enlazada:** Los nodos, aparte de tener un enlace al nodo siguiente, tienen un enlace al nodo anterior.

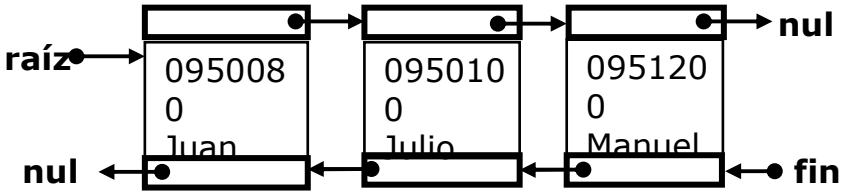


Figura 5.5. Lista lineal doblemente enlazada

1.4. Implementación del TAD lista

Las operaciones de insertar o eliminar nodos en una lista pueden hacerse en cualquier posición de la lista, sin embargo existe una limitación que se presenta por la estructura con la cual fue implementada la lista.

a) Estructuras estáticas (vectores)

- En este tipo de estructura, la estructura de arreglo es la que define cual es el próximo nodo de la lista.
- Su tamaño no cambia durante la ejecución del programa

b) Estructuras dinámicas(asignación dinámica)

- Cuando se implementa bajo esta estructura la direccion del nodo que sigue en una lista, esta determinado por el puntero del nodo actual. Es necesario almacenar la dirección de memoria del primer elemento.
- Es dinámica, es decir, su tamaño cambia durante la ejecución del programa.

1.5. Punteros

Un puntero es una variable que almacena la dirección de memoria de un elemento determinado. Para trabajar con punteros C++ dispone de los operadores * y &.

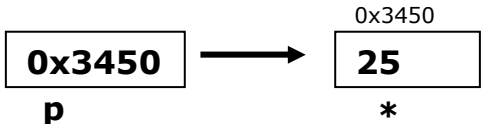
- El operador * es usado para declarar una variable puntero como se observa:

```
tipo *variablePuntero;
```

Donde tipo puede ser int, double,..etc. Por ejemplo:

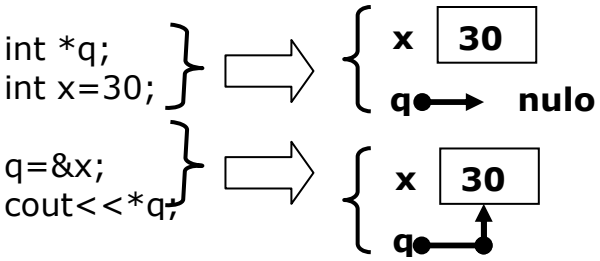
```
int *p;       // declara un puntero a int
*p=25;
```

significa que el valor 25 se registra en una dirección de la memoria(0x3450) a la cual apunta el puntero p.



También nos sirve para acceder directamente al valor almacenado, tal es asi que una instrucción *cout<<*p* visualizara el valor almacenado en **p*.

- El operador & de dirección o también denominado de referencia se utiliza para obtener la dirección de una variable cualquiera, ya que todas las variables se registran en una dirección de memoria. Por ejemplo:



la variable q almacena la direccion de x y ahora apunta a la direccion de x.

Ambos operadores se utilizan también en los parámetros de las funciones, asi como en la declaración de tipos compuestos:

- Cuando usamos en la zona de parámetros la sintaxis *& en la cabecera de una función como se observa en el siguiente ejemplo

```
void ingresaLista(LISTA *&raiz, ALUMNO x)
```

estamos haciendo uso de un parámetro puntero por referencia. Sino se usará el ampersand (&) la lista que se pasa a esta función no se actualizaria una vez terminada la llamada a la función.

- Para declarar un puntero a un objeto se utiliza la misma sintaxis que se usa con cualquier otro tipo de dato como se observa abajo:

```
nombreDeLaClase  *variablePuntero;
```

Por ejemplo:

```
LISTA  *p;  // p es una variable puntero de la clase LISTA
```

- Para asignar memoria dinámica a una variable usamos el operador new

```
nombreDeLaClase *variablePuntero = new nombreDeLaClase;
```

Por ejemplo:

```
LISTA  *ptr = new LISTA;
```

El operador new:

  - Crea el espacio para el objeto
  - Llama al constructor especifico

- Cuando usamos delete, queremos eliminar una variable que esta en la memoria, y a la cual se le asigno memoria con new usando la siguiente sintaxis:

```
delete variablePuntero;
```

Por ejemplo:

```
delete ptr;
```

El operador delete:

  - Llama al destructor
  - Libera el espacio ocupado por el objeto

Otro aspecto importante en el uso de los punteros es cuando se trabajan con TAD's, tal es asi que para acceder a los miembros de una clase, se usa el operador punto(.) si es que el objeto no es un puntero y el operador flecha(->) en caso el objeto si lo sea. Por ejemplo sean los métodos denominados *conseguirDatos*() y *visualizarDatos*(), miembros de la clase PUNTERO, y *obj* un objeto de esta misma clase y *pt* un puntero a un objeto de la clase PUNTERO como se observa abajo:

```
PUNTERO  obj, *pt;
obj.conseguirDatos();
pt->visualizarDatos();
```

A un puntero se le puede asignar un objeto de dos maneras:

- Asignándole la dirección de un objeto existente.
- Localizando espacio en memoria mediante el operador new.

Usando un puntero se puede recibir la dirección de otra variable mediante ampersand (&) y asignar memoria al puntero con el operador new como se observa en el siguiente ejemplo:

```
#include <iostream>
#include <cstdlib>
using namespace std;
class EJEMPLO{
	int num;
	public:
		void capturaEntero(int val)
		{
			num=val;
		}
		void mostrarEntero()
		{
			cout << num << "\n";
		}
};
int main(void)
{
	EJEMPLO ob; //declarar un objeto y un puntero a él
	EJEMPLO *p=new EJEMPLO;
	ob.capturaEntero(1); //acceso a ob directamente
	ob.mostrarEntero();
	p=&ob;
	p->mostrarEntero(); // acceder a ob usando un puntero
	system("PAUSE");
	return 0;
```

```
}
```

Salida del programa:

```
1
1
```

1.6. Primitivas

Usando la capacidad de la asignación dinámica la lista puede aumentar o disminuir su tamaño. Esto es razón suficiente para enfocar nuestro estudio en esta nueva forma de implementación que no se basa en arreglos y del cual ya conocimos su uso y utilidad en el capitulo anterior.

primitiva	FILAS: fila F y variable val con información del registro	ARREGLOS: arreglo T con número de elementos N	LISTA: nodo T y variable val con información del nodo
Inicio	Inicio(F)	i ←0	Crear(T)
Leer	Leer(F, val)	val ← T[i]	Leer(T, val)
Avanzar	Avanza con cada lectura	i ← i +1	Sgte(T)
Verificar final	Ultimo (F)	i < N	T ≠ nulo
escribir	Escribir(F, val)	T [i] ← val	Asignar(T, val)
terminar	Cerrar(F)	No existe	Liberar(T)

Tabla 5.1. Primitivas para Filas, Arreglos y Listas

Comprensión de la notación expuesta en el cuadro anterior:

- Crear(T) : asigna un espacio de memoria para el nodo T
- Sgte(T) : devuelve el siguiente nodo de T
- Asignar(T, j) : asignar un valor *j* a un nodo T.
- Leer(T,val) : captura en una variable *val* el valor que contiene el nodo T
- Liberar(T) : saca al nodo T de memoria, es el caso inverso de Crear.

El esquema gráfico de abajo puede ayudarnos más en la tarea de comprender como en una lista la información y los enlaces están dispuestos en cada nodo y como a su vez los punteros o enlaces relacionan cada nodo de la lista.

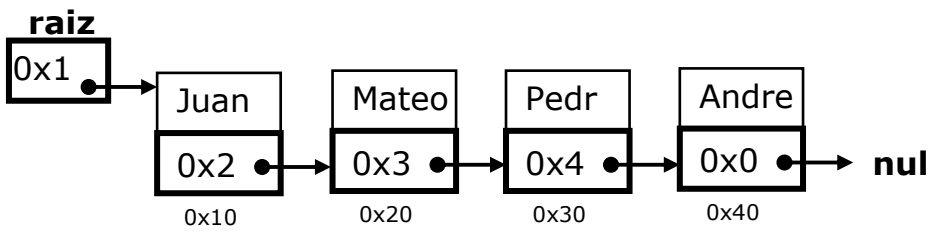


Figura 5.6. Lista lineal visualizando sus posiciones en memoria.

