Algoritmos y Estructura de datos

Clase 07

Lista ordenada y Listas doblemente enlazadas

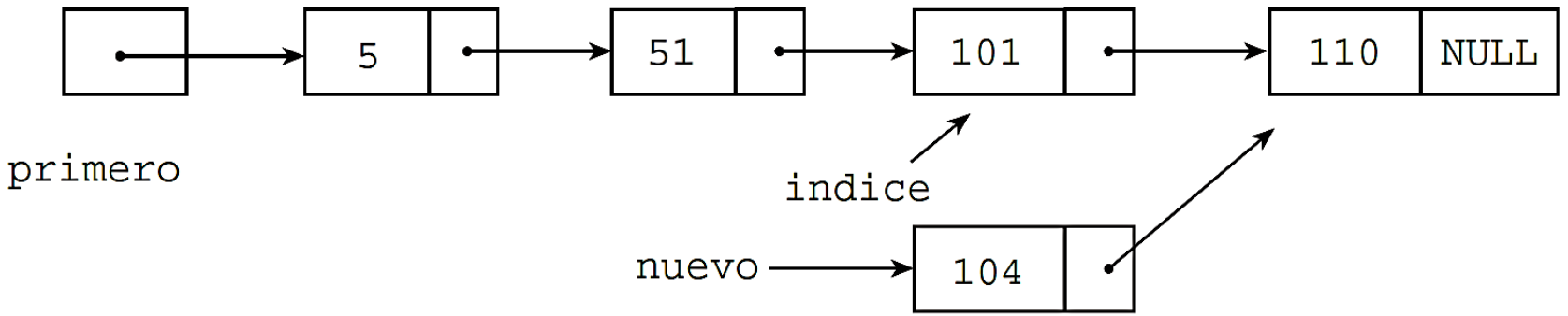
Profesor: Carlos Diaz

Contenido

- Lista simplemente enlazada ordenada ascendentemente
- Lista doblemente enlazada
- Clase NodoDoble
- Clase ListaDoble
- Método crearLista()
- Método visualizar()
- Insertar datos a la lista
- Eliminar datos de la lista
- Ejercicios

