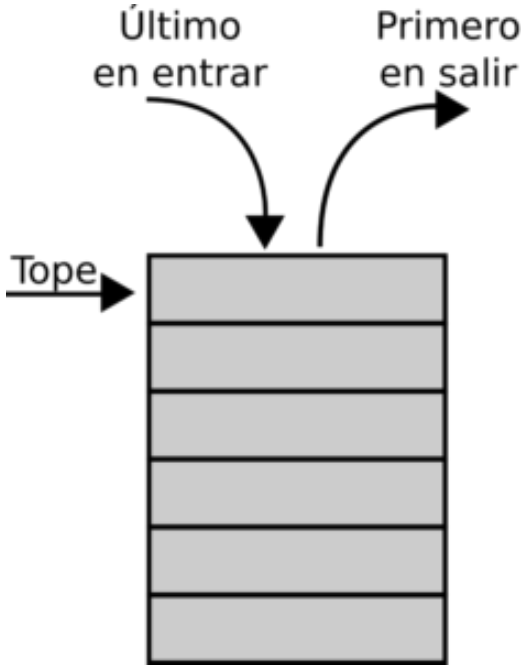
ESTRUCTURAS DE DATOS

LIC. REDES Y SERVICIOS DE CÓMPUTO

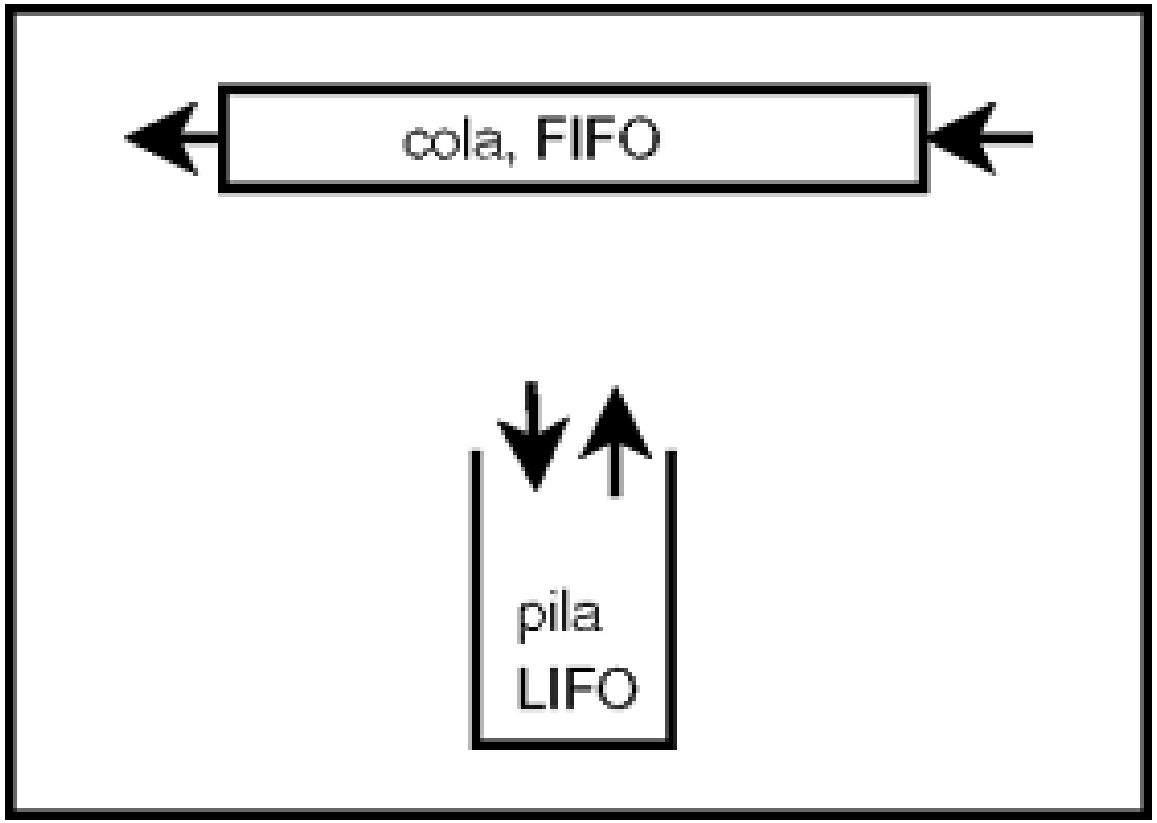
Clase

Unidad VI. Pilas

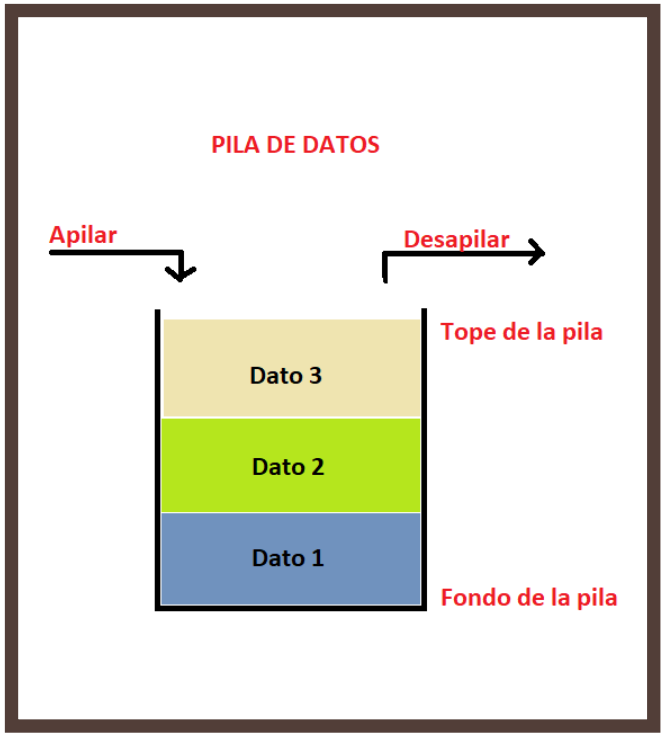
- Fundamentos teóricos
- Implantación de pilas
- Operaciones básicas de pilas
- Operaciones complementarias de pilas
- Aplicaciones de pilas



Una pila es una estructura de datos de entradas ordenadas tales que solo se introduce y elimina por un extremo, llamado cima o tope



Una estructura de datos tipo **pila** permite agregar nodos a la pila y eliminarlos de esta sólo desde su parte superior.
Por esta razón, a una pila se le conoce como estructura de datos UEPS (último en entrar, primero en salir) o LIFO (Last-Input, First-Output).

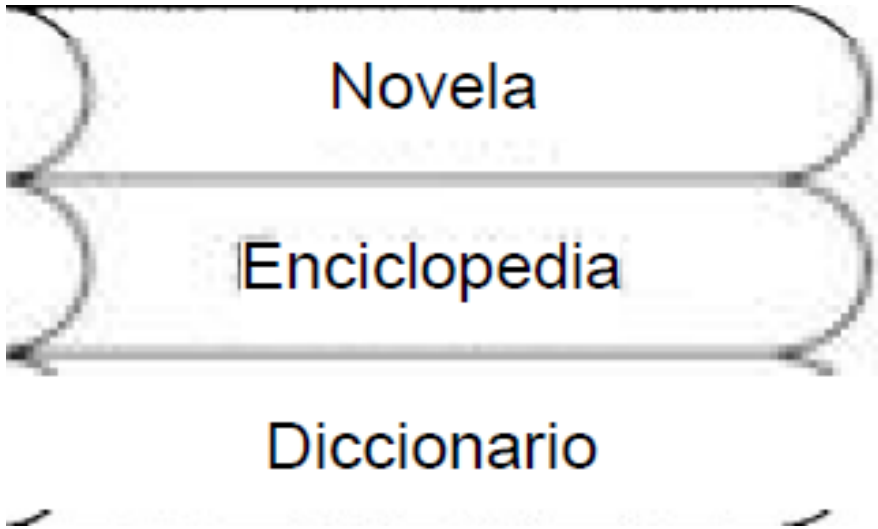


Pilas

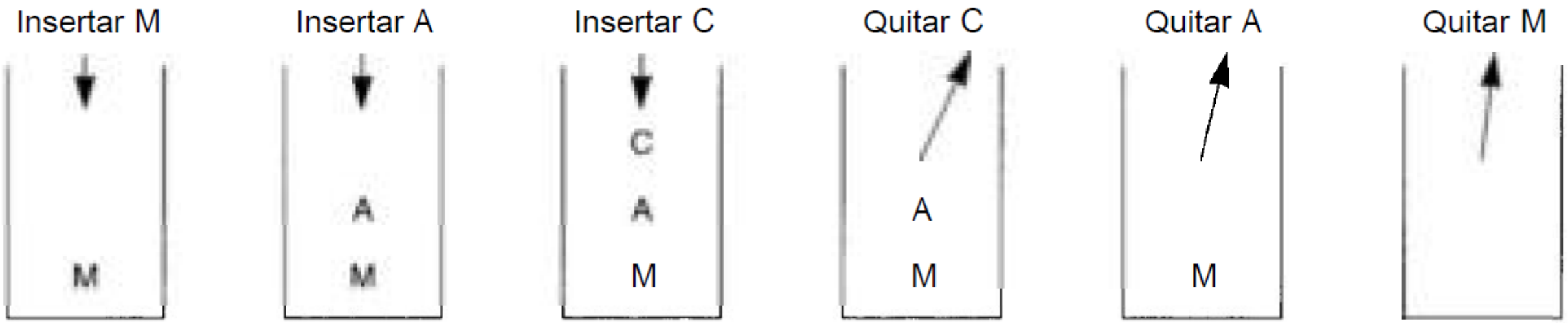
- Los elementos de una pila se eliminan en orden inverso al que se insertaron; es decir, el último elemento que se mete en la pila es el primero que se saca.
- Por ejemplo: Una pila de libros que se exhiben en una librería o una pila de platos. Es de suponer que si el cocinero necesita un plato limpio, tomará el que está encima de todos, que es el último que se colocó en la pila.

Pilas

- Se puede crear una pila de libros, situando primero un diccionario, encima de él una enciclopedia y encima de ambos una novela de modo que la pila tendrá la novela en la parte superior.
- Cuando se quitan los libros de la pila, primero debe quitarse la novela, luego la enciclopedia y, por último, el diccionario.



