Universidad Autónoma de Baja California
Facultad de Ciencias Químicas e Ingeniería

Algoritmos y Estructuras de Datos
Ciclo 2020-1

Práctica 3

"Estructuras FIFO con Listas Enlazadas"

Profesor(a): Thelma Violeta Ocegueda Miramontes

Grupo: 552

Alumnos:	Matrícula:
• Cota Robledo Benjamín	1225836
• Rodríguez Muñoz José Luis	1260368

Tijuana, Baja California, a 19 de febrero de 2020

CÓDIGO EN C

```c
#include <stdio.h> // Librería principal
#include <string.h> // Manejo de cadenas
#include <stdlib.h> // Uso de memoria dinámica
#include <time.h> // Para el uso del delay

// Estructura para manejar al Prisionero
typedef struct
{
    char nombrePrisionero[15]; // Nombre del prisionero
    char apodo[10]; // Apodo del prisionero
    int tipoMuerte; // Tipo de muerte para el prisionero
} Prisionero;

// Estructura para manipular el Nodo de la Cola
struct Nodo
{
    Prisionero prisionero; // Atributo de la estructura Prisionero
    struct Nodo* siguiente; // Atributo de la estructura Nodo con apuntador
};

/* Funciones prototipo */
void capturaDatos(char nombrePrisionero[15], char apodo[10], int*
tipoMuerte); // Captura los datos del prisionero
void delay(int segundos); // Simula el tiempo
void clearBuffer(); // Limpia el buffer para que no haya conflicto con los
gets

/* Funciones para la Cola */
void insertarNodo(struct Nodo** inicio, char nombrePrisionero[15], char
apodo[10], int tipoMuerte); // Inserta prisionero a la cola
void borrarNodo(struct Nodo** inicio, int posicion);
void moverFinal(struct Nodo** inicio); // Interactúa con la lista de los
prisioneros para hacer el corrimiento
void mostrar(struct Nodo* inicio); // Muestra la lista de los prisioneros
agregados

void capturaDatos(char nombrePrisionero[15], char apodo[10], int* tipoMuerte)
{
    int opcion; // Variable para manejar el tipo de ejecución
    printf("\n\n\t Prisionero");
    printf("\n\n Nombre: "); // Pide nombre del prisionero
    gets(nombrePrisionero); // Captura el nombre
    printf("\n Apodo: "); // Pide el apodo del prisionero
    gets(apodo); // Captura el apodo
    printf("\n\n ************************************************** "); //
Cuestión de estética
    printf("\n\n\t Tipo de ejecuci%cn", 162);
    printf("\n\n 1) R%cpida", 160);
    printf("\n 2) Lenta");              // Este bloque imprime los tipos de
ejecución
    printf("\n 3) Dolorosa");
    do
    {
```

```
        if(opcion < 1 || opcion > 3) // Rango de opciones 1-3
            printf("\n\n\t Elige el tipo de ejecuci%cn: ", 162);
            scanf("%d", &opcion); // Lee el tipo de ejecución
            *tipoMuerte = opcion; // Se le asigna la variable tipoMuerte con
apuntador a la variable opción
            printf("\n\n Prisionero listo para ejecutar ...");
    } while(opcion < 1 || opcion > 3); // Mientras la opción sea entre 1 y 3
    delay(1000); // Simula el tiempo
    clearBuffer(); // Limpia el buffer para los gets
}

// Función para simular el tiempo
void delay(int segundos)
{
    clock_t start = clock();
    while(clock() < start + segundos); /* Este bloque realiza un retardo para
ejecutar alguna instrucción */
}

// Función para limpiar el buffer
void clearBuffer()
{
    while(getchar() != '\n'); // Evita conflictos con los gets
}

// Función para insertar elemento a la lista enlazada
void insertarNodo(struct Nodo** inicio, char nombrePrisionero[15], char
apodo[10], int tipoMuerte)
{
    struct Nodo* nuevo; // Atributo de la estructura Nodo
    nuevo = (struct Nodo*)malloc(sizeof(struct Nodo)); // Se utiliza la
memoria dinámica
    strcpy(nuevo -> prisionero.nombrePrisionero, nombrePrisionero); // Copia
la cadena apuntada por origen en la cadena apuntada por destino
    strcpy(nuevo -> prisionero.apodo, apodo); // Copia la cadena apuntada por
origen en la cadena apuntada por destino.
    nuevo -> prisionero.tipoMuerte = tipoMuerte; // Tipo de muerte
    nuevo -> siguiente = NULL; // Si el nuevo es siguiente entonces será nulo