Los números debajo de cada nodo suponen la dirección en la que se ubica cada nodo y que nos servirá para explicar con más detalle el enlazado de los nodos.

Observe que existe un puntero que contiene la dirección del primer nodo cuya dirección es 0x10, asi mismo este nodo tiene un puntero que tiene un valor 0x20 que es la dirección del siguiente nodo que a su vez tiene un puntero con valor 0x30 y que es la dirección del siguiente nodo. Si seguimos veremos que el último nodo tiene un puntero con valor 0x00(nulo) y que nos indica el final de la lista.

1.7. Construcción de listas

Se hará la especificación de una lista con nombre LISTA donde cada nodo contiene como información un número entero.

Especificacion LISTA
variable
info : entero
sgte : LISTA
métodos
ingresarLista(raiz,x) :no retorna valor
mostrarLista(raiz) :no retorna valor
significado
ingresaLista registra en la lista con cabecera raiz del tipo LISTA un número de tipo entero.
mostrarLista muestra todos los números de la lista con cabecera raiz del tipo LISTA
Fin_especificacion

```
Procedimiento ingresarLista (raiz, x)
    // definir variables
    LISTA: q
    crear (n)
    Asignar ( n, x )
    Si raiz = nulo entonces
        raiz ← n
    Sino
        q ← raiz
        Mientras (sgte(q) ≠ nulo) hacer
            q ← sgte(q)
        Fin_mientras
        sgte(q) ←n
    Fin_si
    sgte(n) ← nulo
Fin_procedimiento

Procedimiento mostrarLista (raiz)
```

```
        // Definir variables
        LISTA: ptr
        ptr ← raiz
        Mientras ( ptr ≠ nulo) Hacer
            Leer (ptr, info)
            escribir info
            ptr ← sgte(ptr)
        Fin_mientras
    Fin_procedimiento
```

Implementación del TAD en C++

```
    #include <iostream>
    #include <cstdlib>
    using namespace std;
    class LISTA
    {
        int info;
        LISTA  *sgte;
        public:
        void menu(){
                cout<< "\nMENU DE OPCIONES\n";
                cout<< "----------------\n" ;
                cout<<"<1> Ingresar Lista\n";
                cout<<"<2> Mostrar Lista\n";
                cout<<"<3> Salir \n";
        }
        void ingresaLista (LISTA *&raiz, int  x){
            LISTA *q;
            LISTA *n =new LISTA;
            n->info=x;
            if(raiz==NULL)
                raiz=n;
            else{
                q=raiz;
                while (q->sgte!= NULL)
                    q=q->sgte;
                q->sgte=n;
            }
            n->sgte=NULL;
        }
        void mostrarLista (LISTA *raiz){
            LISTA *ptr;
            ptr= raiz;
            while( ptr!= NULL)      {
                        cout<<"  "<<ptr->info;
                        ptr=ptr->sgte;
            }
```

```
            cout<<endl;
        }
};
int main(){
        char opcion;
        int x;
        LISTA lista;
        LISTA *raiz=NULL;
        do
        {
                lista.menu();
                cout<<"\ningrese opcion : ";
                opcion=cin.get();
                switch(opcion)
                {
                        case '1':
                                cout<<"\n ingrese un numero:
                        ";cin>>x;
                                lista.ingresaLista(raiz,x);
                                break;
                        case '2':
                                lista.mostrarLista(raiz);
                                break;
                }
                cin.ignore();
        }
        while( opcion !='3');
        system("PAUSE");
        return 0;
}
```

1.8. Ejercicios resueltos

Ejemplo 01

Escriba la especificación de los TAD's, los algoritmos y su implementación en C++ para registrar los datos de los alumnos en una lista.

Solución:

Especificación de los TAD'S y los algoritmos

Especificacion ALUMNO
    variable
        codigo : cadena.
        nombre : cadena.
        edad : entero.

métodos

menu() : no retorna valor.
ingresarDatos() : no retorna valor.
mostrarDatos() : no retorna valor.

significado

menu muestra las opciones a escoger.
ingresarDatos ingresa los datos de un alumno.
mostrarDatos visualiza los datos de un alumno.

Fin_especificacion

Especificacion LISTA

usa ALUMNO

variable

info : ALUMNO.
raiz : LISTA.
sgte : LISTA.

métodos

ingresarLista(raiz,x) :no retorna valor.
mostrarLista(raiz) :no retorna valor.

significado

ingresaLista registra en la lista con cabecera raiz del tipo LISTA un alumno *x* del tipo ALUMNO.
mostrarLista muestra todos los alumnos de la lista con cabecera raiz del tipo LISTA.

Fin_especificacion

Implementación del TAD

```
#include <iostream>
#include <cstdlib>
using namespace std;
class ALUMNO
{
	char codigo[10];
	char nombre[40];
	int edad;
	public :
		void menu()
		{
			cout<< "\nMENU DE OPCIONES\n";
			cout<< "----------------\n" ;
			cout<<"<1> Ingresar Lista\n";
			cout<<"<2> Mostrar Lista\n";
			cout<<"<3> Salir \n";
		}
		ALUMNO(){ strcpy(codigo,"");strcpy(nombre,"");int
nota=0; }
```

```
            void ingresarDatos()
            {
                  fflush(stdin);
                  cout<<"ingrese codigo : ";
                  gets(codigo);fflush(stdin);
                  cout<<"ingrese nombre : ";
                  gets(nombre ); fflush(stdin);
                  cout<<"ingrese nota : ";
                  cin>>edad;
            }
            void mostrarDatos()
            {
                  cout<<"\n "<<codigo<<"    "<<nombre<<"
"<<edad;
            }
};
class LISTA
{
      ALUMNO info;
      LISTA  *sig;
      public:
            void ingresaLista (LISTA *&raiz,ALUMNO x)
            {
                        LISTA *q;
                        LISTA *n =new LISTA;
                        n->info=x;
                        if(raiz==NULL)
                              raiz=n;
                        else
                        {
                              q=raiz;
                              while (q->sig!= NULL)
                                    q=q->sig;
                              q->sig=n;
                        }
                        n->sig=NULL;
            }
            void mostrarLista (LISTA *raiz)
            {
                  LISTA *ptr;
                  ptr= raiz;
                  while( ptr!= NULL)
                  {
                        ptr->info.mostrarDatos();
                        ptr=ptr->sig;
                  }
                  cout<<endl;
            }
```

```
};
int main()
{
	char opcion;
	ALUMNO x;
	LISTA lista;
	LISTA *raiz;
	raiz=NULL;
	do
	{
		x.menu();
		cout<<"\ningrese opcion : ";
		opcion=cin.get();
		switch(opcion)
		{
			case '1':
				x.ingresarDatos();
				lista.ingresaLista(raiz,x);
				break;
			case '2':
				lista.mostrarLista(raiz);
				break;
		}
		cin.ignore();
	}
	while( opcion !='3');
	system("PAUSE");
	return 0;
}
```

Ejemplo 02

Escriba la especificación de los TAD's, el algoritmo y su implementación en C++, que permita registrar en una tercera lista, la concatenación de dos listas dinámicas simple. Por ejemplo:

Entrada
- Lista 1: 20, 3, 7, 9
- Lista 2: 5, 6

Salida
- Lista 3: 20, 3, 7, 9, 5, 6

Solución:

Especificación de los TAD's y los algoritmos

Especificación ENTERO
variable
num : entero
métodos
menu() : no retorna valor.
ingresarEntero( ) : no retorna valor.
mostrarEntero( ) : no retornar valor.
significado
menu muestra las opciones a escoger.
ingresarEntero ingresa un valor entero.
mostrarEntero visualiza un valor entero.
Fin_especificacion

