	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):	Práctica #5 - #6
Integrante(s):	Santiago Díaz Edwin Jaret
No. de Equipo de cómputo empleado:	Trabajo en Casa
No. de Lista o Brigada:	42
Semestre:	2021-2
Fecha de entrega:	28/06/2002
Observaciones:	

CALIFICACIÓN: _____

Estructura de Datos Lineales: Pila y Cola // Cola Circular y Cola Doble

Objetivos

Objetivo (5): Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales pilas y colas, con la finalidad que comprendas su estructura y puedas implementarlas.
Objetivo (6): Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales cola circular u cola doble, con la finalidad de que comprendas sus estructuras y puedas implementarlas.

Desarrollo

Estructura Pila

El programa "Pila.c" utiliza la biblioteca "Pila.h" en donde se declaran todas las funciones y la estructura **Pila**.

La estructura **Pila** contiene:

- Una variable de tipo entero para el tope, llamada **tope**.
- Un arreglo de tipo entero en donde se almacenan los elementos. Tiene un tamaño de 100 con el nombre de **lista**.

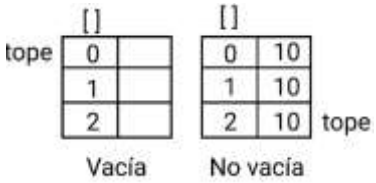
Las funciones son utilizadas en el programa principal y sus utilidades son:

- Función de tipo Pila, *crearPila()*, para crear una estructura, además, recibe un parámetro de tipo entero para obtener el tamaño a utilizar de la Pila, esto, lo ingresará el usuario en un programa siguiente.
  La Pila es creada y se le asigna de nombre "**p**", se le da un valor al **tope** de **0** y se devuelve la estructura.
  Por gusto personal, le agregué a la función un *ciclo for* para que los espacios a utilizar inicien con un valor de **0**, esto es para tener un código más "limpio".

```
Pila p;
    p.tope=0;
    return p;
```

- Función de tipo entero, *isEmpty(),* para comprobar si la pila se encuentra vacía y recibe como parámetro la estructura Pila.
  La función guarda el parámetro en una variable **p**, se utiliza una *sentencia if()* en donde si el **tope** de **p** es igual a **0**, retorna valor de **1**, dando a entender que es **True** y la pila **sí** está vacía. Si el tope es distinto a 0, la pila no está vacía y retorna un valor de **0.**

```
if(p.tope==0)
    return 1;
return 0;
```



- Función de tipo void, *push()*, recibe de parámetro la estructura Pila y una dato de tipo entero. La función agrega el valor de la variable a la estructura Pila.
  Por medio del operador de apuntador flecha, se tiene acceso a la variable **lista** de la pila **p** para que de manera indirecta se modifica a la posición [] del valor de la variable **tope** de la pila **p** a un valor de la variable **x**. Al final se suma 1 al **tope,** para que se recorra el tope.

```
p->lista[p->tope]=x;
    p->tope++;
```

- Función de tipo entero, *pop(),* recibe de parámetro la estructura Pila, esto es para eliminar el último elemento de la pila.

```
if(isEmpty(*p)==1){
        printf("La pila ya est%c vac%ca. \n",160,161);
        return -1;
    }
```

    - Primero, se comprueba por medio de una *sentencia if()* el estado de la Pila, haciendo el uso de la función *isEmpty()*. Esto es debido que no se puede eliminar un elemento de la Pila si esta se encuentra vacía.

```
else{
        int aux = p->lista[p->tope-1];
        p->tope--;
        return aux;
    }
```

    - Si la función devuelve un 1, la pila está vacía, se imprime un mensaje para informar al usuario y retorna **-1**, para terminar la ejecución de la función; por lo contrario, si devuelve **0**, nos da a entender que contiene elementos, y al final, se elimina el último elemento.
      Para eliminar, se crea una variable de tipo entero llamada **aux** en donde accede al último elemento de la **lista** de la estructura **Pila** por medio del apuntador flecha. Se le resta un valor de **1** al **tope,** para disminuir el tamaño y eliminar el elemento, y al final, se regresa la variable que contiene el elemento eliminado.

- Función de tipo entero, *top(),* para conocer el valor del tope de la pila. Se pasa como parámetro la estructura Pila.
  Primero, se comprueba que la Pila no se encuentre vacía por medio de una *sentencia if()* y la función *isEmpty()*. Si se encuentra vacía, se le notifica al usuario imprimiendo un mensaje y termina la función con el retorno de **-1**.

```
else{
        return pila.lista[pila.tope-1];
    }
```

  Si no está vacía, se retorna el valor del último elemento del **tope** por medio del apuntador punto, accediendo a la posición de la **lista** a través del valor de **tope-1,** de la estructura **pila**.

