

Estructura de datos

Estructuras dinámicas

↳ Crecen y se reducen en el tiempo de ejecución // Intro

Listas enlazadas

↳ Datos alineados en filas

↳ Los usuarios pueden agregar o eliminar datos de cualquier parte de la fila

Pilas

↳ Se apilan los datos

↳ Los usuarios solo pueden agregar o eliminar datos de la parte superior

Colas

↳ Líneas de espera

↳ Se insertan datos en el final (rabo)

↳ Se eliminan datos del inicio (cabeza)

Tipos struct simples, boxing y unboxing

- ↳ boolean
- ↳ byte
- ↳ SByte
- ↳ char
- ↳ decimal
- ↳ double
- ↳ single
- ↳ Int 32, 64, 16
- ↳ UInt 32, 64, 16

conversiones boxing y unboxing

↳ todos los tipos heredan la clase value type (ValueType), esta hereda la clase object.

OPORTUNIDAD

cualquier valor tipo simple puede asignarse a una variable object

CONVERSIÓN
BOXING

} un valor de tipo simple
se copia en un objeto
para que pueda
manipularse como object

Puede ser de forma
implícita como explícita

EJEMPLO 2

```
button-click ()  
{  
    Página2 p2 = new Página2();  
    p2.show();  
    this.hide();  
}
```

Algoritmo para
cambiar de
páginas y
cerrar la
actual.

Ejemplo

```
int i = 5; // Crea un valor de tipo int.  
Object objeto1 = (Object)i; // Conversión boxing explícita del valor int.  
Object objeto2 = i; // Conversión boxing implícita del valor int.
```

int i = 5
↓
objeto1 = i → objeto2 = 5

Son dos objetos distintos que copian el int.

Conversion Unboxing

Puede usarse para convertir de forma explícita una referencia Object a un valor simple

Ejemplo

```
int Int1 = (int) objeto1
```

Lo si trata de realizar una conversión unboxing con un valor simple incorrecto produce una excepción

InvalidCastException

Clases Autorreferenciadas

Lo contiene un miembro de referencia que hace referencia a un objeto del mismo tipo de clase.

// una clase autorreferenciada

```
class Nodo
```

```
private int datos; // Almacena datos enteros
```

```
private Nodo siguiente; // Almacena referencia al siguiente Nodo
```

```
public Nodo ( int valorDatos )
```

// Cuerpo del constructor

}

```
public int Dados
```

{

get;

// Cuerpo get

}

Set

{

// Cuerpo set

}

}

Public Nodo Siguiente

get

// cuerpo get

set

// cuerpo set

}

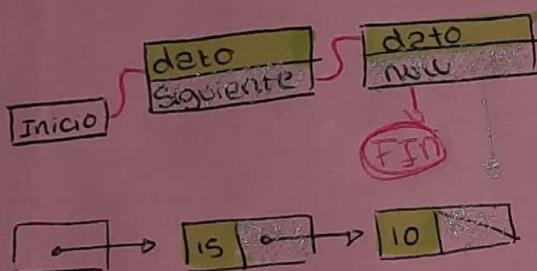
* *

}; // fin de la class Nodo

los objetos autorreferenciados pueden enlazarse para formar pilas, filas y colas

// mas adelante Arboles

Este código representa la estructura base de una lista de siempre enlazada



Estructura de la clase autorreferencial

clase

NO DO

(L)

dato

e

referencia del
siguiente Nodo

dato de
tipo entero

utiliza dos propiedades

→ dato

→ Siguiente

Para crear y mantener
estructuras dinámicas de datos

↳ Asignación dinámica de memoria

OPERADOR NEW

Nodo *Nodo = new Nodo();
Si no hay memoria

↳ OutOfMemoryException

LISTAS ENLAZADAS

La colección lineal de **Nodos** → Puede contener datos de cualquier tipo, incluyendo referencias a objetos o clases.

¿Qué diferencias tiene con los arreglos?

