Pilas y colas

Ing José Luis MARTÍNEZ

1 de agosto de 2020

1. PILAS

Las pilas son estructuras de datos que guardan sus elementos en forma ordenada, los cuales son agregados o quitados solamente de un extremo, que es el tope(Top) de la pila.
En consecuencia es una estructura de datos tipo LIFO (*last in first out*) o en castellano UEPS (*ultimo en entrar primero en salir*). Las pilas son muy utilizadas cuando se hacen llamados a función en los programas, los valores de la función invocadora son guardados en una pila para atender a la función invocada.
Las pilas pueden realizar tres operaciones:

1 PUSH(EMPUJAR): Agrega un elemento desde el tope de la pila, el nuevo elemento pasa a ser el tope.

2 POP(SACAR): Quita el elemento que se encuentra en el tope de la lista, así el elemento que ocupa el segundo lugar debe pasar a ser el tope de pila.

3 PEEK(OJEAR): Observa el valor del elemento tope de lista

El siguiente código programa un pila y sus operaciones:

```
/* DUP que implemente una pila (stack) enlazada y
sus operaciones push(empujar), pop(sacar) y peek(ojear). */
#include <stdio.h>
#include <stdlib.h>


typedef struct stack
{
int dato;
struct stack *siguiente;     // Enlace a la siguiente estructura
}Pila;

Pila *top = NULL;
Pila *push(Pila *, int);
Pila *muestraPila(Pila *);
Pila *pop(Pila *);
int peek(Pila *);

int main()
{
    int val, opcion;
    do
        {
```

```
            printf("\n ***** Menu Principal *****");
            printf("\n 1. PUSH");
            printf("\n 2. POP");
            printf("\n 3. PEEK");
            printf("\n 4. Desplegar los valores de la pila");
            printf("\n 5. Salir");
            printf("\n Ingrese la opcion que desea ejecutar: ");
            scanf("%d", &opcion);
            switch(opcion)
            {
                case 1:
                    printf("\n Ingrese el numero a introducir en la pila: ");
                    scanf("%d", &val);
                    top = push(top, val);
                    break;
                case 2:
                    top = pop(top);
                    break;
                case 3:
                    val = peek(top);
                    if (val != -1)
                        printf("\n El valor en el tope de la pila es: %d", val);
                    else
                        printf("\n La pila se encuentra vacia.");
                    break;
                case 4:
                    top = muestraPila(top);
                    break;
            }
        }while(opcion != 5);
    return 0;
}

/* Introduce los elementos de la pila */
Pila *push(Pila *top, int val)
{
    Pila *ptr;
    ptr = (Pila*)malloc(sizeof(Pila));      //reserva memoria para la estructura y toma el pun
    ptr -> dato = val;                      // carga el valor
    if(top == NULL)                         // Pregunta si la pila estac
        {
            ptr -> siguiente = NULL;        // Al ser el primer dato, tambiserl ltimo por eso
            top = ptr;                      // Le adjudica el valor tope de la pila al puntero
        }
    else
        {
            ptr -> siguiente = top;         // La pila ya tenotro valores por eso el puntero a
            top = ptr;                      // El nuevo valor top es el del puntero
        }
    return top;
}

/* Despliega los valores de la pila */
```

```
Pila *muestraPila(Pila *top)
{
    Pila *ptr;
    ptr = top;
    if(top == NULL)
        printf("\n La pila se encuentra vacia");
    else
        {
            while(ptr != NULL)                      // recorre la pila desde el tope hasta el
            {
                printf("\n %d", ptr -> dato);       // En cada iteraccimprime el valor del dat
                ptr = ptr -> siguiente;             // por la direccion del puntero que se act
            }
        }
    return top;
}

/* Sacar el elemento tope de la pila */
Pila *pop(Pila *top)
{
    Pila *ptr;
    ptr = top;
    if(top == NULL)
        printf("\n Underflow de la pila");                              // lleg fondo de la pi
    else
        {
            top = top -> siguiente;                                    // actualiza el valor
            printf("\n El valor a ser retirado es: %d", ptr -> dato);
            free(ptr);                                                 // elimina el elmento
        }
    return top;
}

/* Ojea el valor tope de la pila*/
int peek(Pila *top)
{
    if(top==NULL)
        return -1;          // co de error no hay valor tope
    else
        return top ->dato;  // devuelve el valor del tope
}
```