- Agregué una función de tipo void, *imprimirPila(),* con parámetro de estructura **Pila** y el tamaño que ingresó el usuario. Esto es para imprimir los valores utilizados que se encuentran en la **Pila**.
  Se utiliza un *ciclo for()* para recorrer todos los espacios de la **lista**. Se imprime la posición y el valor accediendo a la posición [] de la **lista** de la **Pila**.
  Por último, imprime el tope actual de la **Pila**, llamando a la función *top()*.

```
for (i = 0; i < 10; i++)
    {
        printf("Posicion %d \t valor %d \n",i+1, p.lista[i]);
    }
```

Estructura Cola

El programa “Cola.c” utiliza la biblioteca “Cola.h” en donde se declaran las funciones a utilizar y la estructura **Cola**.

```
struct Cola{
    int primero;
    int ultimo;
    int lista[5];
}
```

La estructura Cola contiene:

- Una variable de tipo entero para índice del inicio de la cola, llamada **primero**
- Una variable de tipo entero para índice del final de la cola, llamada **ultimo**.
- Un arreglo de tipo entero para guardar los elementos, llamada **lista** con un tamaño de 100 elementos.

Las funciones utilizadas en el programa junto con su utilidad son:

```
Cola c;
    c.primero=1;
    c.ultimo=0;
```

- Función de tipo estructura Cola, *crearCola()*, recibe como parámetro el tamaño de la Cola a utilizar, ingresada por el usuario.
  Crea una Cola con el nombre **c** y se inicia con el valor de **1,** a la variable **primero** y con el valor de **0,** a la variable **ultimo**.
  Se agregó un *ciclo for()* en donde cada espacio a utilizar de la cola, se inicia con valor de **0**, esto es para evitar confusiones en los programas siguientes.
  Al final regresa Cola **c**.

```
if(c.primero==c.ultimo+1)
        return 1;
    return 0;
```

- Función de tipo entero, *isEmpty(),* con parámetro de la estructura Cola con el nombre de **c.**
  Verifica si se encuentra vacía la Cola **c**, por medio de una *sentencia if()* se valida si la variable **primero** de la Cola **c** tiene el mismo valor que la variable **ultimo+1** de la Cola **c**. Si esto se cumple, retorna **1** la sentencia y, por lo tanto, la función, está dando a entender que si está vacía. Si no se cumple la *sentencia*, la función retorna un **0** y se entiende que no está vacía.

```
c->lista[c->ultimo]=x;
    c->ultimo+=1;
```

- Función de tipo void, *encolar(),* con parámetro de un apuntador de la estructura Cola con el nombre **c**, y una variable de tipo entero con valor de **x**.
  Dentro de la estructura **c**, la variable **lista** con la posición del valor de la variable **último**, se le asigna el valor de **x** y se le suma **1** a la variable **ultimo** debido a que se ocupa un espacio más en la cola y se recorre el valor de **ultimo**, ya que nos indica el índice final de la Cola.

- Función de tipo entero, *desencolar()*, con parámetro de un apuntador la estructura Cola con el nombre **c**.
  - Se utiliza una *sentencia if()* y la función *isEmpty()* para obtener el estado de la cola, si esta se encuentra vacía o no. En caso de que esté vacía, se le notifica al usuario por medio de un mensaje en la pantalla, de lo contrario, se elimina el último elemento.

```
if((isEmptyC(*c))==1)
        printf("la cola est%c vac%ca \n",160, 161);
```

- Para eliminar el primer elemento, se crea una variable de tipo entero llamada **aux**, en donde se le asigna el elemento de la **lista** con posición del valor de la variable **primer-1** de la estructura **c**.

```
int aux = c->lista[c->primero-1];
c->lista[c->primero-1] = 0;
c->primero++;
```

- Se suma **1** a la variable **primero** y, por medio de otra *sentencia if()*, se valida si la variable **primero** de la estructura **c** tiene el mismo valor que la variable **último** de la structura **c**, si esto es así, la cola está vacía y se crea otra cola llamando a la función *crearCola()* y guardándola en la variable **c** con apuntador (el apuntador es para modificar la variable de manera indirecta).
- Al final, la función regresa la variable **aux**.

- Función de tipo entero, *firstC(),* con parámetro de la estructura Cola asignando una variable **c.** Sirve para conocer el primer índice actual de la Cola.
  Retorna el valor del elemento de la **lista** con posición **primero-1** de la estructura **c**.

```
return c.lista[c.primero-1];
```

- Agregué una función de tipo void, *imprimirCola(),* en donde recibe como parámetro la estructura Cola y el tamaño de la cola.
  Con un *ciclo for()* se recorre los espacios utilizados en la Cola para imprimir los valores de la cola junto con su posición utilizada.
  Por último, se imprime el valor del primer y último índice de la cola, la cual, están guardadas en las variables **primero** y **ultimo.**

Estructura Cola Doble

El programa "colaDoble.c" utiliza la biblioteca "colaDoble.h" en donde se declaran las funciones a utilizar y la estructura **Cola Doble**.

```
int primero;
int ultimo;
int tamano;
int disponibles;
int* lista;
```

La estructura Cola Doble contiene:

- Una variable de tipo entero para el índice inicial de la cola, llamada **primero**
- Una variable de tipo entero para el índice final de la cola, llamada **ultimo**.
- Una variable de tipo entero para el tamaño de la cola llamada **tamano**.
- Una variable de tipo entero para los espacios disponibles, llamada **disponibles.**
- Una variable de tipo entero con un apuntador para la guardar los elementos, llamada **lista**.

