UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

Facultad de Ciencias Químicas e Ingeniería

Algoritmos y estructura de datos

Práctica 7. Playlist

Alumno: Caudillo Sánchez Diego

Matricula: 1249199

Grupo: 551

Docente: Alma Leticia Palacios Guerrero

Fecha de entrega: 12/Abril/2019

Introducción

Una cola es una estructura de datos que almacena elementos en una lista y permite acceder a los datos por uno de los dos extremos de la lista. Un elemento se inserta en la cola (parte final) de la lista y se suprime o elimina por la frente (parte inicial, cabeza) de la lista. Las aplicaciones utilizan una cola para almacenar elementos en su orden aparición o concurrencia.

Los elementos se eliminar (se quitan) de la cola en el mismo orden en que se almacenan y, por consiguiente, una cola es una estructura de tipo FIFO (first-in firts-out). El servicio de atención al cliente en un almacén es un ejemplo típico de cola. La acción de gestión de memoria intermedia (buffering) de trabajos o tareas de impresora en un distribuidor de impresoras (spooler) es otro ejemplo típico de cola. Dado que la impresión es una tarea que requiere mas tiempo que el proceso de la transmisión real de los datos desde la computadora a la impresora, se organiza una cola de trabajos de modo que los trabajos se imprimen en el mismo orden en que se recibieron por la impresora. Este sistema tiene el gran inconveniente de que si su trabajo persona consta de una única pagina para imprimir y delante de su petición de impresión existe otra petición para imprimir un informe de 300 páginas, deberá esperar la impresión de esas 300 paginas antes de que se imprima su página.

Desde el punto de vista de estructura de datos, una cola es similar a una pila, en donde los datos se almacenan de un modo lineal y el acceso a los datos solo esta permitido en los extremos de la cola. Las acciones que están permitidas en una cola son:

- Creación de una cola vacía.
- Verificación de que una cola esta vacía.
- Añadir un dato al final de una cola.
- Eliminación de los datos de la cabeza de la cola.

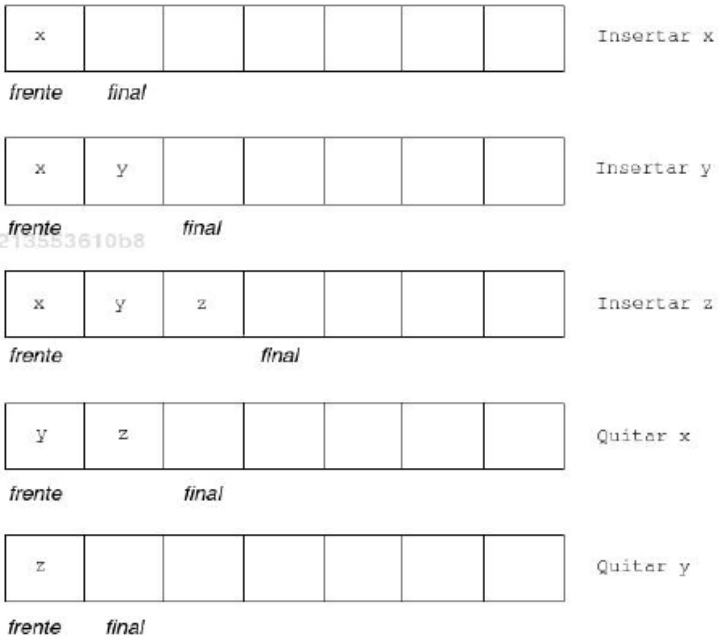

Figura 19.5. Operaciones de `Insertar` y `Quitar` en una Cola.

Competencia: Implementar soluciones de software utilizando eficientemente el principio FIFO en las estructuras estáticas.

Problema

Un dispositivo reproductor de canciones tiene capacidad para una cantidad limitada de canciones, se le pide implementar un programa basado en el principio FIFO para administrar las canciones de la lista. Se desea que a aplicación cuente con las siguientes opciones:

1. Agregar una canción a la lista
2. Remover canción
3. Reproducir lista completa.
4. Salida

Por cada canción se registra la siguiente información:

- Intérprete
- Título
- duración.

