Facultad:	Ingeniería
Escuela:	Computación
Asignatura:	Programación con Estructuras de Datos

Tema: Estructura Lista Enlazada.

Competencia

- Desarrolla sistemas de información informáticos mediante la integración de principios matemáticos, ciencia computacional y prácticas de ingeniería, considerando estándares de calidad y mejores prácticas validadas por la industria del software.

Materiales y Equipo

- Guía Número 3
- Computadora con programa Microsoft Visual C#.

Introducción Teórica

ESTRUCTURA DINÁMICA LISTA

Una lista está formada por una serie de elementos llamados nodos los cuales son objetos que contiene como variable miembro un puntero asignado y variables de cualquier tipo para manejar datos. El puntero sirve para enlazar cada nodo con el resto de nodos que conforman la lista.

De esto podemos deducir que una lista (lista) es una secuencia de nodos en el que cada nodo esta enlazado o conectado con el siguiente (por medio del puntero mencionado anteriormente). El primer nodo de la lista se denomina cabeza de la lista y el último nodo cola de la lista. Este último nodo suele tener su puntero igualado a NULL Para indicar que es el fin de la lista.
En las listas de acuerdo a quién apunte su cabeza y cola y al tipo de nodo que implementa (cuántos punteros tiene) se dividen en cuatro grandes tipos:

- Listas enlazadas (simplemente enlazadas)
- Listas doblemente enlazadas
- Listas circulares (simplemente circulares)
- Listas doblemente circulares

Las listas simplemente enlazadas, permiten recorrer la lista en un solo sentido y desde la cabeza hasta la cola.

Las listas doblemente enlazadas, permiten el recorrido en dos direcciones, de la cabeza a la cola y de la cola hacia la cabeza.

Las listas simplemente circulares, permiten el recorrido en una dirección pero al llegar al último nodo (cola) este se encuentra comunicado o enlazado a la cabeza, haciendo un anillo o circulo si se representa gráficamente.

Las listas doblemente circulares, permiten el recorrido en ambas direcciones y la cabeza y cola se encuentran conectadas en ambas direcciones.

LISTA SIMPLEMENTE ENLAZADA

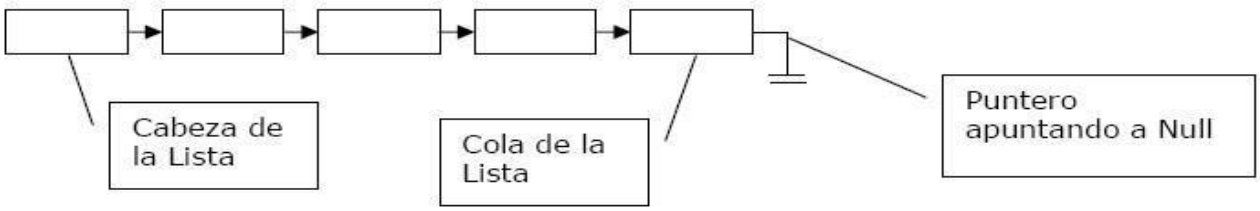
De estos cuatro tipos de lista en esta guía nos enfocaremos en las listas simplemente enlazadas, por ser las más genéricas y las que dan origen a otras estructuras lineales que estudiaremos más adelante.

Los nodos de la lista simplemente enlazada tienen la siguiente forma:

El dato que contenga un nodo puede ir desde un tipo de dato básico como un entero, decimal o carácter hasta algo más complejo como una estructura completa. Los punteros son los que enlazan al nodo con otros nodos exactamente iguales a él.

La lista enlazada es una estructura de datos dinámica cuyos nodos suelen ser normalmente registros y que tienen un tamaño fijo. Ahora bien suelen llamarse estructuras dinámicas porque se crean y destruyen según se vayan necesitando. De este modo se solicita o libera memoria en tiempo de ejecución del programa.

Gráficamente una lista enlazada puede representarse de la siguiente forma:



Las operaciones típicas de la lista incluyen: Crear la lista, verificar si está vacía, insertar elementos, eliminar elementos, mostrar elementos.

La lista no tiene ninguna restricción en cuanto a dónde puede eliminar o agregar nodos y es precisamente esta característica lo que la hace la estructura más versátil.

