



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:

Edgar Tista García

Asignatura:

Estructuras de Datos y Algoritmos I

Grupo:

1

No. de Práctica(s):

5

Integrante(s):

Ugalde Velasco Armando

*No. de Equipo de
cómputo empleado:*

No. de Lista:

38

Semestre:

2020-2

Fecha de entrega:

14 de marzo de 2020

Observaciones:

CALIFICACIÓN: _____

OBJETIVO

Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales Pila y Cola, con la finalidad de que comprendas sus estructuras y puedas implementarlas.

EJERCICIO 1

a) Completa las funciones de push y pop en la biblioteca Pila.h para que el programa “ejercicio1.c” se pueda ejecutar correctamente. Realiza los comentarios respectivos.

La implementación de las funciones se llevó a cabo de la siguiente forma:

```
void push(Pila *p, int x) {
    int *tope = &p->tope;
    if (*tope < 100)
    {
        p->lista[(*tope)++] = x;
    }
    else
    {
        printf("Pila llena\n");
    }
}
```

Función push

Para la función **push**, primero se almacenó un puntero a la variable **tope** de la pila, para modificarla de forma indirecta. Posteriormente, se comprobó si había espacio suficiente en el arreglo para añadir el nuevo elemento, es decir, si el valor de la variable **tope** era un índice válido (menor a 100). De ser así, se asignaba el nuevo elemento al arreglo en la posición correspondiente (variable **tope**) y se realizaba el incremento en ésta. De lo contrario, únicamente se imprime el mensaje “Pila llena”.

```

int pop(Pila *p){
    if (!isEmpty(*p))
    {
        int tope = --p->tope;
        int val = p->lista[tope];
        p->lista[tope] = NULL;
        return val;
    }
    else
    {
        printf("Pila vacía\n");
        return NULL;
    }
}

```

Función pop

En esta función, primero se comprobó si la pila estaba vacía, utilizando la función **isEmpty**. Si no era así, era posible eliminar un elemento.

Primero, se realizaba un decremento a la variable **tope** en la pila, y se asignaba el valor de ésta a una variable **tope** dentro de la función. Posteriormente, se almacenaba el valor a eliminar en la variable **val**. Finalmente, el valor en el arreglo era cambiado a **NULL**, es decir, 0, y se retornaba el valor eliminado.

Si la pila estaba vacía, únicamente se imprimía el mensaje “Pila vacía”.

En el programa **ejercicio1.c**, fue necesario cambiar las funciones **meter** por **push**, y **sacar** por **pop**. En él, se crea una pila con los siguientes elementos:

Tope	5
50 (top)	4
40	3
30	2
20	1
10	0

Los primeros datos que el programa imprime son el tope de la pila y el valor en la punta. Como se puede observar en la captura de pantalla, los datos coinciden con la pila representada.

```
vamos a crear una pila
vamos a ingresar algunos elementos
El tope de la pila es: 5
El valor almacenado hasta arriba es: 50
```

Salida del programa

Después, se ejecutan las siguientes líneas:

```
int a = pop(&miPila); // cambiar funcion sacar por pop
printf("Ahora el tope de la pila es : %d \n", top(miPila));
int b = pop(&miPila);
int c = pop(&miPila);
printf("Ahora el valor hasta arriba es: %d \n", top(miPila));
int d = pop(&miPila);
printf("valor de c es: %d \n", c);
```

Ejecución de pop e impresión de valores

Como podemos observar, se eliminan 3 valores de la pila y se almacenan en las variables a, b, c y d. Después de la primera eliminación se imprime el tope de la pila, el cual debería ser 4, como se muestra en la siguiente representación

Tope	4
40 (top)	3
30	2
20	1
10	0

Luego, se eliminan dos elementos: el número **40**, que se asigna a la variable **b**, y el número **30**, que se asigna a la variable **c**. Por lo tanto, únicamente quedan el valor **20** y el **10** en la pila.

Se imprime el valor “hasta arriba”, que, como ya se mencionó, es 20.

Finalmente, se elimina un valor, y se imprime el valor de la variable **c**: 30.

A continuación, se muestra la salida del programa, la cual coincide con lo enunciado:

```
Ahora el tope de la pila es : 4
Ahora el valor hasta arriba es: 20
valor de c es: 30
```

Salida del programa

b) Completa las funciones de encolar, desencolar y first en Cola.h para que el programa ejercicio1b se pueda ejecutar correctamente.

La implementación de las funciones se llevó a cabo de la siguiente forma:

```
void encolar(Cola *c,int x){  
  
    if (c->ultimo < 100)  
    {  
        c->lista[c->ultimo++] = x;  
    }  
    else  
    {  
        printf("Cola llena\n");  
    }  
}
```

Función encolar

Primero, se comprueba que haya suficiente espacio para encolar el elemento, es decir, se comprueba que el valor de la variable **último** sea un índice válido para acceder al arreglo (menor a 99).