Código

```c
/*
<<<Descripción del programa>>>
Un dispositivo reproductor de canciones tiene capacidad para una cantidad
limitada de canciones, se le pide
implementar un programa basado en el principio FIFO para administrar las
canciones de la lista. Se desea que
la aplicación cuente con las siguientes opciones:
  1) Agregar una canción a la lista
  2) Remover canción
  3) Reproducir lista completa.
  4) Salida

Por cada canción se registra la siguiente información:
  ♣ Intérprete,
  ♣ título
  ♣ duración.
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>

#define MAX 11 // Cantidad máxima de canciones que se pueden guardar.
```

```c
// Estructura de tipo canción que contiene la información
// necesaria para capturar una canción.
typedef struct
{
  char interprete[25]; // interprete de la canción
  char titulo[25]; // Titulo de la canción
  int duracion; // duración de la canción
}song_t;

//Funciones prototipo de las funciones creadas
void clrData(song_t playlist[MAX], int tail);
bool removeSong(song_t playlist[MAX], int *tail);
void showPlaylist(song_t playlist[MAX], int head, int tail);
void showDuration(song_t song);
void delay(int seconds);
void playSong(song_t song);
bool remainMem(int tail);
bool addSong(song_t playlist[MAX], int *tail);
void rmvSong(song_t playlist[MAX]);

/* Inicio del programa*/
int main(int argc, char const *argv[])
{
  int opc, head = 0, tail = 0;
  song_t playlist[MAX] = {0};
  do
  {
    system("cls");
    printf("head: %d\ttail: %d\n",head, tail);
    puts("[1] Agregar una cancion a la lista");
    puts("[2] Remover cancion");
    puts("[3] Reproducir lista completa");
    puts("[4] Salida");
    printf("Seleccione una opcion: "); scanf("%d", &opc);
    setbuf(stdin, 0);
    system("cls");
    switch (opc)
    {
      case 1: if(addSong(playlist, &tail) == false)
      {
        printf("No hay espacio en la lista.");
        getchar();
      }else
      {
```

```
        printf("Agregando canci%cn de la lista ...",162);
        delay(1);
        system("cls");
        showPlaylist(playlist, head, tail);
        getchar();
      }break;

      case 2:if(removeSong(playlist, &tail) == false)
      {
        printf("No hay espacio en la lista.");
        getchar();
      }else
      {
        printf("Removiendo canci%cn de la lista ...",162);
        delay(1);
        system("cls");
        showPlaylist(playlist, head, tail);
        getchar();
      }
      break;

      case 3:showPlaylist(playlist, head, tail);
      getchar();
      while(head < tail)
      {
        puts("\tReproduciendo");
        playSong(playlist[head]);
        head++;
      }puts("Ya no hay mas canciones...");
      getchar();
      break;
    }
  } while(opc != 4);
  system("cls");
  return 0;
}

/*
  Descripción: función que limpia los datos guardados en una
  cancion.

  Parámetros
    -playlist: Array de canciones de la cual se va a limpiar
    los datos de una de ellas.
    -tail: índice que indica la posición de la cancion a la
```

```
    cual sus datos van a ser limpiados.
*/
void clrData(song_t playlist[MAX], int tail)
{
  for(int i = 0; playlist[tail].interprete[i] != 0; i++)
    playlist[tail].interprete[i] = 0;
  for(int i = 0; playlist[tail].titulo[i] != 0; i++)
    playlist[tail].titulo[i] = 0;
}

/*
  Descripción: Función de tipo booleano que remueve una cancion de la lista.
  Devuelve 'true' si la cancion ha sido removida con éxito, de
  lo contario de vuelve 'false' indicando que no se pudo quitar
  la cancion de la lista.

  Parámetros
  -playlist: lista que contiene todas las canciones de la cual se desea
  remover una de ellas, se va a remover la última en haber entrado.
  -tail: índice que nos indica la posición de la cancion que se tiene
  que remover.
*/
bool removeSong(song_t playlist[MAX], int *tail)
{
  *tail -= 1;
  if (remainMem(*tail) == false) return false; // ya no hay espacio disponible
  clrData(playlist, *tail);
  return true;
}

/*
  Descripción: Función que muestra las canciones dentro de una lista.
  La función muestra el intérprete, el título de la cancion
  y la duración. La función no devuelve ningún tipo de
  valor.

  Parámetros
  -playlist: arreglo de tipo 'song_t' el cual contiene la
  lista de las canciones.
  -head: cabeza de la lista la cual tiene la función de