    if(*inicio == NULL)
    {
        *inicio = nuevo; // Se podría decir que este es cómo un caso base
    }
    else
    {
        struct Nodo *p, *q; // Atributos de la estructura Nodo
        p = *inicio; // Se le asigna el inicio con apuntador al atributo p

        while(p != NULL)
        {
            q = p;
            p = p -> siguiente; /* Este bloque agrega un elemento mientras no
sea nulo */
        }
        q -> siguiente = nuevo;
    }
}
```

```
// Función para eliminar nodo de la lista enlazada
void borrarNodo(struct Nodo** inicio, int posicion)
{
    int i; // Variable contador

    if(*inicio == NULL) // Si inicio apunta a nulo es un caso base
        return;
    struct Nodo *aux = *inicio; // Se le asigna el inicio al atributo aux de
la estructura Nodo con apuntador

    if(posicion == 0) // Si posición es 0 entonces realiza el siguiente
bloque
    {
        *inicio = aux -> siguiente; // aux pasa a ser el nodo siguiente
        free(aux); // Libera memoria de aux
        return;
    }
    for(i = 0; aux != NULL && i < posicion - 1; i++) // Itera K veces según
la posición ingresada
        aux = aux -> siguiente; // Se asigna a aux el nodo que hace
referencia como siguiente

    if(aux == NULL || aux -> siguiente == NULL) // Si aux es nulo o aux
siguiente es nulo, esto quiere decir que estas al final de la cola
    {
        //system("cls"); // Limpia pantalla
        printf("\n No existe prisionero en esa posici%cn ...", 162); //
Imprime el mensaje para el usuario
        return;
    }
    struct Nodo *siguiente = aux -> siguiente -> siguiente; // Hace el
corrimiento de los nodos
    free(aux -> siguiente); // Libera memoria del nodo auxiliar siguiente
    aux -> siguiente = siguiente; // Se le asigna el nodo siguiente al nodo
auxiliar siguiente
}

// Función para interactuar con la lista enlazada
void moverFinal(struct Nodo** inicio)
{
    if(*inicio == NULL || (*inicio) -> siguiente == NULL)
        return; // Se podría decir que es cómo un caso base

    struct Nodo* primero = *inicio; // Se le asigna el inicio con apuntador
al atributo primero
    struct Nodo* ultimo = *inicio; // Se le asigna el inicio con apuntador al
atributo ultimo

    while(ultimo -> siguiente != NULL)
    {
        ultimo = ultimo -> siguiente; // Cuando el último sea diferente de
nulo se asigna elemento
    }

    *inicio = primero -> siguiente;
```

```
    primero -> siguiente = NULL;        /* Este bloque realiza los
movimientos en la lista enlazada */
    ultimo -> siguiente = primero;
}

// Función para mostrar el contenido de la lista enlazada
void mostrar(struct Nodo* inicio)
{
    struct Nodo* aux; // Auxiliar de la estructura Nodo
    int i = 0; // Contador para los prisioneros
    aux = inicio; // Se le asigna el inicio al auxiliar
    printf("\n\n\t\t Prisioneros en espera");
    do
    {
        printf("\n\n %d. %s \t %s", i, aux -> prisionero.nombrePrisionero,
aux -> prisionero.apodo); // Muestra el nombre y el apodo en orden

        if(aux -> prisionero.tipoMuerte == 1)
        {
            printf("\t Muerte r%cpida", 160); // Si la opción es 1 entonces
es muerte rápida
        }
        else if(aux -> prisionero.tipoMuerte == 2)
        {
            printf("\t Muerte lenta"); // Si la opción es 2 entonces es
muerte lenta
        }
        else if(aux -> prisionero.tipoMuerte == 3)
        {
            printf("\t Muerte dolorosa"); // Si la opción es 3 entonces es
muerte dolorosa
        }
        aux = aux -> siguiente; // Asignación al auxiliar
        i++; // Incrementa el contador
    } while(aux != NULL); // Mientras que el auxiliar sea diferente de nulo
}

int main()
{
    struct Nodo* inicio = NULL; // Se crea el nodo de la lista enlazada
    char nombrePrisionero[15], apodo[10]; // Nombre y apodo del prisionero
    int tipoMuerte, K; // Variable para el tipo de muerte y para la posición
a eliminar
    char opcion; // Variable para las opciones del menú