De ser así, se procede a asignar el elemento al arreglo en la cola en la posición correspondiente, y se realiza el incremento del índice **último**. De lo contrario, únicamente se muestra en pantalla el mensaje “Cola llena”.

```
int desencolar(Cola *c){  
    if (!isEmpty(*c))  
    {  
        int val = c->lista[c->primero];  
        c->lista[c->primero++] = NULL;  
        return val;  
    }  
    else  
    {  
        printf("Cola vacía\n");  
    }  
}
```

Función desencolar

En esta función, primero se comprueba que la cola no se encuentre vacía. Si no lo está, se procede a eliminar el elemento correspondiente.

Se almacena el valor a eliminar, accediendo a la posición correspondiente en el arreglo de la cola: el índice se encuentra almacenado en la variable **primero**, en la estructura cola. Después, se le asigna el valor **NULL (0)** a este elemento del arreglo y se incrementa la variable primero. Finalmente, se retorna el valor eliminado.

Si la cola está vacía, únicamente se imprime el mensaje “Cola vacía”.

```
int first(Cola c){  
    if (!isEmpty(c))  
    {  
        return c.lista[c.primerero];  
    }  
    else  
    {  
        printf("Cola vacia\n");  
    }  
}
```

Funcion first

Al igual que en la función anterior, primero se comprueba si la cola no está vacía. Si no lo está, se retorna el valor del elemento correspondiente al primero de la cola, utilizando la variable **primero** como índice. Si lo está, únicamente se imprime el mensaje “Cola vacía”.

En el programa **ejercicio1b.c**, primero se crea una cola, y justo después se intenta desencolar un elemento de ésta. En la salida del programa, se observa el mensaje producido por la función desencolar, el cual indica que la cola está vacía.

Luego, se encolan los números 5, 10, 15 y 20, y se imprimen los valores de los índices en la cola. Como se espera, el índice del primer elemento es 0, y el índice del último elemento, o de la **tail** es 4, es decir, se trata del quinto elemento del arreglo, ya que, al encolarse el próximo elemento ésta será la posición donde se almacene.

A continuación, se muestra la salida del programa, que concuerda con lo anterior:

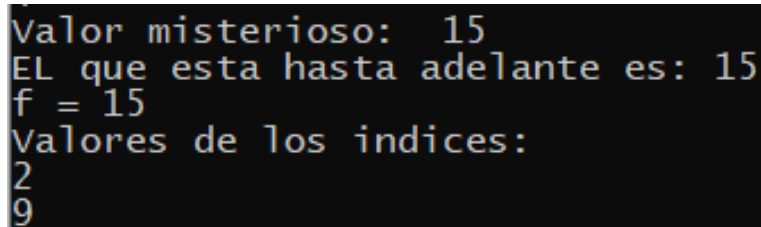
```
Cola vacia  
Valores de los indices:  
0  
4
```

Posteriormente, se encola el número 25 y se desencolan dos números, almacenados en dos variables. Se realiza la suma de ellas y se imprime el valor, el cual es 15, ya que los dos primeros valores ingresados a la cola fueron 5 y 10.

Se encolan los números 30, 35, 40 y 55. Se almacena el primer número de la cola, utilizando la función **first**, y se imprime su valor, el cual es 15, ya que anteriormente se eliminaron el 5 y 10, que eran sus antecesores.

Finalmente, se imprimen los índices presentes en las variables primero y último en la cola, es decir la **head** y la **tail**. Como se eliminaron 2 elementos, ahora el primer elemento se encuentra en el índice 2. Como se agregaron 9 elementos en total, el índice de la **tail** debe encontrarse una posición después del último elemento, es decir, se encuentra en la posición del décimo elemento, donde el valor del índice es 9.

A continuación, se muestra la captura de pantalla con la salida esperada:



```
Valor misterioso: 15
EL que esta hasta adelante es: 15
f = 15
Valores de los indices:
2
9
```

Salida del programa

EJERCICIO 2

Crea un nuevo programa (incluye la biblioteca pila.h) en el que puedas realizar lo siguiente:

- Crea una pila.
- Agrega las instrucciones para solicitar el usuario 10 valores para la pila.
- Utilizando 2 pilas adicionales, escribe las instrucciones necesarias para determinar el mayor de los elementos que ingresó el usuario.

Nota: para el punto anterior no puedes utilizar variables para almacenar datos, solamente puedes utilizar las funciones **push**, **pop**, **top** y **isEmpty**.

Implementación

En el programa, primero se creó una pila y se solicitaron los 10 valores correspondientes al usuario, almacenándolos en ella utilizando la función **push**.