Especificación LISTA
Usa ENTERO
variable
info : ENTERO
sgte : LISTA
métodos
ingresarLista(raiz, x) : no retorna valor.
concatenarLista(raiz1,raiz2, raiz3) : no retorna valor.
mostrarLista(raiz) : no retorna valor.
significado
ingresaLista registra en la lista con cabecera raiz de tipo LISTA un valor x de tipo ENTERO.
concatenarLista tiene como precondición a la lista1 y a la lista2, y como postcondición a la lista3.
mostrarLista muestra los datos de la lista con cabecera raiz de tipo LISTA.
Fin_especificacion

```
Procedimento concatenar(raiz1, raiz2, raiz3)
    // definir variables
     LISTA: p, q
     p←raiz1
    Mientras (p ≠ nulo) Hacer
        Leer (p, info)
        ingresarLista(raiz3, info)
        p← sgte(p)
    Fin_mientras
    q← raiz2
    Mientras (q  ≠  nulo) Hacer
        Leer (q, info)
        ingresarLista(raiz3, info)
        q← sgte(q)
```

```
        Fin_mientras
Fin_procedimiento
```

Implementación del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>

using namespace std;
class ENTERO
{
        int num;
        public:
        void menu()
        {
                cout<< "\nMENU DE OPCIONES\n";
                cout<< "---------------\n" ;
                cout<< "<1>. Ingresar Lista1\n";
                cout<< "<2>. Ingresar Lista2\n";
                cout<< "<3>. Concatenar Lista \n";
                cout<< "<4>. Mostrar  Lista\n";
                cout<< "<5>. Salir\n";
        }
        void ingresarEntero()
        {
                fflush(stdin);
                cout<<"\n Ingresar entero: ";cin>>num;
        }
        void mostrarEntero()
        {
                cout<<num<<setw(4);
        }
};
class LISTA
{
        ENTERO info;
        LISTA *sgte;
        public:
                void ingresarLista(LISTA *&raiz,ENTERO x)
                {
                        LISTA *q;
                        LISTA *n =new LISTA;
                        n->info=x;
                        if(raiz==NULL)
                                raiz=n;
                        else
                        {
```

```
                        q=raiz;
                        while(q->sgte!= NULL)
                              q=q->sgte;
                        q->sgte=n;
                  }
                  n->sgte=NULL;
            }
            void concatenarLista(LISTA *raiz1, LISTA *raiz2, LISTA
      *&raiz3)
            {
                  LISTA *p,*q;
                  p=raiz1;
                  while(p!=NULL)
                  {
                        ingresarLista(raiz3,p->info);
                        p=p->sgte;
                  }
                  q=raiz2;
                  while(q!=NULL)
                  {
                        ingresarLista(raiz3,q->info);
                        q=q->sgte;
                  }
            }
            void mostrarLista(LISTA *raiz)
            {
                  LISTA *p=raiz;
                  while(p!=NULL)
                  {
                        p->info.mostrarEntero();
                        p=p->sgte;
                  }
            }
};
int main()
{
      char opcion;
      ENTERO e;
      LISTA list,*raiz1=NULL,*raiz2=NULL,*raiz3=NULL;
      do
      {
            e.menu();
            cout<<"ingrese opcion : ";
            opcion =cin.get();
            switch (opcion)
            {
                  case '1':
                        e.ingresarEntero();
```

```
                        list.ingresarLista(raiz1,e);
                        break;
                case '2':
                        e.ingresarEntero();
                        list.ingresarLista(raiz2,e);
                        break;
                case '3':
                        list.concatenarLista(raiz1,raiz2,raiz3);
                        cout<<"\n Se concateno exitosamente \n";
                        break;
                case '4':
                        cout<<"\n Lista 1 \n";
                        list.mostrarLista(raiz1);
                        cout<<"\n Lista 2 \n";
                        list.mostrarLista(raiz2);
                        cout<<"\n Lista 3 \n";
                        list.mostrarLista(raiz3);
                        break;
        }
        cin.ignore();
    }
    while (opcion !='5');
    system("PAUSE");
    return 0;
}
```

Ejemplo 03

Escriba la especificación, el algoritmo y la implementación en C++, que permita encontrar la unión de dos listas. La unión de dos listas esta formado por todos los elementos que pertenecen a la Lista1 o a Lista2 o a ambos. Por ejemplo:

Entrada
- Lista 1: 0, 1, 2, 3, 4, 5
- Lista2: 5, 6, 8

Salida
- Lista 3: 0, 1, 2, 3, 4, 5, 6, 8

Solución

Especificación de los TAD's y los algoritmos

En esta parte la especificación es muy parecida al del ejemplo 02 con la diferencia que no es necesario contar con el método concatenarLista, sino con el metodo unionLista que tiene como precondición a la lista1 y a la lista2, y como postcondición, a la lista3:

```
Procedimiento union (raiz1,raiz2, raiz3)
  // Definir variables
  LISTA: p, q
  Entero x,y
  Logico: salir
  p ← raiz1
  Mientras (p ≠ nulo) Hacer
        Leer (p, info)
        ingresarLista(raiz3, info)
      p ← sgte(p)
  Fin_mientras
  q ←raiz2
  Mientras ( q  ≠ nulo) Hacer
    salir ← false
    p ←raiz1
    Mientras ( p  ≠ nulo y no salir) Hacer
      Leer (q, info)
      x ← info.retornarEntero()
      Leer (p, info)
      y ← info.retornarEntero()
           Si ( x  =  y)  entonces
             salir ← true
      Fin_si
       p ← sgte(p)
    Fin_mientras
    Si (salir = false) entonces
      ingresarLista(raiz3, info)
     Fin_si
     q ← sgte(q)
   Fin_mientras
Fin_procedimiento
```

Implementación del TAD

De manera similar a la especificación, en el programa del ejemplo 02 reemplace la función concatenarLista por la función unionLista que se observa abajo:

```
void unionLista(LISTA *raiz1, LISTA *raiz2,LISTA *&raiz3)
{
        LISTA *q,*p;
        int x, y;
        bool salir;
        p=raiz1;
        while( p!=NULL)
        {
                ingresarLista(raiz3,p->info);
            p=p->sgte;
```

```
		}
		q=raiz2;
		while( q!=NULL)
		{
			salir=false;
		p=raiz1;
		while(p!=NULL && !salir)
		{
			x = q->info.retornarEntero();
			y = p->info.retornarEntero();
			if ( x == y)
				salir=true;
				p=p->sgte;
		}
			if (salir==false)
			ingresarLista(raiz3,q->info);
		q=q->sgte;
		}
	}
```

luego modificar el código del main para que quede como se observa a continuación:

```
int main()
{
	char opcion;
	ENTERO e;
	LISTA list,*raiz1=NULL,*raiz2=NULL,*raiz3=NULL;
	do
	{
		e.menu();
		cout<<"ingrese opcion : ";
		opcion=cin.get();
		switch (opcion)
		{
			case '1':
				e.ingresarEntero();
				list.ingresarLista(raiz1,e);
				break;
			case '2':
				e.ingresarEntero();
				list.ingresarLista(raiz2,e);
				break;
			case '3':
				list.unionLista(raiz1,raiz2,raiz3);
				cout<<"\n Se unio exitosamente \n";
				break;
			case '4':
```