2. Colas

Otra estructura de datos común es la cola, que adquiere la misma forma y significado que una cola o fila de un cajero, banco, trámites, etc. Del mismo modo tiene una estructura FIFO(First In First Out) o PEPS(Primero Entra Primero Sale). Cuando un elemento de la cola es atendido pasa al siguiente según el orden que fueron cargados. Las operaciones principales que se pueden hacer son agregar (enqueue) y colorblueretirar(dequeue).
En una cola ligada cada elemento tiene dos partes, uno que almacena el dato y otro que guarda la dirección del elemento siguiente. Vamos a utilizar dos elementos, el de inicio o **cabeza** y otro que apunte al último al que llamamos **talón**. Todas los agregados de elementos se realizan por el **talón** y el retirado de los mismo se hace por la **cabeza**. Si tenemos que $CABEZA = TALÓN = NULL$, la

cola está vacía.

El siguiente programa da un ejemplo de como implementar una cola enlazada.

```
/* DUP que implemente una cola enlazada y sus operaciones bcas. */
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int dato;
    struct node *siguiente;
}Nodo;

typedef struct queue
{
    Nodo *cabeza;               // apunta a dos nodos
    Nodo *talon;
}Cola;

Cola *q;                        // puntero de tipo Cola
void creaCola(Cola *);          // Crea la cola sin inicializarla
Cola *Agregar(Cola *,int);      // inserta un elemento
Cola *RetirarElemento(Cola *);  // Retirar un elemento
Cola *despliega(Cola *);        // muestra los elementos de la cola
int peek(Cola *);               // muestra la cabeza de la cola

int main()
{
    int val, opcion;
    creaCola(q);      // cra una cola vac
    do
        {
            printf("\n ***** Menu Principal *****");
            printf("\n 1. Agregar");
            printf("\n 2. Retirar");
            printf("\n 3. Ojear");
            printf("\n 4. Desplegar");
            printf("\n 5. Salir");
            printf("\n Ingrese su opcion : ");
            scanf("%d", &opcion);
            switch(opcion)
            {
                case 1:
                    printf("\n Ingrese el numero a Agregarr en la cola:");
                    scanf("%d", &val);
                    q = Agregar(q,val);
                    break;
                case 2:
                    q = RetirarElemento(q);
                    break;
                case 3:
                    val = peek(q);
```

```
                    if(val != -1)
                        printf("\n El valor al frente de la cola es : %d", val);
                    break;
                case 4:
                    q = despliega(q);
                    break;
                }
            }while(opcion != 5);
        getchar();
        return 0;
}

/* */
void creaCola(Cola *q)
{
    q -> talon = NULL;   // inicializa dos nodos vac
    q -> cabeza = NULL;
}

/* */
Cola *Agregar(Cola *q,int val)
{
    Nodo *ptr;
    ptr = (Nodo*)malloc(sizeof(Nodo));
    ptr -> dato = val;
    if(q -> cabeza == NULL)
        {
            q -> cabeza = ptr;
            q -> talon = ptr;
            q -> cabeza -> siguiente = q -> talon -> siguiente = NULL;
        }
    else
        {
            q -> talon -> siguiente = ptr;
            q -> talon = ptr;
            q -> talon -> siguiente = NULL;
        }
    return q;
}

/* */
Cola *despliega(Cola *q)
{
    Nodo *ptr;
    ptr = q -> cabeza;
    if(ptr == NULL)
        printf("\n Cola vac);
    else
        {
            printf("\n");
            while(ptr!=q -> talon)
                {
                    printf("%d\t", ptr -> dato);
```

```
                    ptr = ptr -> siguiente;
                }
            printf("%d\t", ptr -> dato);
        }
    return q;
}

/* */
Cola *RetirarElemento(Cola *q)
{
    Nodo *ptr;
    ptr = q -> cabeza;
    if(q -> cabeza == NULL)
        printf("\n Underflow");
    else
        {
            q -> cabeza = q -> cabeza -> siguiente;
            printf("\n El valor que sera eliminado es : %d", ptr -> dato);
            free(ptr);
        }
return q;
}

int peek(Cola *q)
{
    if(q->cabeza==NULL)
        {
            printf("\n No hay elementos en la cola");
            return -1;
        }
    else
        return q->cabeza->dato;
}
```

