Pilas

Especificación genérica

Pilas

Pilas

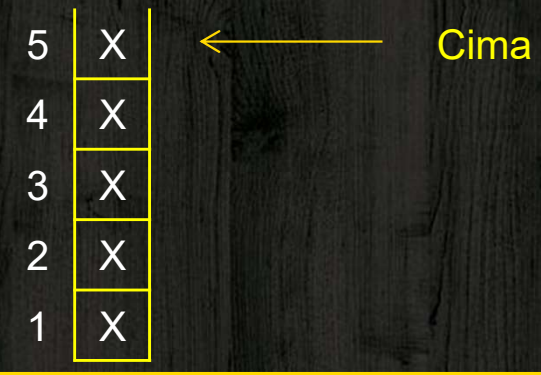
- Una pila es un tipo especial de lista en la que todas las inserciones y supresiones tienen lugar en un extremo denominado *Tope*.
- A las pilas se les llama también “Listas LIFO” (Last in first out) o listas “último en entrar, primero en salir”.
- Una pila es una versión restringida de una lista enlazada.
- A una pila se le pueden añadir y retirar nuevos nodos únicamente de su parte superior.

- El modelo intuitivo de una pila es precisamente una pila de fichas de póquer puesta sobre una mesa, o de libros sobre el piso, o de platos en una estantería, situaciones todas en las que sólo es conveniente quitar o agregar un objeto del extremo superior de la pila, al que se denominará en lo sucesivo “tope”.

- Un tipo de datos abstracto de la familia PILA incluye a menudo las cinco operaciones siguientes:

1. **ANULA(P)** convierte la pila en una pila vacía.
2. **TOPE(P)** devuelve el valor del elemento de la parte superior de la pila P.
3. **SACA(P)**, en inglés POP, suprime el elemento superior de la pila.

4. **METE(x, P)**, en inglés PUSH, inserta el elemento *x* en la parte superior de la pila P. el anterior tope se convierte en el siguiente elemento, y así sucesivamente.
5. **VACIA(P)** devuelve verdadero si la pila P esta vacía, y falso en caso contrario.



Implementación estática

Pilas

Implementación estática

- Para implementar una pila de manera estática mediante un arreglo se tiene en cuenta el hecho de que las inserciones y las supresiones ocurren solo en la parte superior.
- Se puede anclar la base de la pila a la base del arreglo y dejar que la pila crezca hacia la parte superior del arreglo.
- Una variable llamada tope indicará la posición actual del primer elemento de la pila

```
void push(int pila[], int* tope, int v)
{
    if(!full(*tope))
         pila[++(*tope)] = v;
    else
         printf("\nNo es posible agregar un elemento\n");
}
```

```
int full(int tope)
{
    if(tope == MAX-1)
         return 1;
    else
         return 0;
}
```

```
int pop(int pila[], int* tope)
{
    if(!empty(*tope))
         return (pila[(*tope)--]);
    else
         printf("\nNo es posible extraer un elemento\n");
}
```

```
int empty(int tope)
{
    if (tope == -1)
         return 1;
    else
         return 0;
}
```

```
#define MAX 3

int main( )
{
    int tope = -1;
    int pila[MAX];

    push(pila, &tope, 1);
    push(pila, &tope, 2);
    push(pila, &tope, 3);

    while(!empty(tope))
    {
         valor = pop(pila, &tope);
         printf("%i\n",  valor);
    }
}
```

Implementación dinámica

Pilas

Estructura Pila (LIFO)

- La pila (stack) es una estructura lineal sobre la que rigen ciertas restricciones a la hora de agregar o quitar elementos.
- A diferencia de las listas enlazadas en las que desarrollamos operaciones para insertar, agregar y eliminar nodos sin ningún tipo de limitación, decimos que la pila es una estructura lineal restrictiva de tipo LIFO(Last In First Out).
- Esto indica que el último elemento que ingresó a la pila debe ser el primero en salir.

El nodo

- Las estructuras dinámicas se forman “enlazando” nodos.

- Un nodo representa un conjunto de uno o más valores, más un puntero haciendo referencia al siguiente nodo de la colección.

```
typedef struct Nodo
{
    int valor;
    struct Nodo* sig;
}Nodo;
```

Un nodo simplemente es una estructura que define valores más una referencia (puntero de tipo Nodo*) para apuntar al siguiente nodo de la colección.

Implementación dinámica de la estructura pila

- Implementaremos la pila sobre una lista enlazada para la cual solo desarrollaremos dos operaciones: poner (apilar un elemento) y sacar (obtener y eliminar un elemento de la pila).