Posteriormente, se crearon dos pilas auxiliares, como lo indicaba el programa. La variable **aux1**, que contenía a la primera pila auxiliar, sirvió como un punto intermedio para comparar los valores entre la pila original y la segunda pila auxiliar. En la variable **aux2**, donde se almacenó la segunda pila auxiliar, se fueron colocando respectivamente los valores resultado de la comparación entre el valor eliminado de la pila original y la punta de la pila **aux2**. Es decir, en esta pila se almacenaban los valores que cumplieran con la condición de ser mayores, respecto a los datos mencionados.

```
push(&aux2, pop(&pila));

while (!isEmpty(pila))
{
    push(&aux1, pop(&pila));
    if (top(aux1) > top(aux2))
    {
        push(&aux2, pop(&aux1));
    }
}
```

Manipulación de pilas para obtener el elemento mayor

Primero, se eliminó el primer elemento de la pila y se añadió a la segunda pila auxiliar, para obtener el primer valor con el que comparar los demás.

Posteriormente, se ejecutó el ciclo mostrado. La condición era que la pila no estuviera vacía. En cada iteración, se eliminaba un elemento de la pila original, y se añadía a la pila **aux1**, para compararlo después. Se comprobaba si el valor en la parte superior de la cola **aux1** era mayor que el valor en la parte superior en **aux2**. De ser así, se eliminaba de **aux1** y se agregaba a **aux2**, siendo éste el elemento mayor hasta ese punto.

Al terminar el proceso, el elemento mayor se encontraba en la punta de la segunda pila auxiliar, ya que se realizaron comparaciones con todos los elementos de la pila.

Finalmente, se imprimió el valor mencionado, es decir, el valor superior de **aux2**. A continuación, se muestra la salida del programa. Como podemos observar, el número mostrado coincide con la característica enunciada: es el mayor de todos los elementos.

```
Inserte el valor 1 para la pila: 300
Inserte el valor 2 para la pila: 200
Inserte el valor 3 para la pila: 600
Inserte el valor 4 para la pila: 500
Inserte el valor 5 para la pila: 400
Inserte el valor 6 para la pila: 700
Inserte el valor 7 para la pila: 300
Inserte el valor 8 para la pila: 1000
Inserte el valor 9 para la pila: 240
Inserte el valor 10 para la pila: 540
El mayor elemento es: 1000
```

Salida del programa

EJERCICIO 3

Elabora un programa en el cual el usuario ingrese 10 valores enteros que se almacenarán en una pila y con ayuda de una cola (sin usar variables auxiliares) se invierta el orden de los valores de la pila. En la salida se debe mostrar el contenido final de la pila.

Implementación

Al igual que en el programa pasado, primero se creó una pila y se solicitaron los 10 valores correspondientes al usuario, almacenándolos en ella utilizando la función **push**.

Posteriormente, se imprimió el contenido de la pila, utilizando la función auxiliar **imprimirPila**. Se creó una cola utilizando la función correspondiente.

Luego, se ejecutaron los siguientes ciclos:

```
while (!isEmpty(pila))
{
    encolar(&cola, pop(&pila));
}

while (!isEmptyQueue(cola))
{
    push(&pila, desencolar(&cola));
}
```

Ciclos utilizados para invertir el orden de la pila

En el primero, se vaciaron todos los elementos de la pila utilizando la función **pop**, y se encoló cada elemento conforme fueron eliminados.

Sabemos que la cola utiliza el paradigma **FIFO**. Por lo tanto, ahora que tenemos todos los elementos de la pila en la cola, podemos desencolarlos y agregarlos a la pila, invirtiendo el orden original de ésta. Es decir, el último elemento agregado en la pila original es ahora el primero en agregarse, quedando hasta el “fondo” de ésta. Estas instrucciones se ejecutan en el segundo ciclo: mientras la cola no esté vacía, se desencola un elemento y se agrega a la pila, produciendo el resultado deseado.

Finalmente, se imprimen nuevamente los elementos de la pila, para observar el resultado.

A continuación, podemos observar la salida del programa. Como podemos notar, el orden de la pila es invertido.

```
Inserte el valor 1 para la pila: 645
Inserte el valor 2 para la pila: 76
Inserte el valor 3 para la pila: 342
Inserte el valor 4 para la pila: 654
Inserte el valor 5 para la pila: 23
Inserte el valor 6 para la pila: 65
Inserte el valor 7 para la pila: 87
Inserte el valor 8 para la pila: 2
Inserte el valor 9 para la pila: 54
Inserte el valor 10 para la pila: 23

Pila:
23
54
2
87
65
23
654
342
76
645

Pila:
645
76
342
654
23
65
87
2
54
23
```

Salida del programa

EJERCICIO 4

Crea la estructura **Libro**, que tendrá como miembros: título, autor, número de páginas y clave. Elabora un programa en el que, haciendo las modificaciones respectivas a la biblioteca cola, elabores una cola de Libros.