```
                        cout<<"\n Lista 1 \n";
                        list.mostrarLista(raiz1);
                        cout<<"\n Lista 2 \n";
                        list.mostrarLista(raiz2);
                        cout<<"\n Lista 3 \n";
                        list.mostrarLista(raiz3);
                        break;
                }
                cin.ignore();
        }
        while (opcion !='5');
        system("PAUSE");
        return 0;
}
```

1.9. Ejercicios propuestos

1. Escriba la especificación, el algoritmo y el programa para rotar solo los datos impares que hay en una lista cada vez que se escoja la opción rotar impares, respetando la posicion que ocupan los impares. Por ejemplo:

   Entrada
   Lista: 3, 4, 8, 7, 1
   salida

opcion	lista
Rotar	1, 4, 8, 3, 7
Rotar	7, 4, 8, 1, 3
Rotar	3, 4, 8, 7, 1
:	

2. Escriba la especificación, el algoritmo y el programa eliminar de una lista solo los valores enteros consecutivos ascendentes. Si en caso después de una eliminación quedan valores consecutivos se prosigue con la eliminación. Por ejemplo:

   Entrada
   Lista : 5, 9, 2, 3, 10, 1, 9, 8, 7, 1,2, 3,4, 10,9,8
   Salida
   Lista : 5, 1, 9, 8, 7, 10,9,8

3. Escriba la especificación, el algoritmo y el programa para ordenar en forma ascendente solo los valores pares de una lista, respetando la posicion que ocupan los pares en la lista.

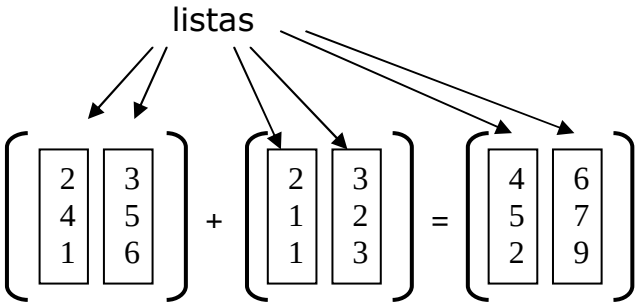
Entrada

Lista: 3, 12, 8, 7, 1, 10

Salida

Lista: 3, 8, 10, 7, 1, 12

4. Escriba la especificación, el algoritmo y el programa para construir matrices usando listas para cada columna. Luego efectue la operación suma de matrices. El resultado debe ser una matriz con las mismas características. Por ejemplo



Lección 6

Otras Estructuras Dinámicas

El capitulo Otras Estructuras Dinámicas es una extensión del capítulo de Listas Lineales Simples en el cual se expondrán estructuras que utilizan la asignación dinámica de memoria con enlace de los elementos o nodos a través de punteros, la misma técnica usada por las listas lineales simples.

6.1. Lista lineal doblemente enlazada o ligada

"Una lista doblemente ligada es una colección de nodos, en la cual cada nodo tiene dos punteros, uno de ellos apuntando a su predecesor y otro a su sucesor." (Cairó, 2002, p. 156). Puede verse la estructura de este tipo de lista a continuación:

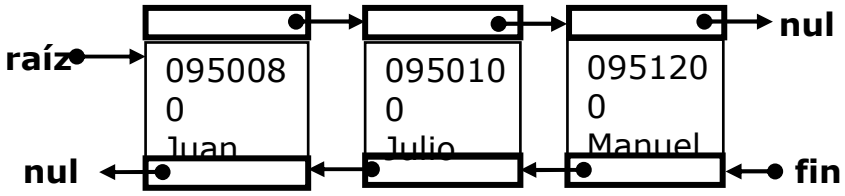


Figura 6.1. Lista lineal doblemente enlazada

6.2. Lista circular

"Las listas circulares tienen la característica de que el último elemento de la misma apunta al primero." (Cairó, 2002, p. 155). Las listas pueden estar estructuradas de dos maneras:

- Listas circulares lineales simples

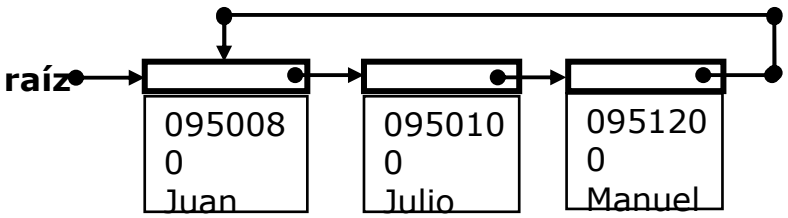


Figura 6.2. Lista circular lineal simple

- Listas circulares lineales dobles

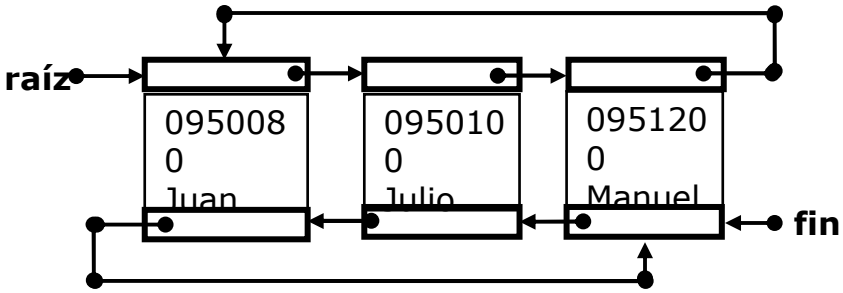


Figura 6.3. Lista circular lineal doble

6.3. Operaciones básicas con listas doblemente enlazadas:

Las operaciones que se aplican tanto en una lista doblemente enlazada como en una lista circular doblemente enlazada o no, son las mismas que se aplican para una lista simple, con la diferencia que el procedimiento para restablecer el enlace de los nodos por ejemplo después de una operación de inserción o eliminación son distintas. Entre las operaciones más comunes tenemos:

- Añadir o insertar nodos.
- Buscar o localizar nodos.
- Borrar nodos.
- Visualizar el contenido de la lista.

6.4. Ventajas y desventajas de las listas dobles y circulares

Al igual que las listas lineales enlazadas simples, las listas doblemente enlazadas circulares o no, son más adecuadas para trabajar mejor en sistemas en las que las operaciones de inserción y eliminación son frecuentes, sin embargo el uso del doble enlace incrementa sustancialmente la velocidad de las operaciones.

Ventajas

- Como el recorrido puede ser en dos sentidos, podemos recorrer una lista doblemente enlazada desde el inicio hasta el final o viceversa, además desde un nodo específico se pueden conocer los nodos que están antes y después del nodo dado de manera rápida.
- Otra ventaja en las listas doblemente enlazadas circulares o no, es en las operaciones de inserción o eliminación, al evitar usar punteros temporales para efectuar las operaciones. Solo basta trabajar con los punteros anterior y siguiente para lograr la reconexion.
- Como ventaja las listas circulares tienen la facilidad de poder recorrer la estructura desde cualquier nodo.

- Una lista circular doblemente enlazada por su estructura de anillo permite un acceso rápido al último nodo de la lista desde la raíz, como también del último nodo al nodo raíz de la lista.

Desventaja

- Una desventaja tal vez frente a una lista enlazada simple sea el que en las listas doblemente enlazadas circulares o no, cada nodo tiene la carga adicional de un puntero más y que por ende ocupara más espacio en cada elemento de la lista.
- Una desventaja que podría darse en una lista circular doblemente enlazada o no, es que si la lista no está debidamente estructurada puede dar lugar a búsquedas que nunca terminen.

6.5. Primitivas

Para el desarrollo de los algoritmos en el uso de las listas doblemente enlazadas o circulares se hace uso de la primitiva anterior(ante(t)) que se adiciona a las primitivas ya expuestas en el capítulo de listas lineales simples:

primitiva	Lista Dinámica: para el nodo T y un valor val
Inicio	Crear(T)
Leer	Leer(T, val)
Avanzar	Sgte(T)
Anterior	Ante(T)
Verificar final	T≠ nulo
escribir	Asignar(T, val)
terminar	Liberar(T)

Tabla 6.1. Primitivas de la Lista dinámica

6.6. Construcción de una lista doblemente enlazada y una lista circular

Se desea construir una lista doblemente enlazada y una lista circular simple, donde cada lista registrara alumnos de una facultad. En cada una de las estructuras cada nodo tendrá como información el código, nombre y edad de un alumno como se observa en la figura de abajo para una lista doblemente enlazada:

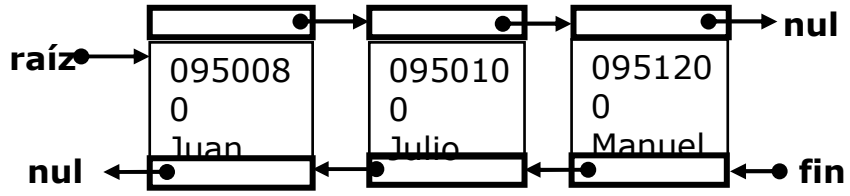


Figura 6.4. Lista doblemente enlazada

Para construir ambas estructuras se requieren dos pasos para cada una:

- La especificación del TAD y los algoritmos
- La implementación en C++

Especificación del TAD y los algoritmos

Especificación ALUMNO
variable
codigo : cadena.
nombre : cadena.
edad : entero.
métodos
ingresarDatos() : no retorna valor.
mostrarDatos() : no retorna valor.
significado
ingresarDatos lee los datos de un alumno.
mostrarDatos visualiza los datos de un alumno.
fin_especificacion

Especificacion LISTA
usa ALUMNO.
variable
a : ALUMNO.
sgte, ante : LISTA.
métodos
ingresarLista(raiz, fin, x) :no retorna valor.
mostrarLista(raiz) :no retorna valor.
significado
ingresarLista registra en la lista doble con inicio en raiz y final fin del tipo LISTA, un alumno x del tipo ALUMNO**.**
mostrarLista muestra todos los alumnos de la lista con inicio raiz del tipo LISTA.
Fin_especificacion