Procedimiento

EJEMPLO1_GUÍA3

Abrir un nuevo proyecto en Visual C# (el proyecto será en entorno CONSOLA).

1. Añada una clase nueva al proyecto, nómbrela **nodo** y codifique lo siguiente:

```
class nodo
{
    public int dato; //atributo que almacena el valor del nodo, en este caso entero
    public nodo siguiente; //atributo que señala al siguiente nodo, clase auto referenciada
}
```

2. Añada una clase nueva al proyecto, nómbrela **lista** y codifique lo siguiente para sus atributos y constructor:

```
class lista
{
    public nodo inicio; //cabeza de la lista

    //constructor por defecto
    public lista()
    { inicio = null; }
```

2.1 Siempre dentro de la clase lista para el método que inserta en la cola de la lista

```
//método para insertar al final de la lista, como lo hace ArrayList
public void InsertarF(int item)
{
    nodo auxiliar = new nodo(); //nodo temporal que todavía no pertenece a la lista
    auxiliar.dato = item; //almacena en el atributo dato el valor recibido en el parámetro
    auxiliar.siguiente = null; //hace que el apuntador señale a null

    if(inicio == null) //verifica si la lista está vacia
    {
        inicio = auxiliar; //hacemos que nodo sea parte de la lista, lo hacemos la cabeza.
    }
    else
    {
        nodo puntero; //utilizamos este nodo para recorrer la lista, así no se mueve la cabeza
        puntero = inicio; //situamos a puntero señalando al mismo nodo que inicio
        while(puntero.siguiente!=null)
        {
            puntero = puntero.siguiente; //se desplaza por todos los nodos de la lista
        }
        puntero.siguiente = auxiliar; // hacemos que último nodo ahora señale al auxiliar
    }
}
```

2.2 Para el método que inserta en la cabeza de la lista

```
//método insertar al inicio (insertar en la cabeza)
public void InsertarI(int item)
{
    nodo auxiliar = new nodo(); //nodo temporal que después se agrega a la lista

    auxiliar.dato = item; //almacena valor en el atributo dato
    auxiliar.siguiente = null; //hacemos que puntero siguiente señale a null

    if(inicio ==null)
    {
        inicio = auxiliar; //al estar vacia la cola lo hacemos la cabeza
    }
    else
    {
        nodo puntero; //para recorrer lista;
        puntero = inicio; //dos apuntadores señalando al mismo nodo
        inicio = auxiliar; //asignamos como cabeza al nodo auxiliar
        auxiliar.siguiente = puntero; //el puntero de auxiliar que ahora es cabeza se enlaza con
        // el nodo que era antes la cabeza y que tiene apuntador puntero
    }

}
```

2.3 Para el método que elimina en la cabeza de la lista

```
//método para eliminar nodo que está a la cabeza de la lista
public void EliminarI()
{
    if(inicio == null) //cuando lista esté vacia
    { Console.WriteLine("Lista vacia, no se puede eliminar elemento"); }
    else
    { inicio = inicio.siguiente; //a quien estaba señalando inicio será nuevo inicio
    }
}
```

2.4 Para el método que elimina en la cola de la lista

```
//método para eliminar nodo al final de la lista
public void EliminarF()
{
    if(inicio ==null) //cuando lista esté vacia
    { Console.WriteLine("Lista vacia, no se puede eliminar elemento"); }
    else
    {
        nodo punteroant, punteropost; //requiero dos punteros para mover porque no declaré la cola
        punteroant = inicio; //inicializo ambos en la cabeza de la lista
        punteropost = inicio;

        while (punteropost.siguiente!=null) //mientras puntero posterior no señale a null
        {
            punteroant = punteropost; //el puntero anterior será a quien señale el posterior
            punteropost = punteropost.siguiente; //puntero posterior será a quien señalaba antes el posterior
        }
        punteroant.siguiente = null; //con esto sacamos el que estaba al final de la lista, el último nodo
        //era el que estaba señalando el punteropost pero ahora el último será en donde se quedó punteroant
    }
}
```