Implementación

Se creó la estructura **Libro**, con los requerimientos señalados. Además, también se creó la nueva estructura **ColaLibros**, como se muestra en la siguiente captura:

```

typedef struct
{
    char titulo[100];
    char autor[100];
    int pags;
    int clave;
} Libro;

typedef struct{
    int primero;
    int ultimo;
    int tamano;
    Libro *lista;
} ColaLibros;

```

Declaración de estructuras

El problema mencionaba que se requerían almacenar los datos en un arreglo dinámico los libros, por lo que se utilizó un puntero a **Libro** en la variable **lista**, para posteriormente crear el arreglo. También se agregó el tamaño, para comprobar posteriormente si había espacio suficiente en la cola.

Para la manipulación de la cola, se definieron las funciones que implementaban las operaciones estándar de la misma forma que en el ejercicio anterior. Sin embargo, en la función **crearColaLibros** se requirió de un parámetro: el tamaño correspondiente de la cola, por lo que, al momento de inicializar la variable **lista** fue necesario utilizar la función **malloc**. Además, fue necesario agregar la función **eliminarColaLibros**, para liberar el espacio almacenado dinámicamente.

En el programa, primero se solicitó el tamaño de la cola, para posteriormente solicitar al usuario que ingresara cada elemento y encolarlo.

Después, se desencoló cada elemento, se imprimió su título y su número de páginas, y el tiempo de lectura correspondiente. Dicha tarea se realizó en el siguiente ciclo:

```

while(!isEmptyQueueLibros(cola))
{
    tmp = desencolarLibro(&cola);
    tiempoTmp = (tmp.pags * 20) / 60;
    printf("LIBRO %s\n", tmp.titulo);
    printf("Numero de paginas: %d\n", tmp.pags);
    printf("Tiempo de lectura: %d minutos\n\n", tiempoTmp);
    tiempoTotal += tiempoTmp;
}

```

Se desencolan los elementos y se imprime la información pertinente

Como podemos observar, primero se desencola un elemento y se almacena en una variable auxiliar. Luego, se calcula el tiempo de lectura para el libro, multiplicando el número de páginas por 20 y dividiendo el resultado entre 60. Finalmente, se imprime el título del libro, el número de páginas y el tiempo de lectura. Además, se suma el tiempo de lectura a la variable **tiempoTotal**, que funge como un acumulador.

Al finalizar el programa, se imprime el tiempo total de lectura y se libera la memoria utilizada para almacenar dinámicamente el arreglo, utilizando la función **eliminarColaLibros**.

A continuación, se observa la salida del programa:

```
Cual es el tamaño de la cola? 5
Titulo: Don Quijote
Autor: Miguel De Cervantes
Paginas: 350
Clave: 1

Titulo: 100 años de soledad
Autor: Gabriel García Marquez
Paginas: 240
Clave: 2

Titulo: Las almas muertas
Autor: Nicolas Gogoi
Paginas: 325
Clave: 3

Titulo: Preludio a la fundacion
Autor: Isaac Asimov
Paginas: 560
Clave: 4

Titulo: La comedia humana
Autor: Honore de Balzac
Paginas: 443
Clave: 5

LIBRO Don Quijote
Numero de paginas: 350
Tiempo de lectura: 116 minutos

LIBRO 100 años de soledad
Numero de paginas: 240
Tiempo de lectura: 80 minutos

LIBRO Las almas muertas
Numero de paginas: 325
Tiempo de lectura: 108 minutos

LIBRO Preludio a la fundacion
Numero de paginas: 560
Tiempo de lectura: 186 minutos

LIBRO La comedia humana
Numero de paginas: 443
Tiempo de lectura: 147 minutos

TIEMPO TOTAL DE LECTURA: 637 minutos
```

Salida del programa

CONCLUSIONES

En la práctica, se hizo uso de las estructuras lineales pila y cola, que son conceptos fundamentales en la computación.

En el primer ejercicio, se implementaron las operaciones básicas de las estructuras, con lo cual se logró comprender su funcionamiento e implicaciones. En los demás ejercicios, se utilizaron las estructuras mencionadas para resolver problemas. Se utilizaron sus operaciones para manipularlas y cumplir con las tareas propuestas.

Gracias a la realización de la práctica, reafirmé los conceptos vistos en clase y además logré implementarlas y comprenderlas correctamente.

Me parece que los ejercicios realizados no se centran del todo en la implementación y utilización de las estructuras mencionadas. Se utilizó una buena parte del tiempo programando otros componentes sin relación directa con las estructuras, lo cual no me parece óptimo para la comprensión de los temas. Además, pienso que es importante recalcar que las definiciones vistas en clase de las funciones para las operaciones de las estructuras pueden realizarse de distintas formas.