1

3. Problemas

P.1 Sigmund Froidentino ha instalado una cabina de consultas astrológicas en un centro de compras, sus clientes pueden pagar uno, dos o tres minutos de consulta. Para garantizar que haya un tráfico fluido de personas, la cola de clientes NO puede ser mayor a diez. Suponga que los clientes se presentan en forma aleatoria y el tiempo de consulta también se distribuye en forma aleatoria (1, 2 o 3 minutos). Realice una simulación con colas determinando:

1 ¿Cuántos clientes atenderá en una hora en promedio?

2 ¿Cuánta será la espera promedio para ser atendido?

3 ¿Cuál será la longitud promedio de la fila?
Para aprovechar la reutilización de código, use los archivos queue.h y queue.c que acompañan al presente apunte.
Utilize una estructura como la siguiente para definir a los clientes y actualice este valor en el archivo encabezado.

[1] Acompaña a este apunte los programas dados como ejemplo, ejemplos del libro de Deitel, ejemplo de programa con múltiples fuentes

```
typedef struct item
{
 long llegada; /* hora a la que el cliente se une a la cola */
 int tiempoConsulta; /* cantidad de minutos que desea para la consulta */
} Item;
```

4. Apéndice Programa

4.1. Programa adivino

Archivos de compilación múltiple a ser modificados para resolver P.1.

```
/* adivino.c  Programa principal-- usa la interface Queue*/
/* compilar con queue.c y queue.h*/
#include <stdio.h>
#include <stdlib.h> /* para rand() y srand() */
#include <time.h> /* para time() */
#include "queue.h" /* no olvidar cambiar Item typedef */
#define MIN_POR_HR 60.0

bool nuevoCliente(double x); /* ¿Llego un nuevo cliente? */
Item tiempoCliente(long cuando); /* setea los parametros del cliente */

int main()
{
 Queue line;
 Item temp; /* datos del nuevo cliente */
 int horas; /* horas de simulacion */
 int porhora; /* cantidad promedio de llegadas por hora */
 long ciclo, cicloLimite; /* contador, limite */
 long abandonos = 0; /* abandonos por estar completa la fila */
 long clientes = 0; /* clientes en la cola */
 long atendidos = 0; /* atendidos durante la simulacion */
 long sum_line = 0; /* longitud de la fila*/
 int tiempoEspera = 0; /* tiempo hasta ser atendido */
 double min_por_cliente; /* tiempo promedio entre arribos */
 long lineaEspera = 0; /* tiempo acumulativo en la fila */
 inicializaQueue(&line);
 srand(time(0)); /* inicializacion aleatoria de rand() */
 puts("Caso de estudio: Simulacion de arribo de clientes a una consulta");
 puts("Ingrese las horas de simulacion:");
 scanf("%d", &horas);
 cicloLimite = MIN_POR_HR * horas;
 puts("Ingrese la cantidad promedio de clientes por hora:");
 scanf("%d", &porhora);
 min_por_cliente = MIN_POR_HR / porhora;
 for (ciclo = 0; ciclo < cicloLimite; ciclo++)
    {
        if (nuevoCliente(min_por_cliente))
        {
            if (queueLlena(&line))
                abandonos++;
            else