2.5 Para trabajar con el método que inserta por una posición dada por el usuario.

```
// método para insertar en una posición específica de la lista
public void InsertarP(int item, int pos)
{
    nodo auxiliar = new nodo(); //definición de nuevo nodo con sus atributos
    auxiliar.dato = item;
    auxiliar.siguiente = null;

    if (inicio == null) //si lista está vacia
    {
        Console.WriteLine("La lista está vacia, por lo tanto se va a insertar en la 1a. posición");
        inicio = auxiliar; //inserto nodo
    }
    else
    {
        nodo puntero;
        puntero = inicio; //para recorrer lista
        if(pos ==1) //si la posición que pidieron es la 1, se inserta en la cabeza
        {
            inicio = auxiliar;
            auxiliar.siguiente = puntero;
        }
```

```
        else //si la posición solicitada no es la 1
        {
            for(int i=1; i<pos-1; i++)
            {
                puntero = puntero.siguiente;
                if (puntero.siguiente == null) //si llegamos al final de la lista
                    break;
            }

            nodo punteronext; //apuntador de ayuda
            punteronext = puntero.siguiente; //a quien señalaba el puntero ahí se ubicará el apuntador punteronext
            puntero.siguiente = auxiliar; //ahora el apuntador puntero señalará a al nodo auxiliar (el nuevo)
            auxiliar.siguiente = punteronext; //el nodo recien ingresado señalará a punteronext
            //con estas cuatro líneas anteriores es cómo se corta momentaneamente la lista y luego se une de nuevo
        }
    }
}
```

2.6 Para el método que muestra el contenido de la lista

```
    //método que muestra el contenido de la lista
    public void mostrar()
    {
        if(inicio ==null) //si la lista está vacia
        { Console.WriteLine("La lista está vacia"); }
        else
        {
            nodo puntero;
            puntero = inicio; //inicializamos a puntero en el mismo nodo que la cabeza;
            Console.Write("{0} -> \t", puntero.dato); //imprimir valor de nodo. Write y no WriteLine
            while(puntero.siguiente!=null)
            {
                puntero = puntero.siguiente; //avanzamos un nodo en la lista
                Console.Write("{0} -> \t", puntero.dato); //usar Write para que no salte de línea
            }
        }
        Console.WriteLine();
    }
} //fin de la clase lista
```

3. Una vez concluida la clase lista para poder utilizarla debemos crear los objetos pertinentes, es por ello que iremos ahora a la clase Program y en el Main escribiremos:

```
static void Main(string[] args)
{
    //creamos una instancia de la clase lista para que podamos utilizar los métodos
    lista listanueva = new lista();

    //le agregamos cuatro nodos a la lista
    listanueva.InsertarI(40);
    listanueva.InsertarI(30);
    listanueva.InsertarI(20);
    listanueva.InsertarI(10);
    //mostramos qué hay contenido dentro de la lista
    listanueva.mostrar();
```

4. Ahora ejecute el código trabajado y observe lo que realiza el programa

5. Posteriormente agregue las siguientes líneas a su código en el Main

```
//insertando en posición
listanueva.InsertarP(220, 3);
listanueva.mostrar();

Console.ReadKey();
```

EJEMPLO2_GUÍA3

Ahora que conocemos el funcionamiento interno de una lista, trabajaremos con la clase List<T> que proporciona C# en su namespace System.Collections.Generic; (agregarla si no la tiene habilitada), realizaremos un juego de memoria.

1. Abra un nuevo proyecto en entorno gráfico (Windows Form) y nombrelo como memoria.
2. Ahora a las propiedades del formulario aplique las siguientes modificaciones:

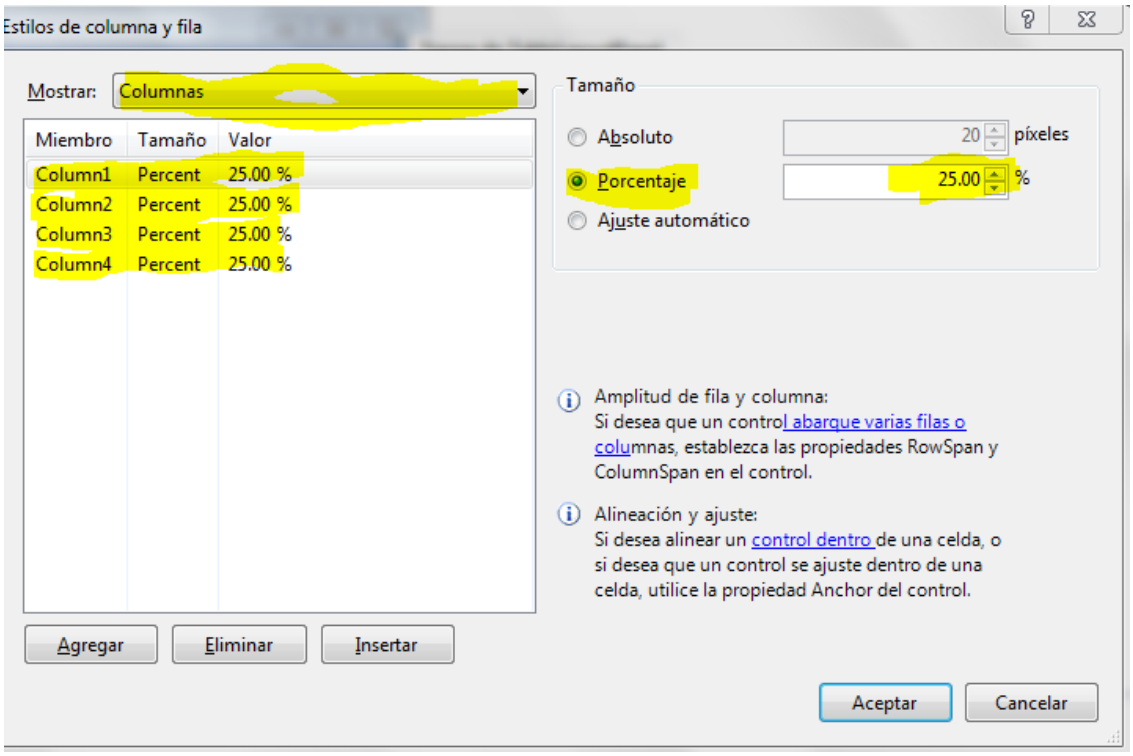
Propiedad	**Valor**
Size	550 , 550

3. Ahora agregue en el formulario un control TableLayoutPanel, con las propiedades de esa herramienta realice modificaciones en el orden indicado en la tabla (respetar ese orden):

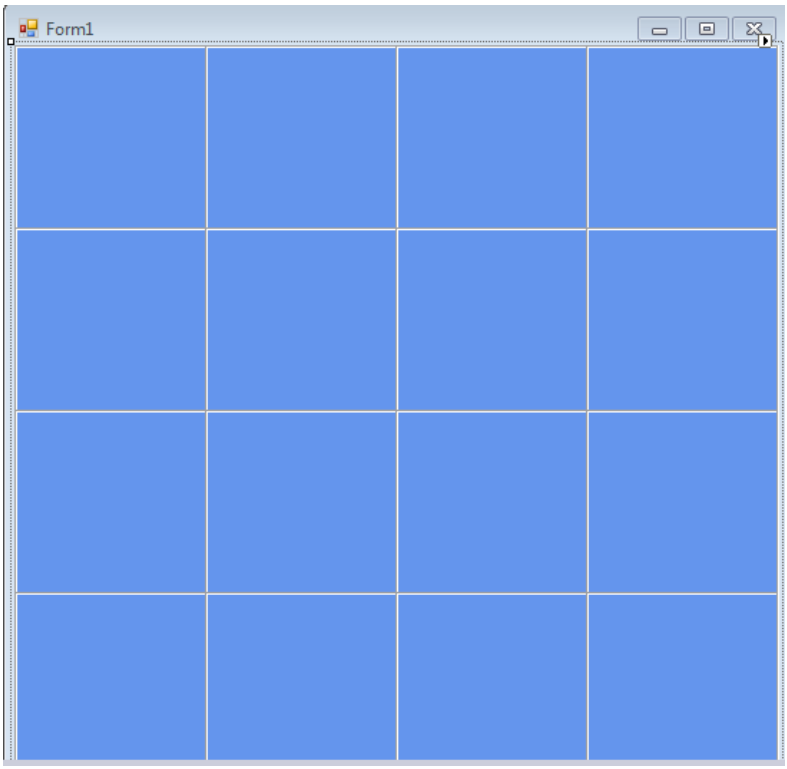
Propiedad	Valor
BackColor: Tipo Web	CornflowerBlue
Dock:	Fill
CellBorderStyle:	Inset

