

Tema: TDA Cola

Apellido y Nombre: Fecha:...../...../.....

EJEMPLOS

Ejemplo 1 – Variante de implementación: Dada la siguiente definición del TDA cola (que prioriza velocidad de proceso), modifique las operaciones del TDA cola de modo que se adapten a la implementación propuesta.

<p>CONSTANTES</p> <p>MAXC=10</p> <p>MAXI=2</p> <p>TIPOS</p> <p>contenedor=ARREGLO [1..MAXC] de CARACTERES</p> <p>indicadores=ARREGLO [1..MAXI] de ENTEROS</p> <p>tcola=REGISTRO</p> <p> datos: contenedor</p> <p> indice: indicadores</p> <p>FIN_REGISTRO</p>	<pre>const int MAXC=10; const int MAXI=2; typedef char contenedor[MAXC]; typedef int indicadores[MAXI]; typedef struct tcola{ contenedor datos; indicadores indice; };</pre>
---	--

Adaptación de las operaciones del TDA cola a la variante de implementación propuesta

En la variante de implementación propuesta los indicadores de la cola se almacenan en un arreglo de 2 posiciones. En este caso, la primera posición se destina al indicador *frente* y la segunda al indicador *final*. Teniendo en cuenta esto, la operación *iniciar_cola* asigna los valores de inicialización a los elementos del campo *índice*.

<p>PROCEDIMIENTO iniciar_cola(E/S q:tcola)</p> <p>INICIO</p> <p> q.indice[1] ← MAXC // frente</p> <p> q.indice[2] ← MAXC // final</p> <p>FIN</p>	<pre>void iniciar_cola(tcola &q) { q.indice[0]=MAXC-1; // frente q.indice[1]=MAXC-1; // final }</pre>
---	---

La operación *cola_vacia* comprueba si la cola está vacía o no. En este caso, la comprobación se realiza mediante la comparación de los indicadores *frente* y *final* cuyos valores se encuentran almacenados en el arreglo *índice* (campo del registro *tcola*). Si *frente* y *final* son iguales entonces la cola está vacía.

<p>FUNCIÓN cola_vacia(E q:tcola):LOGICO</p> <p>INICIO</p> <p> cola_vacia ← q.indice[1]=q.indice[2]</p> <p>FIN</p>	<pre>bool cola_vacia(tcola q) { return q.indice[0]==q.indice[1]; }</pre>
---	--

La operación *cola_llena* comprueba si la cola está llena o no. En este caso, la comprobación se realiza mediante la comparación de los indicadores *frente* y *final* cuyos valores se encuentran almacenados en el arreglo *índice* (campo del registro *tcola*). Si la próxima posición a la que debe apuntar *final* coincide con *frente* entonces la cola está llena.

<p>FUNCIÓN cola_llena(E q:tcola):LOGICO</p> <p>INICIO</p> <p> cola_llena ← q.indice[1]=sig(q.indice[2])</p> <p>FIN</p>	<pre>bool cola_llena(tcola q) { return q.indice[0]==sig(q.indice[1]); }</pre>
--	---

La operación *sig* calcula, a partir del valor actual de un índice, la siguiente posición a la que se hará referencia teniendo en cuenta un almacenamiento circular. Nótese que al alcanzar la última posición del arreglo, la siguiente posición es la primera.

<pre> FUNCIÓN sig(E n:ENTERO):ENTERO INICIO SI (n = MAX) ENTONCES n←1 SINO n←n+1 FIN SI sig←n FIN </pre>	<pre> int sig(int n) { if (n == MAX-1) n=0; else n++; return n; } </pre>
--	--

La operación *agregarCola* permite añadir un nuevo elemento a la cola siempre que exista espacio. Para adaptar esta operación a la implementación propuesta simplemente se utilizó *q.indice[segunda_posición]* como indicador *final* de la cola en las instrucciones que hacen referencia a él.

<pre> PROCEDIMIENTO agregarCola(E/S q:tcola,E nuevo:carácter) INICIO SI (cola_llena(q) = V) ENTONCES ESCRIBIR "COLA LLENA" SINO q.indice[2]←sig(q.indice[2]) q.datos[q.indice[2]]←nuevo FIN SI FIN </pre>	<pre> void agregarCola(tcola &q, char nuevo) { if (cola_llena(q)==true) cout << "COLA LLENA" << endl; else { q.indice[1]=sig(q.indice[1]); q.datos[q.indice[1]]=nuevo; } } </pre>
---	---

La operación *quitarCola* permite extraer un elemento de la cola siempre que no esté vacía. Para adaptar esta operación a la implementación propuesta simplemente se utilizó *q.indice[primera_posición]* como indicador *frente* de la cola en las instrucciones que hacen referencia a él.