```
Procedimiento ingresarLista(raiz, fin, x)
    // Definir variables
    crear(n)
    asignar(n, x)
    Si(raiz = nulo) entonces
        raiz ←n
        fin  ←n
        sgte(n) ← nulo
        ante(n) ← nulo
    Sino
        sgte( n ) ←raiz
```

```
                ante(raiz) ← n
                ante(n) ← nulo
                raiz ←n
        Fin_si
Fin_procedimiento

Procedimiento mostrarLista(raiz)
        // Definir variables
        q ←raiz
        Mientras (q  ≠ nulo) Hacer
                leer(q, a)
                a.mostrarDatos()
                q ←sgte(q )
        Fin_mientras
Fin_procedimiento
```

Implementación del TAD

```
#include <iostream>
#include <cstdlib>
using namespace std;
class ALUMNO
{
        char codigo[10];
        char nombre[40];
        int edad;
        public :
        void menu()
        {
                cout<< "\nMENU DE OPCIONES\n";
                cout<< "----------------\n" ;
                cout<<"<1> Ingresar Lista\n";
                cout<<"<2> Mostrar Lista\n";
                cout<<"<3> Salir \n";
        }

        ALUMNO(){  strcpy(codigo,"");strcpy(nombre,"");int  edad=0;
}
        void ingresarDatos()
        {
                fflush(stdin);
                cout<<"ingrese codigo : ";
                gets(codigo);fflush(stdin);
                cout<<"ingrese nombre : ";
                gets(nombre ); fflush(stdin);
                cout<<"ingrese edad : ";
                cin>>edad;
```

```
        }

        void mostrarDatos()
        {
                cout<<"\n    "<<codigo<<"            "<<nombre<<"
"<<edad;
        }

};
class LISTA{
        ALUMNO a;
        LISTA *sgte,*ante;
        public:
        void ingresarLista(LISTA *&raiz, LISTA *&fin,ALUMNO x)
        {
                LISTA *n=new LISTA;
                n->a=x;
                if(raiz==NULL)
                {
                        raiz=fin=n; n->sgte=NULL; n->ante=NULL;
                 }
                else
                {
                        n->sgte=raiz;
                        raiz->ante=n;
                        n->ante=NULL;
                        raiz=n;
                }
        }
        void mostrarLista(LISTA *raiz)
        {
                LISTA *ptr;
                ptr= raiz;
                while( ptr!= NULL)
                {
                        ptr->a.mostrarDatos();
                        ptr=ptr->sgte;
                }
                cout<<endl;
        }
};
int main(){
        char opcion;
        ALUMNO x;
        LISTA list, *raiz=NULL,*fin=NULL;
        do
        {
                x.menu();
```

```
                cout<<"\ningrese opcion : ";
                opcion=cin.get();
                switch(opcion)
                {
                        case '1':
                                x.ingresarDatos();
                                list.ingresarLista(raiz,fin,x);
                                break;
                        case '2':
                                list.mostrarLista(raiz);
                                break;
                }
                cin.ignore();
        }
        while( opcion !='3');
        system("pause");
        return 0;
}
```

Ahora observe en el gráfico de abajo a la lista enlazada circular simple de la cual se tiene a continuación su especificación y su implementación:

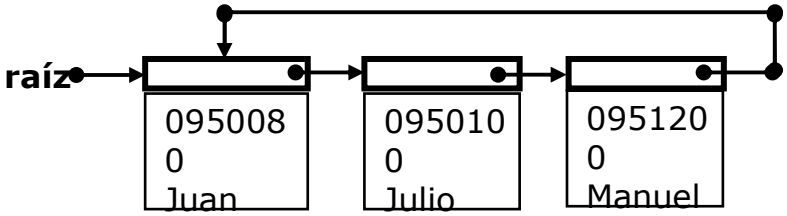


Figura 6.5. Lista enlazada circular simple

Especificación del TAD y los algoritmos

Especificación ALUMNO
variable
codigo : cadena.
nombre : cadena.
edad : entero.
métodos
ingresarDatos() : no retorna valor.
mostrarDatos() : no retorna valor.
significado
ingresarDatos lee los datos de un alumno.
mostrarDatos visualiza los datos de un alumno.
fin_especificacion

Especificacion LISTA
usa ALUMNO
variable
a : ALUMNO.
sgte : LISTA.
métodos
ingresarLista(raiz, x) :no retorna valor.
mostrarLista(raiz) :no retorna valor.
menú() :no retorna valor.
significado
ingresarLista registra en la lista circular con cabecera raíz del tipo LISTA, un alumno x del tipo ALUMNO.
mostrarLista muestra todos los alumnos de la lista con inicio raíz del tipo LISTA.
menú muestra las alternativas del menú.
Fin_especificacion

```
Procedimiento ingresarLista(raiz, x)
    // Definir variables
    crear(n)
    Asignar(n,x)
    Si (raiz = nulo) entonces
        raiz ← n
    Sino
        q ← raiz
        Mientras (sgte(q) ≠ raiz) Hacer
            q ← sgte (q)
        Fin_mientras
        sgte (q) ← n
    Fin_si
    sgte(n) ← raiz
Fin_procedimiento
```

Implementación del TAD

```
#include <iostream>
#include <cstdlib>
using namespace std;
class ALUMNO
{
    char codigo[10];
    char nombre[40];
    int edad;
    public :
    ALUMNO(){ strcpy(codigo,"");strcpy(nombre,"");int nota=0; }
    void ingresarDatos()
```

```
	{
		cout<<"ingrese codigo : "; fflush(stdin);
		gets(codigo);fflush(stdin);
		cout<<"ingrese nombre : ";
		gets(nombre ); fflush(stdin);
		cout<<"ingrese edad : ";
		cin>>edad;
	}
	void mostrarDatos()
	{
		cout<<"\n "<<codigo<<"    "<<nombre<<"  "<<edad;
	}

};
class LISTA
{
	ALUMNO a;
	LISTA *sgte;
	public:
	 void menu()
	{
		cout<< "\nMENU DE OPCIONES\n";
		cout<< "----------------\n" ;
		cout<<"<1> Ingresar Lista\n";
		cout<<"<2> Mostrar Lista\n";
		cout<<"<3> Salir \n";
	}
	 void ingresarLista(LISTA *&raiz,ALUMNO x)
	{
		LISTA *q;
		LISTA *n=new LISTA;
		n->a=x;
		if(raiz==NULL)
			raiz=n;
		else
		{
			q=raiz;
			while(q->sgte!= raiz)
				q=q->sgte;
			q->sgte=n;
		}
		n->sgte=raiz;
	}
	void mostrarLista(LISTA *raiz)
	{
		LISTA *q=raiz; bool salir=false;
		if (raiz!=NULL)
		 {
```