4. Ahora en las tareas del TableLayoutPanel (el triángulo en la parte superior izquierda) seleccione Editar filas y columnas

Construya una tabla con 4 columnas y 4 filas de 25% por cada uno



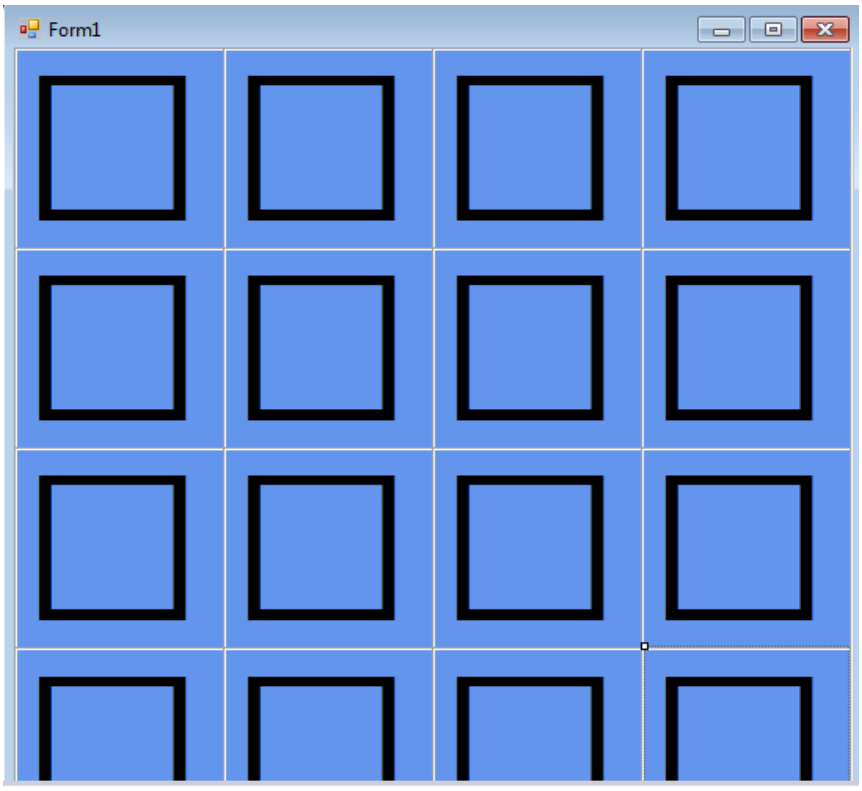
Al terminar este proceso debe obtener una pantalla similar a esta:



5. Con el TableLayoutPanel seleccionado, agregue un control Label a la celda superior izquierda del TableLayoutPanel. Las propiedades de este Label serán:

Propiedad	Valor
BackColor: Tipo Web:	CornflowerBlue
AutoSize:	False
Dock:	Fill
TextAlign:	MiddleCenter
Font:	Webdigns
Estilo de Fuente	Negrita
Tamaño de fuente	70
Texto por defecto:	C

6. Ahora copie ese Label en cada cuadro del TableLayoutPanel. Tendremos una interfaz como en la figura



7. Ahora habrá que trabajar el código del proyecto, para lo cual se programará lo siguiente:

```
public partial class Form1 : Form
{
    Random random = new Random(); //crear un objeto random
    //Utilizamos una lista de C# con elementos de tipo string
    //los valores de la lista ya estarán predeterminados.
    List<string> iconos = new List<string>()
    {
        "!", "!", "N","N", ",",",",",","k","k",
        "b","b","v","v","w","w","z","z"
    };

    //Primer label al que se le ha dado click
    Label primerclick = null;
    //Segundo label al que se le ha dado click
    Label segundoclick = null;
```

```
//método para asignar iconos a los label
1 referencia
private void Asignar()
{ //realiza esta acción para cada label
    foreach (Control control in tableLayoutPanel1.Controls)
    {
        Label iconLabel = control as Label;
        if(iconLabel != null)
        {
            int numeroRandom = random.Next(iconos.Count);
            iconLabel.Text = iconos[numeroRandom];
            iconos.RemoveAt(numeroRandom);
        }

    }
}
```

```
    public Form1()
    {
        InitializeComponent();
        Asignar(); //llamamos al método
    }
```

