

Semana 6: TDA pila y TDA cola

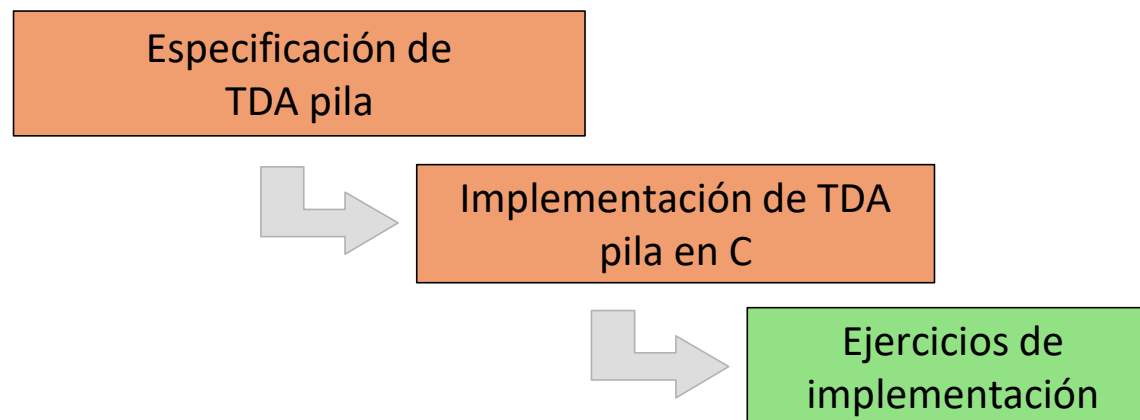
Semestre 2 - 2024



Análisis de Algoritmos y Estructura de Datos

Departamento de Ingeniería Informática



Ruta de trabajo



-  Trabajo en sala general
-  Trabajo en grupo (*implementación individual*)

Especificación e implementación de TDA pila

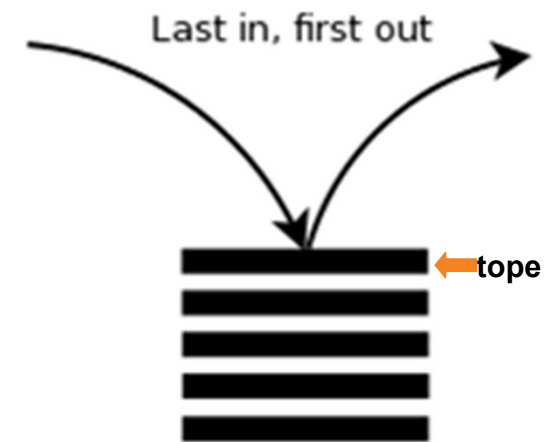




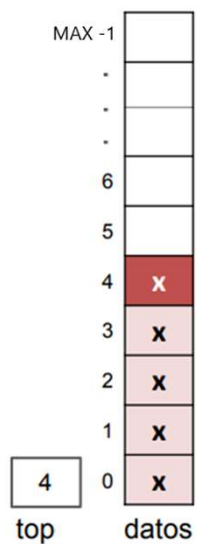
Especificación de TDA pila

- **Estructura de datos:**

- Una pila es una secuencia de elementos que cumple la propiedad **LIFO** (last in, first out), es decir, el último elemento que se introduce en la pila, será el primer elemento en salir de ella
- **Tope de la pila:** extremo de la lista por donde se introducen y extraen los elementos



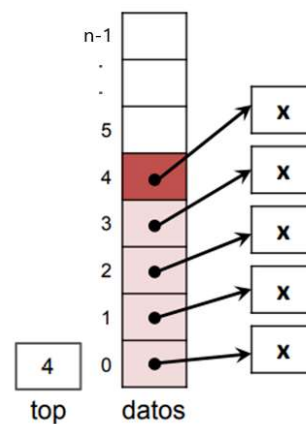
Representación de TDA pila



```

registro pila:
  NUM max
  NUM top
  <tipo de datos> datos[max]

pila *p
  
```



```

registro info:
  NUM x:
  CHAR Y
  :
registro pila:
  NUM max
  NUM top
  info datos[max]

Pila *p
  