Lista ordenada

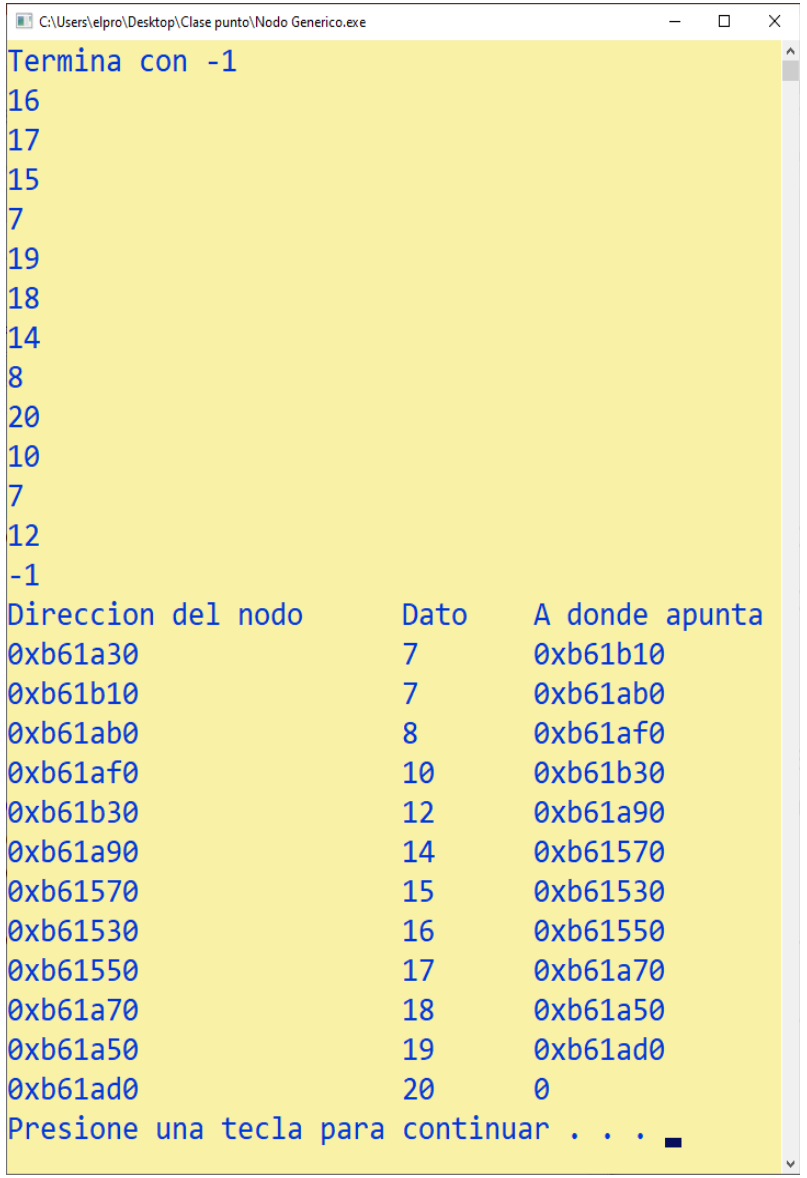
- Para ordenar los datos de la lista a medida que se van ingresando, se determina, en primer lugar, la posición de inserción y, a continuación, se ajusta los enlaces.
- Por ejemplo, en la figura, para insertar el dato 104 es necesario recorrer la lista hasta el nodo con 110, que es el nodo inmediatamente mayor.
- El puntero índice se queda con la dirección del nodo anterior, a partir del cual se realizan los enlaces con el nuevo nodo.




```
C:\Users\elpro\Desktop\Clase punto\Nodo Generico.exe
Termina con -1
16
17
15
7
19
18
14
8
20
10
7
12
-1
Direccion del nodo      Dato    A donde apunta
0xad1b30                12      0xad1b10
0xad1b10                7       0xad1af0
0xad1af0                10      0xad1ad0
0xad1ad0                20      0xad1ab0
0xad1ab0                8       0xad1a90
0xad1a90                14      0xad1a70
0xad1a70                18      0xad1a50
0xad1a50                19      0xad1a30
0xad1a30                7       0xad1570
0xad1570                15      0xad1550
0xad1550                17      0xad1530
0xad1530                16      0
Presione una tecla para continuar . . .
```

Ampliando la clase Lista

- A nuestro clase Lista debemos agregarle la función insertaOrden() que crea la lista ordenada.
- El punto de partida es una lista vacía, a la que se añaden nuevos elementos, de tal forma que en todo momento los elementos están ordenados en orden creciente.
- La inserción del primer nodo de la lista consiste, sencillamente, en crear el nodo y asignar su dirección a la cabeza de la lista.
- El segundo elemento se ha de insertar antes o después del primero, dependiendo de que sea menor o mayor.



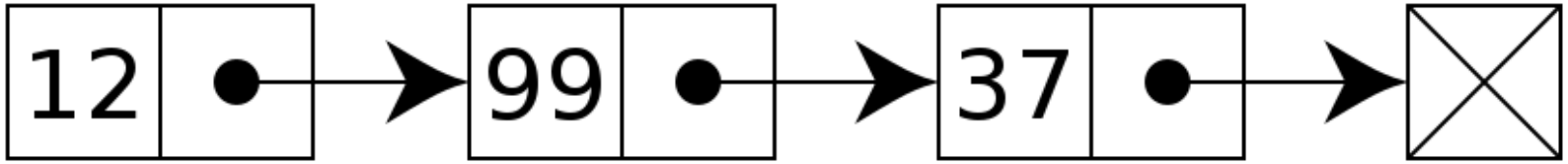
Codificación de la función insertaOrden()

```
void Lista::insertaOrden(int dato)
{
    Nodo<int>* nuevo ;
    nuevo = new Nodo<int>(dato);
    if (primero == NULL) // lista vacía
        primero = nuevo;
    else if (dato < primero -> datoNodo())
    {
        nuevo -> ponerEnlace(primero); //El dato nuevo apunta al primero
        primero = nuevo; //Y ahora el dato nuevo se convierte en primero
    }
    else // búsqueda del nodo anterior al de inserción
    {
        Nodo<int> *anterior, *p; // p es puntero que recorre
        anterior=NULL;
        p = primero;
        while ((p!= NULL) && (dato > p->datoNodo()))
        {
            anterior = p; //Guarda el nodo menor al nuevo dato
            p = p -> enlaceNodo(); //Avanza al siguiente enlace
        }
    // se procede al enlace del nuevo nodo
    nuevo -> ponerEnlace(anterior-> enlaceNodo());
    anterior -> ponerEnlace(nuevo);
    }
}
```