    do
    {
        //system("cls");
        printf("\n\n ************************************************** ");
        printf("\n\n\t FIFO con Listas Enlazadas");
        printf("\n\n 1) Agregar prisionero a la lista");
        printf("\n 2) Ejecuci%cn de prisioneros", 162);
        printf("\n 3) Mostrar lista de prisioneros");       /* Bloque que
muestra el menú */
        printf("\n 4) Salir");
        printf("\n\n\t Introduce una opci%cn: ", 162);
        opcion = getchar(); // Captura la opción
```

```
        switch(opcion)
        {
        case '1':
            //system("cls");
            clearBuffer(); // Limpia el buffer
            capturaDatos(nombrePrisionero, apodo, &tipoMuerte); // Captura
los datos del prisionero
            insertarNodo(&inicio, nombrePrisionero, apodo, tipoMuerte); //
Inserta elemento
            getch(); // Espera un enter para continuar
            break; // Rompe el bucle

        case '2':
            //system("cls");
            clearBuffer(); // Limpia el buffer

            if(inicio != NULL) // Si el inicio es diferente de nulo
            {
                printf("\n Ingresa la posici%cn a remover [0, ... n]: ",
162); // Pide la posición a eliminar
                scanf("%d", &K); // Captura el dato
                mostrar(inicio); // Muestra la lista enlazada
                borrarNodo(&inicio, K); // Borra el nodo
                moverFinal(&inicio); // Hace el corrimiento en la lista
                printf("\n\n
********************************************** "); // Cuestión de
estética
                delay(2000); // Simula el tiempo
                printf("\n\n\t -> Se elimin%c al prisionero ...", 162); //
Monitoreo
                printf("\n"); // Salto de línea para que se vea mejor
                mostrar(inicio); // Muestra la lista enlazada
                getch(); // Espera enter para continuar
            }
            else
            {
                printf("\n No hay prisioneros en cola ..."); // No hay
elementos en la lista
                getch(); // Espera enter para continuar
            }
            break; // Rompe bucle

        case '3':
            //system("cls");
            clearBuffer(); // Limpia el buffer

            if(inicio != NULL) // Si inicio es diferente de nulo
            {
                mostrar(inicio); // Muestra la lista enlazada
                getch(); // Espera enter para continuar
            }
            else
            {
                printf("\n No hay prisioneros en cola ..."); // No hay
elementos en la lista
                getch(); // Espera enter para continuar
            }
```

```
            break; // Rompe bucle
        }
    } while(opcion != '4' && opcion != EOF); // Realiza el menú mientras no
sea opción 4 o Fin de archivo
      {
          //system("cls");
          printf("\n Cierre exitoso ..."); /* Termina el programa */
      }

    return 0;
}
```

CONCLUSIONES

Cota Robledo Benjamín: La realización de la practica me hizo comprender más a fondo el concepto de listas enlazadas lo cual al principio se me dificultaba ya que no tenía idea de cómo poder referenciar nodos que no están al inicio o al final de la cola, y fue algo que necesitábamos implementar en la práctica, de igual manera pasar nodos del inicio hasta el final de la cola sin perder ningún nodo en el proceso.

Rodríguez Muñoz José Luis: Sinceramente está práctica me gustó bastante, ya que era algo bastante dinámico para poder implementar una lista enlazada con colas. Donde descubrí que, en las funciones de insertar, eliminar, se debía usar doble apuntador para poder realizar lo que pedía el ejercicio. Una observación es que nos fue complicado realizar la función para hacer los movimientos de la lista al momento de eliminar a un prisionero dependiendo la posición que ingresara el usuario, y al igual, se necesitó de doble apuntador para poder manipular la cola o bien, la lista enlazada.