                {
                    clientes++;
                    temp = tiempoCliente(ciclo);
                    EnQueue(temp, &line);
                }
        }
        if (tiempoEspera <= 0 && !queueVacia(&line))
            {
                DeQueue (&temp, &line);
                tiempoEspera = temp.tiempoConsulta;
                lineaEspera += ciclo - temp.llegada;
                atendidos++;
            }
        if (tiempoEspera > 0)
            tiempoEspera--;
        sum_line += queueCuentaItems(&line);
    }
    if (clientes > 0)
        {
            printf(" clientes aceptados: %ld\n", clientes);
            printf(" clientes atendidos: %ld\n", atendidos);
            printf(" abandonos: %ld\n", abandonos);
            printf(" largo promedio de la cola: %.2f\n", (double) sum_line / cicloLimite);
            printf(" tiempo promedio de espera: %.2f minutes\n", (double) lineaEspera / atendidos);
        }
    else
        puts("Sin clientes!");
    vaciarQueue(&line);
    puts("Fin");
    return 0;
}

/* x = tiempo promedio entre clientes en mintuos*/
/* devuelve un valor true si el cliente llega en estos minutos */
bool nuevoCliente(double x)
{
 if (rand() * x / RAND_MAX < 1)
    return true;
 else
    return false;
}

/* cuando es la hora a la que llega el cliente */
/* la funcion devuelve una estructura Item con el tiempo de llegada*/
/* seteada a cuando y el tiempo de consulta seteado a un valor aleatorio*/
/* entre 1 y 3 */
Item tiempoCliente(long cuando)
{
 Item client;
 client.tiempoConsulta= rand() % 3 + 1;
 client.llegada = cuando;
 return client;
}
```

```
/* queue.c Funciones utilizadas-- */
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

/* funciones locales*/
static void cargarNodo(Item item, Nodo * pn);
static void cargarItem(Nodo * pn, Item * pi);

void inicializaQueue(Queue * pq)
{
    pq->cabeza = pq->talon = NULL;
    pq->items = 0;
}

bool queueLlena(const Queue * pq)
{
 return pq->items == MAXQUEUE;
}

bool queueVacia(const Queue * pq)
{
 return pq->items == 0;
}

int queueCuentaItems(const Queue * pq)
{
 return pq->items;
}

bool EnQueue(Item item, Queue * pq)
{
 Nodo * pnew;
 if (queueLlena(pq))
    return false;
 pnew = (Nodo *) malloc( sizeof(Nodo));
 if (pnew == NULL)
 {
     fprintf(stderr,"No se pudo reservar memoria! \n");
     exit(1);
 }
 cargarNodo(item, pnew);
 pnew->siguiente = NULL;
 if (queueVacia(pq))
    pq->cabeza = pnew;  /* item va a cabeza */
 else
    pq->talon->siguiente = pnew; /* enlace al final de la cola */
 pq->talon = pnew; /* guarda ubicacion del talon */
 pq->items++; /* agregar un item a la cola*/
 return true;
}

bool DeQueue(Item * pitem, Queue * pq)
{
 Nodo * pt;
 if (queueVacia(pq))
    return false;
 cargarItem(pq->cabeza, pitem);
 pt = pq->cabeza;
 pq->cabeza = pq->cabeza->siguiente;
 free(pt);
 pq->items--;
 if (pq->items == 0)
