TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE TIJUANA

SUBDIRECCIÓN ACADÉMICA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

SEMESTRE: AGOSTO – DICIEMBRE 2021

ING. SISTEMAS COMPUTACIONALES

ESTRUCTURA DE DATOS - 2SC3C

INVESTIGACIÓN
UNIDAD 3 – RECURSIVIDAD

GUILLEN MARTINEZ ANTHONY - 20210575

M.C.C. LUZ ELENA CORTEZ GALVAN

FECHA DE ENTREGA 14/10/202

CONTENIDO DEL TRABAJO:

INDICE

3.1 - Pilas……3

3.2 - Colas……..8

3.3 – Listas Enlazadas…….. 10

Fuentes de Información……14

3.1 – Pilas

Qué es una Pila.

Una pila (stack en inglés) es una lista ordenada o estructura de datos que permite almacenar y recuperar datos, siendo el modo de acceso a sus elementos de tipo LIFO (del inglés Last In, First Out, «último en entrar, primero en salir»). Esta estructura se aplica en multitud de supuestos en el área de informática debido a su simplicidad y capacidad de dar respuesta a numerosos procesos.

Cuáles son las operaciones básicas de una pila.

Habitualmente, junto a las dos operaciones básicas de apilar y desapilar (push, pop), las pilas puede implementar otra serie de funciones:

- Crear (constructor): crea la pila vacía.
- Tamaño (size): regresa el número de elementos de la pila.
- Apilar (push): añade un elemento a la pila.
- Desapilar (pop): lee y retira el elemento superior de la pila.
- Leer último (top o peek): lee el elemento superior de la pila sin retirarlo.
- Vacía (empty): devuelve cierto si la pila está sin elementos o falso en caso de que contenga alguno.

Una pila puede implementarse fácilmente ya sea mediante una matriz o una lista enlazada. Lo que identifica a una estructura de datos como una pila en cualquier caso no es su estructura sino su interfaz: al usuario solamente se le permite colocar y extraer datos en el modo que se espera de una pila y algunas otras operaciones auxiliares.

Aplicaciones de una Pila.

Las pilas suelen emplearse en los siguientes contextos:

- Evaluación de expresiones en notación postfija (notación polaca inversa).
- Reconocedores sintácticos de lenguajes independientes del contexto.
- Implementación de recursividad.

Programa ejemplo en C# de un TDA Pila.

```
01 using System;
02 using System.Collections.Generic;
03 using System.Linq;
04 using System.Text;
05 using System.Collections;
06
```

```
07 namespace conStack
08 {
09     class Program
10     {
11         static void Main(string[] args)
12         {
13 
14             Stack miPila = new Stack();
15 
16             int opcion;//opcion del menu
17 
18             do{
19                 Console.Clear();//se limpia consola
20 
21                 opcion = menu();//muestra menu y espera opción
22 
23                 switch (opcion)
24                 {
25                     case 1:
26                         agregar(ref miPila);
27                         break;
28                     case 2:
29                         eliminar(ref miPila);
30                         break;
31                     case 3:
32                         limpiar( ref miPila);
33                         break;
34                     case 4:
35                         imprimir(miPila);
36                         break;
37                     case 5: break; //salir
38                     default:
39                         mensaje("ERROR: la opción no es valida. Intente de n
uevo.");
40                         break;
41                 }
42 
43             }
```

```
44              while(opcion!=5);
45
46              mensaje("El programa a finalizado.");
47          }
48
49          /** añade un nuevo elemento a la pila */
50          static void agregar( ref Stack pila )
51          {
52              Console.Write("\n>Ingrese valor: ");
53              try
54              {
55                  int valor = Convert.ToInt32(Console.ReadLine());
56                  if (valor > 99 || valor <= 0)
57                  {
58                      mensaje("Solo números del 1 al 99");
59                  }
60                  else
61                  {
62                      pila.Push(valor);
63                      imprimir(pila);
64                  }
65              }
66              catch {
67                  mensaje("Error: Solo números del 1 al 99");
68              }
69
70          }
71
72          /** Elimina todo los elementos de la pila */
73          static void limpiar( ref Stack pila )
74          {
75              pila.Clear();
76              imprimir(pila);
77          }
78
79          /** Elimina elemento de la pila */
80          static void eliminar( ref Stack pila)
```

