



REFERENCIAS

Definición

Una referencia es un tipo especial de variable, que almacena el valor de una **dirección de memoria**, esta dirección puede ser la de una variable individual, pero más frecuentemente será la de un elemento de un array, una estructura u objeto de una clase. Las referencias, al igual que una variable común, pertenecen a un tipo (type).

TAD

Las estructuras dinámicas son una implementación de TDAs o TADs (Tipos Abstractos de Datos). En estos tipos el interés se centra más en la estructura de los datos que en el tipo concreto de información que almacenan.

Un TAD es un conjunto de objetos estructurados de alguna forma y con ciertas operaciones definidas sobre ellos.

Técnica basada en los TADs:

- Identificar los distintos tipos de datos que intervienen en el sistema y la función principal de dicho sistema.
- Caracterizar cada tipo de datos en función de las operaciones que se puedan realizar con los objetos de los distintos tipos (haciendo abstracción de sus representaciones concreta).
- Componer el sistema utilizando objetos de los tipos definidos junto con sus operaciones características.
- Implantar cada uno de los tipos utilizados.

Piénsese, por ejemplo, en una calculadora; los elementos que maneja son cantidades numéricas y las operaciones definidas son operaciones aritméticas. Otro ejemplo posible es el TAD matriz matemática; los elementos que maneja son las matrices y las operaciones son las que nos permiten crearlas, sumarlas, invertirlas, etc. Otro tipo de TAD es cualquier tipo de cola de espera: en el autobús, cine, la compra, etc. Los elementos de las colas son las personas y las operaciones más usuales son entrar o salir de la cola, pero no son las únicas. Desde un punto de vista abstracto podemos pensar en operaciones como crear un elemento de la cola, comprobar si la cola está vacía, si está llena, etc.

LISTAS LINEALES

Las estructuras básicas disponibles en C# tienen una importante limitación: no pueden cambiar de tamaño durante la ejecución, sin embargo en muchas ocasiones se necesitan estructuras que puedan cambiar de tamaño durante la ejecución del programa.

Las estructuras dinámicas nos permiten crear estructuras de datos que se adapten a las necesidades reales a las que suelen enfrentarse nuestros programas. Pero no sólo eso, como veremos, también nos permitirá crear estructuras de datos muy flexibles, ya sea en cuanto al orden, la estructura interna o las relaciones entre los elementos que las componen.

Las estructuras de datos están compuestas de otras pequeñas estructuras a las que llamaremos **nodos** o elementos, que agrupan los datos con los que trabajará nuestro programa y además una o más referencias a objetos del mismo tipo nodo.

Dependiendo del número de referencias y de las relaciones entre nodos, podemos distinguir varios tipos de estructuras dinámicas:



TEMA3. ESTRUCTURAS DE DATOS COMPUESTAS

- **Listas abiertas:** cada elemento sólo dispone de una referencia, que apuntará al siguiente elemento de la lista o valdrá NULL si es el último elemento.
- **Listas circulares:** o listas cerradas, son parecidas a las listas abiertas, pero el último elemento hace referencia al primero. De hecho, en las listas circulares no puede hablarse de "primero" ni de "último". Cualquier nodo puede ser el nodo de entrada y salida.
- **Listas doblemente enlazadas:** cada elemento dispone de dos referencias, una al siguiente elemento y el otra al elemento anterior. Al contrario que las listas abiertas anteriores, estas listas pueden recorrerse en los dos sentidos.
- **Pilas:** son un tipo especial de lista, conocidas como listas LIFO (Last In, First Out: el último en entrar es el primero en salir). Los elementos se "amontonan" o apilan, de modo que sólo el elemento que está encima de la pila puede ser leído, y sólo pueden añadirse elementos encima de la pila.
- **Colas:** otro tipo de listas, conocidas como listas FIFO (First In, First Out: El primero en entrar es el primero en salir). Los elementos se almacenan en fila, pero sólo pueden añadirse por un extremo y leerse por el otro.
- **Arboles:** cada elemento dispone de dos o más referencias, pero las referencias nunca son a elementos anteriores, de modo que la estructura se ramifica y crece igual que un árbol.
- **Arboles binarios:** son árboles donde cada nodo sólo puede apuntar a dos nodos.
- **Arboles binarios de búsqueda (ABB):** son árboles binarios ordenados. Desde cada nodo todos los nodos de una rama serán mayores, según la norma que se haya seguido para ordenar el árbol, y los de la otra rama serán menores.