<pre> FUNCIÓN quitarCola(E/S q:tcola):CARACTER VARIABLES extraido:CARACTER INICIO SI (cola_vacia(q) = V) ENTONCES extraido←' ' SINO q.indice[1]←sig(q.indice[1]) extraido←q.datos[q.indice[1]] FIN SI quitarCola←extraido FIN </pre>	<pre> char quitarCola(tcola &q) {char extraido; if (cola_vacia(q)==true) extraido=' '; else { q.indice[0]=sig(q.indice[0]); extraido=q.datos[q.indice[0]]; } return extraido; } </pre>
--	--

La operación *primero* permite consultar el elemento de la cola será el próximo en salir. Para adaptar esta operación a la implementación propuesta simplemente se utilizó *q.indice[primera_posición]* como indicador *frente* de la cola en las instrucciones que hacen referencia a él. Nótese que *frente* no indica el primer elemento de la cola sino uno que ya se extrajo (según se definió en la clase de teoría), por esta razón se utiliza la función *sig* para apuntar al primer elemento válido de la cola.

<pre> FUNCIÓN primero(E/S q:tcola):CARACTER VARIABLES pri:CARACTER INICIO SI (cola_vacia(q) = V) ENTONCES pri←' ' SINO pri←q.datos[sig(q.indice[1])] FIN SI primero←pri FIN </pre>	<pre> char primero(tcola &q) {char pri; if (cola_vacia(q)==true) pri=' '; else pri=q.datos[sig(q.indice[0])]; return pri; } </pre>
--	--

La operación *último* permite consultar el último elemento agregado a la cola. Para adaptar esta operación a la implementación propuesta simplemente se utilizó *q.indice[segunda_posición]* como indicador *final* de la cola en las instrucciones que hacen referencia a él. El indicador *final* apunta al último elemento de la cola, por lo que no es necesario hacer ningún ajuste.

```

FUNCIÓN ultimo(E/S q:tcola):CARACTER
VARIABLES
    ulti:CARACTER
INICIO
    SI (cola_vacia(q) = V) ENTONCES
        ulti ← ' '
    SINO
        ulti ← q.datos[q.indice[2]]
    FIN_SI
    ultimo ← ulti
FIN

```

```

char ultimo(tcola &q)
{char ulti;
  if (cola_vacia(q)==true)
    ulti=' ';
  else
    ulti=q.datos[q.indice[1]];
  return ulti;
}

```

Ejemplo 2 – Aplicación del TDA cola: Sabiendo que el cociente entre 2 valores enteros puede calcularse mediante restas sucesivas, desarrolle un algoritmo que aplique el TDA cola y sus operaciones básicas para resolver el problema propuesto. Indique además la definición de las estructuras de datos utilizadas en la solución planteada.

Solución propuesta

A fin de resolver el problema planteado (calcular el cociente entre 2 valores enteros) se propone diseñar una función entera que, utilizando el concepto de pila y restas sucesivas, calcule el cociente de una división entera.

```

FUNCIÓN cociente(a:ENTERO, b:ENTERO):ENTERO
VARIABLES
    coc:ENTERO
    q:tcola
INICIO
    iniciarCola(q)
    MIENTRAS (a >= b) ENTONCES
        agregarCola(q,1)
        a ← a-b
    FIN_MIENTRAS
    coc ← 0
    MIENTRAS (cola_vacia(q) = F) HACER
        coc ← -coc + quitarCola(q)
    FIN_MIENTRAS
    cociente ← coc
FIN

```

```

int cociente(int a, int b)
{int coc;
  tcola q;
  iniciarCola(q);
  while (a >= b)
  { agregarCola(q,1);
    a=a-b; }
  coc=0;
  while (cola_vacia(q)==false)
    coc=coc + quitarCola(q);
  return coc;
}

```

El primer bucle *MIENTRAS* ejecuta la resta sucesiva del dividendo (*a*) respecto del divisor (*b*), almacenando un valor 1 en la cola por cada resta realizada. El segundo bucle extrae los valores almacenados en la cola acumulándolos en la variable *coc*, la que finalmente se asigna a la función cociente.

Respecto a la **definición** del *TDA cola*, a continuación se indican 3 alternativas (existen muchas más) que podrían utilizarse en este problema. Es importante destacar que, más allá de la implementación utilizada, el algoritmo anterior hace referencia de **forma general** al TDA cola y sus operaciones por lo que emplear una u otra resulta indistinto.