```



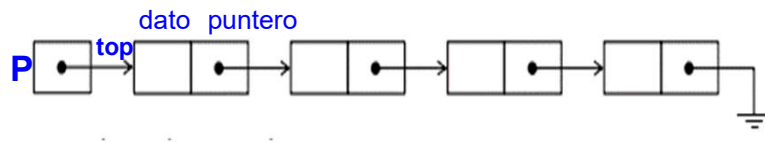
Representación de TDA pila

ALTERNATIVA 1: (datos en el mismo registro)

registro **nodo**:
<tipo> dato
nodo *puntero

registro **pila**:
nodo *tope

pila *p



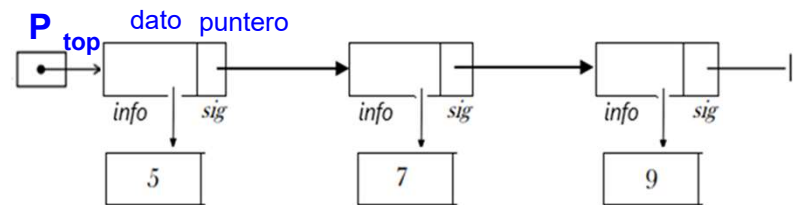
ALTERNATIVA 2 (caso general)

registro **info**:
num dato

registro **nodo**:
info *dato
nodo *puntero

registro **pila**:
nodo *tope

pila *p





Especificación de TDA pila

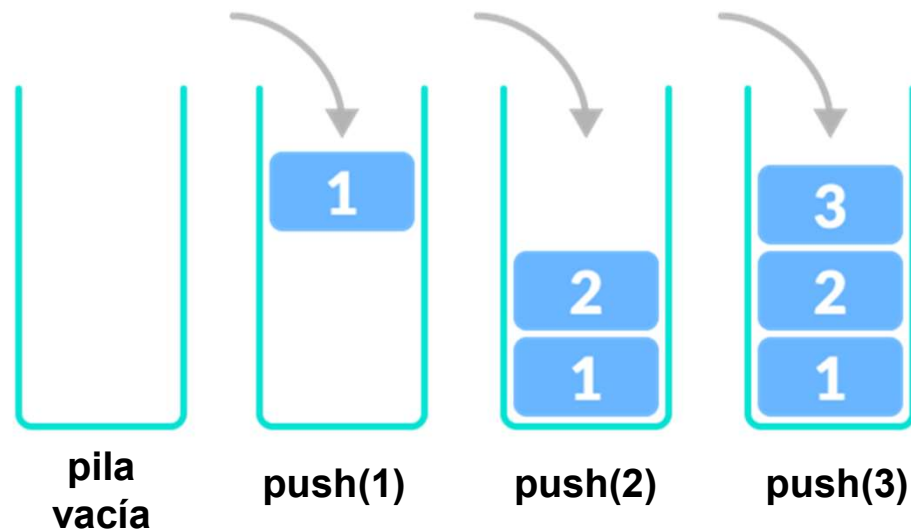
- Operaciones:

- **apilar(pila,dato)**: agrega un elemento al comienzo de la pila (tope) → **push**
- **desapilar(pila)**: quita el primer elemento en el tope de la pila, sin devolverlo → **pop**
- **tope(pila)**: devuelve el primer elemento de la pila, sin extraerlo → cima, **top**
- **es_pila_vacia(pila)**: comprueba si la pila está vacía

Operación *apilar*

- **Ejemplo:**

- Supongamos que tenemos una pila de enteros vacía y queremos apilar los valores 1, 2 y 3:

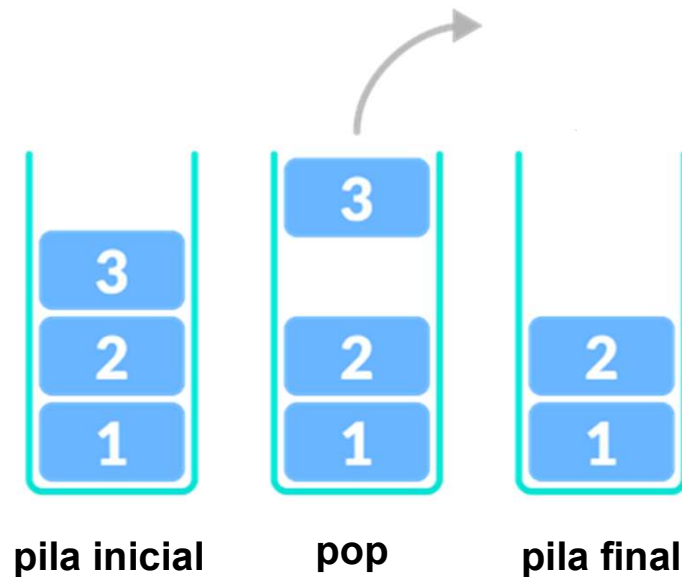




Operación *desapilar*

- **Ejemplo:**

- Supongamos que tenemos una pila de enteros con los valores 1,2 y 3, siendo 3 el elemento del tope de la pila. Necesitamos extraer el elemento del tope:

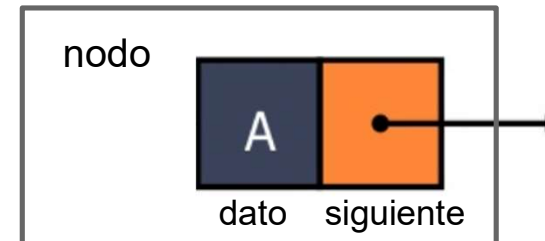




Implementación de estructura de datos de TDA Pila

- La estructura de datos que representa un nodo es la siguiente:

```
typedef struct nodo{  
    int dato;  
    struct nodo *siguiente;  
}nodo;
```

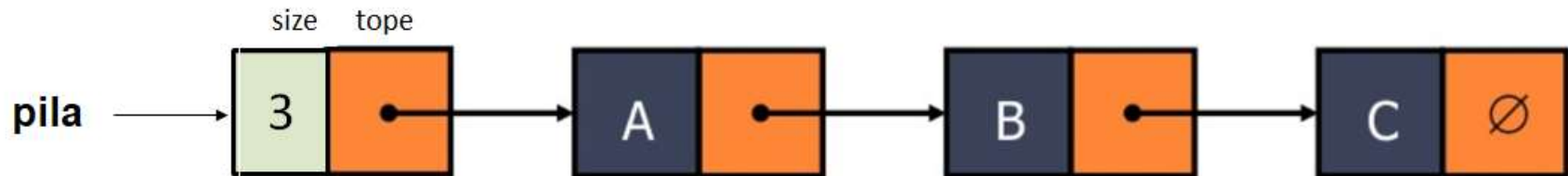




Implementación de estructura de datos de TDA lista enlazada

- La estructura de datos que representa una lista es la siguiente:

```
typedef struct{  
    int size;  
    nodo *tope;  
}pila;
```





Implementación de operaciones de TDA pila

```
pila* nueva_pila(void){
    pila *p = (pila*)malloc(sizeof(pila));
    p->size = 0;
    p->tope = NULL;
    return p;
}

int es_pila_vacia(pila *p){
    if(p->size == 0){
        return 1;
    }
    else{
        return 0;
    }
}
```

Ejercicios de implementación



Actividad de implementación 1

1. Descargar los archivos **TDApila.h**, **TDApila_implementacion.c** y **lab07-pila.c**
2. Compilar y ejecutar **lab07-pila.c** y **TDApila_implementacion.c**
3. Implementar las siguientes funciones:

```
void apilar(pila *p, int d);  
  
void desapilar(pila *p);  
  
nodo* tope(pila *p);
```

Operación APILAR

- La operación *apilar* (PUSH) permite agregar un elemento a una pila, siempre por el tope

```
apilar(pila *p, num valor)
    nodo *n_nodo ← nodo_vacio()
    n_nodo→dato ← valor
    n_nodo→siguiente ← p→tope
    p→tope ← n_nodo
    p→size ← p→size + 1
```

} O(1)

Orden de complejidad: $O(1)$

Similar a inserción al inicio de lista enlazada simple

Operación TOPE

- La operación *tope* (TOP) devuelve el primer elemento de la pila, sin extraerlo (puntero a nodo)

```
tope(pila *p): nodo*  
    if (es_pila_vacia(p) = 0) then  
        return p→tope  
    return NULL
```

} O(1)

Orden de complejidad: O(1)

Operación DESAPILAR

- La operación *desapilar* (POP) permite extraer el elemento del tope de una pila, pero no devuelve dicho elemento

```
desapilar(pila *p)
  if (es_pila_vacia(p) = 0) then
    nodo *ptr_aux ← p→tope
    p→tope ← p→tope→siguiente
    p→size ← p→size - 1
    liberar(ptr_aux)
```

