

USO

@

```
int main()  
{  
    return 0;  
}
```


Facultad de Ingeniería  
y Ciencias Naturales



*Pilas*

*Programación III.*

# Objetivos

- 
- Conocer el modelo del tipo abstracto de datos pila
  - Elaborar ejercicios de aplicación de pilas

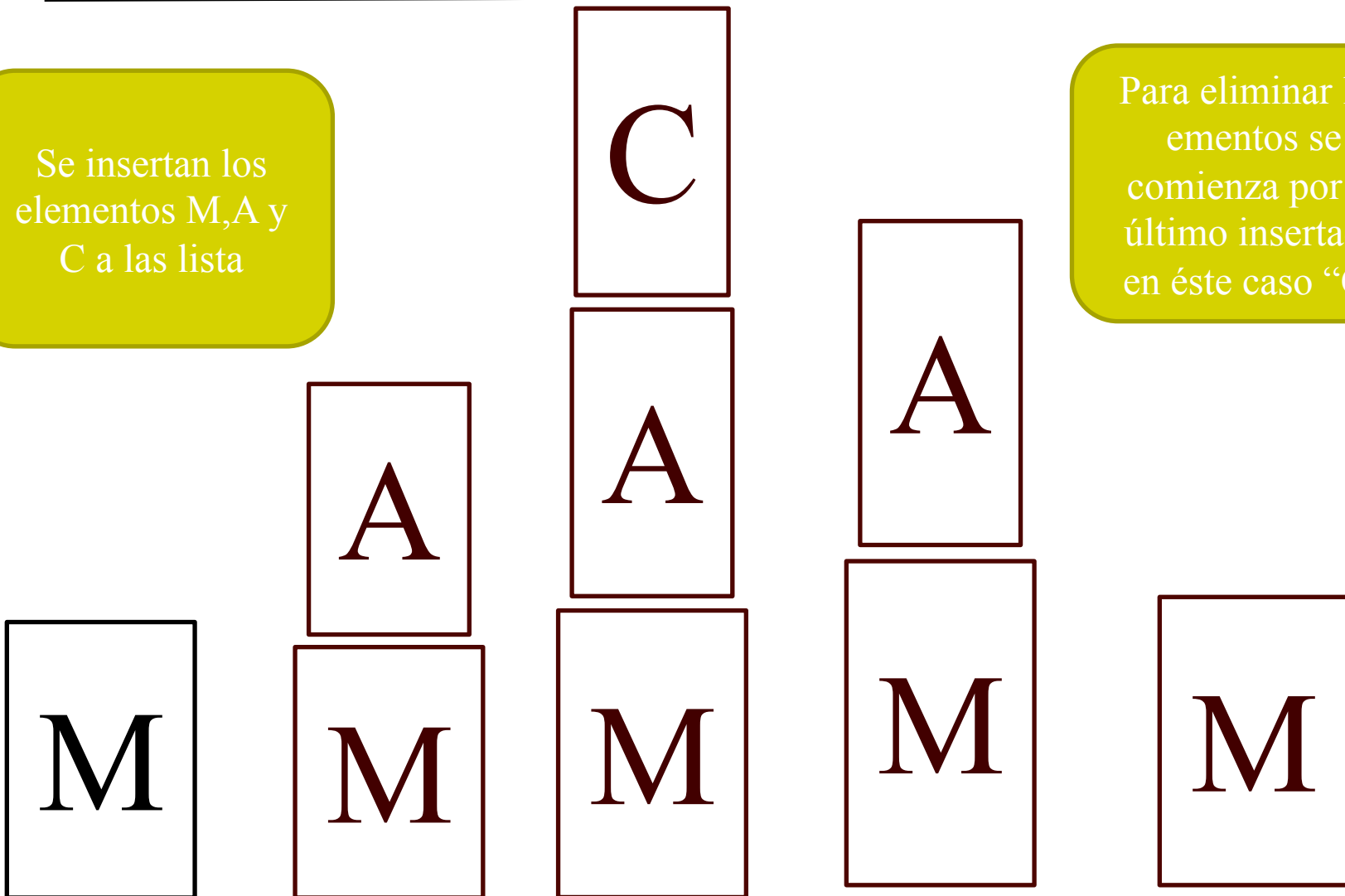
# Pilas

- Una pila(stack) es un tipo especial de lista abierta en la que sólo se pueden insertar y eliminar nodos en uno de los extremos de la lista. Estas operaciones se conocen como (push)"empujar" y "tirar"(pop). Además, las escrituras de datos siempre son inserciones de nodos, y las lecturas siempre eliminan el nodo leído.