```
                while(!salir)
                {
                        q->a.mostrarDatos();
                        q=q->sgte;
                        if(q==raiz) salir=true;
                }
        }
    }
};
int main()
{
    char opcion;
    ALUMNO x;
    LISTA list, *raiz=NULL;
    do
    {
        list.menu();
        cout<<"\ningrese opcion : ";
        opcion=cin.get();
        switch(opcion)
        {
        case '1':
            x.ingresarDatos();
            list.ingresarLista(raiz,x);
            break;
        case '2':
            list.mostrarLista(raiz);
            break;
        }
        cin.ignore();
    }
    while( opcion !='3');
    system("pause");
    return 0;
}
```

6.7. Ejercicios resueltos.

Ejercicio 01

Escriba la especificación, el algoritmo y el programa para una lista lineal doblemente enlazada en la que se quiere realizar la operación de eliminar un número entero según su ubicación en la lista. Hay que observar que cuando se elimina un elemento de una lista doble pueden darse los siguientes casos:

Solucion:

Especificación de los TAD's y los algoritmos

Especificación LISTA
variable
num : entero.
sgte, ante : LISTA.
métodos
ingresarLista(raiz,fin, x) : no retorna valor.
eliminarPorPosicion(raiz,fin, posicion) : retorna valor lógico.
mostrarLista(raiz) : no retorna valor.
significado
ingresaLista registra en la lista doble un numero *x*.
eliminarPorPosicion tiene como precondición a la lista y a su posición, y como postcondición, a la lista.
mostrarLista muestra los datos de la lista.
Fin_especificación

```
Funcion eliminarPorPosicion(raiz, fin, posicion):logico
    // Definir variables
    LISTA  : q ← raiz
    logico: enconto ← false
    entero:  i ←0
    Mientras ( q ≠ NULL y no encontro) Hacer
       Si (posicion = i) entonces
        Si (raiz=fin) entonces
                raiz ← nulo
                  fin←nulo
            sino
                Si (raíz=q) entonces
                    raíz← sgte(q)
                   ante(raiz)←nulo
                    sino
                      si(fin=q) entonces
                            fin←ante(q)
                            sgte(fin) ←nulo
                         sino
                            sgte(ante(q))←sgte(q)
                            ante(sgte(q))←ante(q)
                      Fin_si
                  Fin_si
             Fin_si
             liberar(q)
          encontro ←true
        Sino
            q ← sgte (q)
```

```
                i←i+1
            Fin_si
        Fin_mientras
        retornar  encontro
Fin_funcion
```

Implementación del TAD

```
#include <iostream>
#include <cstdlib>
using namespace std;
class LISTA
{
        int num;
        LISTA *sgte,*ante;
        public:
          void menu()
        {
                cout<< "\nMENU DE OPCIONES\n";
                cout<< "---------------\n" ;
                cout<<"<1> Ingresar Lista\n";
                cout<<"<2> Eliminar por posicion \n";
                cout<<"<3> Mostrar Lista\n";
                cout<<"<4> Salir \n";
        }
        void ingresarLista(LISTA *&raiz, LISTA *&fin,int x)
        {
                LISTA *n=new LISTA;
                n->num=x;
                if(raiz==NULL)
                {
                        raiz=fin=n; n->sgte=NULL; n->ante=NULL;
                 }
                else
                {
                        n->sgte=raiz;
                        raiz->ante=n;
                        n->ante=NULL;
                        raiz=n;
                }
        }
         bool     eliminarPorPosicion(LISTA     *&raiz,LISTA     *&fin,int
posicion)
        {
                LISTA *q=raiz;
                bool encontro =false;
                int i=0;
                while((q != NULL) && !encontro)
```

```
            {
                  if(posicion == i)
                  {
                        if(raiz==fin){
                                    raiz=fin=NULL;
                              }
                              else if(raiz==q){
                                          raiz=q->sgte;
                                          raiz->ante=NULL;
                                    }
                                    else if(fin==q){
                                                fin=q->ante;
                                                fin->sgte=NULL;
                                     }
                                    else{
                                        (q->ante)->sgte=q->sgte;
                                        (q->sgte)->ante=q->ante;
                                    }
                        delete q;
                        encontro =true;
                  }
                  else
                  {
                        q=q->sgte;
                        i++;
                  }
            }
             return encontro;
      }
      void mostrarLista(LISTA *raiz)
      {
            LISTA *ptr;
            ptr= raiz;
            while( ptr!= NULL)
            {
                  cout<<"   "<<ptr->num;
                  ptr=ptr->sgte;
            }
            cout<<endl;
      }
};
int main()
{
      char opcion;
      int posicion;
      int x;
      LISTA list, *raiz=NULL,*fin=NULL;
      do
```

```
        {
                list.menu();
                cout<<"\ningrese opcion : ";
                opcion=cin.get();
                switch(opcion)
                {
                case '1':
                        cout<<"\n ingrese un numero entero: ";cin>>x;
                        list.ingresarLista(raiz,fin,x);
                        break;
                case '2':
                        cout<<"\n Ingrese posicion de dato a eliminar: ";
                             cin>>posicion;
                        if(list.eliminarPorPosicion(raiz,fin,posicion))
                                cout<<"\n Se elimino exitosamente";
                        else
                                cout<<"\n no se elimino ";
                        break;
                case '3':
                        list.mostrarLista(raiz);
                        break;
                }
                cin.ignore();
        }
        while( opcion !='4');
        system("pause");
        return 0;
}
```

Ejemplo 02

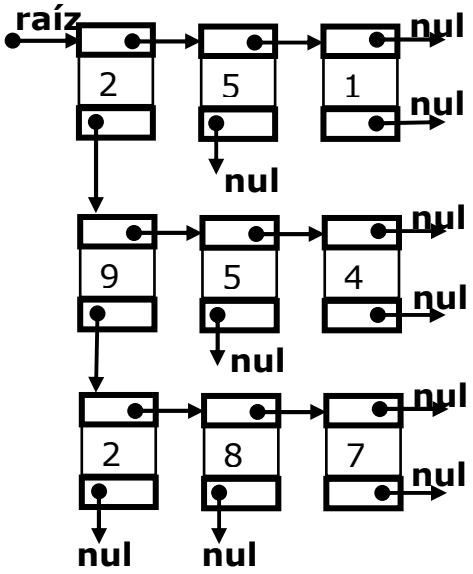
Escribir la especificación, el algoritmo y el programa para crear la estructura que pueda registrar los valores enteros de una matriz.

Solución:

Por ejemplo si la matriz que se quiere ingresar es la siguiente:

$$\begin{bmatrix} 7 & 8 & 2 \\ 4 & 5 & 9 \\ 1 & 5 & 2 \end{bmatrix}$$

La estructura que guardara estos valores se formará como se observa abajo:



Especificación del TAD y los algoritmos

Especificacion MATRIZ
variable
num : entero.
sgte,abaj : LISTA.
métodos
ingresarMatriz(raiz, x, y) :no retorna valor.
mostrarMatriz(raiz) :no retorna valor.
menu() :no retorna valor.
significado
ingresarMatriz registra en la estructura una matriz.
mostrarMatriz muestra todos los valores la matriz.
menu muestra las alternativas del menú.
Fin_especificación

```
Procedimiento ingresarMatriz(raiz,   x,  y)
      MATRIZ :q, r, n
      Desde i ← 0 Hasta i< y con incremento 1 Hacer
            r←nulo
            Desde j ← 0 Hasta j< x con incremento 1 Hacer
                  crear(n)
                  leer  num
                  asignar(n,num)
                  Si(r=nulo) entonces
                        r←n
                        sgte(r)←nulo
                  Sino
                        sgte(n)←r
                        r←n
                  Fin_si
```

```
            Fin_desde
            Si(raiz=nulo) entonces
                  raiz←r
                  abaj(raiz)←nulo
            Sino
                  abaj( r)←raiz
                  raiz←r
            Fin_si
      Fin_desde
Fin_procedimiento
```

Implementación del TAD

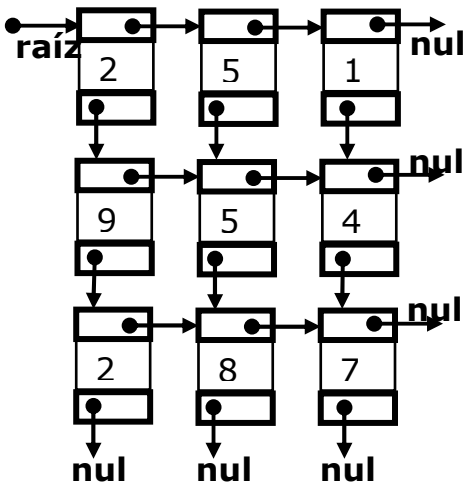
```
#include <iostream>
#include <cstdlib>
using namespace std;
class MATRIZ{
      int num;
      MATRIZ *sgte,*abaj;
      public:
      void menu()
      {
            cout<< "\nMENU DE OPCIONES\n";
            cout<< "----------------\n" ;
            cout<<"<1> Ingresar Matriz\n";
            cout<<"<2> Mostrar Matriz\n";
            cout<<"<3> Salir \n";
        }

      void ingresarMatriz(MATRIZ *&raiz,int x,int y){
            MATRIZ *q,*r,*n;
            for(int i=0; i<y; i++){
                  r=NULL;
                  cout<<"\n";
                  for(int j=0; j<x; j++){
                        MATRIZ *n=new MATRIZ;
                        cout<<"  ingrese numero entero: ";
                        cin>>n->num;
                        if(r==NULL){ r=n; r->sgte=NULL; }
                        else{ n->sgte=r; r=n; }
                  }
                  if(raiz==NULL){ raiz=r; raiz->abaj=NULL; }
                  else{ r->abaj=raiz; raiz=r; }
            }
      }
      void mostrarMatriz(MATRIZ *raiz){
            MATRIZ *p,*q=raiz;
            while (q!=NULL){
```