Las funciones utilizadas en el programa junto con su utilidad son:

- Función de tipo Cola, *crearCola(),* con parámetro de una variable de tipo entero para el tamaño.

Se crea una estructura de tipo Cola llamada **c**. Se asignan valores a las variables de la estructura:

- La variable **primero** se le asigna valor de **1**
- La variable **ultimo** se le asigna un valor de **0**.
- La variable **tamano** se le asigna el valor de la variable utilizada como parámetro de la función.
- La variable **disponibles** se le asigna el valor la variable **tamano** de la Cola **c**.
- La variable **lista** tendrá una memoria reservada hecha por *calloc()* con el número de espacios reservados del valor de la variable **tamano** y con un tamaño 4 bytes.

```
Cola c;
c.primero=1;
c.ultimo=0;
c.tamano=tamano;
c.disponibles = c.tamano;
c.lista = (int*)calloc(c.ta-
mano,sizeof(int));
return c;
```

Al final, se retorna la Cola Doble **c**.

- Función de tipo entero, *isEmpty(),* con parámetro de la estructura Cola.
  Se valida que la cola se encuentre vacía, esto es por medio de una *sentencia if()* en donde, dentro de la estructura **c,** si se cumple que el valor de la variable **primero** es igual al valor de la variable **último+1** y si además se cumple que la variable **disponible** tiene el mismo valor que la variable **tamaño** , esto **devuelve 1,** dando a entender que **si** está vacía.

```
if((c.primero==c.ultimo+1)&&(c.disponibles==c.tamano))
    return 1;
return 0;
```

- Función de tipo void, *encolarFinal(),* con parámetro de la estructura Cola llamada **c** y una variable de tipo entero llamada **x.**

```
return c.lista[c.primero-1];
```

  Primero, se comprueba por medio de una *sentencia if()* el espacio disponible de la cola, si la variable **disponible** es igual a 0, se imprime en la pantalla un mensaje al usuario informándole que la Cola se encuentra llena.
  Por lo contrario, si no está llena:
  - Se le suma **1** al resultado de obtener el módulo del valor **último** entre **tamano**, y se guarda en la variable **ultimo** de la estructura Cola **c**, esto es para recorrer el valor de índice final.
  - Se le asigna el valor a encolar a la posición del valor del **último-1** de la lista. Esto es, encolar al final el valor.
  - Se le resta **1** a la variable **disponible**, debido a que al utilizar un nuevo espacio, se reduce el espacio disponible de la Cola **c**.

- Función de tipo void, *encolarInicio(),* con parámetro de la estructura Cola asignando una variable **c** y una variable de tipo entero llamada **x**.

```
return c.lista[c.primero-1];
```

  - Por medio de una *sentencia if()*, si el valor de la variable **ultimo** de la estructura **c** es **0**, se ejecuta la función *encolarFinal()* con parámetro de **c** y **x**. Esto es porque la Cola doble no tiene ningún elemento y se debe de encolar en la posición [0].

- Si no se cumple la *sentencia*, por medio de otra *sentencia if()*, se comprueba si la variable **disponible** de la estructura **c** es **0,** si esta se cumple, la Cola **c** está llena y se le notifica al usuario y no agrega nada.
- Si no se cumple la *sentencia*, se utiliza otra *sentencia if()* en donde si el índice **primero** es igual a **1**, la variable primero se le sumará el valor del **tamano-1,** debido a que, será la última posición de la Cola.
  - A la posición de **primero-1** de la **lista** se le asigna de **x**.
  - Al final se le resta **1** a la variable **disponible**, debido a que el espacio se reduce.
  - De lo contrario, el valor del residuo de la división entre la suma de la variable **primero** + el **tamano** entre el valor del **tamano+1,** será asignada a la variable **primero**. Esto modifica el índice **primero** debido a que un nuevo elemento se agrega al inicio.
  - A la posición de **primero-1** de la **lista**, se le asigna el valor de **x** (el valor agregado a encolar)
  - Después se resta **1** a la variable **disponible**, debido a que el espacio se reduce.
  - Se le resta **1** a la variable **disponibles**, debido a que hay un espacio nuevo utilizado.