- Finalmente, en el método crearLista() ya no se insertará el nodo directamente.
- Así que la sentencia primero=new Nodo<int>(x,primero); la ponemos como comentario.
- Y colocamos una llamada a la función insertaOrden(x)

Lista doblemente enlazada

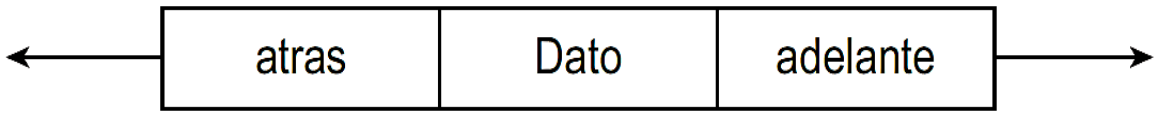
- Hasta ahora el recorrido de una lista se ha realizado en un solo sentido (hacia adelante).



- Desde un nodo de una lista doblemente enlazada se puede avanzar al siguiente, o bien retroceder al nodo anterior.
- Cada nodo de una lista doble tiene tres campos, el dato y dos punteros, uno apunta al siguiente nodo de la lista y el otro al nodo anterior.

Nodo de una lista doblemente enlazada

- La clase NodoDoble define para cada objeto nodo dos enlaces, uno adelante y uno atrás.
- Cada nodo creado apunta al inicio a NULL por ambos lados.
- El método datoNodo() devuelve el valor contenido en el nodo y el método fijarDato() asigna un dato al nodo.
- Los métodos adelanteNodo() y atrasNodo() devuelven los nodos de adelante y atrás respectivamente.
- Los métodos ponerAdelante() y ponerAtras() colocan un nodo antes o después respectivamente.




```
class NodoDoble
{
    private:
    int dato;
    NodoDoble* adelante; //A la derecha
    NodoDoble* atras; //A la izquierda
    public:
    NodoDoble(int t)
    {
        dato = t;
        adelante = atras = NULL;
    }
    int datoNodo() { return dato; }
    void fijarDato(int a){dato=a;}
    NodoDoble* adelanteNodo() { return adelante; }
    NodoDoble* atrasNodo() { return atras; }
    void ponerAdelante(NodoDoble* a) { adelante = a; }
    void ponerAtras(NodoDoble* a) { atras = a; }
};
```