  delimitar el inicio de la lista.
  -tail: la cola de la lista es el que pone el límite de
  canciones que van a ser impresas.
*/
void showPlaylist(song_t playlist[MAX], int head, int tail)
```

```
{
  puts("\t\t>>>LISTA DE REPRODUCCION<<<");
    printf("T%ctulo               Interprete
Duraci%cn\n",161,162);
  for(int i = head; i < tail; i++)
  {
    printf("\n%s                %s                    ",
playlist[i].titulo,playlist[i].interprete);
    showDuration(playlist[i]);
  }
}

/*
  Descripción: función que imprime la duración de una cancion
  con formato de minutos y segundos. Ejemplo: 1:39

  Parámetros
  -song: cancion a la cual se le va extraer el tiempo
  capturado para darle formato al tiempo.
*/
void showDuration(song_t song)
{
  int sec = 0, min = 0;
  sec = song.duracion - ((song.duracion/3600)*3600);
  min = sec/60;
  sec -= (min*60);
  printf("%d:%d", min, sec);
}

/*
  Descripción: función que realiza un dela. El tiempo del delay es en función a
segundos
  y el valor se recibe como un numero tipo entero.

  Parámetros
  -seconds: número de segundos que
*/
void delay(int seconds)
{
  int milli_seconds = 1000 * seconds;
  clock_t start_time = clock();
  while (clock() < start_time + milli_seconds);
}

/*
```

```
  Descripción: Función que reproduce las canciones mostrando la información
  y el tiempo transcurrido. Cuando llegue al final de la lista
  muestra un mensaje que ya no hay canciones en la lista.

  Parámetros
  -song: cancion que actualmente se está reproduciendo.
*/
void playSong(song_t song)
{
  system("cls");
  int sec = 0, min = 0;
  for (int i = 0; i < song.duracion; i++)
  { sec++;  // aumentan los segundos
    if(sec > 59)
    {
      sec  = 0; // si segundos llega a 59, se reinicia a 0
      min++;  // y se aumentan los minutos.
    }printf("%s by %s\t%d:%d/", song.titulo, song.interprete, min, sec); //
impresion de los datos.
    showDuration(song);
    delay(1); // delay de un segundo para dar la impresion de que es un contador.
    system("cls"); // se limpia pantalla para que se actualicen los datos en
pantalla
  }
}

/*
  Descripción: Función que devuelve un dato tipo booleano. La función verifica si
en la cola
  aún existe espacio disponible para almacenar datos en ella, si es así devuelve
  'true', caso contrario 'false'.
  Parámetros
  -tail: verifica que el apuntador de la cola no haya rebasado el límite
permitido de datos
  a guardar en la cola.
*/
bool remainMem(int tail)
{
  if(tail <  MAX || tail >= 0) return (true); //Aun hay espacio disponible
  else return (false); // Ya no hay espacio disponible en la cola
}

/*
  Descripción: Función que captura los datos de una cancion. Los datos a capturar
  son el intérprete, el título de la cancion y la duración, la cual
```

```
  es capturada en segundos.

  Parámetros
  -playlist: Arreglo de tipo canciones, la cual contiene los datos a capturar.
  -tail: entero el cual se va desplazando a la siguiente posición de la cola
  para seguir insertando más datos. Funciona como un índice para ver en que
  posición se va agregar la siguiente cancion.
*/
void getData(song_t playlist[MAX], int tail)
{
  puts("<<< Ingrese los siguientes datos >>>");
  printf("Interprete: ");              gets(playlist[tail].interprete);
  printf("Titulo: ");                  gets(playlist[tail].titulo);
  printf("Duraci%cn (segundos): ",162); scanf("%d", &playlist[tail].duracion);
  getchar();
  system("cls");
}

/*
  Descripción: Función de tipo booleano que agrega una cancion a la cola.
  Retorna "true" si la cancion se agregó con éxito. Caso con-
  traro regresa "false" indicando que ya no hay espacio dis-
  ponible.

  Parámetros
  -playlist: Array de tipo 'song' el cual tiene las canciones
  que se pueden agregar a la cola.
  -tail: entero que sirve como índice de la posición en la que
  se ve a agregar la siguiente cancion. Incrementa en 1 si la
  cancion se agregó correctamente.
*/
bool addSong(song_t playlist[MAX], int *tail)
{
  if (remainMem(*tail) == false) return false; // ya no hay espacio disponible
  getData(playlist, *tail); // Llamada a la función para capturar los datos de la
cancion.
  *tail += 1; // Se incrementa para apuntar a la siguiente posición del array.
  return true; // Se ha agregado con éxito la cancion.
}
```