- Función de tipo entero, *desencolarInicio(),* con parámetro de la estructura Cola asignando una variable **c**.

```
c->disponibles++;
int aux = c->lista[c->primero-1];
c->lista[c->primero-1]=0;
```

  - Por medio de una *sentencia if()*, si al llamar la función *isEmpty()* retorna **1**, nos da a entender que la cola está vacía y se le notifica al usuario.
  - De lo contrario, se le suma **1** a la variable **disponible** (debido a que se va a eliminar un elemento), se crea una variable **aux** que contiene el elemento a eliminar de la **lista** con posición del valor **primero-1** de la estructura **c**.
  - Agregué una línea de código donde a la **lista** se le asigna el valor de **0** (así se elimina).
  - Se comprueba por medio de una *sentencia if()* si el índice **primero** es distinto al índice **ultimo**, la cola no está llena y al índice **primero** se le aumenta **1**, esto es, para recorrer su posición hacia adelante.
  - Después de haber cumplido al función anterior y agregar **1** a primero, por medio de otra *sentencia if()*, si el índice **primero** es igual al índice de **ultimo+1**, la cola está llena y por lo tanto, se crea otra Cola Doble llamando a la función *crearCola()*, pasándole como parámetro el tamaño de la cola llena y se guarda en la variable **c** como apuntador, para que pueda ser editada y utilizada de manera global.

- Función de tipo entero, *desencolarFinal(),* con parámetro de la estructura Cola.

```
c->disponibles++;
int aux = c->lista[c->ultimo-1];
c->lista[c->ultimo-1]=0;
c->ultimo--;
```

  - Por medio de una *sentencia if()*, se verifica el estado de la cola llamando a la función *isEmpty()*. Si la función retorna **1**, la cola está vacía y se le notifica al usuario.
  - Si no se encuentra vacía, se puede eliminar el elemento al final de la cola, se agrega **1** a la variable **disponibles** (porque habrá un nuevo espacio vacío), se crea y guarda en una variable de tipo entero llamada **aux** el elemento a eliminar (Se

encuentra en índice **ultimo-1** de la **lista**) y se le resta **1** a la variable **ultimo**.

- Con la ayuda de otra *sentencia if()*, si el valor de la variable **ultimo** es 0, se comprueba por otra *sentencia if()* si el valor de la variable **primero** es igual al **último+1**, se crea una Cola Doble, con el mismo tamaño de la cola actual guardándola en la variable **c** como apuntador, debido a que la cola se ha quedado vacía.
- Si no se cumple la segunda *secuencia if()* , se le suma el valor del **tamano** de la cola **c** a la variable **ultimo**.
- Por último, con otra *secuencia if()*, si el valor de la posición **primero** es igual al valor de la posición **ultimo+1**, la cola se ha quedado vacía y por lo tanto, crea otra cola llamando a la función *crearCola()* guardándola en la variable **c** como apuntador debido a que la cola se ha quedado vacía. AQUÍ ME QUDEEE

- Función de tipo void, *mostrarValores(),* con parámetro de la estructura Cola asignando una variable **queue1**.
  Se crea una variable de tipo entero llamada **i** con valor de **0** para ser utilizada en un *ciclo for()*, recorriendo desde 0 hasta el valor de la variable **tamano** de la estructura **queue1**, y así, imprimir el número de posición (**i+1**) y el elemento de dicha posición (**queue1.lista[i]**)

```
for(i=0;i<queue1.tamano;i++){
        printf("Posicion %d \t valor %d \n\n",i+1,queue1.lista[i]);
    }
```

- Función de tipo void, *mostrarIndices()*, con parámetro de la estructura Cola asignando una variable llamada **queue1**.
  - Imprime el valor de la variable **primero** de la estructura **queue1** (índice inicial de la Cola Doble)
  - Imprime el valor de la variable **ultimo** de la estructura **queue1** (índice final de la Cola Doble).
  - Imprime el valor de la variable **disponibles** de la estructura **queue1** (los espacios disponibles de la Cola Doble).

```
printf("Primero = %d \n",queue1.primero);
printf("Ultimo = %d \n",queue1.ultimo);
printf("Disponibles = %d \n",queue1.disponibles);
```

Previo1.c – Pila

El programa utiliza las funciones *crearPila(), isEmpty() , push(), pop(), imprimirPila()* del programa "Pila.c".
Se le solicita al usuario con qué estructura lineal se desea trabajar, con **Pila** o con **Cola.**
Si se trabaja con **Pila**, se le solicita al usuario con cuántos elementos se va a trabajar, para fines didácticos, se va a trabajar con **5**.

El programa ejecuta las funciones en orden siguiente:

- *crearPila()*. Con parámetro del valor del tamaño ingresado por el usuario.
- *isEmpty()*. Verificar e imprimir si se encuentra vacía o no.
- *Push()*. Se agrega a al final de la pila un valor de **5**.
- *Push()*. Se agrega a al final de la pila un valor de **8**.
- *Push()*. Se agrega a al final de la pila un valor de **10**.
- *Pop()*. Se elimina el último elemento de la Pila
- *Pop()*. Se elimina el último elemento de la Pila.
- *isEmpty()*. Verificar e imprimir si se encuentra vacía o no.

```
Se hace un push para agregar en la última posición
Posicion 1      valor 5
Posicion 2      valor 8
Posicion 3      valor 10
Posicion 4      valor 0
Posicion 5      valor 0
El tope es actual es el elemento: 10

Se hace un pop para eliminar el último elemento
Posicion 1      valor 5
Posicion 2      valor 8
Posicion 3      valor 0
Posicion 4      valor 0
Posicion 5      valor 0
El tope es actual es el elemento: 8
```