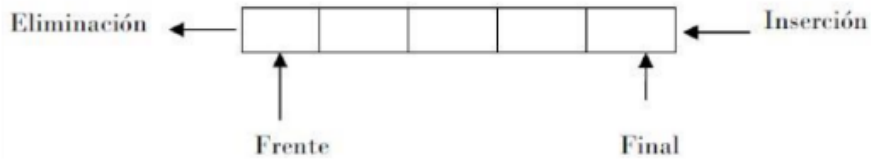
```
81          {
82              if (pila.Count > 0)
83              {
84                  int valor = (int)pila.Pop();
85                  mensaje("Elemento " + valor + " eliminado");
86              }
87              else {
88                  imprimir(pila);
89              }
90
91          }
92
93          /** muestra menu y retorna opción */
94          static int menu()
95          {
96              //Console.Clear();
97              Console.WriteLine("\n            Stack Menu\n");
98              Console.WriteLine(" 1.- Agregar elemento");
99              Console.WriteLine(" 2.- Eliminar elemento");
100              Console.WriteLine(" 3.- Vaciar Pila");
101              Console.WriteLine(" 4.- Ver pila");
102              Console.WriteLine(" 5.- Termina programa");
103              Console.Write(" JC:> Ingresa tu opción: ");
104              try
105              {
106                  int valor = Convert.ToInt32( Console.ReadLine() );
107                  return valor;
108              }
109              catch {
110                  return 0;
111              }
112          }
113
114          /** Muestra mensaje del programa al usuario */
115          static void mensaje( String texto )
116          {
117              if (texto.Length > 0)
```

```
118             {
119                 Console.WriteLine("\n    ================================
==================");
120                 Console.WriteLine(" JC:> {0}",texto);
121                 Console.WriteLine(" ====================================
================");
122                 Console.WriteLine("\n    JC:>Presione cualquier tecla para
continuar...");
123                 Console.ReadKey();
124             }
125         }
126 
127         /** Imprime pila */
128         static void imprimir(Stack pila)
129         {
130             if (pila.Count > 0)
131             {
132                 Console.WriteLine("");
133                 foreach (int dato in pila)
134                 {
135                     Console.WriteLine(" | |");
136                     if( dato <10)
137                         Console.WriteLine(" | 0{0} |", dato);
138                     else
139                         Console.WriteLine(" | {0} |", dato);
140                     Console.WriteLine(" |_____|");
141                 }
142                 Console.WriteLine("\nPresione cualquier tecla para continua
r...");
143                 Console.ReadKey();
144             }
145             else
146             {
147                 mensaje("La Pila esta vacia");
148             }
149         }
150     }
151 }
```

3.2 - Colas

Qué es una Cola

Las colas son llamadas también estructuras FIFO (first-in, first-out, primero en entrar, primero en salir). Sus aplicaciones son numerosas: colas de las tareas realizadas por una impresora, acceso a almacenamiento en disco, sistema compartido y uso del CPU. Como su definición lo dice, en una cola los elementos deben insertarse por la final de la cola y eliminarse por el frente, así se cumple la definición primero en entrar, primero en salir. Gráficamente una cola puede representarse de la siguiente manera:



Clasificaciones de las Colas (simple, doble=bicola, circular, prioridades).

Cola simple: Se inserta por un sitio y se saca por otro, en el caso de la cola simple se inserta por el final y se saca por el principio. Para gestionar este tipo de cola hay que recordar siempre cual es el siguiente elemento que se va a leer y cual es el último elemento que se ha introducido.

Colas dobles: Permiten realizar las operaciones de inserción y eliminación por cualquiera de sus extremos. Una cola doble también puede ser circular, en dicho caso, será necesario que los métodos de inserción y eliminación (sobre cualquiera de los métodos de inserción y eliminación (sobre cualquiera de los extremos) considere el movimiento adecuado de los punteros.

Colas circulares: Es aquella en la cual el sucesor del último elemento es el primero. Por lo tanto, el manejo de las colas como estructuras circulares permite un mejor uso del espacio de memoria reservando para la implementación de las pilas.

Colas de Prioridad: En ellas, los elementos se atienden en el orden indicado por una prioridad asociada a cada uno. Si varios elementos tienen la misma prioridad, se atenderán de modo convencional según la posición que ocupen. Hay dos formas de implementación.