Pilas



Entrada:
MAC

Salida:
CAM

Pilas

Las pilas son estructuras de datos lineales, como los arreglos, ya que los componentes ocupan lugares sucesivos en la estructura y cada uno de ellos tiene un único sucesor y un único predecesor, con excepción del último y del primero, respectivamente

Una pila se define formalmente como una colección de datos a los cuales se puede acceder mediante un extremo, que se conoce generalmente como tope.

Las pilas no son estructuras fundamentales de datos Para su representación requieren el uso de otras estructuras de datos, como arreglos o listas.

Pilas

Implementación de Pilas
Constructor
Está vacía
Agregar elemento
Eliminar elemento
Mostrar los datos de la pila
Destructor
Obtener el nodo del tope
Mover la pila p1 a la pila actual

Pilas

Implementación de Pilas

Eliminar un elemento de la pila

Contar el número de elementos de una pila

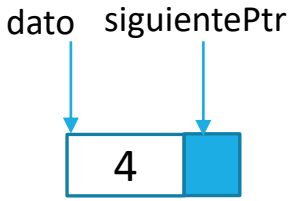
Encontrar el número mayor en los elementos de una pila

Concatenar la pila p1 al inicio de la pila actual

Pilas

- La clase Nodo

- Crea un Nodo formado por los campos:
  - **dato** (de tipo entero) que corresponde al valor que contiene el nodo y
  - **siguientePtr** (de tipo apuntador) que corresponde a la dirección del siguiente nodo en la pila.




```
// definicion del nodo d
class Nodo
{

public:
    int dato;    // INFO,
    Nodo *siguientePtr;

};
```

Pilas

- La clase Pila
- Crea la estructura Pila, formada por el nodo topePtr. Que contiene la dirección del nodo en la cima o tope de la pila.
- Se definen los métodos de la clase Pila.

```
// definicion de la pila de nodos
class Pila
{

private:
    Nodo *topePtr;  // P, apuntador al
    bool estaVacia();

public:
    Pila();
    ~Pila();
    void mete(int valor);
    int saca();
    int nodoInicio();
    void mueve(Pila &p1);
    void muestra();
    int cantidadElementos();
    int nodoMayor();
    void eliminaNodo(int referencia);
    void concatena(Pila &p1);
};
```

Pilas

- Método Pila::Pila()
- Es el constructor que inicializa el tope de la pila con NULL.

```
// constructor predeterminado
Pila::Pila()
{
    topePtr = NULL;
}
```

topePtr  = NULL

Pilas

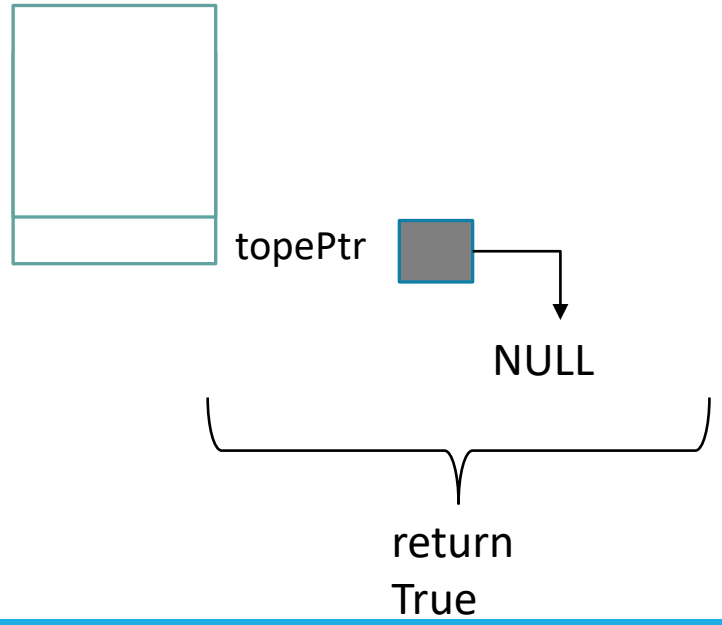
- Método Pila::~Pila()

- Es el destructor. Libera el espacio en memoria ocupado por los nodos de la Pila.
- Emplea el método saca()

```
// destructor predeterminado
Pila::~Pila()
{
    while( !estaVacia() )
    {
        saca(); // saca el ultimo elemento q
    }
}
```

Pilas

- Método Pila::estaVacia()
- Verifica si topePtr apunta a Null; si es el caso, la pila no tiene elementos.



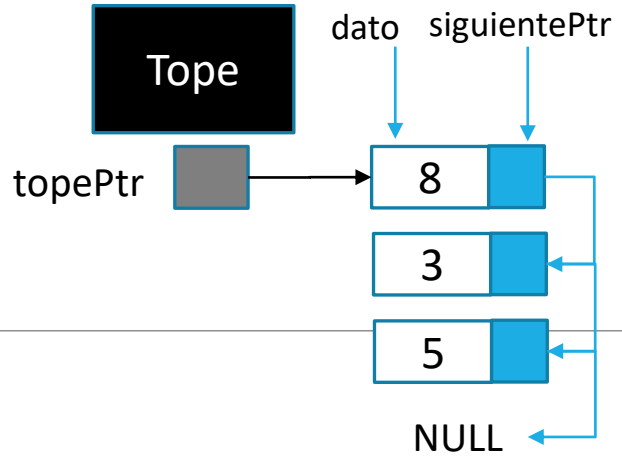

```
// verifica si la pila esta vacia
bool Pila::estaVacia()
{
    return topePtr == NULL;
}
```

Pilas

- Método Pila::mete(int valor)
- Mete un elemento a la pila, siempre al inicio o tope de la pila.

```
// mete un elemento de la pila, siempre al
void Pila::mete(int valor)
{
    Nodo *nuevoPtr = new Nodo();     // var
    nuevoPtr->dato = valor;          // DAT
    nuevoPtr->siguientePtr = topePtr;   //

    topePtr = nuevoPtr; // apunta el tope
}
```

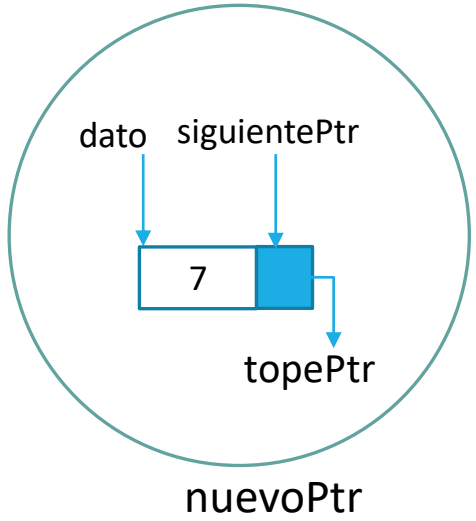
Pilas

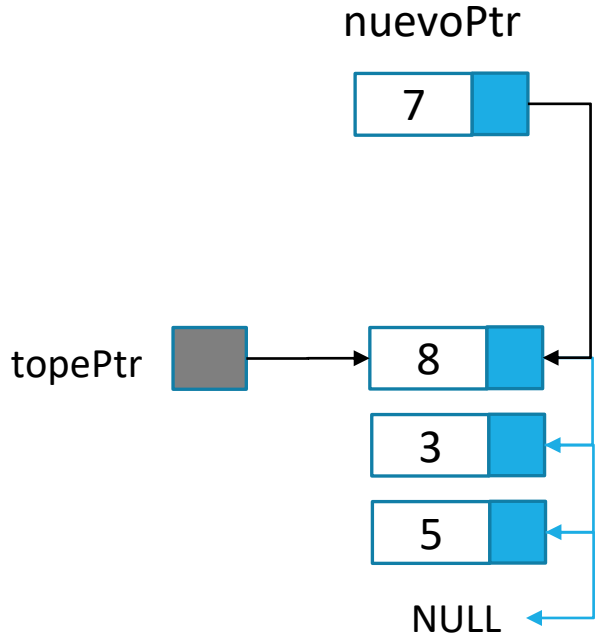


- Método Pila::mete(int valor)

Valor=7

```
// mete un elemento de la pila, siemp
void Pila::mete(int valor)
{
    Nodo *nuevoPtr = new Nodo();     /
    nuevoPtr->dato = valor;          /
    nuevoPtr->siguientePtr = topePtr;

    topePtr = nuevoPtr; // apunta el
}
```



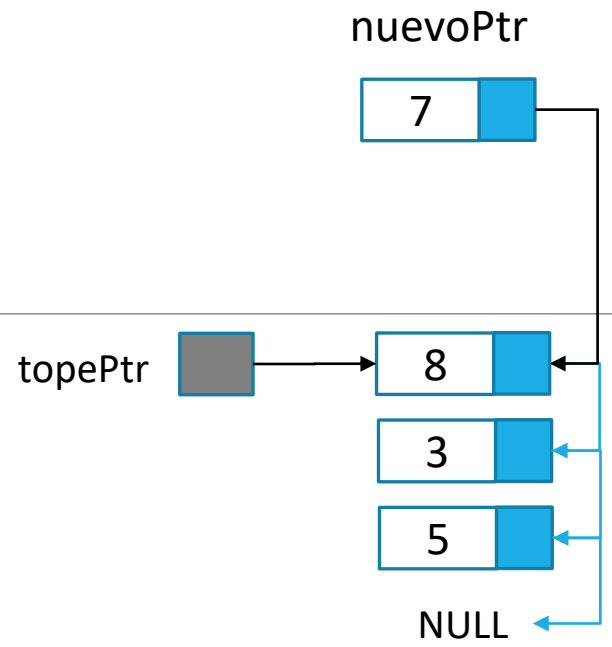


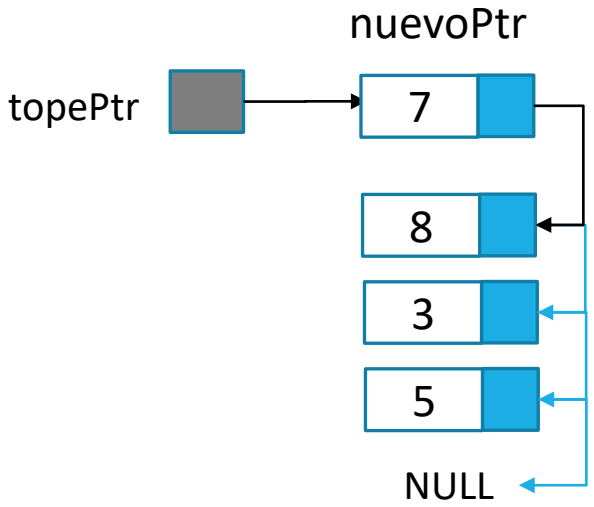
Pilas

- Método Pila::mete(int valor)

Valor=7

```
// mete un elemento de la pila, siemp
void Pila::mete(int valor)
{
    Nodo *nuevoPtr = new Nodo();     /
    nuevoPtr->dato = valor;          /
    nuevoPtr->siguientePtr = topePtr;

    topePtr = nuevoPtr; // apunta el
}
```



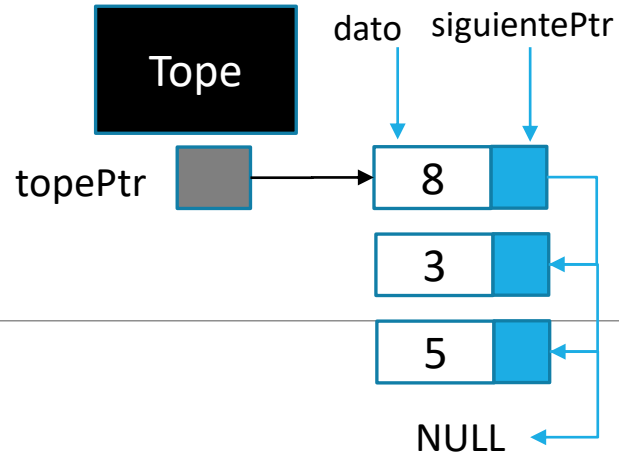


Pilas

- Método Pila::saca()
- Saca el elemento que está siempre al inicio de la pila.