Evidencia de ejecución

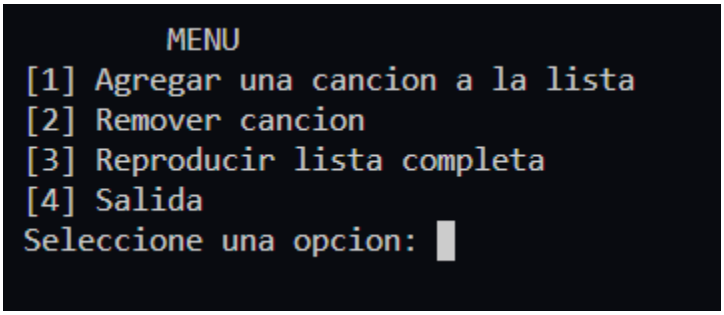

Figura 1. Menú principal

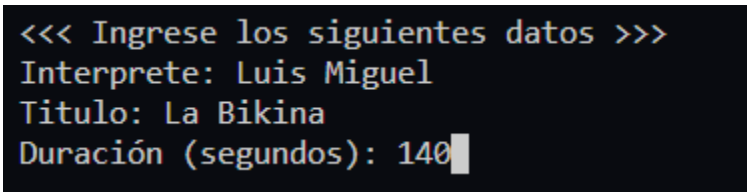

Figura2. Captura de los datos

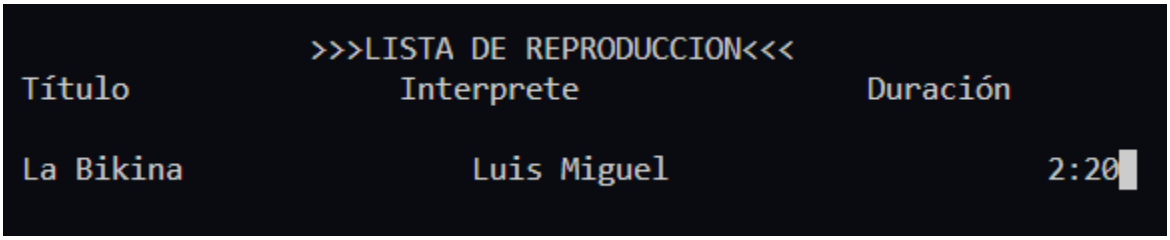

Figura 3. Muestra las canciones de la lista

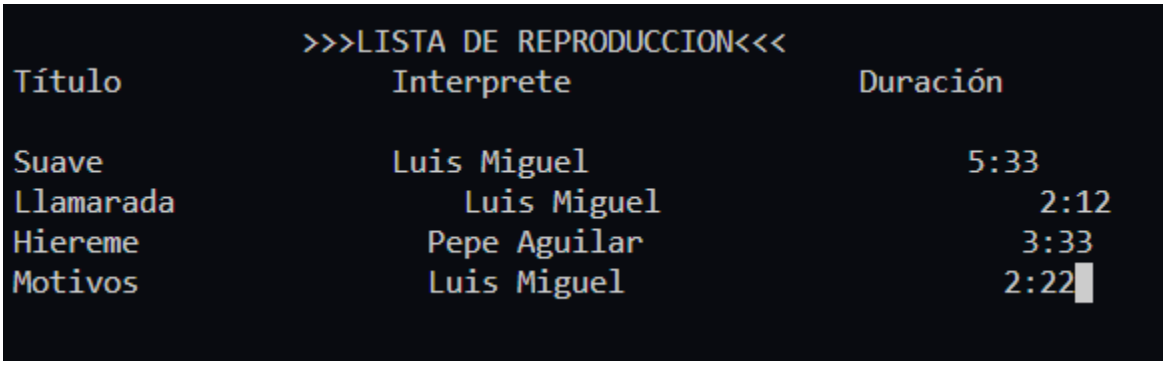

Figura 5. Lista de canciones en la cola.

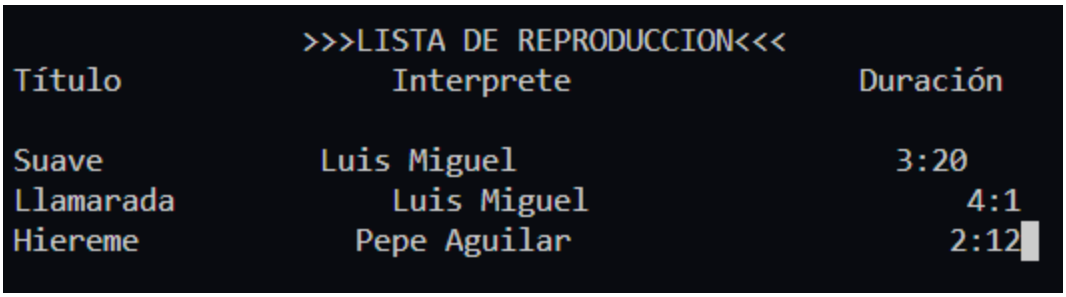

Figura 6. Eliminar un dato de la cola

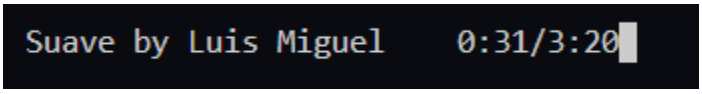

Figura 7. Reproducción de la lista

Conclusión

Con la realización de esta práctica fue de gran ayuda para comprender de qué manera se utilizan las colas y sus operaciones que se le pueden aplicar, como agregar y eliminar un dato de la cola. Unos de los componentes mas importantes de las colas son apuntadores cola y cabeza para saber en que posición estamos actualmente y poder hacer las operaciones correctamente.

Bibliografía

Luis Joyanes (2005), *Programación en C. Metodología, algoritmos y estructura de datos*. Madrid: McGrawHill.