 pq->talon = NULL;
 return true;
}

/* vaciar la cola */
void vaciarQueue(Queue * pq)
{
 Item dummy;
 while (!queueVacia(pq))
    DeQueue(&dummy, pq);
}

/* funciones locales */
static void cargarNodo(Item item, Nodo * pn)
{
 pn->item = item;
}

static void cargarItem(Nodo * pn, Item * pi)
{
 *pi = pn->item;
}



/*Archivo de encabezado*/
#ifndef QUEUE_H_INCLUDED
#define QUEUE_H_INCLUDED
#include <stdbool.h>
/* Ingrese el tipo de dato aqui*/
/* por ejemplo, */
typedef struct item
{
 long llegada; /* hora a la que el cliente se une a la cola */
 int tiempoConsulta; /* cantidad de minutos que desea para la consulta */
} Item;
/* O typedef struct item {int gumption; int charisma;} Item; */
#define MAXQUEUE 10

typedef struct node
{
```

```
 Item item;
 struct node * siguiente;
} Nodo;

typedef struct queues
{
 Nodo * cabeza; /* puntero a la cabeza de la cola */
 Nodo * talon; /* puntero al talon de la cola */
 int items; /* cantidad de elementos en la cola */
}Queue;

/* operacion: inicializar la cola */
/* precondicion: pq apunta a cola(queue) */
/* postcondicion: queue es inicializada para estar vacia */
void inicializaQueue(Queue * pq);

/* operacion: comprubea si queue esta llena */
/* precondicion: pq apunta a la cola previamente inicializada */
/* postcondicion: devuelve True si queue esta llena, else False */
bool queueLlena(const Queue * pq);  /* OJO, estandar C99*/

/* operacion: comprueba si queue esta vacia */
/* precondicion: pq apunta a la cola previamente inicializada */
/* postcondition: devuelve True si queue esta vacia, else False */
bool queueVacia(const Queue *pq); /* OJO, estandar C99*/

/* operacion: determina cantidad de items en queue */
/* precondicion: pq apunta a la cola previamente inicializada*/
/* postcondicion: devuelve la cantidad de items en queue */
int queueCuentaItems(const Queue * pq);

/* operacion: agrega un item al talon de queue */
/* precondicion: pq apunta a la cola previamente inicializada */
/* item será colocado en el talon de la cola */
/* postcondicion: si la cola no esta vacia, el item es colocado */
/* en el talon de la cola y la funcion devuelve True */
/* de lo contrario la cola no cambia y la funcion devuelve False */
bool EnQueue(Item item, Queue * pq);

/* operacion: quita elemento de la cabeza de la cola */
/* precondicion: pq apunta a la cola previamente inicializada */
/* postcondicion: si queue no esta vacia, el item en la cabeza de la */
/* cola es copiado a *pitem y luego eliminado, la funcion devuelveTrue;*/
/* si la operacion vacia la cola, entonces es reseteada a vacia*/
/* Si la cola ya esta vacía, entonces no cambia y devuelve False */
bool DeQueue(Item *pitem, Queue * pq);

/* operacion: vacía la cola */
/* precondicion: pq apunta a una cola previamente inicializada */
/* postcondicions: la cola fue vaciada */
void vaciarQueue(Queue * pq);

#endif
```