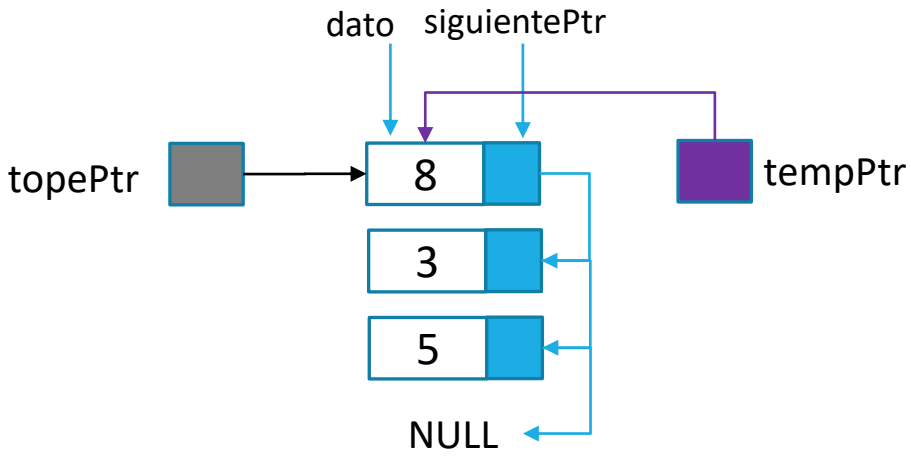
```
// saca un elemento de la pila, siempre del inicio
int Pila::saca()
{
    if( estaVacia() )   // la pila esta vacia
    {
        cout << "\nError en saca(). La pila esta vacia\n\n";
        exit(1);
    }

    int x = topePtr->dato;  // guardo el dato de tope
    Nodo *tempPtr = topePtr;    // guardo el nodo a sacar y

    topePtr = topePtr->siguientePtr;    // el nodo segundo s

    delete tempPtr; // libera la memoria

    return x;   // regresa el dato sacado
}
```

Pilas. Método Pila::saca()



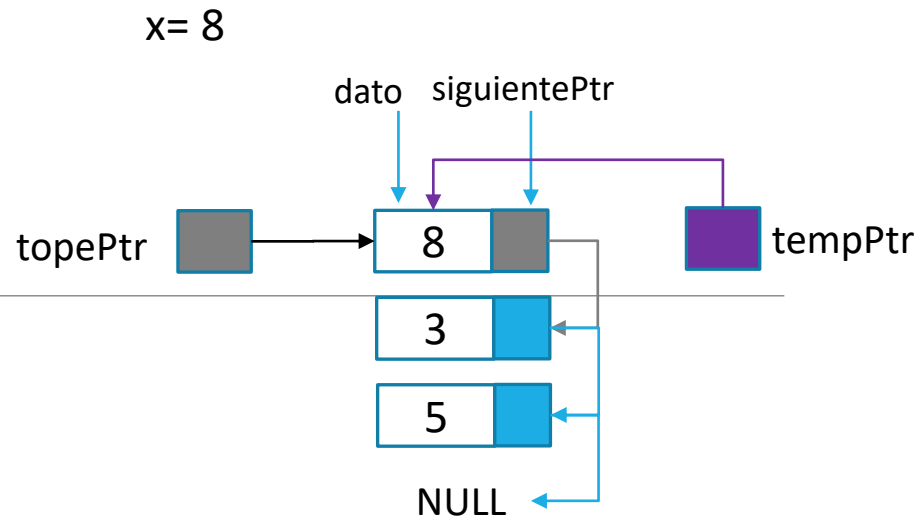

```
// saca un elemento de la pila, siempre del inicio
int Pila::saca()
{
    if( estaVacia() )   // la pila esta vacia
    {
        cout << "\nError en saca(). La pila esta vacia\n\n";
        exit(1);
    }

    int x = topePtr->dato;  // guardo el dato de tope
    Nodo *tempPtr = topePtr;    // guardo el nodo a sacar y

    topePtr = topePtr->siguientePtr;    // el nodo segundo s

    delete tempPtr; // libera la memoria

    return x;   // regresa el dato sacado
}
```

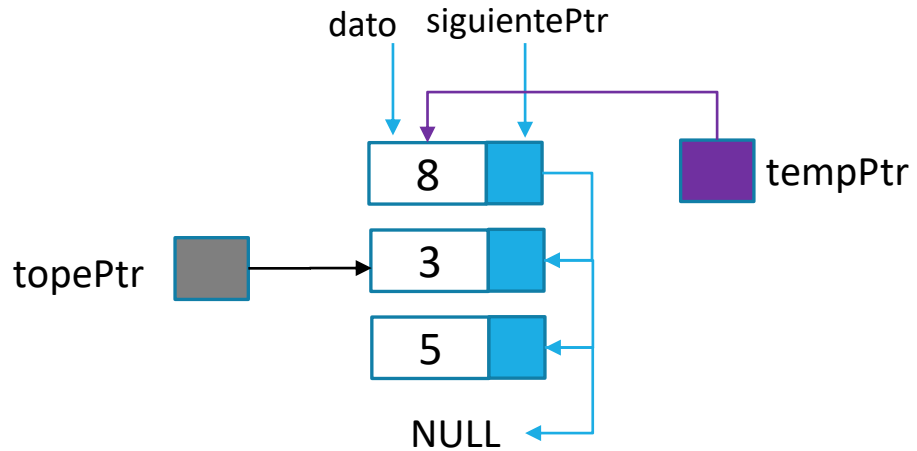
x= 8



Pilas. Método Pila::saca()

```
// saca un elemento de la pila, siempre del inicio
int Pila::saca()
{
    if( estaVacia() )   // la pila esta vacia
    {
        cout << "\nError en saca(). La pila esta vacia\n\n";
        exit(1);
    }

    int x = topePtr->dato;  // guardo el dato de tope
    Nodo *tempPtr = topePtr;    // guardo el nodo a sacar y

    topePtr = topePtr->siguientePtr;    // el nodo segundo s

    delete tempPtr; // libera la memoria

    return x;   // regresa el dato sacado
}
```

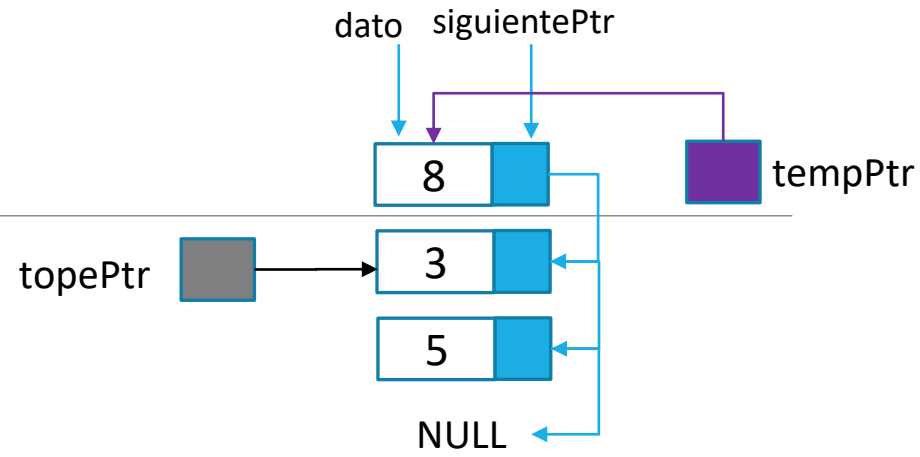


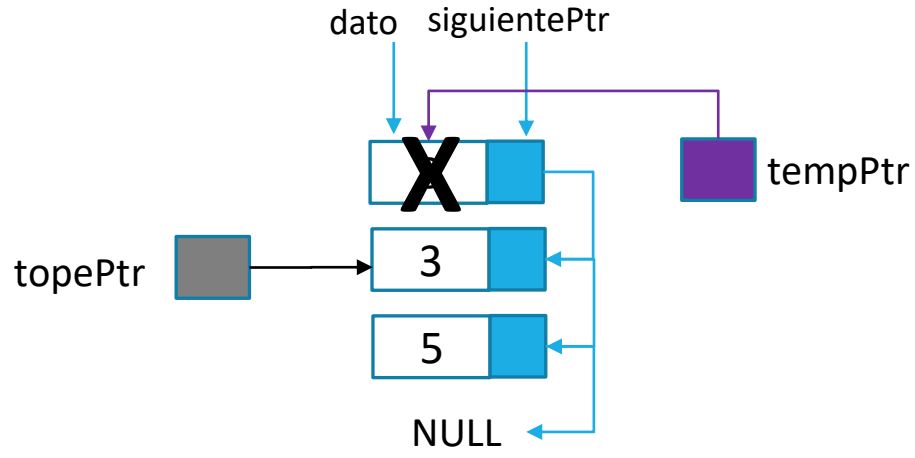


Pilas. Método Pila::saca()

```
// saca un elemento de la pila, siempre del inicio
int Pila::saca()
{
    if( estaVacia() )   // la pila esta vacia
    {
        cout << "\nError en saca(). La pila esta vacia\n\n";
        exit(1);
    }

    int x = topePtr->dato;  // guardo el dato de tope
    Nodo *tempPtr = topePtr;    // guardo el nodo a sacar y

    topePtr = topePtr->siguientePtr;    // el nodo segundo s

    delete tempPtr; // libera la memoria

    return x;   // regresa el dato sacado
}
```

x= 8



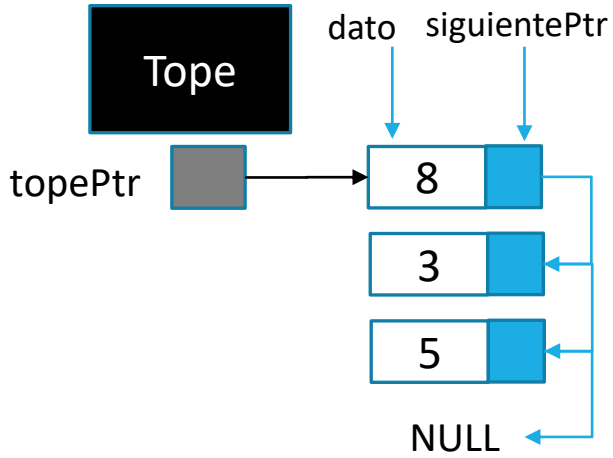


Return x= 8

Pilas

- Actividad en línea:

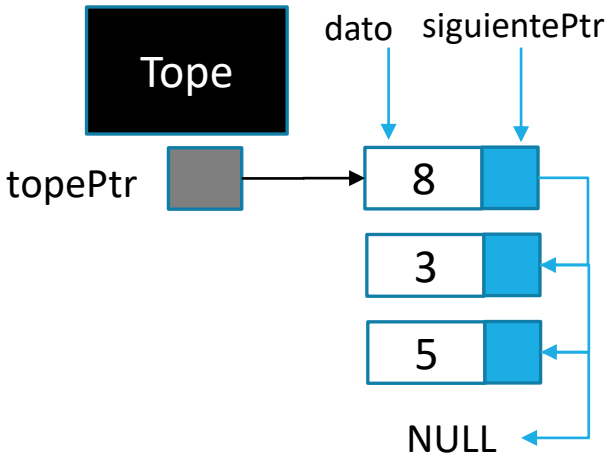
Elabora el método int Pila::nodoInicio(). Es decir el método para obtener el dato del tope o inicio de la pila:



Pilas

- Método Pila::nodoInicio()

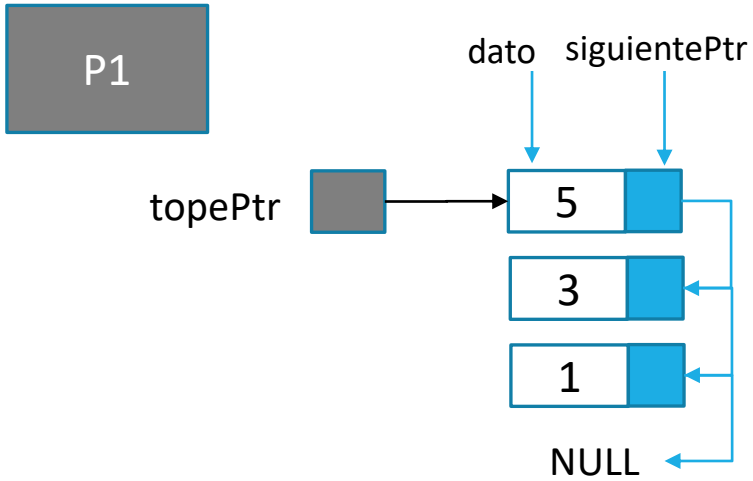
```
// muestra el primer dato de la pila
int Pila::nodoInicio()
{
    if( topePtr == NULL )    // No hay inicio
    {
        cout << "\nError en nodoInicio(). La pila esta vacia\n\n";
        exit(1);
    }

    return topePtr->dato;
}
```

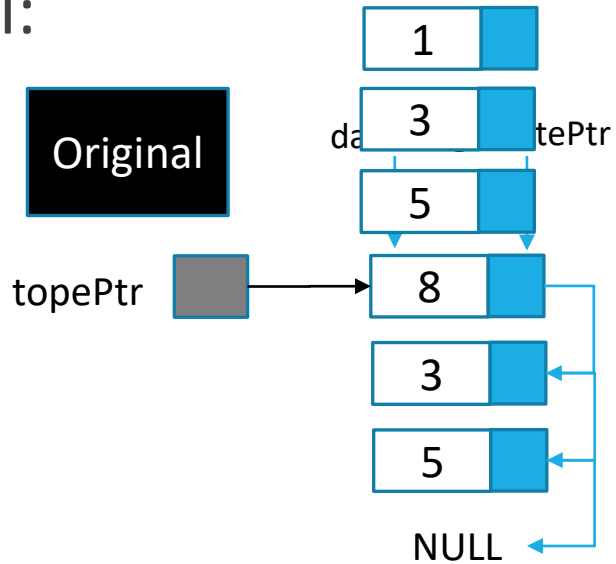


Pilas

- Actividad fuera de línea:

Elabora el método Pila::mueve(Pila &p1). Es decir el método para mover los elementos de una pila p1, a la pila original:





Pilas

- Método Pila::mueve(Pila &p1)
- Mueve los elementos de una pila p1, a la pila original.

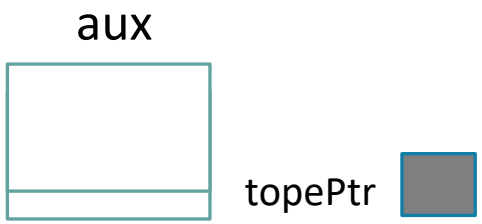
```
// mueve los elementos de una
void Pila::mueve(Pila &p1)
{
    int x;  // variable tempo

    // regresa los datos desd
    while( !p1.estaVacia() )
    {
        x = p1.saca();
        mete(x);    // mete e
    }
}
```

Pilas

- Método Pila::muestra()
- Muestra los datos de la pila

```
// muestra los datos de la pila
void Pila::muestra()
{
    if( estaVacia() )   // la pila esta vacia
    {
        cout << "\nLa pila esta vacia\n\n";
        system("pause");
        return;
    }

    Pila aux;   // pila auxiliar para no perder lo
    int x;  // varieble temporal para el dato actu

    cout << "\nLos elementos de la pila son: ";

    // muestra los datos y los guarda en aux
    while( !estaVacia() )
    {
        x = saca();
        cout << x << " -> ";    // muestra el dato
        aux.mete(x);    // guarda el dato en la pi
    }

    // regresa los datos de la pila aux a la pila
    mueve( aux );

    cout << "\n\n";
    system("pause");
}
```

Pilas



- Método Pila::muestra()

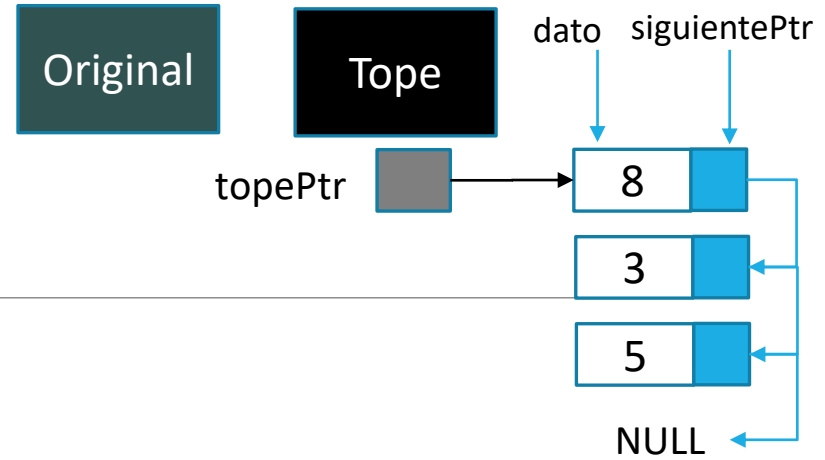
```
Pila aux;   // pila auxiliar para no perder lo
int x;  // varieble temporal para el dato actu

cout << "\nLos elementos de la pila son: ";

// muestra los datos y los guarda en aux
while( !estaVacia() )
{
    x = saca();
    cout << x << " -> ";    // muestra el dato
    aux.mete(x);    // guarda el dato en la pi
}

// regresa los datos de la pila aux a la pila
mueve( aux );

cout << "\n\n";
system("pause");
}
```

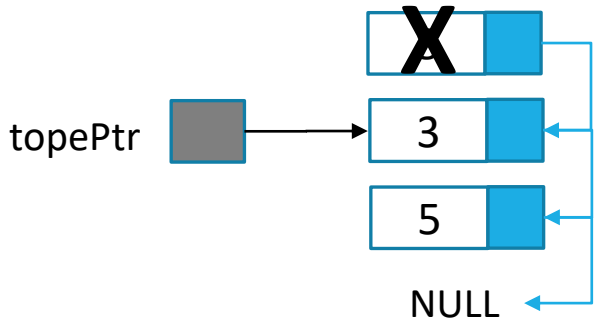
Declara la variable de tipo entero **x**. **x** servirá para almacenar los valores de la pila original, escribirlos en pantalla y meterlo a la pila aux.