```
                    p=q;
                    cout<<"\n";
                    while (p!=NULL){
                        cout<<"  "<<p->num;
                        p=p->sgte;
                    }
                    q=q->abaj;
                }
        }
 };
int main(int argc, char *argv[]){
        char opcion;
        MATRIZ ma, *raiz=NULL;
        do
        {
                ma.menu();
                cout<<"\ningrese opcion : ";
                opcion=cin.get();
                switch(opcion){
                        case '1':
                                ma.ingresarMatriz(raiz, 3,3);
                                break;
                        case '2':
                        ma.mostrarMatriz(raiz);break;
                }
                cin.ignore();
        }
        while( opcion !='3');
        system("pause");
        return 0;
}
```

Ejemplo 03

Escriba la especificación, el algoritmo y el programa para crear una matriz con la estructura siguiente que pueda registrar una matriz con valores enteros:



Solución:

Especificación del TAD y los algoritmos

Especificacion MATRIZ
variable
num : entero.
sgte,abaj : LISTA.
métodos
MATRIZ() :inicializa los enlaces.
ingresarMatriz(raiz, nn) :no retorna valor.
mostrarMatriz(raíz, nn) :no retorna valor.
menu() :no retorna valor.
nodo_dev(raíz, a, b) : retorna un tipo MATRIZ
significado
MATRIZ inicializa los apuntadores sgte y abaj a nulo.
ingresarMatriz registra en la lista una matriz.
mostrarMatriz muestra todos los valores la matriz.
menu muestra las alternativas del menú.
nodo_dev retorna un apuntador del tipo MATRIZ.
Fin_especificación

```
Procedimiento ingresarMatriz(raiz,  nn)
        MATRIZ : n,q,aux←nulo
        entero :  i, j
        Desde i ← 0 Hasta i< nn con incremento 1 Hacer
                cout<<"\n";
                Desde j ← 0 Hasta j< nn con incremento 1 Hacer
                        crear(n)
                        leer num
                        asignar(n, num)
```

```
                    Si(raiz=nulo) entonces
                        raiz←n
                    Sino
                        Si(i=0) entonces
                            q←raiz
                            Mientras(sgte(q)≠nulo) hacer
                                q←sgte(q)
                            fin_mientras
                            sgte(q)←n
                        sino
                            Si(j ≠0) entonces
                                sgte(aux)←n
                            Fin_si
                            abaj(nodo_dev(raiz,i-1,j))←n
                            aux←n
                        Fin_si
                    Fin_si
                    abaj(n)←nulo
                    sgte(n)←nulo
                Fin_desde
        Fin_desde
Fin_procedimiento

Funcion  nodo_dev(raiz,  a,  b): MATRIZ
        MATRIZ: q
        entero:  i, j
        crear(q)
        q←raiz
        Desde i ← 0 Hasta i< a con incremento 1 Hacer
                q←abaj(q )
        Fin_desde
        Desde j ← 0 Hasta j< b con incremento 1 Hacer
                q←sgte(q )
        Fin_desde
        retornar q
Fin_funcion
```

Implementación del TAD

```
#include <iostream>
#include <cstdlib>
using namespace std;

class MATRIZ{
        int num;
        MATRIZ *abaj;
        MATRIZ *sgte;
        public:
```

```
MATRIZ(){abaj=sgte=NULL;}
void menu()
{
	cout<< "\nMENU DE OPCIONES\n";
	cout<< "---------------\n" ;
	cout<<"<1> Ingresar Matriz\n";
	cout<<"<2> Mostrar Matriz\n";
	cout<<"<3> Salir \n";
}
void ingresarMatriz(MATRIZ *&raiz,int nn){
MATRIZ *n,*q,*aux=NULL;
	int i,j;
	for(i=0;i<nn;i++)
	{
		cout<<"\n";
		for(j=0;j<nn;j++)
		{
			n=new MATRIZ;
			cout<<" ingrese un numero:  ";
			cin>>n->num;
			if(raiz==NULL)
				raiz=n;
			else
			{
				if(i==0)
				{
					q=raiz;
					while(q->sgte!=NULL)
						q=q->sgte;
					q->sgte=n;
				}
				else
				{
					if(j!=0)
						aux->sgte=n;
					nodo_dev(raiz,i-1,j)->abaj=n;
					aux=n;
				}
			}
			n->abaj=NULL;
			n->sgte=NULL;
		}
	}
}
void mostrarMatriz(MATRIZ *raiz, int nn){
	 int p,q;
	for(p=0;p<nn;p++)
	{
```