- Puede alterarse el tamaño de forma dinámica
- Su límite de almacenamiento es la memoria del sistema

Implementación de una lista enlazada

// declaración de NodoLista y lista

```
namespace BibliotecaListasEnlazadas
```

```
// clase para representar un nodo en una lista
```

```
class NodoLista
```

```
{
```

```
    private Object datos; // almacena los datos para el nodo
```

```
    private NodoLista siguiente; // almacena la referencia del
```

```
    siguiente nodo
```

```
// constructor para crear un NodoLista que haga referencia a
```

```
// ValorDatos y sea el último nodo de la lista
```

```
public NodoLista(Object valorDatos)
```

```
: this(valorDatos, null)
```

```
{
```

```
} // fin del constructor predeterminado
```

* nos permiten
inicializar un objeto
NodoLista que
se ubicará donde sea
deseadlo

```
// constructor para crear un NodoLista que haga referencia a ValorDatos  
// y se refiere al siguiente NodoLista en la lista.
```

```
public NodoLista(Object valorDatos, NodoLista siguienteNodo)
```

```
{
```

```
    datos = valorDatos;
```

```
    siguiente = siguienteNodo;
```

```
} // fin del constructor
```

*
*

// propiedad **Siguiente** → Propiedad de referencia

```
public NodoLista Siguiente
```

```
{
```

```
    get
```

```
{
```

```
        return siguiente;
```

```
}
```

```
Set
```

```
{
```

```
    siguiente = value;
```

```
} // fin de set
```

```
} // fin de la propiedad Siguiente
```

// Propiedad datos
Public Object Datos

{
Get
}

return datos;
} // Fin del get

} // Fin de la propiedad Datos

// Declaración de la clase Lista
Public class Lista

{

Private Nodolista PrimerNodo;

Private Nodolista UltimoNodo;

Private String nombre; // cadena a mostrar, tal como "lista"
// construye una lista vacía con el nombre especificado

Public Lista (String nombreLista)

{
nombre = nombreLista;
PrimerNodo = UltimoNodo = null;

} // Fin del constructor

// construye lista vacía como 'lista' con su nombre

Public Lista ()

: this ("lista")

{

}

// insertar objeto al frente de la lista si está vacía,

// primer nodo y último nodo harán referencia al mismo nodo.

// en caso contrario, primer nodo hace referencia al nuevo nodo.

public void InsertarAlFrente (Object insertarElemento)

{

if (estaVacia ())

{ PrimerNodo = UltimoNodo = new Nodolista (insertar elemento); } ;

else

{ PrimerNodo = new Nodolista (insertar Elemento, PrimerNodo); }

} // Fin de insertar al frente

// insertar objeto al final de la lista. Si está vacía,

// primer nodo y el último nodo harán referencia al mismo objeto.

// en caso contrario, la propiedad siguiente de último nodo hace referencia al nuevo nodo

Son los métodos principales

```
Public void InsertarAlFinal (Object insertarElemento)
{
    if (estávacía())
        PrimerNodo = ÚltimoNodo = new NodoLista (insertarElemento);
    else
        ÚltimoNodo = ÚltimoNodo.Siguiente = new NodoLista (insertarElemento);
}
```

? //Fin del metodo Insertar al final

//eliminar el primer nodo de la lista

```
Public Object EliminarAlFrente ()
{
}
```

```
if (Estávacía())
    throw new ExceptionListaVacia (nombre);
```

```
Object eliminarElemento = PrimerNodo.Dato //recupera los datos
```

// establece las referencias primerNodo y ultimoNodo

```
if (PrimerNodo == ÚltimoNodo)
    PrimerNodo = ÚltimoNodo = null;
```

```
else
    PrimerNodo = primerNodo.Siguiente;
```

```
return eliminarElemento; //devuelve los datos eliminados
```

?

//eliminar del final

```
Public Object EliminarDelFinal ()
{
}
```

```
if (Estávacía())
    throw new ExceptionListaVacia (nombre);
```

```
Object eliminarElemento = ÚltimoNodo.Datos; //obtiene los datos
```

//re establece la referencia del primerNodo y el ÚltimoNodo

```
if (PrimerNodo == ÚltimoNodo)
    PrimerNodo = ÚltimoNodo = null;
```

```
else
    PrimerNodo = primerNodo.Siguiente; ÚltimoNodo.Datos; //obtiene los datos
```

```
nodoLista actual = PrimerNodo;
```