Pilas

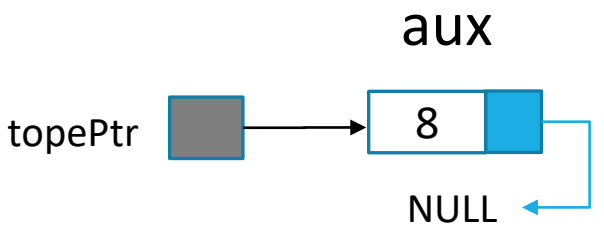
- Método Pila::muestra()

```
Pila aux;   // pila auxiliar para no perder lo
int x;  // varieble temporal para el dato actu

cout << "\nLos elementos de la pila son: ";

// muestra los datos y los guarda en aux
while( !estaVacia() )
{
    x = saca();
    cout << x << " -> ";    // muestra el dato
    aux.mete(x);    // guarda el dato en la pi
}

// regresa los datos de la pila aux a la pila
mueve( aux );

cout << "\n\n";
system("pause");
}
```

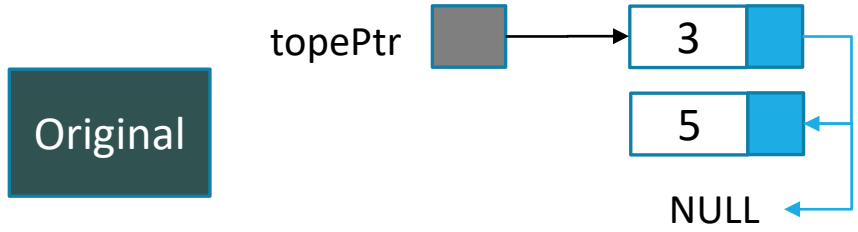




Escribe 8→

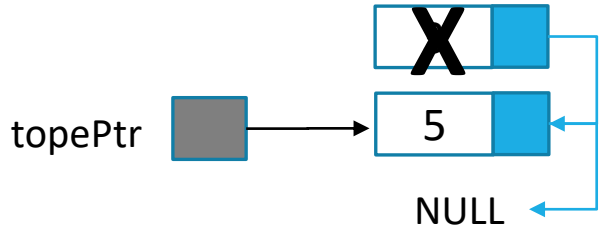


Pilas



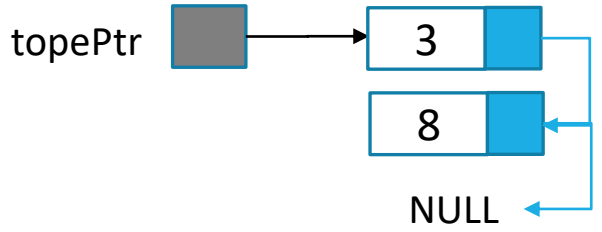
- Método Pila::muestra()

```
Pila aux;   // pila auxiliar para no perder lo
int x;  // varieble temporal para el dato actu

cout << "\nLos elementos de la pila son: ";

// muestra los datos y los guarda en aux
while( !estaVacia() )
{
    x = saca();
    cout << x << " -> ";    // muestra el dato
    aux.mete(x);    // guarda el dato en la pi
}

// regresa los datos de la pila aux a la pila
mueve( aux );

cout << "\n\n";
system("pause");
}
```

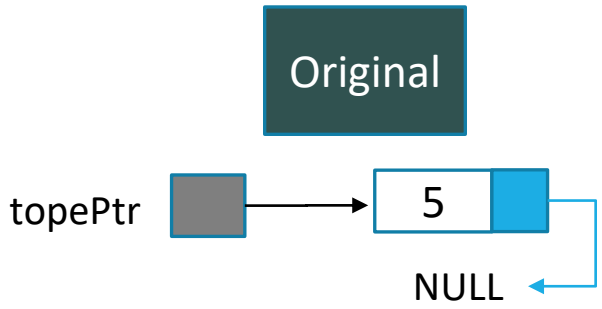


Escribe 3→8→

aux

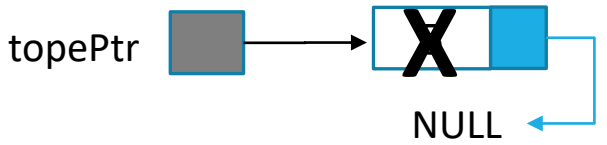


Pilas

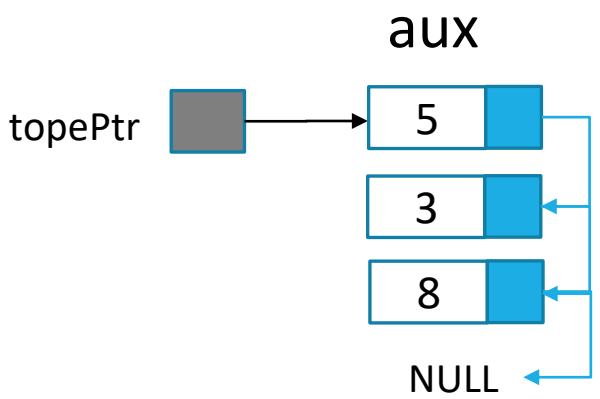


- Método Pila::muestra()

```
Pila aux;   // pila auxiliar para no perder lo
int x;  // varieble temporal para el dato actu

cout << "\nLos elementos de la pila son: ";

// muestra los datos y los guarda en aux
while( !estaVacia() )
{
    x = saca();
    cout << x << " -> ";    // muestra el dato
    aux.mete(x);    // guarda el dato en la pi
}

// regresa los datos de la pila aux a la pila
mueve( aux );

cout << "\n\n";
system("pause");
}
```



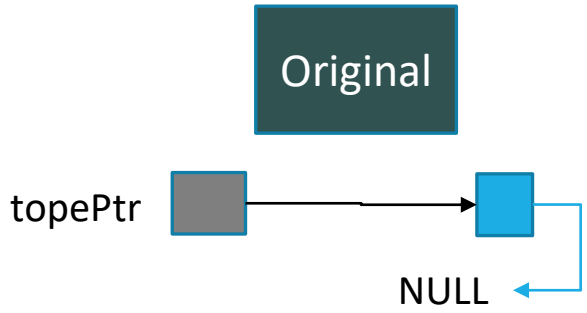
Escribe 5→3→8→

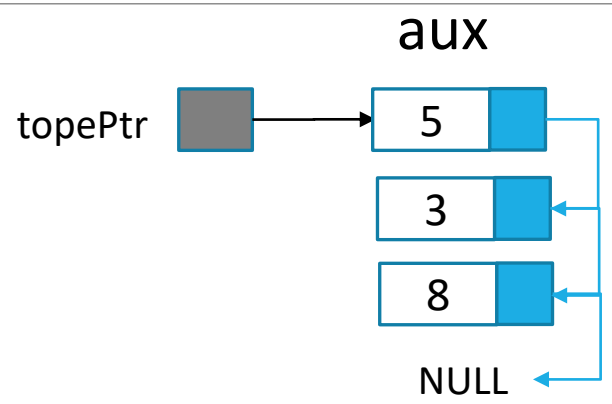


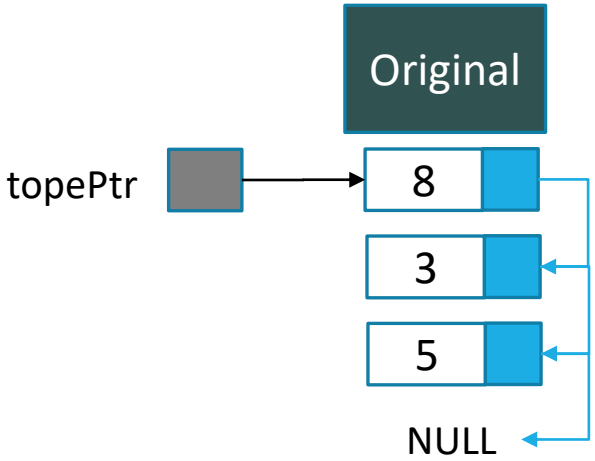
Pilas

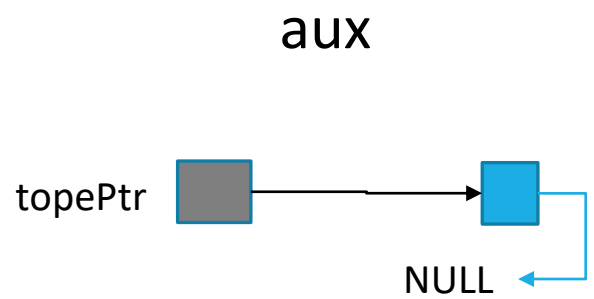
- Método Pila::muestra()

```
Pila aux;   // pila auxiliar para no perder lo
int x;  // varieble temporal para el dato actu

cout << "\nLos elementos de la pila son: ";

// muestra los datos y los guarda en aux
while( !estaVacia() )
{
    x = saca();
    cout << x << " -> ";    // muestra el dato
    aux.mete(x);    // guarda el dato en la pi
}

// regresa los datos de la pila aux a la pila
mueve( aux );

cout << "\n\n";
system("pause");
}
```









Pilas

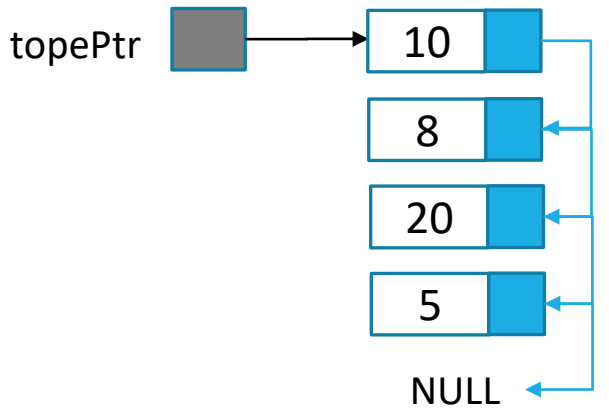
- Método Pila::cantidadElementos()
- Regresa la cantidad de elementos existentes en la pila.

```
// regresa la cantidad de elemen
int Pila::cantidadElementos()
{
    if( estaVacia() )   // la pi
    {
        return 0;
    }

    Pila aux;   // pila auxiliar
    int x;  // varieble temporal
    int cont = 0;   // variable

    // muestra los datos y los g
    while( !estaVacia() )
    {
        x = saca(); // saca el d
        aux.mete(x);    // guard
        cont++;     // aumenta e
    }

    // regresa los datos de la p
    mueve( aux );

    return cont;
}
```

Pilas

- Actividad en línea:

Elabora la prueba de escritorio para el método Pila::nodoMayor(). Es decir el método para encontrar el dato mayor para la siguiente pila:



Pilas

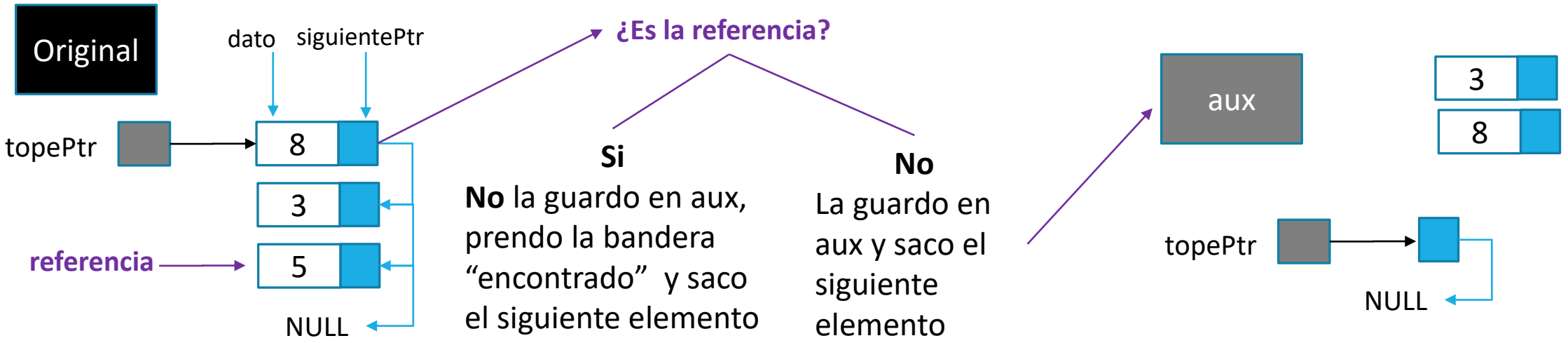
- Método Pila::nodoMayor().
- Regresa el dato con el numero mayor de la pila.

```
// regresa el dato con el numero mayor
int Pila::nodoMayor()
{
    if( estaVacia() )   // la pila esta vacia
    {
        cout << "\nError en nodoMayor(). La pila esta vacia\n\n";
        exit(1);
    }

    Pila aux;   // pila auxiliar para no perder los datos de la pil
    int mayor;  // variable temporal para el dato mayor. Inicia con
    int x;  // variable temporal para el dato actual

    mayor = saca(); // en caso de que solo haya un nodo, lo hacemos
    aux.mete(mayor);    // guarda el dato en la pila aux

    // muestra los datos y los guarda en aux
    while( !estaVacia() )
    {
        x = saca(); // saca el dato
        aux.mete(x);    // guarda el dato en la pila
        if( x > mayor )
        {
            mayor = x;
        }
    }

    // regresa los datos de la pila aux a la pila actual
    mueve( aux );

    return mayor;
}
```

Pilas

- Actividad fuera de línea:

Elabora el método Pila::eliminaNodo(int referencia). Es decir el método para eliminar el nodo con el dato otorgado como referencia:



Pilas

- Método Pila::eliminaNodo(int referencia)
- Elimina el nodo de la pila con el dato dado como referencia.

```
void Pila::eliminaNodo(int referencia)
{
    if( estaVacia() )   // la pila esta vacia
    {
        cout << "\nEl nodo dado como referencia no se encuentra en la pila.\n";
        return;
    }

    Pila aux;   // pila auxiliar para no perder los datos de la pila original
    int x;  // varieble temporal para el dato actual
    bool encontrado = false;    // bandera que indica si encontro el nodo

    // muestra los datos y los guarda en aux
    while( !estaVacia() )
    {
        x = saca(); // saca el dato
        if( referencia == x )
        {
            cout << "\nEliminando el nodo: " << x << "\n";
            encontrado = true;
        }
        else
        {
            aux.mete(x);    // guarda el dato en la pila aux
        }
    }

    // regresa los datos de la pila aux a la pila actual
    mueve( aux );

    if( !encontrado )
        cout << "\nEl nodo dado como referencia no se encuentra en la pila.\n";
}
```

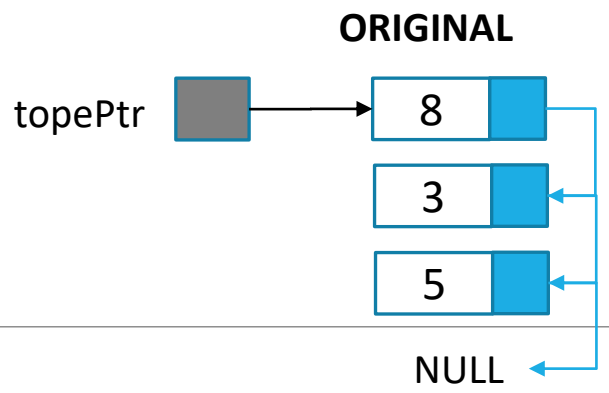
Pilas

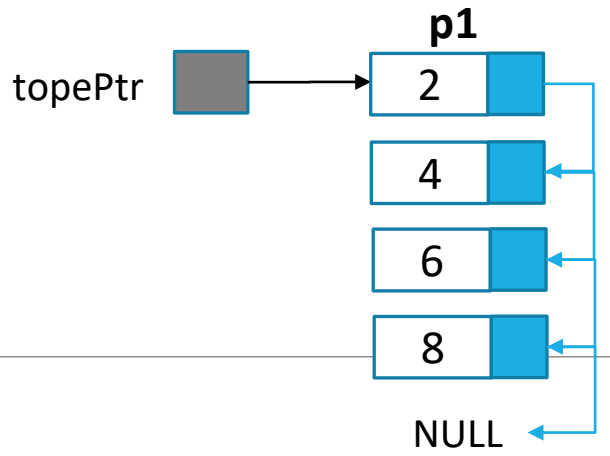
- Pila::concatena(Pila &p1)
- Copia los datos de la pila p1 en orden al inicio de la pila actual.

```
// copia los datos de la pila p1 en orden al inicio de
void Pila::concatena(Pila &p1)
{
    Pila aux;   // pila auxiliar para no perder los da
    int x;  // variebles temporales para guardar los a

    // guarda los datos en aux en orden invertido
    aux.mueve( p1 );

    // agrega los datos a la pila actual
    mueve( aux );
}
```

Pilas

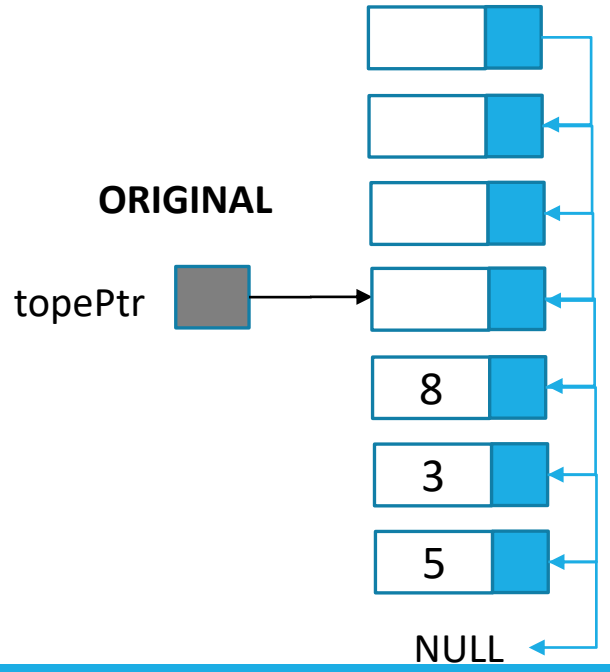
Pila::concatena(Pila &p1)





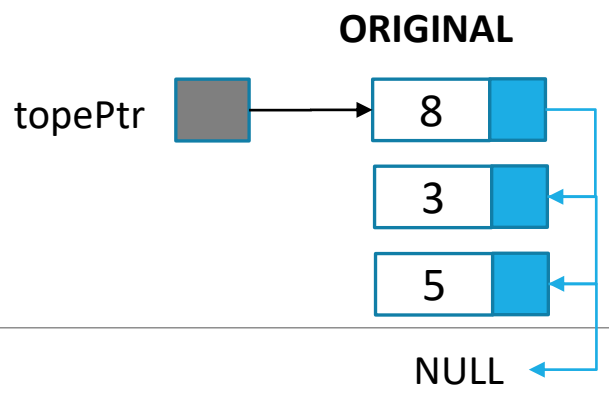
Datos ingresados por el usuario a p1:
8, 6, 4, 2

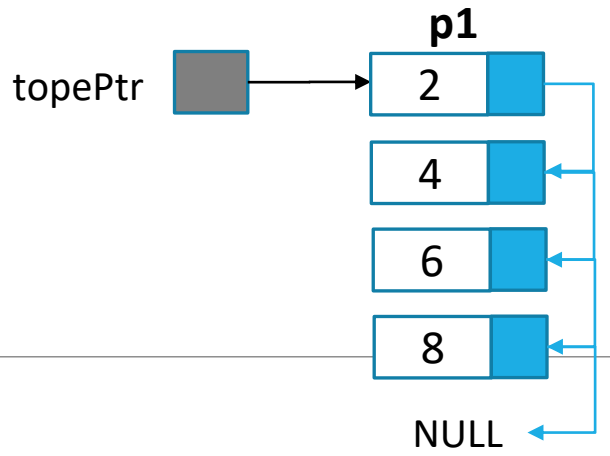
```
// copia los datos de la pila p
void Pila::concatena(Pila &p1)
{
    Pila aux;    // pila auxilia
    int x;  // variebles tempor

    // guarda los datos en aux
    aux.mueve( p1 );

    // agrega los datos a la pi
    mueve( aux );
}
```



Pilas

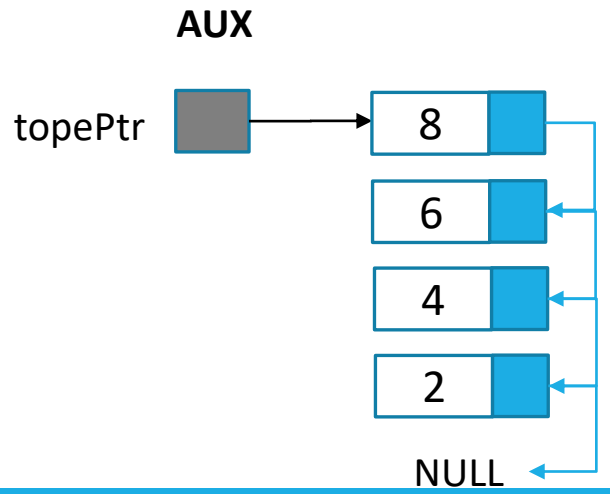
Pila::concatena(Pila &p1)

