UNIDAD VII TDA Cola

2.1.1 Descripción del TDA Cola

Ahora conoceremos el concepto de Cola y se definirá como "una estructura de almacenamiento donde los datos van a ser insertados por un extremo y serán extraídos por otro". El concepto anterior se puede simplificar como **FIFO (first-in, first-out**), esto es, el primer elemento en entrar debe ser el primero en salir.

Los ejemplos sobre este mecanismo de acceso, son visibles en las filas de un banco, los clientes llegan (entran) se colocan en la fila y esperan su turno para salir

Varios hechos de la vida que podemos aplicar en el concepto de colas son, por ejemplo, cuando las personas hacen fila para esperar el uso de un teléfono público, para subir al transporte escolar o un autobús, para realizar los pagos en un supermercado, para imprimir varios documentos etc.

2.1.2 Especificación del TDA Cola

Partiendo de lo establecido en la Unidad 1. Especificación Informal tenemos lo siguiente:
Elementos que conforman la estructura

TDA Cola (**VALORES** todos los valores numéricos de cualquier tipo, **OPERACIONES** crea, vacía, primero, poner, sacar)

OPERACIONES

crear (C: Cola)
Utilidad: Sirve para inicializar la Cola
Entrada: Cola C
Salida: Ninguna
Precondición: Ninguna.
Poscondición: Cola inicializada

Vacía (c:Cola)
Utilidad: Sirve para indicar si la cola esta vacia
Entrada: Cola C
Salida: Valor booleano
Precondición: Ninguna.
Poscondición: Ninguna

Primero (C: cola)
Utilidad: Devuelve el elemento situado al frente de la cola

Entrada: C cola
Salida: Elemento
Precondición: La cola es no vacía.
Poscondición: Ninguna

Poner (ES C:Cola, E : Elemento)
Utilidad: Añade el elemento E a la cola, quedando éste situado al final de los anteriores elementos
Entrada: C Cola y Elemento E
Salida: Ninguna
Precondición: Cola con capacidad.
Poscondición: Cola modificada con un elemento más en su contenido

Sacar (ES C:Cola , ES E: Elemento)
Utilidad: Suprime el elemento situado en el frente de la cola y lo retorna
Entrada: Cola C y receptor de elemento E
Salida: ninguna, el valor del elemento retorna en el parámetro por referencia E
Precondición: Cola no vacia.
Poscondición: Cola modificada con un elemento menos en su contenido

2.1.3 Aplicaciones con Cola

En esta sección se puede apreciar que ya estamos en condiciones para plantear algoritmos usando el TDA cola, abstrayéndonos de la forma como esta implementado.

Concatenacion de dos colas

```
concatenar(Cola cola1, Cola cola2 ES Cola Cola3)
 inicio
    mientras No (cola1.vacia()= Verdadero)
    inicio
        cola1.sacar(E ) colaAux.Poner(E)
    fin
     mientras no(colaAux.vacia())
      inicio
          colaAux.Sacar(E);
          cola3.Poner(E);
          cola1.poner(E)
      fin
    mientras No (cola2.vacia()= Verdadero)
     inicio
        cola2.sacar(E ) colaAux.Poner(E)
     fin
    mientras no(colaAux.vacia())
     inicio
       colaAux.Sacar(E);
       cola3.Poner(E);
       cola2.poner(E)
    fin
 fin
```

```
Determinar si una expresión es palindrome
Espalindrome ( Cad : Cadena )
Inicio
  Paraca cada i = 1 hasta longitud de cadena
   inicio
        Letra=cad [i]
        p.meter(letra)
        c.poner(letra)
    fin
ok=verdadero
mientras  (no c.vacia ) y (ok=verdadero)
     inicio
            p.sacar(elempila)
            c.sacar(elemcola)
            si elempila<>elemcola entoces ok=falso
     fin
 Retornar ok
fin
```

2.1.4 Implementaciones del TDA Cola

En esta sección mostraremos tres implementaciones para el TDA Cola:

- **Implementación con lista**
- **Implementación con Vectores**
- **Implementación con Simulación de Memoria/Punteros**

2.1.4.1 Implementación con lista.

```
Clase Cola
Atributos
     L : Lista
Metodos
   Crear()
  booleano Vacia()
  Poner(E : TipoElemento)
  Sacar( ES E: TipoElemento)
  Tipoelemento Primero()
Fin
Constructor Cola.Crear
inicio
    // Crear Objeto L
fin
Cola.Poner( E: TipoElemento)
  Inicio
  l.inserta(l.primero(),E)
 fin
Cola.Sacar ( es E : TipoElemento)
inicio
    l.recupera(l.fin(),E)
    l.suprime(l.fin)
```

```
fin
booleano Cola.vacia()
inicio
   retornar l.vacia
fin
Tipoelemento Cola.Primero()
inicio
  l.recupera(l.fin,E)
 retornar E
fin
```

2.1.4.2 Implementación con vectores

Implementación con vectores considerando una sola vez el uso máximo de su capacidad.