Además, a realizar las operaciones *Push()* y *Pop()* se imprime los valores con sus respectivas posiciones de la **Pila**, para observar cómo está siendo modificada.
Este es el resultado final.

```
Se hace un pop para eliminar el último elemento
Posicion 1      valor 5
Posicion 2      valor 0
Posicion 3      valor 0
Posicion 4      valor 0
Posicion 5      valor 0
El tope es actual es el elemento: 5

Se verifica su estado, si se encuentra vacía o no
0 : No esta vacia
```

Previo1.c Cola

El programa utiliza las funciones *crearCola(), isEmptyC(), encolarC(), desencolarC(), imprimirCola()* del archivo "Cola.c".
Si se trabaja con **Cola**, se le solicita al usuario con cuántos elementos se va a trabajar, para fines didácticos, se va a trabajar con **5**.
El programa ejecuta las funciones en el orden siguiente:

- *crearCola()*. Se le da de parámetro el tamaño deseado de la Cola.
- *isEmpty()*. Verifica e imprime si se encuentra vacía o no.
- *encolarC()*. Se agrega el elemento **1** al principio de la Cola.
- *encolarC()*. Se agrega el elemento **7** al principio de la Cola.
- *encolarC()*. Se agrega el elemento **15** al principio de la Cola.
- *desencolarC()*. Se elimina el primer elemento de la Cola.
- *desencolarC()*. Se elimina el primer elemento de la cola.
- *imprimirCola()*. Imprime los valores y sus posiciones.

```
Se hace un encolar para agregar en la primera posición
Posicion 1      valor 1
Posicion 2      valor 7
Posicion 3      valor 15
Posicion 4      valor 0
Posicion 5      valor 0
El indice del primero: 1

El indice del ultimo: 3
```

- *isEmpty()*. Revisa e imprime el estado de la Cola.

```
Se hace un desencolar para eliminar el primera elemento
Posicion 1      valor 0
Posicion 2      valor 7
Posicion 3      valor 15
Posicion 4      valor 0
Posicion 5      valor 0
El indice del primero: 2
```

```
Posicion 1      valor 0
Posicion 2      valor 0
Posicion 3      valor 15
Posicion 4      valor 0
Posicion 5      valor 0
El indice del primero: 3

El indice del ultimo: 3

Se verifica su estado, si se encuentra vacía o no
0 : No esta vacía
```

Además, a realizar las operaciones *encolarC()* y *desencolarC()* se imprime los valores con sus respectivas posiciones de la **Cola**, para observar cómo está siendo modificada.
Este es el resultado final.

Previo2.c Cola Doble

El objetivo del programa es crear y utilizar las funciones de una Cola Doble en donde se encuentran en el archivo *colaDoble.c.* Las funciones son: *crearCola(), isEmpty(), encolarInicio(), encolarFinal(), desencolarFinal(), desencolarInicio(), mostrarValores(), mostrarIndices()*.

Las funciones son llamadas en el siguiente orden:

```
crearCola(6); /Crea una Cola con tamaño de 6.
isEmpty(cc) /Verifica e imprime el estado de la cola.
encolarFinal(&cc, 6); /Encola el elemento 6 en la última posición [0].
encolarFinal(&cc, 7); /Encola el elemento 7 en última posición [1].
encolarInicio(&cc, 4); /Encola el elemento 4 en la primera posición [7].
encolarInicio(&cc, 1); /Encola el elemento 1 en la primera posición [6].
desencolarFinal(&cc); /Desencola el último elemento [1].
desencolarInicio(&cc); /Desencola el primer elemento [6].
mostrarValores(cc); /Imprime los elementos y sus posiciones.
mostrarIndices(cc); /Muestra los índices en uso de la cola.
```

Además, en el momento de realizar las operaciones *encolarInicio(), encolarFinal(), desencolarInicio* y *desencolarFinal(),* se imprime los valores (llamando a la función *mostrarValors()*) de la Cola Doble con sus respectivas posiciones, para observar cómo está siendo modificada.
Los índices de la Cola Doble son:
Primero[5] = **4**
Ultimo[0] = **6**
Los valores concuerdan con el resultado final de la Cola.

```
Se muestran los índices
Primero = 6
Ultimo = 1
Disponibles = 4

Se verifica su estado, si se encuentra vacia o no
0 : No está vacía
```

```
Se hace un encolar al inicio
Posicion 0      valor 6

Posicion 1      valor 7

Posicion 2      valor 0

Posicion 3      valor 0

Posicion 4      valor 1

Posicion 5      valor 4
```

```
Se hace un desencolar al final
Posicion 0      valor 6

Posicion 1      valor 0

Posicion 2      valor 0

Posicion 3      valor 0

Posicion 4      valor 1

Posicion 5      valor 4
```

```
Se hace un desencolar al inicio
Posicion 0      valor 6

Posicion 1      valor 0

Posicion 2      valor 0

Posicion 3      valor 0

Posicion 4      valor 0

Posicion 5      valor 4
```