8. Si se ha programado de forma correcta, al ejecutar el programa verá una interfaz similar a la imagen

9. Ahora ocultaremos los iconos para que el usuario no pueda verlos, agregue la línea

```
private void Asignar()
{ //realiza esta acción para cada label
    foreach (Control control in tableLayoutPanel1.Controls)
    {
        Label iconLabel = control as Label;
        if(iconLabel != null)
        {
            int numeroRandom = random.Next(iconos.Count);
            iconLabel.Text = iconos[numeroRandom];
            iconos.RemoveAt(numeroRandom);
        }
        iconLabel.ForeColor = iconLabel.BackColor;
    }
}
```

10. Procedemos a añadir un timer al proyecto, darle un intervalo de 750, el código asociado al Timer es:

```
1 referencia
private void timer1_Tick(object sender, EventArgs e)
{
    timer1.Stop(); //detener timer
    primerclick.ForeColor = primerclick.BackColor; //color fondo click 1
    segundoclick.ForeColor = segundoclick.BackColor; //color fondo click 2
    primerclick = null; //retorna el primer click a null
    segundoclick = null; //retorna el segundo click a null
}
```

11. Ahora se programará el evento Click del Label1

```
private void label1_Click(object sender, EventArgs e)
{
    if (timer1.Enabled == true) //si el timer está habilitado
        return;
    Label clickedLabel = sender as Label; //registra si el label ha sido
    //seleccionado
    //si ese label en concreto ha sido seleccionado ( no está nulo)
```

```
if(clickedLabel != null)
{ //el forecolor será negro
    if (clickedLabel.ForeColor == Color.Black)
        return;
    VerificaGana(); //llama al método de Verificar Ganador
    if (primerclick == null) //si no ha habido un click antes
    { //se convierte en primer elemento clickeado
        primerclick = clickedLabel;
        //el color de forecolor en negro
        primerclick.ForeColor = Color.Black;
        //retorna
        return;
    }
        //si ya hay algo en primer click entonces será el segundo
        segundoclick = clickedLabel;
        //color en negro
        segundoclick.ForeColor = Color.Black;
        //elemento clickeado es segundo
        if(primerclick.Text== segundoclick.Text)
        {
            primerclick = null;
            segundoclick = null;
            return;
        }
        timer1.Start();
    }
}
```

12. Realice esta misma acción para todos los Label de la tabla. Recuerde activar el evento Click para cada Label

13. Incluya la declaración del método VerificaGana()

```
private void VerificaGana()
{
}
```

14. Ejecute el programa y verifique su funcionamiento.

Desarrollo de habilidades

Ejercicio No. 1

Modifique el ejemplo de forma haya un menú de opciones para acceder a los métodos que contiene la clase.

a. Insertar al Frente
b. Insertar al Final
c. Insertar en una posición específica
d. Eliminar al Frente
e. Eliminar al Final
f. Mostrar lista
g. Salir

El menú deberá estar siempre disponible hasta que el usuario seleccione la opción g es decir salir de la aplicación

Ejercicio No. 2

Modifique el método de insertar por posición para que en caso de que la posición seleccionada no exista envíe un mensaje de alerta y que inserte al final (es decir que inserte en la cola).

Ejercicio No. 3

Agregue un método más a la clase lista, de forma que los datos se ingresen en forma ascendente, es decir que se comparará con el primer valor que encuentra y si es menor se ingresa antes y sino sigue hasta encontrar la ubicación que le corresponde. Haga la correspondiente prueba en el Main para verificar su funcionamiento.

Ejercicio No. 4

Utilizando la clase dada, pase al entorno gráfico. De forma que los valores de los nodos ingresados se reflejen en una herramienta como ListBox u otra y al momento de suprimirlos también se actualice y el dato no aparezca más en el entorno. El diseño es totalmente libre.

Ejercicio No. 5

Programe el método VerificaGana( ) para el segundo ejemplo, de forma que al descubrir todas las figuras envíe un mensaje de juego terminado y que cierre la aplicación.