Dirección de tipo Entero

Clase Cola

Atributos

V[Max] vector de tipo TipoElemento

Ini,

Fin de tipo Direccion

Metodos

```
  Crear()
  booleano Vacia()
  Poner(E : TipoElemento)
  Sacar( ES E: TipoElemento)
  Tipoelemento Primero()
```

Fin

```
cola.Crear
  inicio
      fin=0
      ini=1
  Fin
Booelano Cola.vacia()
inicio
retornar (ini>fin)
Fin
Cola.poner ( E : TipoElemento)
inicio
    si fin<MAX entonces
            fin = fin +1
            v[fin]=E
 Fin
Cola.Sacar ( ES E: TipoElemento )
Inicio
Si no Vacia() entonces
   E= V[ini]
   ini = ini +1
Fin
TipoElemento cola.primero()
```

```
Inicio
si no vacia() entonces
     retornar v[ ini ]
fin
```

Implementación con vectores desplazando cada vez que se retire un elemento de la cola

Direccion de tipo entero

Clase Cola

Atributos

V[Max] vector de tipo TipoElemento

Ini,

Fin de tipo Direccion

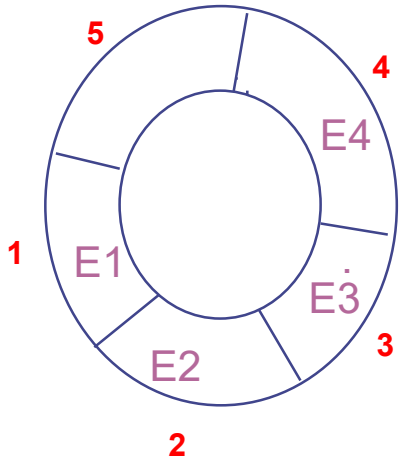
Metodos

```
  Crear()
  booleano Vacia()
  Poner(E : TipoElemento)
  Sacar( ES E: TipoElemento)
  Tipoelemento Primero()
```

Fin

```
cola.Crear
  inicio
       fin=0
       ini=1
  Fin
Booelano Cola.vacia()
inicio
retornar (ini>fin)
Fin
Cola.poner ( E : TipoElemento)
inicio
    si fin<MAX entonces
            fin = fin +1
            v[fin]=E
 Fin
Cola.Sacar ( ES E: TipoElemento )
Inicio
Si no Vacia() entonces
   E= V[1]
   desplazar(1)
   fin = fin -1
Fin
TipoElemento cola.primero()
Inicio
si no vacia() entonces
     retornar v[ 1 ]
fin
```

Implementación con vectores simulando que los extremos del vector están unidos



En esta forma de implementación se usa un vector para guardar los elementos y se pierde una casilla que sirve como frontera entre el ultimo y primer elemento de la cola.

Asi mismo se puede apreciar que es necesario tener una función que retorne la dirección siguiente por lo que se hace necesario implementarla:

```
Siguiente ( dir )
Inicio
   si dir = MAX entonces retornar 1
            caso contrario retornar dir+1
Fin
```

Si no se desea implementar la funcion siguiente se podria reemplazar esta por

dir = (dir modulo MAX) +1

Direccion de tipo entero
Clase Cola
Atributos
V[Max] vector de tipo TipoElemento
Ini,
Fin de tipo Direccion
Metodos
Privados
Direccion siguiente(dir)
publicos
Crear()
booleano Vacia()
Poner(E : TipoElemento)
Sacar(ES E: TipoElemento)
Tipoelemento Primero()
Fin
cola.Crear

```
  inicio
      fin=0
      ini=1
  Fin
Booelano Cola.vacia()
inicio
retornar (siguiente(fin)=ini)
Fin
Cola.poner ( E : TipoElemento)
inicio
  si  Siguiente(Siguiente(Fin))<>ini entonces
          fin = siguiente(fin)
           v[fin]=E
Fin

Cola.Sacar ( ES E: TipoElemento )
Inicio
   Si no Vacia() entonces
      E= V[ini]
      ini=siguiente(ini)
 Fin

TipoElemento cola.primero()
Inicio
si no vacia() entonces
     retornar v[ ini ]
fin
```