Ejercicio 1

El programa contiene la biblioteca "pila.h" para crear una **Pila**, se le solicitar al usuario por medio de un *ciclo for()*, que ingrese valores numéricos para llenar 10 posiciones de la Pila, y además, se devolverá el elemento de mayor tamaño que ingresó el usuario.

```
De que tamaño se desea la pila?. Se recomienda que sea de 8
8
Ingrese los valores para llenar la pila
0: 9
1: 10
2: 44
3: 77
4: 99
5: 1111
6: 231
7: 042
```

Para devolver el elemento de mayor tamaño se implementa un algoritmo en el cual, con la ayuda de 2 pilas vacías, un *ciclo for()* y una *sentencia if()* se ingresa el elemento mayor a la Pila **vacia2** y los demás, a la Pila **vacía1**.

El algoritmo consiste en:

- Recorrer las posiciones de los elementos de la **lista** de la estructura **Pila** (10 elementos) y si el elemento de dicha posición cumple que "es **mayor** al elemento siguiente de la **lista** y el elemento es distinto al elemento siguiente y a la vez, el elemento es **mayor** al elemento que se encuentra en la posición 0 de la lista de la **vacía2**", se ejecuta las 3 siguientes instrucciones.

```
Posicion 1      valor 9
Posicion 2      valor 10
Posicion 3      valor 44
Posicion 4      valor 77
Posicion 5      valor 99
Posicion 6      valor 1111
Posicion 7      valor 231
Posicion 8      valor 34
El tope es actual es el elemento: 34
```

- Se ejecuta la función *push()* para agregar a la Pila **vacía1** el elemento de la posición 0 de la Pila **vacia2**.
- Se elimina el elemento contenido de pila **vacia2** (ya que solo puede haber un elemento en esa Pila) y si esta no contiene ningún elemento, se le notifica al usuario y continúa con la siguiente línea.
- Se le agrega el elemento a la pila **vacia**,2 teniendo el elemento mayor actual.
- Si de lo contrario, no se cumple la *sentencia if()*, se ejecuta la función *push()* para agregar el elemento a la pila **vacia1**.

Al final, se imprime ambas pilas. La primera pila contiene los elementos menores y la segunda pila contiene el elemento mayor.

```
La pila ya está vacía.
La pila con los elementos menores es:
Posicion 1      valor 9
Posicion 2      valor 10
Posicion 3      valor 44
Posicion 4      valor 77
Posicion 5      valor 99
Posicion 6      valor 0
Posicion 7      valor 231
Posicion 8      valor 34
El tope es actual es el elemento: 34
```

```
El elemento mayor es: 1111
Posicion 1      valor 1111
Posicion 2      valor 0
Posicion 3      valor 0
Posicion 4      valor 0
Posicion 5      valor 0
Posicion 6      valor 0
Posicion 7      valor 0
Posicion 8      valor 0
El tope es actual es el elemento: 1111
```

Ejercicio 2.

El programa utiliza la biblioteca "*ejercicio2.h*" donde contiene las operaciones de la cola, la estructura **Cola** y la estructura de **Documento**. La estructura documento contiene:

- Una variable de tipo carácter con el nombre de **nombre**
- Una variable de tipo carácter con el nombre de **autor**.
- Una variable de tipo entero con el nombre de **numPaginas**.
- Una variable de tipo entero con el nombre **tamano**.

Se crea una **Cola** utilizando la función *crearCola()*. La cola será utilizada para guardar los documentos en cada posición de la **Cola**.

Para llenar los documentos:

- Se le solicita al usuario el número de documentos a registrar.
  Para tener la cantidad de documentos que el usuario ingresó, se crea variable con apuntador de tipo un **Documento**, y con *calloc()*, se reserva la memoria. Los espacios reservados es la cantidad de documentos ingresados.
- Se le solicitará al usuario por medio de un *ciclo for* la información de los documentos.
- Dentro del *ciclo for* y después de llenar las 4 variables del **documento**, se utiliza la función *encolarC(),* en donde el **documento** número **i** se va a encolar a la **Cola** creada.
- Después de llenar la cantidad de documentos deseados (cuando el *ciclo for()* termina), se manda a ejecutar la función *imprimirDatos(),* con parámetros de la **cantidad de documentos** y la **Cola.**
- Dentro de la función *imprimirDatos()*, se crea otro **Documento**, en donde se guardará el elemento a desencolar de la Cola. Para desencolar, se utiliza la función de tipo Documento, *desencolarC()*, recibe el parámetro de la Cola. Del documento desencolado, se imprime el **Nombre** y su **tiempo de ejecución.** Esta se obtiene por medio de multiplicar la cantidad de páginas ingresadas por 4 (cantidad de segundos que tarda en imprimir). El tiempo de ejecución se guarda en una variable de tipo entero llamada **tiempo**, en donde se guarda el valor en segundos.
- Por último, el valor **tiempo de ejecución** es impreso a través de una función *imprimirTiempo()*, que recibe de parámetro la variable **tiempo**, en donde, con una *secuencia if(),* si:
  - El **tiempo** es mayor o igual a 3600 (equivalente a 1 hora):
    - `horas = tiempo / 3600; /Se guarda la división de tiempo entre 3600, repre-senta las horas.`

```
tiempo -= horas * 3600; /Se le restará el valor de la multiplicación de la variable horas por 3600. Esto es para restarle los segundos ocupados por horas.
minutos = tiempo / 60; /De los segundos restantes, se divide entre 60 y se guarda en la variable, representando los minutos
tiempo -= (minutos * 60); /Se le restará el valor de la multiplicación de minutos por 60. Esto es para quitarle los segundos ocupados por la variable minutos. Al final la variable tiempo se queda con los segundos y este es menor a 60.
```