Ejemplo

- Crear una lista doblemente enlazada usando tres nodos.

```
#include <iostream>
using namespace std;
int main() {
	NodoDoble* x;
	NodoDoble* y;
	NodoDoble* z;
	x=new NodoDoble(7);
	y=new NodoDoble(9);
	z=new NodoDoble(5);
	cout<<"Atras"<<"\t"<<"DirDelDato"<<"\t"<<"Dato"<<"\t"<<"Adelante"<<endl;
	cout<<x->atrasNodo()<<"\t"<<x<<"\t"<<x->datoNodo()<<"\t"<<x->adelanteNodo()<<endl;
	cout<<y->atrasNodo()<<"\t"<<y<<"\t"<<y->datoNodo()<<"\t"<<y->adelanteNodo()<<endl;
	cout<<z->atrasNodo()<<"\t"<<z<<"\t"<<z->datoNodo()<<"\t"<<z->adelanteNodo()<<endl;
	cout<<"Enlazando dos nodos x e y:\n";
	x->ponerAdelante(y);y->ponerAtras(x);
	cout<<x->atrasNodo()<<"\t\t"<<x<<"\t"<<x->datoNodo()<<"\t"<<x->adelanteNodo()<<endl;
	cout<<y->atrasNodo()<<"\t"<<y<<"\t"<<y->datoNodo()<<"\t"<<y->adelanteNodo()<<endl;
	z->ponerAdelante(x);x->ponerAtras(z);
	cout<<"Enlazando el tercer nodo z:\n";
	cout<<x->atrasNodo()<<"\t"<<x<<"\t"<<x->datoNodo()<<"\t"<<x->adelanteNodo()<<endl;
	cout<<y->atrasNodo()<<"\t"<<y<<"\t"<<y->datoNodo()<<"\t"<<y->adelanteNodo()<<endl;
	cout<<z->atrasNodo()<<"\t\t"<<z<<"\t"<<z->datoNodo()<<"\t"<<z->adelanteNodo()<<endl;
	return 0;
}
```

La clase ListaDoble

- La clase ListaDoble encapsula las operaciones básicas de las listas doblemente enlazadas (*insertar, eliminar, buscar, recorrer, etc.)*
- La clase dispone del puntero variable cabeza para acceder a la lista, apunta al primer nodo.
- El constructor de la clase inicializa la lista vacía.
- Se puede añadir nodos a la lista de distintas formas, según la posición donde se inserte. La posición de inserción puede ser:
  - En cabeza de la lista.
  - Al final de la lista.
  - Antes de un elemento especificado.
  - Después de un elemento especificado.

Codificación de la clase ListaDoble

- El constructor inicializa cabeza a NULL, (lista vacía).
- La función crearLista() construye iterativamente insertando por la cabeza.
- La función visualizar() recorre cada nodo mostrando su dirección, dato y dirección anterior y posterior a donde apunta.
- Las demás funciones básicas se declaran similarmente.

```
//La clase ListaDoble
class ListaDoble
{
    private:
    NodoDoble* cabeza;
    public:
    ListaDoble() //Constructor
    {
        cabeza = NULL;
    }
    void crearLista();
    void visualizar();
    void insertarCabezaLista(int);
    void insertaDespues(int, int);
    void eliminar(int);
    //Continua las demás funciones
};
```

Implementación de crearLista()

- La función insertará los datos por la cabeza de la lista.
- Es decir que el primer dato terminará siendo la cola y el último dato la cabeza de la lista.

```
//Crear una lista doble
#include <iostream>
using namespace std;
void ListaDoble::crearLista()
{
    int x;
    cout << "Termina con -1" << endl;
    do
    {
        cin >> x;
        if (x != -1)
        {
            insertarCabezaLista(x);
        }
    }while (x != -1);
};
```

Insertar por la cabeza

- Algoritmo:

1. Crear un nodo con el nuevo elemento.
2. Hacer que el campo adelante del nuevo nodo apunte a la cabeza (primer nodo) de la lista original, y que el campo atrás del nodo cabeza apunte al nuevo nodo.
3. Hacer que cabeza apunte al nodo creado.

```
void ListaDoble::insertarCabezaLista(int dato)
{
    NodoDoble* nuevo;
    nuevo = new NodoDoble (dato);

    nuevo -> ponerAdelante(cabeza);

    if (cabeza != NULL )
        cabeza -> ponerAtras(nuevo);
    cabeza = nuevo;
}
```

Implementación de visualizar()