4.2. Ejemplo de Pila obtenido del libro de Deitel & Deitel

```
/* Figura 12.8: fig12_08.c
   programa de pila dinámica */
#include <stdio.h>
#include <stdlib.h>

/* estructura auto-referenciada */
struct nodoPila {
   int dato;                  /* define un dato como int */
   struct nodoPila *ptrSiguiente; /* apuntador a nodoPila */
}; /* fin de la estructura nodoPila */

typedef struct nodoPila NodoPila; /* sinónimo de la estructura nodoPila */
typedef NodoPila *ptrNodoPila; /* sinónimo para NodoPila* */

/* prototipos */
void empujar( ptrNodoPila *ptrCima, int info );
int sacar( ptrNodoPila *ptrCima );
int estaVacia( ptrNodoPila ptrCima );
void imprimePila( ptrNodoPila ptrActual );
void instrucciones( void );

/* la función main comienza la ejecución del programa */
int main()
{
   ptrNodoPila ptrPila = NULL; /* apunta al tope de la pila */
   int eleccion; /* elección de menú del usuario */
   int valor;  /* entrada int del usuario */

   instrucciones(); /* despliega el menú */
   printf( "? " );
   scanf( "%d", &eleccion );

   /* mientras el usuario no introduzca 3 */
   while ( eleccion != 3 ) {

      switch ( eleccion ) {

         /* empuja el valor dentro de la pila */
         case 1:
            printf( "Introduzca un entero: " );
            scanf( "%d", &valor );
            empujar( &ptrPila, valor );
            imprimePila( ptrPila );
            break;
```

```
         /* saca el valor de la pila */
         case 2:

            /* si la pila no esta vacía */
            if ( !estaVacia( ptrPila ) ) {
               printf( "El valor sacsdo es %d.\n", sacar( &ptrPila ) );
            } /* fin de if */

            imprimePila( ptrPila );
            break;

         default:
            printf( "Eleccion no valida.\n\n" );
            instrucciones();
            break;

      } /* fin de switch */

      printf( "? " );
      scanf( "%d", &eleccion );
   } /* fin de while */

   printf( "Fin del programa.\n" );

   return 0; /* indica terminación exitosa */

} /* fin de main */

/* despliega las instrucciones del programa para el usuario */
void instrucciones( void )
{
   printf( "Introduzca su eleccion:\n"
      "1 para empujar un valor dentro de la pila\n"
      "2 para sacar un valor dwe la pila\n"
      "3 para terminar el programa\n" );
} /* fin de la función instrucciones */

/* Inserta un nodo en la cima de la pila */
void empujar( ptrNodoPila *ptrCima, int info )
{
   ptrNodoPila ptrNuevo; /* apuntador al nuevo nodo */

   ptrNuevo = malloc( sizeof( NodoPila ) );

   /* inserta el nodo en la cima de la pila */
   if ( ptrNuevo != NULL ) {
      ptrNuevo->dato = info;
      ptrNuevo->ptrSiguiente = *ptrCima;
      *ptrCima = ptrNuevo;
   } /* fin de if */
   else { /* no queda espacio disponible */
      printf( "%d no se inserto. Memoria insuficiente.\n", info );
   } /* fin de else */

} /* fin de la función empujar */

/* Elimina un nodo de la cima de la pila */
int sacar( ptrNodoPila *ptrCima )
{
   ptrNodoPila ptrTemp; /* apuntador a un nodo temporal */
   int valorElim; /* valor del nodo */

   ptrTemp = *ptrCima;
   valorElim = ( *ptrCima )->dato;
   *ptrCima = ( *ptrCima )->ptrSiguiente;
   free( ptrTemp );

   return valorElim;

} /* fin de la función sacar */

/* Imprime la pila */
void imprimePila( ptrNodoPila ptrActual )
{

   /* si la pila esta vacía */
   if ( ptrActual == NULL ) {
      printf( "La pila esta vacia.\n\n" );
   } /* fin de if */
   else {
      printf( "La pila es:\n" );

      /* mientras no sea el final de la pila */
      while ( ptrActual != NULL ) {
         printf( "%d --> ", ptrActual->dato );
         ptrActual = ptrActual->ptrSiguiente;
      } /* fin de while */

      printf( "NULL\n\n" );
   } /* fin de else */

} /* fin de la función imprimePila */

/* Devuelve 1 si la pila está vacía, de lo contrario 0 */
int estaVacia( ptrNodoPila ptrCima )
{
   return ptrCima == NULL;

} /* fin de la función estaVacia */
```