- Añadir un campo a cada nodo con su prioridad. Resulta conveniente mantener la cola ordenada por orden de prioridad.
- Crear tantas colas como prioridades haya, y almacenar cada elemento en su cola

Cuáles son las operaciones básicas de una cola

- **InicializarCola:** nos permite dejar inicialmente vacía la cola una vez creada.
- **Enqueue:** permite añadir un elemento al final de la cola.
- **Dequeue:** se usará para sacar un elemento de la cola.
- **ColaVacia:** devolverá cierto si la cola está vacía antes de sacar un elemento de la misma.

- **ColaLlena:** sólo en aquellos casos en los que sea necesario determinar si la cola se encuentra llena antes de añadir un nuevo elemento debido a la implementación utilizada (sólo en el caso de implementación con tablas), emplearemos esta operación de tipo lógico.

Aplicaciones de una Cola.

En general, operaciones en redes de computadoras.

- Trabajos enviados a una impresora
- Solicitudes a un servidor.

Clientes solicitando ser atendidos por una telefonista.

Mas aplicaciones de las estructuras tipo cola

Programa ejemplo en C# de un TDA Cola.

```
using System;
using System.Collections;

public class Ejemplo_11_03a
{
    public static void Main()
    {
        string palabra;

        Queue miCola = new Queue();
        miCola.Enqueue("Hola,");
        miCola.Enqueue("soy");
        miCola.Enqueue("yo");

        for (byte i=0; i<3; i++)
        {
            palabra = (string) miCola.Dequeue();
            Console.WriteLine( palabra );
        }
    }
}
```

3.3 - Listas

Qué es una Lista

La Lista es una estructura de datos muy importante en los lenguajes de programación donde, representa una colección de elementos ordenados, puede contener elementos repetidos y cada elemento de la lista tiene un índice que lo ubica dentro de la misma.

Clasificaciones de Listas (simplemente enlazada, doblemente enlazada, circulares).

Listas simplemente enlazadas: Una lista simplemente enlazada pertenece a las estructuras de datos fundamentales. Suele utilizarse para implementar otras estructuras de datos. Está estructurada en una secuencia de nodos, en los que se guardan los datos y un puntero que apunta (contiene la dirección de la ubicación) al siguiente nodo.

Listas doblemente enlazada: Una lista doblemente enlazada es una lista lineal en la que cada nodo tiene dos enlaces, uno al nodo siguiente, y otro al anterior. Las listas doblemente enlazadas no necesitan un nodo especial para acceder a ellas, pueden recorrerse en ambos sentidos a partir de cualquier nodo, esto es porque a partir de cualquier nodo, siempre es posible alcanzar cualquier nodo de la lista, hasta que se llega a uno de los extremos.

Listas circulares: Una lista circular es una lista lineal en la que el último nodo a punta al primero. Las listas circulares evitan excepciones en las operaciones que se realicen sobre ellas.

Cuáles son las operaciones básicas de una lista.

Añadir un elemento a una lista doblemente enlazada vacía.
Insertar un elemento en la primera posición de la lista.
Insertar un elemento en la última posición en la lista.
Insertar un elemento a continuación de un nodo cualquiera de una lista.
Buscar o localizar elementos.
Borrado de elementos.
Eliminar único elemento de una lista doblemente enlazada
Eliminar el primer elemento de la lista doblemente enlazada.
Eliminar el último elemento de una liara doblemente enlazada.
Eliminar un elemento intermedio de una lista doblemente enlazada.

Aplicaciones de una lista.

La lista doble se puede utilizar para almacenar información como los botones de adelante y atrás del browser.
La lista sencilla se pude utilizar para hallar los elementos de una fila de personas.
La lista circular se podría aplicar a una red de datos en la cual el último elemento necesita saber cuál es el primero.