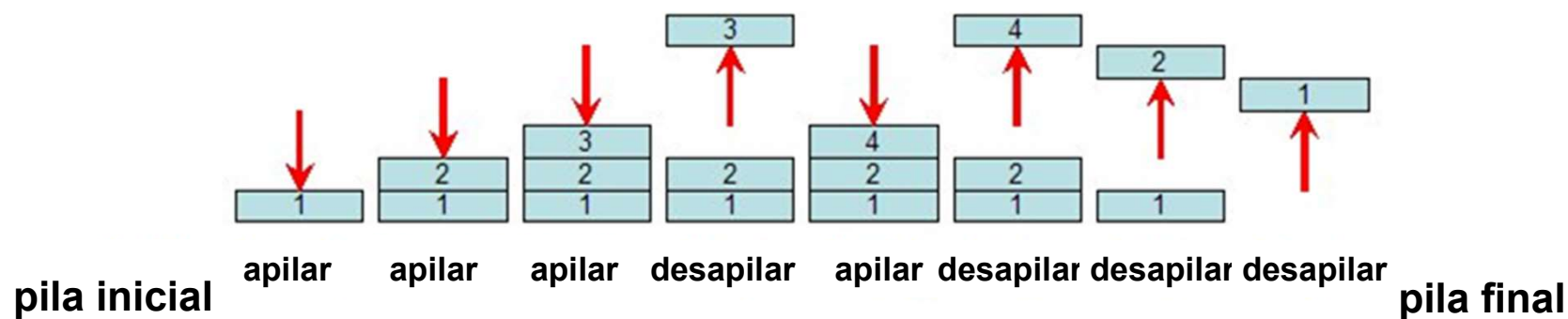
} O(1)

Orden de complejidad: O(1)

Similar a eliminación al inicio de lista enlazada simple

Actividad de implementación 2

Evaluar todas las funciones creadas representando el siguiente ejemplo a través de llamadas desde función *main()* de **lab07-pilas.c** y mostrando tope de pila después de cada operación:



Actividad de implementación 3

1. Usando operaciones implementadas, escribir una función en C para que, dada una pila y un valor entero, devuelva *verdadero* (1) en caso de encontrar el valor en la pila, y *falso* (0) en caso contrario.

```
int busca_dato_pila(pila *p, int d)
```

1. Evaluar la función escrita generando secuencia de llamadas desde función *main()* en **lab07-pila.c** para realizar las siguientes operaciones:
 - Crear una pila
 - Apilar 5 valores sucesivos (1..5)
 - Buscar el dato 4
 - Buscar el dato 8

Actividad de implementación 3

```
busca_dato_pila (pila *p, num dato): num
  pila *pila_aux <- nueva_pila()
  num valor_tope
  num encontrado <- 0
  while (es_pila_vacia(p) = 0) and (not encontrado) do
    valor_tope <- tope(p) ->dato
    if valor_tope = dato then
      encontrado <- 1
      apilar(pila_aux, valor_tope)
      desapilar(p)
  while (es_pila_vacia(pila_aux) = 0) do
    apilar(p, tope(pila_aux) ->dato)
    desapilar(pila_aux)
  return encontrado
```

Actividad de implementación 4

Usando operaciones implementadas, escribir una función en C que muestre por consola los valores de una pila.

```
void imprime_pila(pila *p)
```

Evaluar la función escrita generando secuencia de llamadas desde función *main()* en **lab07-pila.c** para realizar las siguientes operaciones:

- Crear una pila
- Apilar 5 valores sucesivos (1, ..., 5)
- Imprimir
- Desapilar
- Imprimir



Actividad de implementación 5

- Implemente en C una función para intercambiar los elementos de dos posiciones distintas en una pila de números enteros. Para su solución considere que el elemento en la posición nro. 1 es el tope de la pila (el que está más arriba). Por ejemplo, en la figura se muestra el intercambio de los elementos de las posiciones nro. 2 (12) y nro. 4 (6) de una pila de números. Verifique el funcionamiento de la función con la pila del ejemplo.

38
12
22
6
20
18

38
6
22
12
20
18

Actividad de implementación 5

```
intercambia_elementos(pila *p, num pos_1 , num pos_2):  
    pila *pila_antes <- nueva_pila()  
    pila *pila_entre <- nueva_pila()  
    num pos_actual <- 1  
    MIENTRAS (pos_actual <> pos_1) HACER  
        apilar(pila_antes, tope(p) ->dato)  
        desapilar(p)  
        pos_actual <- pos_actual + 1  
    num valor_1 <- tope(p) ->dato  
    desapilar(p)  
    pos_actual <- pos_actual + 1  
    MIENTRAS (pos_actual <> pos_2) HACER  
        apilar(pila_entre, tope(p) ->dato)  
        desapilar(p)  
        pos_actual <- pos_actual + 1  
    num valor_2 <- tope(p) ->dato  
    desapilar(p)  
    apilar(p, valor_1)  
    MIENTRAS (es_pila_vacia(pila_entre) = 0) HACER  
        apilar(p, tope(pila_entre) ->dato)  
        desapilar(pila_entre)  
    apilar(p, valor_2)  
    MIENTRAS (es_pila_vacia(pila_antes) = 0) HACER  
        apilar(p, tope(pila_antes) ->dato)  
        desapilar(pila_antes)
```

Para seguir aprendiendo...



