Tipo Abstracto de Datos Cola
T.A.D. COLA

SISTEMAS DE DATOS
Licenciatura en Sistemas de Información

COLAS

- Las Colas son secuencias de elementos que pueden crecer y contraerse siguiendo la política : Primero en Entrar, Primero en Salir

- Entre los elementos existe un *orden temporal*

- Se las suele llamar también Listas FIFO- First In First Out- , o Listas "primero en entrar primero en salir", o Queue.

T.A.D. Cola – Especificación (1)

Cola: Secuencia de cero o mas elementos de un tipo determinado que obedece a la política FIFO.

C = (a1, a2, ..., an) , n>=0

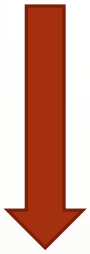
El orden temporal de inserciones en este conjunto determina completamente el orden en el cual los elementos serán recuperados.

Si n>0:

- a1: primer elemento ingresado, próximo elemento a ser retirado.
- an: último elemento ingresado.

T.A.D. Cola – Especificación(2)

C = (a1, a2, ..., an) , n>=0

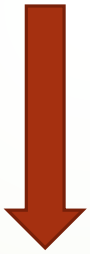


Insertar(C,X)

C = (a1, a2, ..., an, X) , n>0

T.A.D. Cola – Especificación(3)

C = (a1, a2, ..., an) , n>0



Suprimir(C,X)

C = (a2, ..., an) n>=0, X=a1

T.A.D. Cola – Especificación(4)

Operaciones Abstractas

Sean **C**: Cola y **X**: elemento

NOMBRE	***ENCABEZADO***	***FUNCION***	***ENTRADA***	***SALIDA***
Insertar	Insertar (C,X)	Ingresa el elemento X en la cola C	C y X	$C=(a_1,a_2,...,a_n,X)$
Suprimir	Suprimir(C,X)	Si C no está vacía, elimina el elemento que mas tiempo ha permanecido en la cola	C	si n>0: $C=(a_2,..., a_n)$ y $X=a_1$; Error en caso contrario
Recorrer	Recorrer(C)	Procesa todos los elementos de C siguiendo la política FIFO	C	Está sujeta al proceso que se realice sobre los elementos de C
Crear	Crear(C)	Inicializa C	C	C=( )
Vacía	Vacía(C)	Evalúa si C tiene elementos	C	Verdadero si C No tiene elementos, Falso en caso contrario.

T.A.D. Cola – Aplicación (1)

Realizar la simulación informática correspondiente al comportamiento de una cola de clientes que esperan efectuar una transacción en un cajero automático.

El objetivo de esta simulación es conocer el tiempo promedio de espera de los clientes en la cola.

Se construye una simulación regida por el tiempo, monitoreando minuto a minuto lo que ocurre en el sistema.

T.A.D. Cola – Aplicación (2)

¿Qué eventos pueden ocurrir en un minuto particular?

- Puede o no llegar un cliente a la cola.
- Si el servidor está libre, debe comenzar a atender a un nuevo cliente.
- Los clientes que aun están en la cola, esperan un minuto mas.

T.A.D. Cola– Aplicación (3)

¿Cuáles son los componentes relevantes del sistema?

¿Cómo representar cada uno de los componentes?

Cajero (servidor). → Variable booleana? →Contador.

Cola de clientes. → Cola de tiempos

T.A.D. Cola – Aplicación (4)

Ingresar : Frecuencia de llegada de Clientes (x),
Tiempo de atención de Cajero (y) ,Tiempo máximo de Simulación (TMS).
Inicializar Reloj
MIENTRAS (Reloj no alcance el TMS) **HACER**
SI (llega Cliente)
1 / x
ENTONCES Insertar(Cola, Cliente)
FIN SI
SI (Cajero libre)
ENTONCES
SI (No (Vacía(Cola)))
ENTONCES
Suprimir (Cola, Cliente)
Computar el Tiempo de espera del Cliente en la Cola
Acumular Tiempo de espera del Cliente
Contar cliente atendido
Inicializar Cajero Ocupado
Cajero ocupado←y
FIN SI
SINO (Cajero Ocupado)
Actualizar Cajero
FIN SI