Programa ejemplo en C# de un TDA Lista.

```
namespace TDA_Lista_ejemplo
{
    class Program
    {
        static Nodo inn;
        static Nodo plo;
        static Nodo wer;
```

```
static int tpr;
static int nro;
public static int tmo = (3);
static int opp;
static string en = null;
static string er = null;
class Nodo
{
    public int d;
    public Nodo di;
}
static void Main(string[] args)
{
    Console.Title = "USANDO LISTA ENLAZADA SIMPLE";
    plo = null;
    inn = null;
    Menu();
}
public static void Menu()
{
    do
    {
        Console.WriteLine("Usando Lista enlazada simple (" + tmo +")\n");
        Console.WriteLine("================================");
        Console.WriteLine("(1) Inserta =");
        Console.WriteLine("(2) Elimina =");
        Console.WriteLine("(3) Recorre =");
        Console.WriteLine("(4) Salida =");
        Console.WriteLine("================================\n");
        Console.WriteLine("Inciso a escoger: ");
        en = Console.ReadLine();
        Console.Clear();
        switch (en)
        {
            case "1":
                do
                {
                    Console.WriteLine("Inserta\n");

                    Mos();

                    Console.WriteLine("Ingresa calificación");
                    nro = Int32.Parse(Console.ReadLine());

                    Ins();
                    Console.WriteLine("\nIngresar otra calificación: ");

                    Console.WriteLine("(1) Si");
                    Console.WriteLine("(2) No");
                    er = Console.ReadLine();
                    Console.Clear();
                } while (er.Equals("1"));
                break;
            case "2":
                do
                {
                    Console.WriteLine("Elimina\n");
```

```
                    Mos();
                    Console.WriteLine("\nEliminar una calificación: ");
                    Console.WriteLine("(1) Si");
                    Console.WriteLine("(2) No");
                    er = Console.ReadLine();
                    if (er.Equals("1"))
                    {
                        Eli();
                    }

                    Console.Clear();
                } while (er.Equals("1"));
                break;
            case "3":
                Console.WriteLine("Recorre");

                Mos();
                Console.WriteLine("\nTeclee para regresar ");

                Console.ReadKey();
                Console.Clear();
                break;
            default:
                opp = 1;
                break;
        }
    } while (opp == 0);
}
public static void Ins()
{
    if (inn == null)
    {
        inn = new Nodo
        {
            d = nro
        };
        plo = inn;
        plo.di = null;

    }
    else
    {
        wer = new Nodo
        {
            d = nro
        };

        plo.di = wer;
        plo = wer;
        plo.di = null;

    }
}
public static void Eli()
{
    if (inn == null)
    {
        tpr = inn.d;
```

```
                inn = null;
                plo = null;

            }
            else
            {
                tpr = inn.d;
                wer = inn;
                inn = inn.di;
            }
        }
        public static void Mos()
        {
            Console.WriteLine("\nTamaño (" + tmo + ")\n");
            if (nro > 0)
            {
                wer = inn;
                while (wer != null)

                {
                    Console.WriteLine("(" + wer.d + ")");
                    wer = wer.di;
                }
            }
            else
            {
                Console.WriteLine("Vacio");
            }
        }
    }
}
```

Fuentes de Información.

3.1 - Pilas

http://www.nachocabanes.com/csharp/curso2015/csharp11b.php#:~:text=Una%20pila%20nos%20permitir%C3%A1%20introducir,es%20lo%20primero%20en%20salir
https://es.wikipedia.org/wiki/Pila_(inform%C3%A1tica)
https://www.jc-mouse.net/net/c-sharp/pilas-con-c-sharp-ejercicio-resuelto

3.2 – Colas

http://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-con-estructuras-de-datos/2020/i/guia-4.pdf
http://tesciedd.blogspot.com/2011/01/unidad-2-clasificacion-de-colas.html
http://www.iuma.ulpgc.es/users/jmiranda/docencia/programacion/Tema4_ne.pdf
http://agrega.juntadeandalucia.es/repositorio/02122016/a5/es_an_2016120212_9131705/34_colas.html
http://ccc.inaoep.mx/ingreso/programacion/corto2015/Curso-PROPE-PyED-5-Pilas-Colas.pdf
https://www.ecured.cu/Cola_(Estructura_de_datos)#Colas_en_C.23

3.3 – Listas Enlazadas

https://sites.google.com/site/programacioniiuno/temario/unidad-3---estructuras-de-datos-comunes-y-colecciones/la-estructura-lista
https://analisisyprogramacionoop.blogspot.com/2017/07/lista-simplemente-enlazada-C-sharp.html
http://c.conclase.net/edd/?cap=005
http://estr-datos-omar.blogspot.com/2011/10/listas-circulares.html#:~:text=Una%20lista%20circular%20es%20una,uno%20anterior%20y%20uno%20siguiente
http://estructura-de-datos-u3.blogspot.com/2014/10/311-operaciones-basicas-con-listas.html
https://easynetstudio.wixsite.com/easynetstudio/listas