LISTAS

Una *lista* es una estructura de datos que cumple con lo siguiente:

- Todos sus *componentes son del mismo tipo*.
- Cada *elemento o nodo va seguido de otro* del mismo tipo o de ninguno.
- Sus componentes se almacenan según *cierto orden*.

Tipos de listas

- Listas abiertas (lineales enlazadas)
- Listas doblemente enlazadas
- Listas circulares

Operaciones:

- Insertar elementos
- Eliminar elementos
- Recorrer listas
- Búsqueda de datos

Manejo de listas en C#

Una estructura básica de un nodo para crear listas de datos sería:

```
class nodo
{
    public int dato;
    public nodo nodosiguiente;
}
```

Para mayor referencia consulte la presentación de listas.



Ejercicios de la Práctica:

Ejercicio 1.

1. Crea una nueva solución llamada **Practica15Listas** y crea un nuevo proyecto llamado **listas**. Dentro del proyecto escribe el código de la parte inferior (**Utiliza tu librería de Validaciones**):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Validaciones;

namespace listas
{
    class Program
    {
        nodo LISTA = null;
        static void Main(string[] args)
        {
            Program objeto_lista = new Program();
            imprime("Vamos a crear una lista de 10 numeros");
            for (int i = 0; i < 10; i++)
            {
                Console.Write("{0}", i + 1);
                objeto_lista.insertar_nodo(Numeros..entero("Escriba un número."));
            }
            imprime("La lista es ");
            objeto_lista.recorrer_lista();
            imprime("Vamos a buscar un número");
            objeto_lista.buscar_nodo(Numeros.entero("Escriba el número que desea buscar en la lista"));
            imprime("Vamos a eliminar un nodo");
            objeto_lista.eliminar_nodo(objeto Numeros.entero("Escriba el número que desea eliminar de la lista"));
            imprime("La lista sin el nodo eliminado es ");
            objeto_lista.recorrer_lista();
        }
        void insertar_nodo(int N)
        {
            nodo P = new nodo();
            P.n = N;
            P.siguiete = LISTA;
            LISTA = P;
        }
        void recorrer_lista()
        {
            nodo Q = LISTA;
            while (Q != null)
            {
                Console.WriteLine(Q.n);
                Q = Q.siguiete;
            }
        }
        void buscar_nodo(int N)
        {
            nodo Q = LISTA;
            bool Ultimo = false;
            while ((Q.n != N) && (Ultimo == false))
            {
                if (Q.siguiete == null)
                    Ultimo = true;
                else
                    Q = Q.siguiete;
            }
        }
    }
}
```



TEMA3. ESTRUCTURAS DE DATOS COMPUESTAS

```
}
if (Ultimo == false)
    Console.WriteLine("Se encontro el numero");
else
    Console.WriteLine("No se encontro el numero");
}
void eliminar_nodo(int N)
{
    nodo Q = LISTA;
    nodo T = null;
    bool Ultimo = false;
    while ((Q.n != N) && (Ultimo == false))
    {
        if (Q.siguiete == null)
            Ultimo = true;
        else
        {
            T = Q;
            Q = Q.siguiete;
        }
    }
    if (Ultimo == false)
    {
        T.siguiete = Q.siguiete;
        Console.WriteLine("Se encontro el numero y fue eliminado.");
    }
    else
        Console.WriteLine("No se encontró el numero.");
}
static void imprime(string cadena)
{
    Console.WriteLine(cadena);
}
}
class nodo
{
    public int n;
    public nodo siguiete;
}
}
```

2. Adiciona al proyecto los siguientes dos métodos:
 - a) Insertar un nodo al final de la lista.
 - b) Modificar valor de un nodo especificado.

3. Utiliza el ejercicio de la práctica anterior (préstamo de libros en biblioteca) y adiciónale los siguientes métodos:
 - a) Menu con las siguientes opciones: mostrar lista de libros, agregar libro al inicio de la lista, agregar libro al final de la lista, buscar libro, eliminar libro, actualizar libro.
 - b) Crea un método para cada opción del menú.
 - c) Utiliza tu librería Validaciones para solicitar las variables.