2.1.4.3 Implementación con simulación de Memoria / Punteros

```
Nodo
          elemento TipoElemento
          sig      Puntero a Nodo
// fin definicion

 Direccion Puntero a espacio de memoria de tipo Nodo
 Clase cola
          Atributos
              Ini,fin  Puntero de tipo Direccion
          Metodos
             Crear()
            booleano Vacia()
            Poner(E : TipoElemento)
            Sacar( ES E: TipoElemento)
            Tipoelemento Primero()
          Fin
Constructor crear
Inicio
   ini=-1
```

```
  fin=-1
fin
Booelano Cola.vacia()
inicio
retornar (ini=nulo)
Fin

cola.poner( E : TipoElemento)
inicio
      aux = // Pedir Espacio de memoria para Nodo
      si aux <> nulo entoces
                  ponerdato(aux,'->elemento',E);
                  Ponerdato(aux, '->sig',nulo)
               si vacia() entoces
                        ini=aux
                        fin=aux
                        caso contrario
                         ponerdato(fin, '->sig',aux)
                         fin = Aux
            caso contrario // Error
fin
TipoElemento cola.primero()
   inicio
     Si(vacia()) Entoces  // Error
      caso contrario
         return obtenerdato(ini,'->elemento')
fin
cola.sacar(ES E:TipoElemento)
inicio
    si no vacia entonces
            E=obtenerdato(ini,'->elemento')
            X= ini
            Ini=obtenerdato(ini,'->sig')
            Delete_espacio(x)
fin
```

2.2.1 DICOLAS

Las *Dicolas* son casos particulares de listas y generalizaciones de colas en donde las eliminaciones e inserciones pueden realizarse en ambos extremos de la lista.

El conjunto de operaciones de una dicola consiste en las operaciones de una cola junto con las siguientes:

```
publico Elemento ultimo();
public void poner_frente(Objecto elemento);
public void sacar_final(Objeto Elemento)
```

2.2.1.1 Especificación del TDA DiCola

Partiendo de lo establecido en la Unidad 1. Especificación Informal tenemos lo siguiente:
Elementos que conforman la estructura

TDA DiCola (**VALORES** todos los valores numéricos de cualquier tipo, **OPERACIONES** crea, vacía, primero, poner, sacar,poner_frente,sacar_final,ultimo)

OPERACIONES

crear (D: DiCola)
Utilidad: Sirve para inicializar la DiCola
Entrada: DiCola D
Salida: Ninguna
Precondición: Ninguna.
Poscondición: DiCola inicializada

Vacía (D:DiCola)
Utilidad: Sirve para indicar si la Dicola esta vacia
Entrada: Dicola D
Salida: Valor booleano
Precondición: Ninguna.
Poscondición: Ninguna

Primero (D: Dicola)
Utilidad: Devuelve el elemento situado al frente de la Dicola
Entrada: D Dicola
Salida: Elemento
Precondición: La Dicola es no vacía.
Poscondición: Ninguna

Poner (ES D:DiCola, E : Elemento)
Utilidad: Añade el elemento E a la Dicola, quedando éste situado al final de los anteriores elementos
Entrada: D DiCola y Elemento E
Salida: Ninguna
Precondición: DiCola con capacidad.
Poscondición: DiCola modificada con un elemento más en su contenido

Sacar (ES C:DiCola , ES E: Elemento)
Utilidad: Suprime el elemento situado en el frente de la cola y lo retorna
Entrada: Cola C y receptor de elemento E
Salida: ninguna, el valor del elemento retorna en el parámetro por referencia E
Precondición: Cola no vacia.
Poscondición: Cola modificada con un elemento menos en su contenido

Poner_frente (ES D:DiCola, E : Elemento)
Utilidad: Añade el elemento E a la Dicola, quedando éste situado al inicio de los anteriores elementos
Entrada: D DiCola y Elemento E
Salida: Ninguna
Precondición: DiCola con capacidad.
Poscondición: DiCola modificada con un elemento más en su contenido

Sacar_Final(ES D:DiCola , ES E: Elemento)
Utilidad: Suprime el elemento situado en el final de la cola y lo retorna
Entrada: Di Cola D y receptor de elemento E

Salida: ninguna, el valor del elemento retorna en el parámetro por referencia E
Precondición: DiCola no vacia.
Poscondición: DiCola modificada con un elemento menos en su contenido

Ultimo (D: Dicola)
Utilidad: Devuelve el elemento situado al final de la Dicola
Entrada: D Dicola
Salida: Elemento
Precondición: La Dicola es no vacía.
Poscondición: Ninguna

Implementar el TDA Dicola tomando en cuenta todas las opciones planteadas para las colas vale decir, Dicola con Vectores v1. Dicola con Vectores V2. Dicola con vectores siumlando estar unido por los extremos, Dicola con Simulacion de Memoria, Dicola con Lista y Finalmente Dicola con Punteros.