- 
- ❑ Los elementos de la lista se añaden o se quitan(borran) de la misma sólo por su parte superior(cima).
  - ❑ Su característica fundamental es que al extraer se obtiene siempre el último elemento que acaba de insertarse.

# Lifo (Last Input, First Output)

Se insertan los  
elementos M, A y  
C a las lista



Para eliminar los  
elementos se  
comienza por el  
último insertado  
en éste caso "C"

# Implementacion de Pilas

- ❑ Las pilas pueden implementarse mediante arrays, mediante punteros o listas enlazadas.
- ❑ Una pila puede estar vacia o llena. Si un programa intenta sacar elementos de una lista vacia se producirá un “underflow”(desbordamiento negativo), si se intenta agregar un elemento a una lista llena(arrays) entonces se producirá un error llamado desbordamiento(overflow).

# Operaciones Básicas de las pilas

---

- ❑ Push: Añadir un elemento al final de la pila.
- ❑ Pop: Leer y eliminar un elemento del final de la pila.

# Push

- ❑ Partiremos de que ya tenemos el nodo a insertar y, por supuesto un puntero que apunte a él, además el puntero a la pila valdrá NULL:

El proceso es muy simple, bastará con que:

- ❑ nodo->siguiente apunte a NULL.
- ❑ Pila apunte a nodo

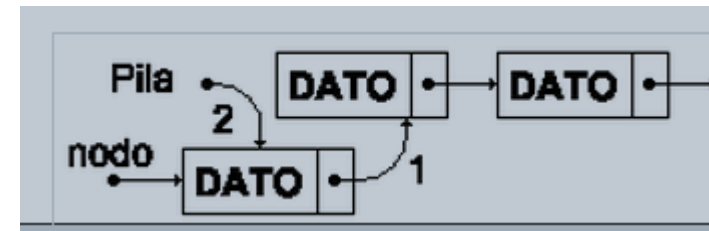
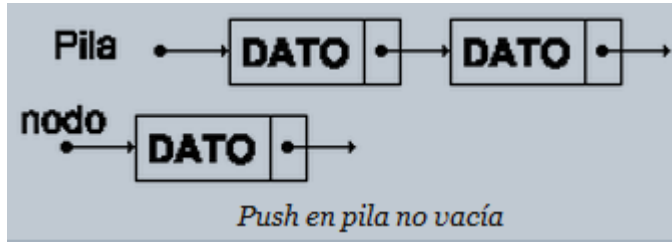


- 
- Por esta razón también se conocen como estructuras de datos LIFO (del inglés *Last In First Out*).

# Push en una pila no vacía

- ❑ Podemos considerar el caso anterior como un caso particular de éste, la única diferencia es que podemos y debemos trabajar con una pila vacía como con una pila normal.
- ❑ De nuevo partiremos de un nodo a insertar, con un puntero que apunte a él, y de una pila, en este caso no vacía:

- ❑ El proceso sigue siendo muy sencillo:
- ❑ Hacemos que nodo->siguiente apunte a Pila.
- ❑ Hacemos que Pila apunte a nodo.