```
                cout<<"\n";
                for(q=0;q<nn;q++)
                {
                        cout<<"  "<<nodo_dev(raiz,p,q)->num;
                }
                cout<<"\n";
        }
    }
    MATRIZ *nodo_dev(MATRIZ *raiz,int a, int b){
        MATRIZ *q=new MATRIZ;
        q=raiz;
        int i,j;
        for(i=0;i<a;i++)
            q=q->abaj;
        for(j=0;j<b;j++)
            q=q->sgte;
        return q;
        }
 };
int main()
{
    char opcion;
    MATRIZ ma, *raiz=NULL;
    do
    {
        ma.menu();
        cout<<"\ningrese opcion : ";
        opcion=cin.get();
        switch(opcion){
            case '1':
                ma.ingresarMatriz(raiz, 3);
                break;
            case '2':
                ma.mostrarMatriz(raiz,3);break;
        }
        cin.ignore();
    }
    while( opcion !='3');
    system("pause");
    return 0;
}
```

6.8. Ejercicios propuestos.

1. Escriba la especificación, el algoritmo y el programa para una lista doblemente enlazada de números enteros en la cual se quiere crear un método para destruir todos los nodos de la lista de manera automática.

2. Escriba la especificación, el algoritmo y el programa para una lista doblemente enlazada circular de personas (dni y nombre) en la cual se quiere crear un método para destruir todos los nodos de la lista de manera automática.

3. Escriba la especificación, el algoritmo y el programa para una lista doblemente enlazada de números enteros en la cual se quiere crear un método para insertar un elemento por valor . Por ejemplo:

   Entrada
   Lista : 2, 4, 6, 1, 9
   Elemento a insertar : 20
   Elemento donde se insertará : 6
   Salida
   Lista : 2, 4, 20, 6, 1, 9

4. Escriba la especificación, el algoritmo y el programa para una lista doblemente enlazada circular de números enteros en la cual se quiere crear un método para eliminar un elemento por valor . Por ejemplo:

   Entrada
   Lista : 2, 4, 6, 1, 9
   Elemento a eliminar : 6
   Salida
   Lista : 2, 4, 1, 9

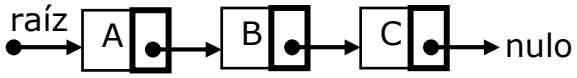
5. Escriba la especificación, el algoritmo y el programa para una doblemente enlazada de números enteros en la cual se quiere eliminar todos los elementos que coincidan con los elementos de una lista enlazada circular simple. Por ejemplo:

   Entrada
   Lista doble : 2, 6, 8, 9, 1, 3
   Lista circular : 7, 2, 3, 34
   Salida
   Lista doble : 6, 8, 9, 1

Resumen

Una lista es un TAD lineal, con elementos del mismo tipo que guardan un orden y en la que las restricciones de acceso no existen, la inserción y eliminación de elementos puede hacerse en cualquier ubicación. Ademas es una estructura dinámica que se construye en

memoria primaria. Cada elemento de una lista se denomina nodo, donde cada nodo tiene un espacio para la información que almacena y un espacio para los enlaces o apuntadores. En el grafico siguiente se observa una lista simple, con información de una letra en cada nodo y un apuntador al siguiente elemento (flecha), note además que la lista posee un apuntador que señala el inicio(raiz) y una dirección vacia(nulo) que marca el final de la lista:



En una lista se dan operaciones de inserción, eliminación o búsqueda de un nodo que son aplicables a los diferentes tipos de listas como listas simples(cada nodo tiene un enlace), listas doblemente enlazadas(cada nodo tiene dos enlaces) o listas circulares(el ultimo nodo se enlaza con el primer nodo de la lista). Una lista puede implementarse mediante arreglos o mediante el uso de gestión dinámica de memoria y apuntadores.

Como todo TAD tiene un conjunto de primitivas para describir las operaciones que se aplican sobre este tipo de estructura a través de algoritmos. Las primitivas definen la creación de un nodo, la lectura de un nodo, liberación de memoria, etc. Por otro lado la utilidad de la gestión de memoria dinámica y enlace no solo es para estructuras lineales sino también para estructuras que no guardan una forma lineal y que nos servirán para construir estructuras de forma matricial o cilíndrica que se podrán usar para diversos fines.

Lectura

Asignacion Dinámica de Memoria

Además de prevenir problemas innecesarias de desbordamiento, el uso de apuntadores tiene grandes ventajas en un ambiente de programación múltiple (tiempo compartido). Si utilizamos arreglos para reservar con anticipación la máxima cantidad de memoria que el programa deberá necesitar, esta memoria se nos asigna y ya no estará disponible para otros usuarios. Si es necesario paginar nuestro trabajo fuera de la memoria, puede perderse memoria a medida que la memoria sin usar se copia en un disco y a partir de él. En vez de emplear arreglos para retener todos nuestro elementos, podemos comenzar con muy poco: con espacio solo para las instrucciones del programa y variables simples, y donde quiera que necesitemos espacio para un elemento adicional podemos solicitar al sistema la memoria necesaria.

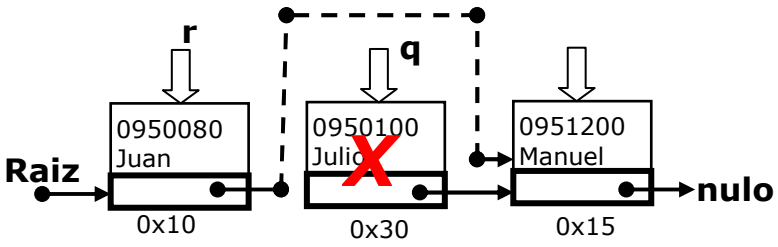
En forma similar, cuando un elemento ya no es necesario, su espacio puede recuperarse para el sistema y este puede entonces asignarlo a otro usuario. En esta forma un programa puede arrancar con poco espacio y crecer solo cuando sea preciso, de tal modo que cuando es pequeño corre más eficazmente y cuando es necesario puede crecer hasta alcanzar los límites del sistema de cómputo. Al proceso que asigna y reasigna la memoria en esta forma se le denomina asignación dinámica de memoria.

Aun con un solo usuario, este control dinámico de la memoria puede resultar útil. Durante una parte de una tarea puede necesitarse una gran cantidad de memoria para algún propósito. Se la puede liberar mas tarde y luego asignarla de nuevo para otro propósito, quizás ahora con datos de un tipo completamente diferente a los anteriores.

Robert L. Kruse (1988) *Estructura de Datos y Diseño de Programas*. México D.F. Prentice-Hall Hispanoamericana S.A. p. 57

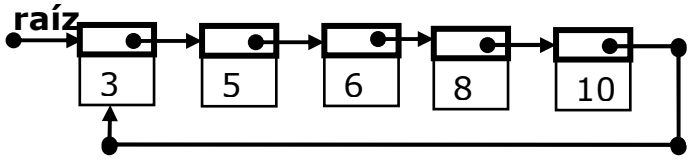
Autoevaluación

1. Al aplicar la operación eliminar en una lista dinámica simple puede suceder el caso mostrado abajo en el grafico. Señale usted cual es valor del apuntador sgte de r y el valor del apuntador q después de la eliminacion



a) 0x30 y nulo b) 0x10 y 0x30 c) 0x15 y nulo d) ninguna

2. En una lista circular como la que se observa abajo, el puntero raíz avanza 3 veces, de dos en dos nodos. Se pide visualizar toda la lista en cada uno movimiento desde la raiz



a) 6, 8, 10, 3, 5
10, 3, 5, 6,

b) 3, 5, 6, 8, 10
6, 8, 10, 3,

c) 6, 8, 10, 3, 5
8, 10, 3, 5,

d) ninguna

3. Se quiere destruir una matriz. Para ello se requiere el siguiente método en el cual tiene que colocar las instrucciones en la secuencia correcta que pueden estar en alguna de las alternativas

```
procedimiento destruirMatriz(raiz)
    MATRIZ : z, p, q
    mientras  (raiz ≠ nulo) Hacer
        [            ]
        [            ]
        mientras (q  ≠ nulo) Hacer
            [            ]
            [            ]
            [            ]
        Fin_mientras
        [            ]
    Fin_mientras
Fin_procedimiento
```

a) q←raiz
p← sgte(q)
z← abaj(q)
liberar(q)
raiz←z
q←p

b) q←raiz
z← abaj(q)
p← sgte(q)
liberar(q)
q←p
raiz←z

c) q←raiz
z← abaj(q)
p← sgte(q)
p← abaj(p)
q←p
raiz←z

d) ninguna

4. En una lista dinámica doblemente enlazada se ha creado un método que cada vez que se invoca suma el valor de todos los nodos y con la suma crea un nuevo nodo que se adiciona a la lista como se observa en el ejemplo:

Entrada
Lista: 2, 3, 6
Salida
Lista: 2, 3, 6, 11

El método creado es el siguiente:

```
procedimiento aumentarLista(raiz, fin)
   LISTA : ptr←raiz
   entero :  s←0
   mientras ( ptr ≠ nulo) Hacer
        {
             [                ]
             [                ]
             [                ]
   Fin_mientras
   [                 ]
Fin_procedimiento
```

Escoja la alternativa que tenga la secuencia adecuada de instrucciones para reemplazarlas en los corchetes

a) ingresarLista(raiz,fin, s)
s←s+num
leer(ptr, num)
ptr← sgte(ptr)

b) ingresarLista(raiz,fin, s)
leer(ptr, s)
num←num+s
ptr← sgte(ptr)

c) leer(ptr, num)
s←s+num
ptr← sgte(ptr)
ingresarLista(raiz,fin, s)

d) ninguno

Claves: 1:c; 2:a; 3:b; 4:c;

Enlaces

http://www.calcifer.org/documentos/librognome/glib-lists-queues.html
http://es.kioskea.net/faq/sujet-2872-listas-doblemente-enlazadas
http://www.zator.com/Cpp/E1_8a.htm
http://c.conclase.net/edd/?cap=005b
http://fcc98.tripod.com/tutores/ed1/ed1.html#CONTE

Bibliografía

Larry R. Nyhoff (2006) *Estructuras de datos y resolución de problemas con C++,* Madrid, Prentice Hall. Cap. 6: Listas, pp. 253-312.

Allen Weiss, M., (1995) *Estructura de Datos y Algoritmos*, México D.F., Addison-Wesley Iberoamericana. Cap. 4: Listas, pp. 56-87.

Cairó O., Guardati M.C. S., (2002) *Estructura de Datos*, 2ª. Ed. México D.F., Mc Graw-Hill / Interamericana Editores S.A. Cap. 5: Listas, pp. 230-278.

Brassard, G., Bratley, P., (1998) *Fundamentos de Algoritmia*, 2ª. Ed. México D.F., Prentice Hall. Cap. 7: Programación dinámica, pp. 125-169.