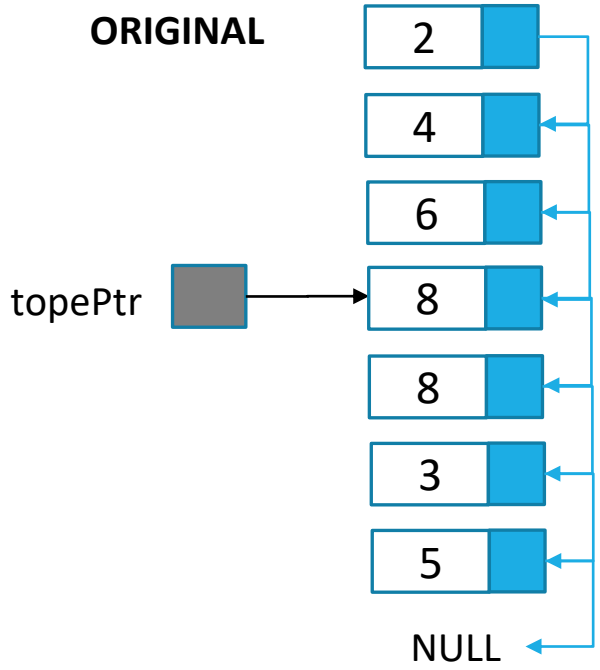





```
// copia los datos de la pila p
void Pila::concatena(Pila &p1)
{
    Pila aux;    // pila auxilia
    int x;  // variebles tempor

    // guarda los datos en aux
    aux.mueve( p1 );

    // agrega los datos a la pi
    mueve( aux );
}
```

Datos ingresados por el usuario a p1:
5, 20, 8, 10





Pilas

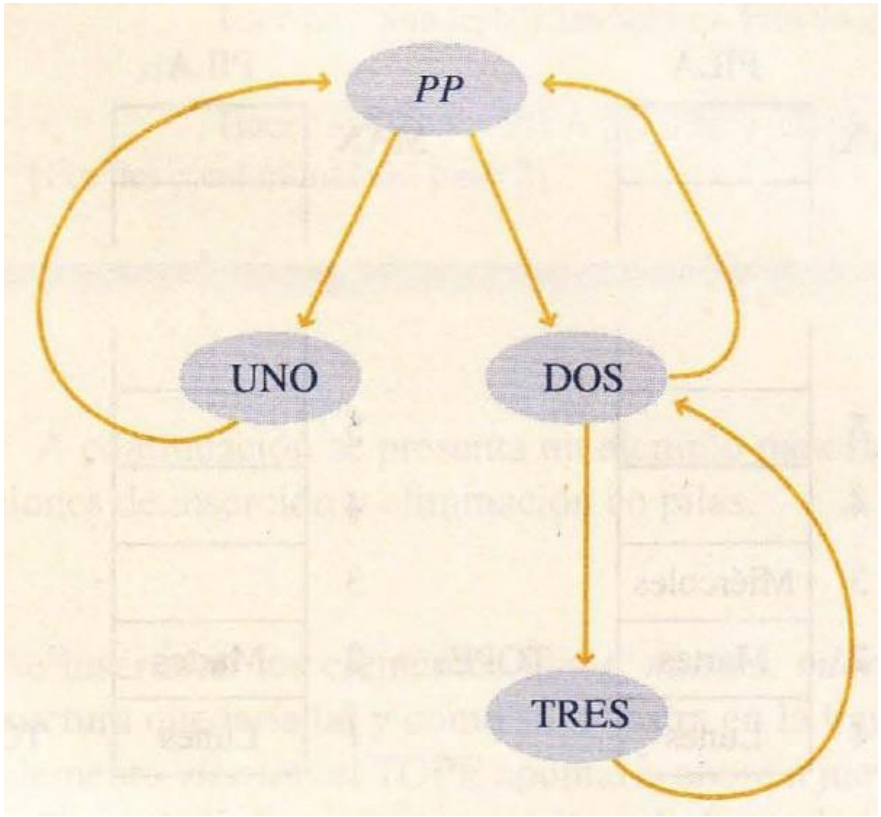
Aplicaciones de las Pilas
Llamadas a subprogramas
Recursividad
Tratamiento de expresiones aritméticas
Ordenación

Pilas. Aplicaciones

- **Llamadas a subprogramas.**
  - Cuando se tiene un programa que llama a un subprograma, módulo o función, internamente se usan pilas para guardar el estado de las variables del programa, y las instrucciones pendientes de ejecución.
  - Cuando termina la ejecución del subprograma, los valores almacenados en la pila se recuperan para continuar con la ejecución del programa en el punto en el cual fue interrumpido.
  - Además de las variables se recupera la dirección del programa en la que se hizo la llamada, porque a esa posición se regresa el control del proceso.

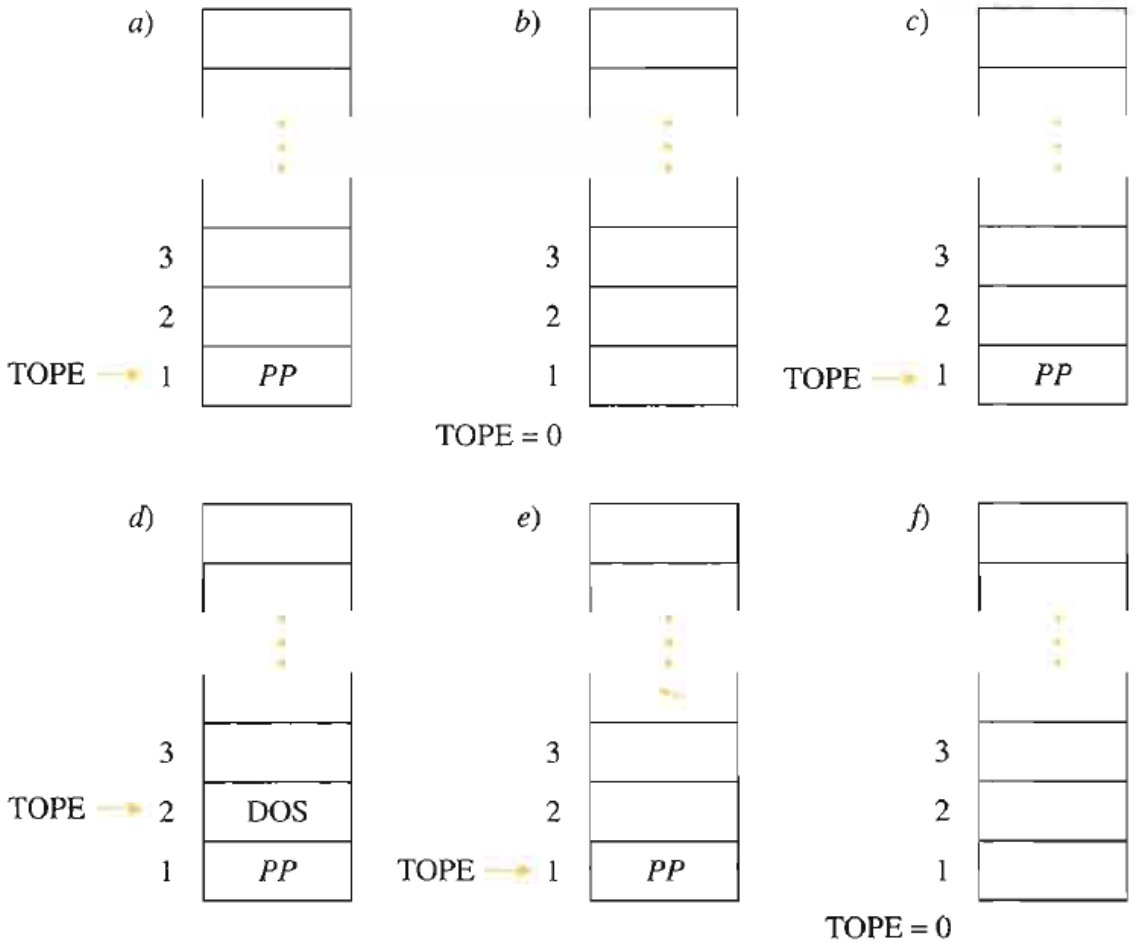
Pilas. Aplicaciones

- **Llamadas a subprogramas.**
  - Por ejemplo, se tiene un programa principal (PP) que llama a los subprogramas UNO y DOS.
  - A su vez, el subprograma DOS llama al TRES.
  - Cada vez que la ejecución de uno de los subprogramas concluye, se regresa el control al nivel inmediato superior.



Pilas. Aplicaciones

- **Llamadas a subprogramas.**
  - Cuando el programa PP llama a UNO, se guarda en una pila la posición en la que se hizo la llamada (a).
  - Al terminar UNO, el control se regresa a PP recuperándose previamente la dirección de la pila (b).
  - Al llamar a DOS, nuevamente se guarda la dirección de PP en la pila (c).
  - Cuando DOS llama a TRES, se pone en la pila la dirección de DOS (d).
  - Después de procesar TRES, se recupera la posición DOS para continuar con su ejecución (e).
  - Al terminar DOS se regresa el control a PP, obteniendo previamente la dirección guardada en la pila (j).

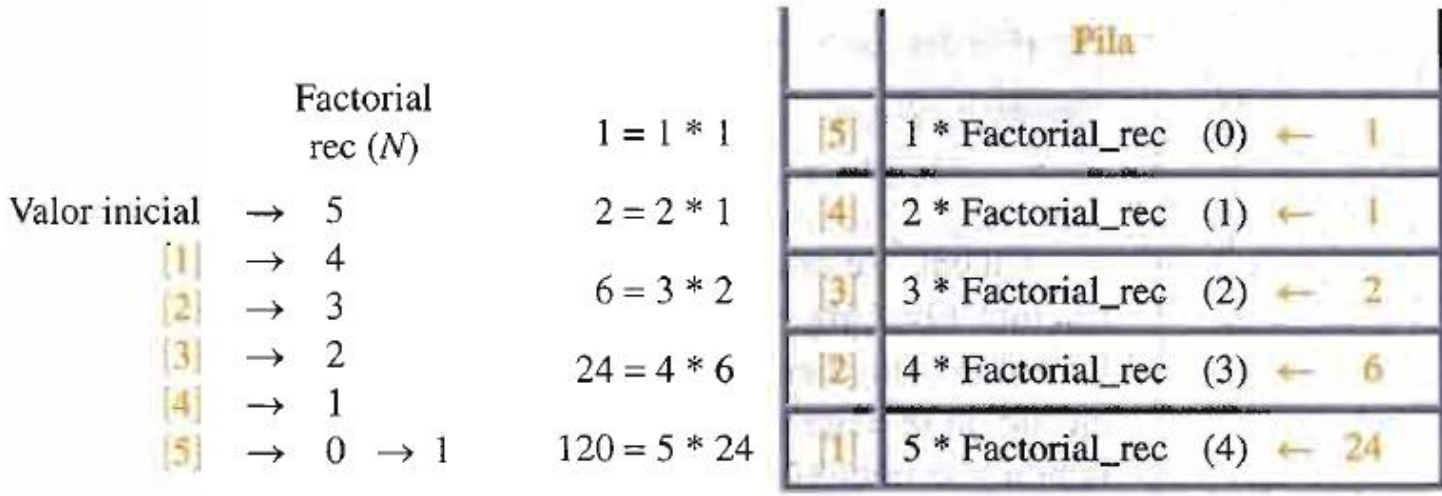


Pilas. Aplicaciones

- **Recursividad**

Ejemplo. Factorial de un número.

```
1. Si (N = 0)
       entonces
           Hacer Factorial_rec ← 1 {Paso básico}
       si no
           Hacer Factorial_rec ← N * Factorial_rec (N – 1)
               {Llamada —paso— recursiva}
2. {Fin del condicional del paso 1}
```



Utiliza una pila, donde se van almacenando las instrucciones a ejecutar con los valores de las constantes y las variables en ese momento.
Cuando se termina la ejecución se llega al estado básico, se toma la instrucción del tope de la pila y se continua ejecutando; hasta que la pila queda vacía.