Alternativa 1

```

CONSTANTES
    MAX=10
TIPOS
tcontenedor=ARREGLO [1..MAX] de ENTEROS
tcola=REGISTRO
    datos:tcontenedor
    frente,final:ENTERO
FIN_REGISTRO

```

Alternativa 1

```

const int MAX=10;
typedef int tcontenedor[MAX];
typedef struct tcola{
    tcontenedor datos;
    int frente,final;
};

```

Alternativa 2

```

CONSTANTES
    MAX=12
TIPOS
tcola=ARREGLO [1..MAX] de ENTEROS

```

Alternativa 2

```

const int MAX=12;
typedef int tcola[MAX];

```

Alternativa 3

```

CONSTANTES
    MAXC=10
    MAXI=3

```

Alternativa 3

```

const int MAXC=10;
const int MAXI=3;

```

```

TIPOS
  contenedor=ARREGLO [1..MAXC] de ENTEROS
  indicadores=ARREGLO [1..MAXI] de ENTEROS
  tcola=REGISTRO
      datos: contenedor
      indice: indicadores
  FIN_REGISTRO

typedef int contenedor[MAXC];
typedef int indicadores[MAXI];

typedef struct tcola{
    contenedor datos;
    indicadores indice;
};

```

Ejemplo 3 - Implementación del TDA bicola: Utilizando listas simples, con punteros de *inicio* y *final*, implemente el TDA bicola (con salida restringida, que almacene caracteres) y las operaciones *iniciar_bicola*, *agregar_bicola*, *quitar_bicola*, *primero* y *último*.

Definición de la estructura

Para implementar el TDA bicola utilizando listas simples es necesario definir los nodos que formarán la bicola y los indicadores que permitirán manejarla. En este caso, se optó por hacer corresponder *frente* con el *inicio* de la lista y *final* con el *final* de la lista, aunque podría haberse planteado al revés sin ningún problema (extraer datos del final (*frente* de la bicola) y agregar datos al inicio (*final* de la bicola)).

```

typedef pnode *tnodo;
typedef struct tnodo {
    char dato;
    pnode sig;
};
typedef struct tbicola{
    pnode frente; // inicio de la lista
    pnode final; // final de la lista
};

```

La operación *iniciar_bicola* es simplemente la operación básica *iniciar_lista*. En esta operación los indicadores *frente* y *final* de la bicola se inicializan en NULL, lo que genera una bicola vacía.

```

void iniciar_bicola(tbicola &bq)
{
    bq.frente=NULL;
    bq.final=NULL;
}

```

La operación *agregar_bicola*, que debe trabajar sobre ambos extremos de la estructura, se construye combinando las operaciones básicas *agregar_inicio* y *agregar_final*. El extremo sobre el que se realizará el agregado se determina en función de la variable lógica *primero* (*true* para agregar por el frente, *false* para agregar por el final).

```

void agregar_bicola(tbicola &bq, pnode nuevo, bool primero)
{
    if (bq.frente==NULL)
    { bq.frente=nuevo;
      bq.final=nuevo; }
    else
    {
        if (primero==true)
        { nuevo->sig=bq.frente;
          bq.frente=nuevo; }
        else
        { bq.final->sig=nuevo;
          bq.final=nuevo; }
    }
}

```

Bicola vacía

Agregar inicio

Agregar final

La operación *quitar_bicola*, que sólo opera con el frente de la bicola, es simplemente la operación básica *quitar_inicio*. Esta operación contempla 3 posibles casos: 1) una bicola vacía, 2) una bicola con un único elemento (se modifican los punteros *frente* y *final*) y 3) una bicola con 2 o más elementos (se modifica únicamente el puntero *frente*).

```

pnode quitar_bicola(tbicola &bq)
{ pnode extraido;
  if (bq.frente==NULL) } Bicola vacía
    extraido=NULL;
  else
    if (bq.frente==bq.final) } Único elemento
      { extraido=bq.frente;
        bq.frente=NULL;
        bq.final=NULL;
      }
    else } 2 o más elementos
      { extraido=bq.frente;
        bq.frente=extraido->sig;
        extraido->sig=NULL;
      }
  return extraido;
}

```

La operación *primero* devuelve la dirección del primer elemento de la bicola (referido por el puntero *frente*) mientras que la operación *ultimo* devuelve la dirección del último elemento de la bicola (referido por el puntero *final*). Nótese que si la bicola está vacía, ambas operaciones retornan nulo.

```

pnode primero(tbicola bq)
{
  return bq.frente;
}