- En primer lugar, declara índice como un puntero al NodoDoble.
- Se recorre desde el nodo cabeza, que fue el último que ingreso a la lista.
- Repite el ciclo mientras el indice no sea NULL.
- Para seguir al siguiente iteración el indice a punta al siguiente nodo.

```
//Visualizar lista doble
void ListaDoble::visualizar()
{
    NodoDoble* indice;
    indice = cabeza;
    cout<<"Atras"<<"\t"<<"DirDelDato"<<"\t"<<"Dato"<<"\t"<<"Adelante"<<endl;
    while (indice != NULL)
    {
        cout<<indice->atrasNodo()<<"\t"<<indice<<"\t"<<indice->datoNodo()<<"\t"<<indice->adelanteNodo()<<endl;
        indice = indice -> adelanteNodo();
    }
}
```

Insertar después de un nodo

- Algoritmo:

1. Crear un nodo, nuevo, con el elemento.
2. Poner el enlace adelante del nodo creado apuntando al nodo siguiente de anterior.
3. El enlace atras del nodo siguiente a dato (si anterior no es el último nodo) tiene que apuntar a nuevo.
4. Hacer que el enlace adelante del nodo anterior apunte al nuevo nodo dato.
5. A su vez, el enlace atras del nuevo nodo dato debe de apuntar a anterior.

```
void ListaDoble::insertaDespues(int datoAnterior, int dato)
{
	NodoDoble* nuevo;
	NodoDoble* anterior;
	nuevo = new NodoDoble(dato);
	// Bucle de búsqueda del anterior
	NodoDoble* indice;
	indice = cabeza;
	while (indice != NULL)
	{
		if(indice -> datoNodo() == datoAnterior)
			break;
		else
			indice = indice -> adelanteNodo();
	}
	//Inserta despues
	if(indice != NULL)
	{
		anterior=indice;
		nuevo -> ponerAdelante(anterior -> adelanteNodo());
		if (anterior -> adelanteNodo() != NULL)
			anterior -> adelanteNodo() -> ponerAtras(nuevo);
		anterior-> ponerAdelante(nuevo);
		nuevo -> ponerAtras(anterior);
	}
}
```

Eliminar un nodo de una lista doblemente enlazada

- Algoritmo:

1. Búsqueda del nodo que contiene el dato.
2. El puntero adelante del nodo anterior tiene que apuntar al puntero adelante del nodo a eliminar (si no es el nodo cabecera).
3. El puntero atras del nodo siguiente a borrar tiene que apuntar a donde apunta el puntero atras del nodo a eliminar (si no es el último nodo).
4. Si el nodo que se elimina es el primero, se modifica cabeza para que tenga la dirección del nodo siguiente.
5. La memoria ocupada por el nodo es liberada.

```
void ListaDoble::eliminar (int dato)
{
	NodoDoble* indice;
	indice = cabeza;
	// Bucle de búsqueda
	while (indice != NULL)
	{
		if(indice -> datoNodo() == dato)
			break;
		else
			indice = indice -> adelanteNodo();
	}
	// Enlace de nodo anterior con el siguiente
	if (indice != NULL)
	{
		//distingue entre nodo cabecera o resto de la lista
		if (indice == cabeza)
		{
			cabeza = indice -> adelanteNodo();
			if (indice -> adelanteNodo() != NULL)
				indice -> adelanteNodo() -> ponerAtras(NULL);
		}
		else if (indice -> adelanteNodo() != NULL) // No es el último
		{
			indice->atrasNodo()->ponerAdelante(indice->adelanteNodo());
			indice->adelanteNodo()->ponerAtras(indice->atrasNodo());
		}
		else // último nodo
			indice->atrasNodo()->ponerAdelante(NULL);
	delete indice;
	}
}
```

Ejercicio 1

- Programe el método insertaAntes() para insertar antes de un nodo.

Ejercicio 2

- Programe un método que invierta una la lista doblemente enlazada. Ejemplo si la lista es 7, 6, 3, 8, 2, 5 debe invertirla a 5, 2, 8, 3, 6, 7

Ejercicio 3

- Programa el método ordenar() que ordene una lista doblemente enlazada, que ya tiene datos, de forma ascendente. Utilice el método de ordenación de su preferencia. Ejemplo si la lista es 7, 6, 3, 8, 2, 5 debe ordenarla a 2, 3, 5, 6, 7, 8

FIN