//itera mientras el nodo actual NO sea el ÚltimoNodo

```
while (actual.Siguiente != ÚltimoNodo)
```

```
{ actual = actual.Siguiente //avanza al siguiente nodo
```

```
}
```

//actual es el nuevo ÚltimoNodo

```
ÚltimoNodo = actual;
```

```
actual.Siguiente = null;
```

? //Fin del else

```
return eliminarElemento; //devuelve los datos eliminados
```

? //Fin del Metodo eliminar al final

```

    // devuelve Verdadero si la lista está vacía
    Public bool EstaVacia()
    {
        return primerNodo == null;
    }

```

// imprime en pantalla la lista

```

    Public void Imprimir()
    {
        ...
    }

```

Metodo Insertar al Frente

① llama a "Esta Vacia()" para determinar si está vacío

* (línea 7)

IF (estaVacia())

{

... *

*

② Si es vacia, establece que el último nodo y el primer nodo hacen referencia a un nuevo Nodolista

③ else

el nodo se "enlaza" a la lista, estableciendo primer nodo hace referencia a Nodolista

Primer nodo



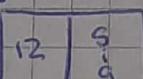
nodo lista



P.
nodo

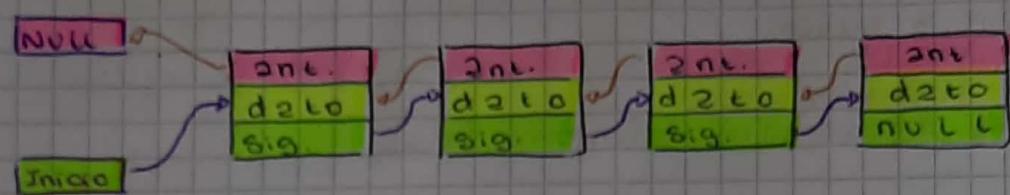
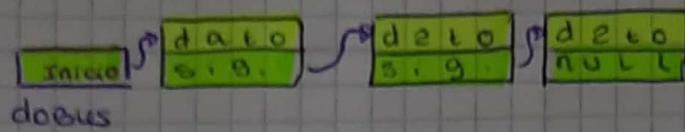


N.
nodo

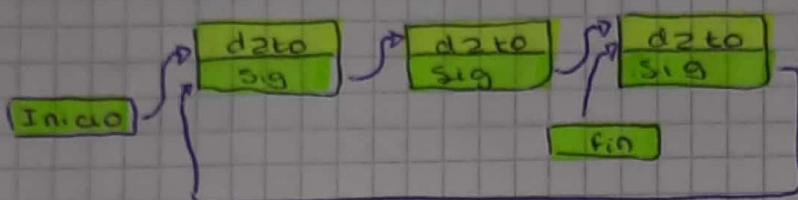


listas Enlazadas

EJEMPLOS



CIRCULAR



PASOS para VS-C#

1º crear clase para nodo
 { dato → Public string Dato;
 Siguiente → Public nodo Siguiente;
 }

2º crear clase lista
 {
 Public Nodo Inicio;
 }

3º Instanciamos un nodo
→ Usamos la clase que creamos en el punto anterior
lista en lazo doble miLista2 (2125) = new ListaEnlazada();

4º Programar eventos

//ejemplo

en el evento de click de inicio

```
if (txtNombre.Text == "")  
    {  
        nodo = unNuevoNodo = new Nodo()  
        unNuevoNodo.Nombre = txtNombre.Text;  
    }
```

5

// Procedimiento de agregar al inicio

if (Inicio == null) // es el primer valor agregado

{ nodo ingresado
 Inicio = un Nodo;

}

else

{

 nodo_aux = Inicio;

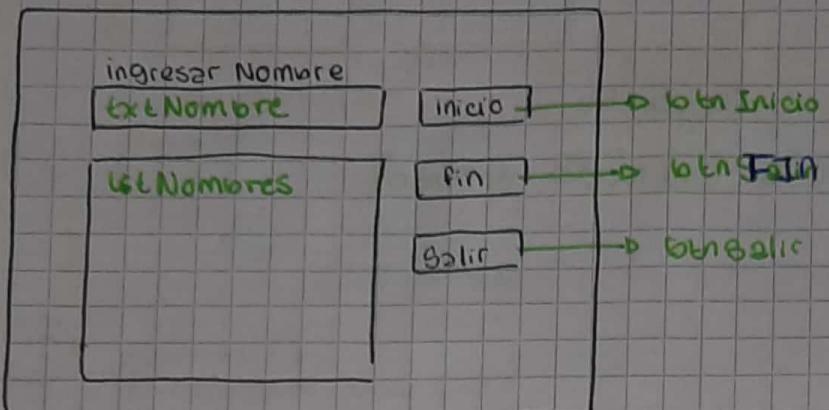
 Inicio = un Nodo;