Ejercicio de implementación 1

Suponga que necesitamos encontrar el valor de una expresión aritmética simple que involucra la multiplicación y la suma de dígitos, por ejemplo:

$$5 * ((9 + 8) * (4 * 6)) + 7$$

En la notación sufija, la expresión que necesitamos evaluar está en una forma en la que cada operador aparece después de sus dos argumentos, en lugar de entre ellos:

$$5\ 9\ 8\ +\ 4\ 6\ *\ * 7\ +\ *$$

Ejercicio de implementación 1

Construya un programa en C para evaluar una expresión sufija. Por ejemplo, la expresión:

$$5\ 9\ 8 + 4\ 6 * * 7 + *$$

Idea: moviéndose de izquierda a derecha, se interpreta cada operando como el comando para guardar el operando en una pila y cada operador como el comando para sacar los dos operandos de la pila, realizar la operación y guardar en la pila el resultado.

Para su solución cree un nuevo programa cliente y utilice el TDA pila que construyó

Ejercicio de implementación 2

1. Tomando como ejemplo le TDA pila crear **TDACola.h** y **TDACola_implementacion.c**.
Específicamente, poner atención a:

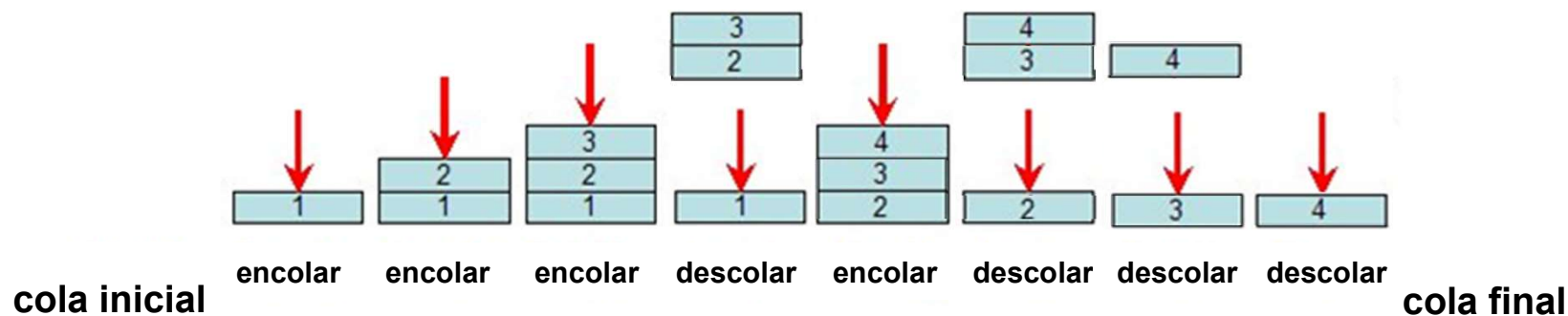
a) **Estructura de datos** – TDA cola implementado como una lista enlazada simple con frente y final

b) **Operaciones** - deben existir las siguientes funciones:

- cola* nuevaCola ()
- int esColaVacia(cola* c)
- encolar(cola *c, int dato)
- descolar(cola *c)
- nodo* frente(cola *c)
- nodo* final(cola* c)

Ejercicio de implementación 2

2. Crear archivo **lab07-colas.c** (similar a lab07-pilas.c) y evaluar funcionamiento de implementación de TDA cola, representando el siguiente ejemplo a través de llamadas desde función *main()* y mostrando frente y final de cola después de cada operación:



Entrega de actividad de laboratorio

- Entrega obligatoria
- Subir actividades 3, 4 y 5 de esta sesión en buzón de Campus Virtual, en único archivo **s6_coordinacion_apellido_nombre.zip**
- Se espera TDAlista.h, TDAlista_implementacion.c y lab07-pila.c modificados para responder actividades 3, 4 y 5
- Plazo: durante el **horario** laboratorio de cada coordinación