2.3.1 Cola de Prioridades

Una cola de prioridad es una cola cuyo comportamiento esta en función de la prioridad que tienen los elementos para su atención en función de la asignación de prioridad, de forma que el orden en que los elementos son procesados sigue las siguientes reglas:

- **El elemento con mayor prioridad es procesado primero.**
- **Dos elementos con la misma prioridad son procesados según el orden en que fueron introducidos en la cola**

TDA ColadePrioridad (**VALORES** todos los valores numéricos de cualquier tipo, **OPERACIONES** crea, vacía, primero, poner, sacar, Asignar_Frecuencia_prioridad)

OPERACIONES
crear (CP: ColaPrioridad)
Utilidad: Sirve para inicializar la Colaprioridad
Entrada: Colaprioridad CP
Salida: Ninguna
Precondición: Ninguna.
Poscondición: Colaprioridad inicializada

Vacía (CP:ColaPrioridad)
Utilidad: Sirve para indicar si la ColaPrioridad esta vacia
Entrada: ColaPrioridad CP
Salida: Valor booleano
Precondición: Ninguna.
Poscondición: Ninguna

Primero (CP: ColaPrioridad)
Utilidad: Devuelve el elemento situado al frente de la ColaPrioridad
Entrada: CP ColaPrioridad
Salida: Elemento
Precondición: La ColaPrioridad es no vacía.
Poscondición: Ninguna

Poner (ES CP :Colaprioridad, E : Elemento, Pri: prioridad)

Utilidad: Añade el elemento E a la ColaPrioridad, quedando éste situado al final de los anteriores elementos en la prioridad Pri.
Entrada: CP ColaPri , Elemento E y Prioridad Pri
Salida: Ninguna
Precondición: ColaPrioridad con capacidad.
Poscondición: ColaPrioridad modificada con un elemento más en su contenido

Sacar (ES CP:ColaPrioridad, ES E: Elemento)
Utilidad: Suprime el elemento situado en el frente de la colaprioridad considireando la prioridad y frecuencia y lo retorna
Entrada: CP Colaprioridad y receptor de elemento E
Salida: ninguna, el valor del elemento retorna en el parámetro por referencia E
Precondición: Colaprioridad no vacia.
Poscondición: ColaPrioridad modificada con un elemento menos en su contenido

Asignar_frecuencia_prioridad (ES CP:ColaPrioridad, frec , pri :Entero)
Utilidad: Establece con que frecuencia Fre serán atendidos los elementos de la cola con prioridad Pri
Entrada: CP Colaprioridad, Frec Frecuencia y Pri que establece la cola a la que pertence la frecuencia
Salida: ninguna,
Precondición: Ninguna.
Poscondición: ColaPrioridad modificada estableciendo su comportamiento a posteriori.

2.3.2 Implementación de la ColadePrioridad

Clase ColaPri
Atributos
VC arreglo(MAX) Ccola
VF arreglo[MAX] numero
colaAct numero
cant numero
Metodos
Crear()
vacia();
primero();
poner(elemento E, entero prioridad)
Asignar_frecuencia_Prioridad (entero frec, entero prioridad)
sacar(ES Elemento E);
Fin

```
Constructor ColaPri.Crear
 inicio
  // Crear Objetos vc[1], vc[2], vc[3] ..
  fin
ColaPri.Poner( E: Elemento , pri :numero)
inicio
    vc[pri].poner(E);
Fin
ColaPri.Asignar_Frecuencia_prioridad( frec,prio)
inicio
 vf[prio]= frec
Fin
```

ColaPri.Sacar(ES E: Elemento)

```
inicio
si no vc[ColaAct].vacia()
      entoces
       si cant < vf[colaAct]
          cant = cant +1
          vc[ColaAct].sacar(e)
          si cant=vf[colaAct]
            entoces
              colaAct = colaAct +1
              cant = 0
          //retornar e
Fin
```