 Inicio.siguiente = aux;

 mi lista.Agregar al inicio (un nuevo nodo)

Vídeo por partes

① listas enlazadas (Diseño)



② Crear una clase Nodo = | | |-------| | dato | | s.ig. |

// explorador de soluciones, click derecho, agregar, clase.

2. ~~Nombre~~
class Nodo

Public String Nombre;
Public Nodo Siguiente;

3

③ Crear una clase para listas enlazadas.

class listaEnlazada

2

Public Nodo Inicio;

3



④ programar acciones (eventos)

① Crear una lista enlazada

// Previo a la iniciación del componente

listaEnlazada milista = new listaEnlazada(); // Instanciación

② evento click de agregar a inicio

2

if (txtNombre.text != "")

Nodo unNuevoNodo = new Nodo(); // Instanciamos un nodo

unNuevoNodo.Nombre = txtNombre.text; // le ponemos el dato

... ⑥ ⑩

⑤ en la clase lista enlazada

~~public void AgregarInicio (Nodo unNodo)~~

if (inicio == null

{

 inicio = unNodo;

}

use

{

 Nodo aux = inicio;

 inicio = unNodo;

 inicio.Siguiente = aux;

 misListas.agregarAlInicio(unNuevoNodo)

}

⑥ misListas.AgregarInicio (unNuevoNodo);

⑦ vamos a hacer un procedimiento para que ordene los nodos en una lista // arriba del evento click

public void Muestra ()

{

 if

 lstNombre.Items.clear (); // metodo para limpiar la lista
 if (misListas.Inicio != null)

 } // agregar un item al list box

 // funcion recursiva

 ⑧ ⑨ AgregarItem (misListas.Inicio) // le paso el nodo inicio

{

Forms 1

⑤ void AgregarItem (Nodo UnNodo) // procedimiento recursivo
 { if (UnNodo != null) // siempre y cuando el nodo no apunte a null

}

 ListNombre.Items.add (UnNodo.Nombre)
 AregarItem (UnNodo.siguiente)

}

ns



dato
sig

dato
sig

dato
null

⑩ mostrar ();
txtNombre.Text = " ";
txtNombre.Focus ();

// terminó el agregar al inicio

⑪ private void event

if (txtNombre.Text == "") {

Nodo UNNuevoNodo = newNodo();

UNNuevoNodo.Nombre = txtNombre.Text;

... ⑫

⑬

⑫ hacer el procedimiento de agregar al final

public void AgregarAlFinal (nodo UnNodo)

{ if (Inicio == null)

{

 Inicio = UNnodo;

{

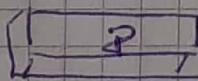
else

{

 Nodo aux; // creamos el nodo auxiliar

... ⑭

⑮ crear un procedimiento que se encargue de buscar el último nodo



public Nodo BuscarUltimo (nodo unNodo) //Funcion recursiva

{ if (unNodo.siguiente = null)

{ return unNodo;

{ else

{ return BuscarUltimo (unNodo.Siguiente)

}

}

(14) nodo aux = BuscarUltimo (Inicio) //crea un nodo auxiliar y le *
aux.siguiente = unNodo; //nodo nuevo que le estoy pasando
* //asigna el valor del nodo encontrado y apunta a sig.

/a el nuevo nodo
(15) mlista.AgregarFinal (unNuevoNodo); //voy a llamar al metodo
//agregar al final, ese metodo lo que va a hacer es buscar
//el ultimo nodo y le va a asignar como siguiente
//este nuevo nodo, y este se le va a asignar el val
//null

(16) mostrar ()

txtNombre.text = " ";
txtNombre.Focus();

(17) ~~btSalir~~

{ Application.Exit ();

}