# Pop, leer y eliminar un elemento

---

- Ahora sólo existe un caso posible, ya que sólo podemos leer desde un extremo de la pila. Partiremos de una pila con uno o más nodos, y usaremos un puntero auxiliar, nodo:

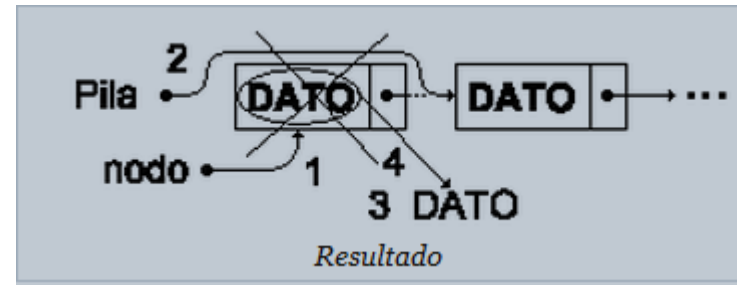
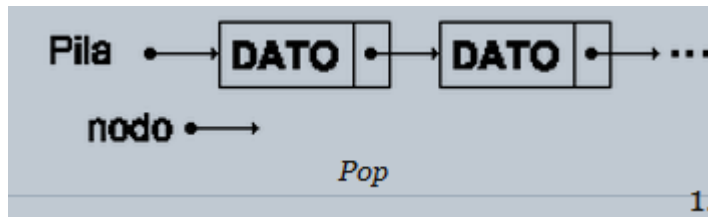
# Aplicaciones

---

- ❑ Las pilas se utilizan en muchas aplicaciones que utilizamos con
- ❑ frecuencia.

- ❑ Hacemos que nodo apunte al primer elemento de la pila, es decir a Pila.
- ❑ Asignamos a Pila la dirección del segundo nodo de la pila: Pila->siguiente.
- ❑ Guardamos el contenido del nodo para devolverlo como retorno, recuerda que la operación pop equivale a leer y borrar.

- ❑ Liberamos la memoria asignada al primer nodo, el que queremos eliminar.
- ❑ Si la pila sólo tiene un nodo, el proceso sigue siendo válido, ya que el valor de Pila->siguiente es NULL, y después de eliminar el último nodo la pila quedará vacía, y el valor de Pila será NULL



# Definición de la Pila

```
#ifndef NODO_H_INCLUDED
#define NODO_H_INCLUDED
class nodo{
    private:
        int valor;
        nodo *siguiente;
    public:
        nodo(int v, nodo *sig = NULL)
            this->valor = v;
            this->siguiente = sig;
        }
    friend class pila;
};
typedef nodo *pnodo;
#endif // NODO H INCLUDED
```




# Insertar un elemento en la pila

```
pila pl;
```

```
pl.meter(1);
```

```
pl.meter(2);
```



```
void meter(int v) {
    pnode nuevo;
    if(pilaVacía()) {
        primero = new nodo(v, primero);
    }
    else {
        ultimo();
        actual->siguiente = new nodo(v, NULL);
    }
}
```

```
void sacar() {
    pnode anterior, nodo;
    nodo = primero;
    anterior = NULL;
    while (nodo) {
        if (nodo ->siguiente == NULL) {
            if (anterior == NULL) {
                primero = NULL;
            }
            else {
                anterior -> siguiente = NULL;
                break;
            }
        }
        else {
            anterior = nodo;
            nodo = nodo -> siguiente;
        }
    }
    delete nodo;
}
```

# Operación pop

# Operaciones de verificación de estado

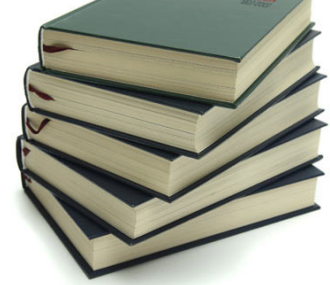
---

- ❑ Se debe proteger la integridad de la pila.
- ❑ Por tanto la clase que implemente debe proporcionar operaciones que comprueben el estado de la pila: vacía o pila llena.



```
int main()  
{  
    return 0;  
}
```





# Bibliografía

---

- <http://c.conclase.net/edd/?cap=002#inicio>
- [http://es.wikipedia.org/wiki/Pila\\_\(inform%C3%A1tica\)](http://es.wikipedia.org/wiki/Pila_(inform%C3%A1tica))