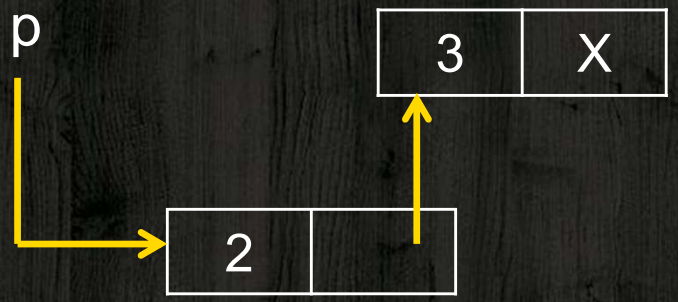
Operaciones poner (push) y sacar (pop)

- Supongamos que queremos poner en una pila los elementos del siguiente conjunto: {3, 2, 1}.

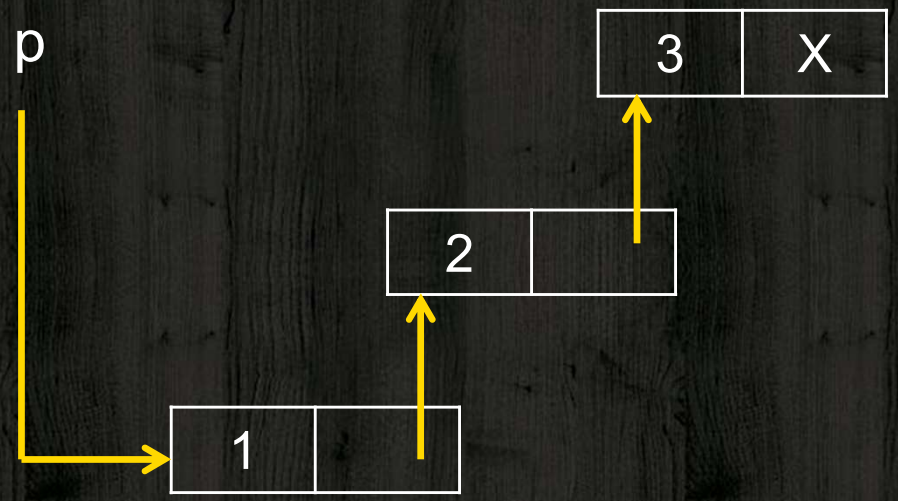
- Para esto, definimos un puntero **p** de tipo **Nodo*** inicializado en NULL y luego agregamos cada uno de los elementos del conjunto al inicio de la lista apuntada por **p.**

- Agregamos el 3 (primer elemento del conjunto):

- Agregamos el 2 (segundo elemento del conjunto):



- Agregamos el 1 (último elemento del conjunto):



- Luego, para sacar un elemento de la pila siempre tomaremos el primero de la lista.

Función Poner (Push)

```
void push(Nodo** p, int v)
{
        Nodo* nuevo = (Nodo*) malloc(sizeof(Nodo));
        nuevo->valor = v;
        nuevo->sig = *p;
        *p = nuevo;
}
```

Función Sacar (Pop)

```
int pop(Nodo** p)
{
	Nodo* aux = *p;
	int ret = aux->valor;

	*p = aux->sig;
	free(aux);

	return ret;
}
```

Notemos que al finalizar el programa no es necesario liberar la memoria ocupada por la pila porque la función ***sacar***, además de retornar el valor del elemento ubicado en la cima, desenlaza el nodo y lo libera con la función ***free*** de C.

Determinar si la pila tiene elementos o no

- La operación **pilaVacia** retorna ***true*** o ***false*** según la pila tenga o no elementos apilados.
- Esta operación es innecesaria ya que la pila estará vacía cuando el puntero al primer nodo de la lista sea NULL

```
int pilaVacia(Nodo* p)
{
    return p==NULL;
}
```

Recordemos que en C los valores booleanos se manejan con valores enteros.

Así, el número **0** (cero) equivale al valor booleano ***false*** y el número **1** (uno) o cualquier otro valor diferente de 0 equivale a ***true***.

Ejemplo de uso de una pila

- Se ingresa por teclado un conjunto de valores que finaliza con la llegada de un 0 (cero).
- Se pide mostrar los elementos del conjunto en orden inverso al original.
- Para resolver este programa utilizaremos una pila en la que apilaremos los valores a medida que el usuario los vaya ingresando.
- Luego, mientras que la pila no este vacía, sacaremos uno a uno sus elementos para mostrarlos por pantalla.