4.3. Ejemplo de cola obtenido del libro de Deitel & Deitel

```
/* Figura 12.13: fig12_13.c
   Operación y mantenimiento de una cola */

#include <stdio.h>
#include <stdlib.h>

/* estructura autoreferenciada */
struct nodoCola {
   char dato;                 /* define dato como un char */
   struct nodoCola *ptrSiguiente; /* apuntador nodoCola */
}; /* fin de la estructura nodoCola */

typedef struct nodoCola NodoCola;
typedef NodoCola *ptrNodoCola;

/* prototipos de las funciones */
void imprimeCola( ptrNodoCola ptrActual );
int estaVacia( ptrNodoCola ptrCabeza );
char retirar( ptrNodoCola *ptrCabeza, ptrNodoCola *ptrTalon );
void agregar( ptrNodoCola *ptrCabeza, ptrNodoCola *ptrTalon,
              char valor );
void instrucciones( void );

/* la función main comienza la ejecución del programa */
int main()
{
   ptrNodoCola ptrCabeza = NULL; /* incializa ptrCabeza */
   ptrNodoCola ptrTalon = NULL;  /* incializa ptrTalon */
   int eleccion;                 /* elección de menú del usuario */
   char elemento;                /* entrada char del usuario */

   instrucciones(); /* despliega el menú */
   printf( "? " );
   scanf( "%d", &eleccion );

   /* mientras el usuario no introduzca 3 */
   while ( eleccion != 3 ) {

      switch( eleccion ) {

         /* agrega el valor */
         case 1:
            printf( "Introduzca un caracter: " );
            scanf( "\n%c", &elemento );
            agregar( &ptrCabeza, &ptrTalon, elemento );
            imprimeCola( ptrCabeza );
            break;

         /* retira el valor */
         case 2:

            /* si la cola no está vacía */
            if ( !estaVacia( ptrCabeza ) ) {
               elemento = retirar( &ptrCabeza, &ptrTalon );
               printf( "se desenfilo %c.\n", elemento );
            } /* fin de if */

            imprimeCola( ptrCabeza );
            break;

         default:
            printf( "Eleccion no valida.\n\n" );
            instrucciones();
            break;

      } /* fin de switch */

      printf( "? " );
      scanf( "%d", &eleccion );
   } /* fin de while */

   printf( "Fin de programa.\n" );

   return 0; /* indica terminación exitosa */

} /* fin de main */

/* despliega las instrcciones del programa para el usuario */
void instrucciones( void )
{
   printf ( "Introuzca su elección:\n"
            "   1 para retirar un elemento a la cola\n"
            "   2 para eliminar un elemento de la cola\n"
```

```
          "   3 para terminar\n" );
} /* fin de la función instrucciones */

/* inserta un nodo al final de la cola */
void agregar( ptrNodoCola *ptrCabeza, ptrNodoCola *ptrTalon,
              char valor )
{
   ptrNodoCola ptrNuevo; /* apuntador a un nuevo nodo */

   ptrNuevo = malloc( sizeof( NodoCola ) );

   if ( ptrNuevo != NULL ) { /* es espacio disponible */
      ptrNuevo->dato = valor;
      ptrNuevo->ptrSiguiente = NULL;

      /* si esta vacía inserta un nodo en la cabeza */
      if ( estaVacia( *ptrCabeza ) ) {
         *ptrCabeza = ptrNuevo;
      } /* fin de if */
      else {
         ( *ptrTalon )->ptrSiguiente = ptrNuevo;
      } /* fin de else */

      *ptrTalon = ptrNuevo;
   } /* fin de if */
   else {
      printf( "no se inserto %c. No hay memoria disponible.\n", valor );
   } /* fin de else */

} /* fin de la función agregar */

/* elimina el nodo de la cabeza de la cola */
char retirar( ptrNodoCola *ptrCabeza, ptrNodoCola *ptrTalon )
{
   char valor;           /* valor del nodo */
   ptrNodoCola tempPtr; /* apuntador a un nodo temporal */

   valor = ( *ptrCabeza )->dato;
   tempPtr = *ptrCabeza;
   *ptrCabeza = ( *ptrCabeza )->ptrSiguiente;

   /* si la cola está vacía */
   if ( *ptrCabeza == NULL ) {
      *ptrTalon = NULL;
   } /* fin de if */

   free( tempPtr );

   return valor;

} /* fin de la función retirar */

/* Devuelve 1 si la cola está vacía, de lo contrario devuelve 0 */
int estaVacia( ptrNodoCola ptrCabeza )
{
   return ptrCabeza == NULL;

} /* fin de la función estaVacia */

/* Imprime la cola */
void imprimeCola( ptrNodoCola ptrActual )
{

   /* si la cola esta vacía */
   if ( ptrActual == NULL ) {
      printf( "La cola esta vacia.\n\n" );
   } /* fin de if */
   else {
      printf( "La cola es:\n" );

      /* mientras no sea el final de la cola */
      while ( ptrActual != NULL ) {
         printf( "%c --> ", ptrActual->dato );
         ptrActual = ptrActual->ptrSiguiente;
      } /* fin de while */

      printf( "NULL\n\n" );
   } /* fin de else */

} /* fin de la función imprimeCola */
```