```

```

pnode ultimo(tbicola bq)
{
  return bq.final;
}

```

EJERCICIOS

- 1) De acuerdo a la definición del *TDA cola*, implemente el TDA y sus operaciones fundamentales, considerando:
 - *TDA cola* requiere un contenedor de datos e indicadores del primer y último elemento de la cola.
 - Una operación de inicialización que permita crear (inicializar) una cola vacía.
 - Una operación de inserción que permita agregar un nuevo elemento a la cola (siempre como último elemento).
 - Una operación que determine si el contenedor de datos está completo.
 - Una operación que extraiga elementos de la cola (siempre el elemento que está al principio de la cola).
 - Una operación que determine si la cola no contiene elementos (cola vacía).
 - Una operación que permita consultar el elemento que está al principio (frente) de la cola.
 - Una operación que permita consultar el elemento que está al final de la cola.
 - Una operación que permita consultar la cantidad de elementos almacenados en la cola.

Suponga que la implementación corresponde a una cola de números reales, realizándose en 2 variantes:

- a) implementación *TDA cola* que priorice velocidad de procesamiento
 - b) implementación *TDA cola* que priorice espacio de almacenamiento
- 2) Modifique la implementación del TDA cola que *prioriza espacio de almacenamiento* considerando que ÚNICAMENTE se cuenta con un arreglo de 15 posiciones para construir el TDA. Utilice las posiciones extremas del arreglo para almacenar los indicadores de la cola.

- 3) Modifique la definición de la estructura y operaciones del TDA cola que *prioriza velocidad de proceso* de modo que el almacenamiento y recuperación de datos se realice recorriendo el arreglo forma inversa (desde las últimas posiciones hacia las primeras). Considere que la función *siguiente* se renombra como *anterior*.
- 4) Suponiendo que la definición del tamaño de arreglos está restringida a 7 elementos, modifique la implementación (definición de la estructura y operaciones) que *prioriza velocidad de proceso* de modo que se pueda construir colas de 14 elementos. ¿Cómo se modificaría esta implementación si las primeras posiciones del segundo arreglo se utilizan para almacenar los indicadores de la cola? Explique brevemente.
- 5) Dadas las siguientes definiciones del *TDA cola* implemente las operaciones básicas *iniciar cola*, *quitar cola*, *siguiente* y *cola_llena* para cada variante

Cola que prioriza velocidad de proceso

CONSTANTES

MAX=12

TIPOS

tcola=ARREGLO [1..MAX] de ENTEROS

Considere que los indicadores ocupan las posiciones finales del arreglo.

Cola que prioriza velocidad de proceso

CONSTANTES

MAX=10

TIPOS

tcontenedor=ARREGLO [1..MAX] de CARACTERES

tindicadores=REGISTRO

ind1:ENTERO

ind2:ENTERO

FIN_REGISTRO

tcola=REGISTRO

datos:tcontenedor

indicador:tindicadores

FIN_REGISTRO

Cola que prioriza espacio de almacenamiento

CONSTANTES

MAX=5

TIPOS

tcontenedor=ARREGLO [1..MAX] de ENTEROS

tindicadores=ARREGLO [1..3] de ENTEROS

tcola=REGISTRO

datos1:tcontenedor

datos2:tcontenedor

indice:tindicadores

FIN_REGISTRO

Cola que prioriza espacio de almacenamiento

CONSTANTES

MAX=6

TIPOS

tcontenedor=ARREGLO [1..MAX] de ENTEROS

tcola=REGISTRO

datos1:tcontenedor

datos2:tcontenedor

datos3:tcontenedor

FIN_REGISTRO

Considere que los indicadores ocupan las posiciones iniciales del tercer arreglo.

- 6) Utilizando listas enlazadas implemente un *TDA cola* de reales y sus operaciones. Para llevar a cabo la implementación considere las siguientes variantes:
 - a) listas simples (queda a libre elección el uso de 1 o 2 punteros a la lista)
 - la operación de **inserción** se realiza por el **final** de la lista mientras que la **eliminación** se realizan por el **principio** de la lista. ¿Qué extremo funciona como frente de la cola? ¿Qué extremo funciona como final de la cola?
 - la operación de **inserción** se realiza por el **principio** de la lista mientras que la **eliminación** se realizan por el **final** de la lista. ¿Qué extremo funciona como frente de la cola? ¿Qué extremo funciona como final de la cola?
 - b) listas dobles (queda a libre elección el uso de 1 o 2 punteros a la lista)
 - la operación de **inserción** se realiza por el **final** de la lista mientras que la **eliminación** se realizan por el **principio** de la lista. ¿Qué extremo funciona como frente de la cola? ¿Qué extremo funciona como final de la cola?
 - la operación de **inserción** se realiza por el **principio** de la lista mientras que la **eliminación** se realizan por el **final** de la lista. ¿Qué extremo funciona como frente de la cola? ¿Qué extremo funciona como final de la cola?
- 7) Aplicando el *TDA cola* y sus operaciones básicas, desarrolle las estructuras y algoritmos que permitan convertir una cadena ingresada por el usuario a mayúsculas/minúsculas según elección de éste. Por ejemplo:

Cadena ingresada: Analista Programador 2023
 Selección de conversión M (mayúsculas) o m (minúsculas): M
 Cadena en mayúsculas: ANALISTA PROGRAMADOR 2023

- 8) Desarrolle un algoritmo que aplique el **TDA cola** y sus operaciones básicas para convertir una cadena de caracteres que almacena dígitos de un valor decimal a su correspondiente valor numérico. Para ello, tenga en cuenta que un valor fraccionario puede expresarse mediante la siguiente ecuación:

$$0,5327 = 5 \times 10^{-1} + 3 \times 10^{-2} + 2 \times 10^{-3} + 7 \times 10^{-4} \quad (\text{note que los exponentes son negativos})$$

Indique la **definición de las estructuras** de datos utilizadas en la solución planteada.

Nota: SUPONGA QUE LOS VALORES A CONVERTIR SIEMPRE TIENEN PARTE ENTERA CERO.

- 9) Aplicando el TDA cola y sus operaciones básicas desarrolle los algoritmos que permitan:

a) Calcular el inverso de un número

$$1 = 1^3$$

b) Calcular el cubo de un número N mediante la suma de N valores impares.

$$3 + 5 = 2^3$$

$$7 + 9 + 11 = 3^3$$

c) Calcular un término de la serie de Fibonacci.

$$13 + 15 + 17 + 19 = 4^3$$

$$21 + 23 + 25 + 27 + 29 = 5^3$$

- 10) Una variante del **TDA cola es la bicola de salida restringida** cuya implementación modifica la operación de inserción de elementos de modo que ésta se realice por el *frente* o el *final* de la estructura según el valor de un parámetro de opción. Considerando esto, indique la definición de datos de la estructura e implemente la operación *agregar_bicola* (y las operaciones que ésta invoque). Considere que la bicola prioriza la velocidad de proceso y que **únicamente** se cuenta con un arreglo (10 posiciones) para construir el TDA.
- 11) Una variante del **TDA cola es la bicola de entrada restringida** cuya implementación modifica la operación de eliminación de elementos de modo que ésta se realice por el *frente* o el *final* de la estructura según el valor de un parámetro de opción. Considerando esto, indique la definición de datos de la estructura e implemente la operación *quitar_bicola* (y las operaciones que ésta invoque). Considere que la bicola espacio de almacenamiento y que **únicamente** se cuenta con un arreglo (15 posiciones) para construir el TDA.
- 12) Modifique la implementación del **TDA bicola con entrada restringida** considerando que **únicamente** se utilizan 2 arreglos (6 posiciones cada uno) para almacenar los datos y los indicadores. Suponga que la implementación prioriza velocidad de proceso y que las últimas posiciones del segundo arreglo guardan los indicadores de la bicola. Indique la **definición** correspondiente a esta estructura e implemente las **operaciones** *bicola_vacía*, *quitar_bicola*, *siguiente* y *anterior*.
- 13) Aplicando el **TDA bicola** y sus operaciones básicas desarrolle un algoritmo que permita determinar si un número entero es capicúa o no. Para ello, elija el tipo de *bicola* más apropiado para resolver el problema planteado.
- 14) Utilizando listas enlazadas implemente el **TDA bicola** (con entrada restringida) y sus operaciones básicas. Para ello, considere las siguientes variantes:
- un único puntero de inicio (a libre elección simples o dobles)
 - punteros de inicio y final (a libre elección simples o dobles)
- 15) Utilizando listas enlazadas (queda a libre elección el uso de 1 o 2 punteros a la lista):
- defina el **TDA bicola con salida restringida** (bicola de valores caracteres) e implemente las operaciones básicas de la bicola. Considere que la cola está limitada a 15 caracteres.
 - defina el **TDA bicola con entrada restringida** (bicola de valores enteros) incluyendo en esta definición un elemento que indique cuántos valores primos almacena la estructura, e implemente las operaciones básicas de la bicola.