- Y si no es mayor o igual a 3600:

```
minutos = tiempo / 60; /Se divide los segundos entre 60 dando la cantidad de minutos.
tiempo -= (minutos * 60); /La cantidad de minutos se multiplica por 60, para obtener los segundos y restárselos a la variable tiempo, ya que estos están transformados en minutos. Al final quedará la cantidad en segundos y este debe de ser menor a 60.
```

Ejemplo:

Se ingresó 2 documentos.

```
Ingrese el nombre del Documento [1]:
Hábitos Atómicos
Ingrese el Autor del Documento [1]:
James Clear
Ingrese el Número de Páginas del Documento [1]:
40
Ingrese el tamaño del Documento[1]:
1
```

```
Nombre: 'Hábitos Atómicos'
Tiempo de Impresion:

        Minutos: 2
        Segundos: 40

Nombre: 'Si lo crees, lo creano se'
Tiempo de Impresion:

        Minutos: 5
        Segundos: 12
```

2 * 60 = 120 segundos
120 + 40 = 160 segundo
160 / 4 = 40 paginas

```
Ingrese el nombre del Documento [2]:
Si lo crees, lo creas
Ingrese el Autor del Documento [2]:
Brian Tracy
Ingrese el Número de Páginas del Documento [2]:
78
Ingrese el tamaño del Documento[2]:
3
```

5 * 60 = 300 segundos
300 + 12 = 312 segundos
312 / 4 = 78 paginas

```
Tiempo total:
        Minutos: 7
        Segundos: 53
```

Segundos: 40 + 12 = 52
Minutos: 2 + 5 = 7

De esta manera, se comprueba los resultados del tiempo de impresión que concuerden con el número de páginas por documento que ingresó el usuario. Aunque, en el tiempo total aparezca que son 53 segundos, se revisó el algoritmo y no encontré ese pequeño error que marca una diferencia de 1 segundo.

Si fuéramos en un caso extremo de páginas, esto sucedería.

```
Ingrese el nombre del Documento [1]:
Club de las 5
Ingrese el Autor del Documento [1]:
Robin S. Sharma
Ingrese el Número de Páginas del Documento [1]:
1639
Ingrese el tamaño del Documento[1]:
100
```

```
Nombre: 'Club de las 5 '
Tiempo de Impresion:

        Horas: 1
        Minutos: 49
        Segundos: 16


Tiempo total:
        Horas: 1
        Minutos: 49
        Segundos: 17
```

1 * 60 = 60 minutos
49 + 60 = 109 minutos
109 * 60 = 6540 segundos
6540 + 16 = 6556 segundos
6556 / 4 = 1639 paginas

Se comprueba los resultados obtenidos del tiempo de impresión, sin embargo, como el ejemplo anterior, tiene una diferencia de **1** segundo en el tiempo total.

Ejercicio 3
El programa utiliza el archivo "*cola.c*" para utilizar la estructura **Cola** con un tamaño de 8 posiciones, y además, se utilizan las operaciones de una Cola y están escritas en forma de funciones con los nombres de *encolarC()* y *desencolarC()*.

Se ejecutan las funciones de la siguiente manera:

```
1. encolarC(&queue, 8);
2. encolarC(&queue, 14);
3. encolarC(&queue, 22);
4. encolarC(&queue, 28);
5. encolarC(&queue, 30);
6. encolarC(&queue, 33);
7. encolarC(&queue, 40);
8. desencolarC(&queue);
9. desencolarC(&queue);
10.desencolarC(&queue);
11.encolarC(&queue, 50);
12.desencolarC(&queue);
```

```
Se encola el valor de 40
Posicion 1      valor 8
Posicion 2      valor 14
Posicion 3      valor 22
Posicion 4      valor 28
Posicion 5      valor 30
Posicion 6      valor 33
Posicion 7      valor 40
Posicion 8      valor 0
El tope es actual es: 8
```

```
Se desencola el primer elemento
Posicion 1      valor 0
Posicion 2      valor 14
Posicion 3      valor 22
Posicion 4      valor 28
Posicion 5      valor 30
Posicion 6      valor 33
Posicion 7      valor 40
Posicion 8      valor 0
El tope es actual es: 14
```

Cada que se mandan a llamar las operaciones para agregar o quitar un elemento de la Cola Circular, se imprime la **Cola** para ver los cambios realizados por cada operación.