Actualizar tiempo de espera de clientes en cola
Actualizar Reloj

FIN MIENTRAS
Tiempo Promedio de espera = $\frac{\text{Tiempo de espera Acumulado}}{\text{Cantidad de Clientes Atendidos}}$
Reportar (Tiempo Promedio de espera)

T.A.D. Cola – Aplicación(5)

- Llega cliente: número aleatorio (random) entre 0 y 1/ Frecuencia de llegada de un cliente
- Cuando el cliente llega, ingresa a la Cola (Reloj o 0)
- Cajero ocupado/ libre: tiempo atención cajero ***y*** minutos

  Cajero = 0 representa Cajero libre

  Cajero > 0 representa Cajero Ocupado
- Inicializar Cajero Ocupado: 0 o Tiempo de atención de cajero
- Para controlar el Tiempo máximo de Simulación

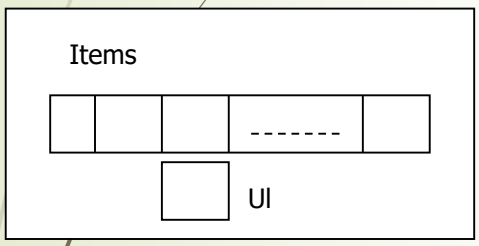
  Reloj = 0 Inicio simulación

  Reloj = Tiempo Máximo Simulación Fin Simulación

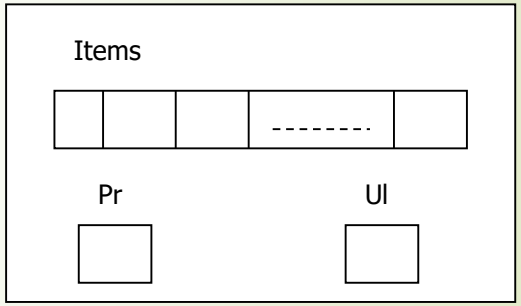
T.A.D. Cola – Representación (1)

Representación secuencial

C = (a1, a2, ..., an)

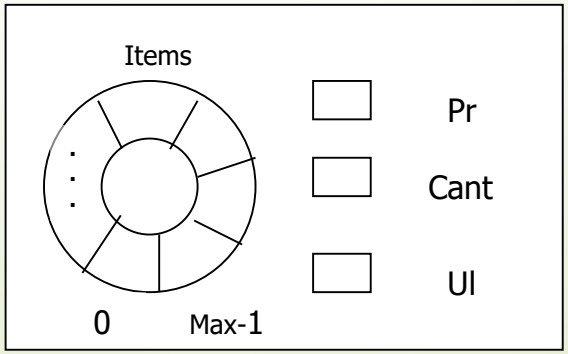


a)



b)

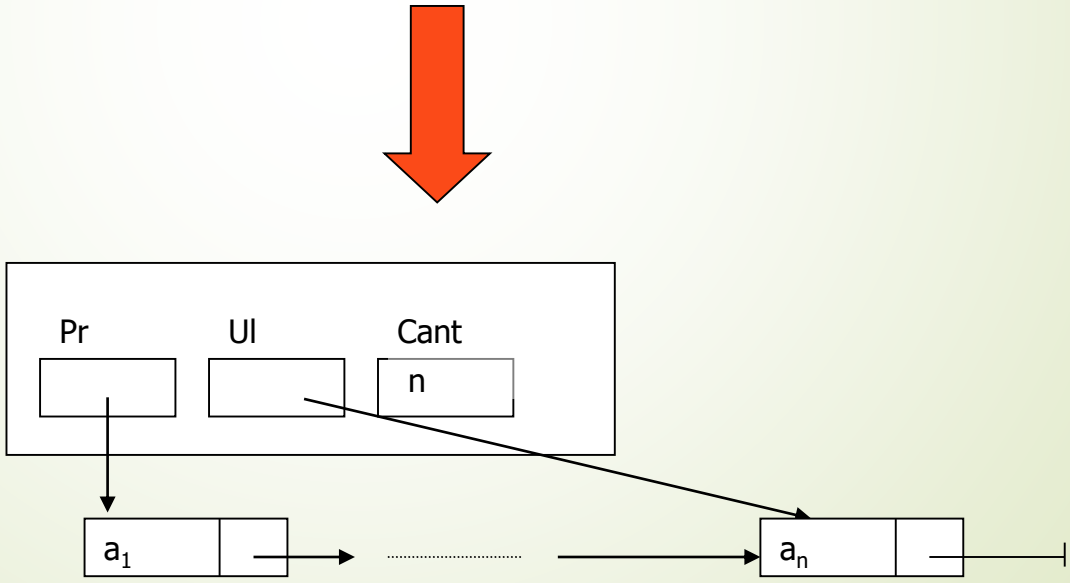
c)