Después de las operaciones, la cola contiene 4 espacios disponibles y 4 espacios utilizados y con índice **primero** de 6 y **ultimo** de 8

```
Posicion 1      valor 0
Posicion 2      valor 0
Posicion 3      valor 0
Posicion 4      valor 0
Posicion 5      valor 30
Posicion 6      valor 33
Posicion 7      valor 40
Posicion 8      valor 50
El tope es actual es: 30
```

Ahora, si se utiliza la cola con un tamaño de 5 elementos y utilizamos las mismas operaciones, no es posible encolar la operación número 6 ni 7, debido a que la cola se encuentra llena.
Ejemplo de la *operación #7*. No se agrega a la cola.

```
Se encola el valor de 40
Posicion 1      valor 8
Posicion 2      valor 14
Posicion 3      valor 22
Posicion 4      valor 28
Posicion 5      valor 30
El tope es actual es: 8
```

Así está la cola al final de las operaciones, con índice **primero** de 5 y el **ultimo** como 6

```
Posicion 1      valor 0
Posicion 2      valor 0
Posicion 3      valor 0
Posicion 4      valor 0
Posicion 5      valor 30
El tope es actual es: 30
```

Ejercicio 4
El programa utiliza el archivo *"colaDoble.c"* en donde contiene la estructura de la **Cola Doble** y operaciones escritas en funciones para un mejor manejo.
Para empezar, se debe de crear la Cola Doble llamando a la función *crearCola()* en donde se le pasa de parámetro el tamaño deseado, el cual, es 8. Para obtener los mismos resultados en los 3 estados, se debe de llamar las funciones de la siguiente manera:

Estado 1:

```
encolarFinal(&doble, 101); /Agrega en la posición 0 de la Cola el valor de 101
encolarFinal(&doble, 102); /Agrega en la posición 1 de la Cola el valor de 102
encolarFinal(&doble, 103); /Agrega en la posición 2 de la Cola el valor 103
encolarFinal(&doble, 104); /Agrega en la posición 3 de la Cola el valor 103
encolarFinal(&doble, 105); /Agrega en la posición 4 de la Cola el valor 104.
desencolarInicio(&doble); /Se elimina el elemento de la posición 0.
mostrarValores(doble); /Imprime los valores actuales
```

```
Estado 1:
Posicion 0      valor 0

Posicion 1      valor 102

Posicion 2      valor 103

Posicion 3      valor 104

Posicion 4      valor 105

Posicion 5      valor 0

Posicion 6      valor 0

Posicion 7      valor 0
```

Estado 2:

```
desencolarInicio(&doble); /Elimina el elemento de la posición 1.
encolarInicio(&doble, 99); /Agrega en la posición 1 el elemento 99
encolarInicio(&doble, 98); /Agrega en la posición 0 el elemento 98
encolarInicio(&doble, 97); /Agrega en la posición 7 el elemento 97
desencolarFinal(&doble); /Elimina el elemento de la posición 4
encolarFinal(&doble, 120); /Agrega en la posición 4 el elemento 120.
encolarFinal(&doble, 121); /Agrega en la posición 5 el elemento 121.
mostrarValores(doble); /Imprime los valores actuales
```

```
Estado 2:
Posicion 0      valor 98

Posicion 1      valor 99

Posicion 2      valor 103

Posicion 3      valor 104

Posicion 4      valor 120

Posicion 5      valor 121

Posicion 6      valor 0

Posicion 7      valor 97
```

Estado 3:

```
encolarFinal(&doble, 122); /Agrega en la posición 6 el elemento 122.
desencolarInicio(&doble); /Elimina el elemento de la posición 1
desencolarInicio(&doble); /Elimina el elemento de la posición 0
desencolarInicio(&doble); /Elimina el elemento de la posición 7
mostrarValores(doble); /Imprime los valores actuales
```

```
Estado 3:
Posicion 0      valor 0

Posicion 1      valor 0

Posicion 2      valor 103

Posicion 3      valor 104

Posicion 4      valor 120

Posicion 5      valor 121

Posicion 6      valor 122

Posicion 7      valor 0
```

Conclusiones.

La práctica contiene ejemplos, aplicaciones, procedimientos de las estructuras lineales pila, cola circular y cola doble. Esto cumple los objetivos de la práctica. Además, la práctica se cumplió en su totalidad abarcando los puntos solicitados

Al conocer las estructuras lineales, se abren las puertas a nuevas posibilidades del uso de los

datos del código implementando distintos algoritmos donde se puede mejorar la eficiencia del programa y cuidar de la memoria. Además, las colas se encuentran en todos lados y se pueden utilizar de todo tipo de datos, esto lo hace más interesnte.

La práctica me costó trabajo para implementar la lógica correcta para cumplir lo solicitado, pero fuera de esto, se que me sirvió mucho pues se me ocurrían muchísismos algoritmos, tener un mejor criterio y corregir mi código.