T.A.D. Cola – Representación (2)

Representación encadenada:

C = (a1, a2, ..., an)



T.A.D. Cola –
Construcción de operaciones abstractas (1)

REPRESENTACIÓN SECUENCIAL

```
class cola
{   int *items;
    int pr;
    int ul;
    int cant;
    int max;
    public:
    cola(int xmax=0):
    max(xmax)
    {  pr=0;
       ul=0;
       cant=0;
       items=new int[max];
     }
    int vacia(void)
    {   return (cant==0);    }
    int insertar(int x)
   {   if (cant<max)
          {     items[ul]=x;
                ul=(ul+1)%max;
                cant++;
                return (x);
     }
     else return (0);
   }
```

```
   int suprimir(void)
   { int x;
     if (vacia())
        {    printf("%s","Pila vacia");
             return(0);
         }
        else
       {      x=items[pr];
              pr=(pr+1)%max;
              cant--;
              return(x);
       }
     }
  void recorrer(void)
   { int i,j;
     if (!vacia())
     {   i=pr;
         j=0;
         for (i; j<cant ; i=(i+1)%max,j++)
           {    cout<<items[i]<<endl;
            }
      }
     }
};
```

T.A.D. Cola –
Construcción de operaciones abstractas (2)

REPRESENTACIÓN ENCADENADA

```
class celda
{
  int item;
  celda *sig;
  public:
  int obteneritem(void)
  {
    return(item);
  }
  void cargaritem(int xitem)
  {
    item=xitem;
  }
  void cargarsig(celda* xtope)
  {
    sig=xtope;
  }
  celda* obtenersig(void)
  {
    return(sig);
  }
};
```

T.A.D. Cola –
Construcción de operaciones abstractas (3)

REPRESENTACIÓN ENCADENADA

```
class cola
{  int cant;
   celda *pr;
   celda *ul;
   public:
   cola(celda* xpr=NULL,celda *xul=NULL,int
xcant=0):
   pr(xpr),ul(xul),cant(xcant)
   { }
   int vacia(void)
   {   return (cant==0);
    }
   int insertar(int x)
  {   celda *ps1;
      ps1=new(celda);
      ps1->cargaritem(x);
      ps1->cargarsig(NULL);
      if (ul==NULL)
       {pr=ps1;}
        else
       {ul->cargarsig(ps1);}
        ul=ps1;
        cant++;
        return(ul->obteneritem());
   }
```

```
 int suprimir(void)
  { celda *aux;
  int x;
  if (vacia())
    {    printf("%s","Cola vacia");
         return(0);
    }
  else
  {   aux=pr;
      x=pr->obteneritem();
      pr=pr->obtenersig();
      cant--;
       free(aux);
       return(x);
    }
   }
  celda* recuperapr(void)
  {
   return(pr);
  }
  void recorrer(celda *aux)
  {
    if (aux!=NULL)
    {  cout<<aux->obteneritem()<<endl;
       recorrer(aux->obtenersig());
    }
  }
};
```