Pilas.

- **Actividad:**
- **Haciendo el uso de la estructura de Pilas, escribe un programa en C++ que lea una expresión en notación infija y la traduzca a notación posfija.**

Pilas.

- **Actividad:**
  - La expresión A + B se dice que ésta se encuentra en notación infija, porque el operador (+) se encuentra entre los operandos (A y B).
  - La expresión AB+ se dice que se encuentra en notación postfija, por que el operador (+) se encuentra después de los operandos (A y B).

La ventaja de usar expresiones en notación postfija o prefija radica en que no son necesarios los paréntesis para indicar orden de operación, ya que éste queda establecida por la ubicación de los operadores con respecto a los operandos.

Pilas.

- **Actividad:**
- Para convertir una expresión dada en notación infija a una en notación postfija se establecen primero ciertas condiciones:
  - Solamente se manejarán los siguientes operadores y se presentan de acuerdo a la prioridad de ejecución:

Operador	Nombre de la operación
^	Potencia
* /	Multiplicación y división
+ –	Suma y resta

Pilas.

- **Actividad:**
- Para convertir una expresión dada en notación infija a una en notación postfija se establecen primero ciertas condiciones:
  - Los operadores de más alta prioridad se ejecutan primero.
  - Si hubiera en una expresión dos o más operadores de igual prioridad, entonces procesarán de izquierda a derecha.
  - Las subexpresiones que se encuentran entre paréntesis tendrán más prioridad que cualquier operador.

Pilas.

- **Actividad:**
  - Ejemplos:

Expresión infija: X + Z * *W*

Expresión posfija: *XZW**+

1. El primer operador que se procesa durante la traducción de la expresión es la multiplicación, paso 1, debido a que es el de más alta prioridad.
2. Se coloca el operador tal manera que los operandos afectados por él lo precedan.
3. Para el operador de suma se sigue el mismo criterio, los dos operandos lo preceden. En este caso el primer operando es X y el segundo es ZW*.

Paso	Expresión
0	*X* + *Z* * *W*
1	*X* + *ZW**,
2	*XZW** +

Pilas.

- **Actividad:**
  - Ejemplos:

Expresión infija: (X + Z)* W / T ^ Y - V
Expresión postfija: XZ+W*TY^/V-

1. En el paso 1 se convierte la subexpresión que se encuentra entre paréntesis por ser la de más alta prioridad. Luego se sigue con el operador de potencia, paso 2,
2. Y así sucesivamente con los demás, según su jerarquía.
3. Como consecuencia de que la multiplicación y la división tienen igual prioridad, se procesa primero la multiplicación por encontrarse más a la izquierda en la expresión, paso 3.
4. El operador de la resta es el último que se mueve, paso 5.

0	$(X + Z)^* \ W/T \ ^\wedge \ Y - V$
1	$XZ + ^*W/T \ ^\wedge \ Y - V$
2	$XZ + ^*W/TY \ ^\wedge - V$
3	$XZ + W^*/TY \ ^\wedge - V$
4	$XZ + W * TY \ ^\wedge / - V$
5	$XZ + W * TY \ ^\wedge / V -$

Pilas.

- **Actividad:**
  - Ejemplos:

Expresión infija: (X + Z)* W / T ^ Y - V
Expresión postfija: XZ+W*TY^/V-

1. En el paso 1 se convierte la subexpresión que se encuentra entre paréntesis por ser la de más alta prioridad. Luego se sigue con el operador de potencia, paso 2,
2. Y así sucesivamente con los demás, según su jerarquía.
3. Como consecuencia de que la multiplicación y la división tienen igual prioridad, se procesa primero la multiplicación por encontrarse más a la izquierda en la expresión, paso 3.
4. El operador de la resta es el último que se mueve, paso 5.

0	$(X + Z)^* \ W/T \ ^\wedge \ Y - V$
1	$XZ + ^*W/T \ ^\wedge \ Y - V$
2	$XZ + ^*W/TY \ ^\wedge - V$
3	$XZ + W^*/TY \ ^\wedge - V$
4	$XZ + W * TY \ ^\wedge / - V$
5	$XZ + W * TY \ ^\wedge / V -$

Conv_postfija (EI, EPOS)

{Este algoritmo traduce una expresión infija —*EI*— a postfija —EPOS—, haciendo uso de una pila —PILA—. MAX es el número máximo de elementos que puede almacenar la pila}

```
1. Hacer TOPE ← 0
2. Mientras (EI sea diferente de la cadena vacía) Repetir
       Tomar el símbolo más a la izquierda de EI. Recortar luego la expresión
   2.1 Si (el símbolo es paréntesis izquierdo)
           entonces {Poner símbolo en PILA. Se asume que hay espacio en PILA}
                Llamar a Pone con PILA, TOPE, MAX y símbolo
           si no
       2.1.1 Si (el símbolo es paréntesis derecho)
                    entonces
             2.1.1.1 Mientras (PILA[TOPE] ≠ paréntesis izquierdo) Repetir
                            Llamar a Quita con PILA, TOPE y DATO
                            Hacer EPOS ← EPOS + DATO
```

```
            2.1.1.2 {Fin del ciclo del paso 2.1.1.1}
                    Llamar a Quita con PILA, TOPE y DATO
                    {Se quita el paréntesis izquierdo de PILA y no se agrega
                     a EPOS}
                        si no
            2.1.1.3 Si (el símbolo es un operando)
                        entonces
                            Agregar símbolo a EPOS
                        si no {Es un operador}
                    Llamar Pila_vacía con PILA, TOPE y BAND
          2.1.1.3A  Mientras (BAND = FALSO) y (la prioridad del
                            operador sea menor o  igual que la prioridad
                            del operador que está en la cima de PILA)
                            Repetir
                            Llamar a Quita con PILA, TOPE y DATO
                            Hacer EPOS ← EPOS + DATO
                            Llamar a Pila_vacía con PILA, TOPE y BAND
          2.1.1.3B  {Fin del ciclo del paso 2.1.1.3A}
                    Llamar a Pone con PILA, TOPE, MAX y símbolo
            2.1.1.4 {Fin del condicional del paso 2.1.1.3}
        2.1.2 {Fin del condicional del paso 2.1.1}
    2.2 {Fin del condicional del paso 2.1}
3. {Fin del ciclo del paso 2}
4. Llamar a Pila_vacía con PILA, TOPE y BAND
5. Mientras (BAND = FALSO) Repetir
       Llamar a Quita con PILA, TOPE y DATO
       Hacer EPOS ← EPOS + DATO
       Llamar a Pila_vacía con PILA, TOPE y BAND
6. {Fin del ciclo del paso 5}
7. Escribir EPOS
```

Pilas.

- A continuación se presentan los pasos necesarios para hacer la traducción.

Expresión infija: X + Z * *W*

Expresión posfija: *XZW**+

Siguiendo el algoritmo anterior.

Pilas.

Expresión infija: X + Z * *W*

En los pasos 1, 3 Y5 el símbolo analizado -un operando- se agrega directamente a EPOS. Al analizar el operador +, paso 2, se verifica si en PILA hay operadores e mayor o igual prioridad.

En este caso, PILA está vacía; por tanto, se pone el símbolo en el tope de ella. Con el operador *, paso 4, sucede algo similar.

En PILA no existen operadores de mayor o igual prioridad -la suma tiene menor prioridad que la multiplicación-, por lo que se agrega el operador * a PILA.
En los dos últimos pasos, 6 y 7, se extraen de PILA sus elementos, agregándolos a EPOS.

Paso	*El*	Símbolo analizado	Pila	EPOS
0	*X + Z * W*			
1	*+ Z * W*	*X*		*X*
2	*Z * W*	+	+	*X*
3	** W*	*Z*	+	*XZ*
4	*W*	*	+ *	*XZ*
5		*W*	+ *	*XZW*
6			+	*XZW **
7				*XZW * +*

Pilas.

- A continuación se presentan los pasos necesarios para hacer la traducción.

Expresión infija: (X + Z)* W / T ^ Y - V
Expresión postfija: XZ+W*TY^/V-

Siguiendo el algoritmo anterior.

Los pasos que se consideran más relevantes son: en el paso 5, al analizar el paréntesis derecho se extraen repetidamente todos los elementos de PILA (en este caso sólo el operador +), agregándolos a EPOS hasta encontrar un paréntesis izquierdo.

El paréntesis izquierdo se quita de PILA pero no se incluye en EPOS -recuerde que las expresiones en notación posfija no necesitan de paréntesis para indicar prioridades.

Cuando se trata el operador de división, paso 8, se quita de PILA el operador * y se agrega a EPOS, ya que la multiplicación tiene igual prioridad que la división.

Al analizar el operador de resta, paso 12, se extraen de PILA y se incorporan a EPOS todos los operadores de mayor o igual prioridad, en este caso son todos los que están en ella –la potencia y la división-, agregando finalmente el símbolo en PILA.

Luego de agregar a EPOS el último operando, y habiendo revisado toda la expresión inicial, se vacía PILA y se incorporan los operadores (en este caso el operador -) a la expresión postfija.

Paso	*EI*	Símbolo analizado	Pila	EPOS
0	$(X + Z)^* W / T \wedge Y - V$			
1	$X + Z)^* W / T \wedge Y - V$	(	(	
2	$+ Z)^* W / T \wedge Y - V$	X	(	X
3	$Z)^* W / T \wedge Y - V$	+	(+	X
4	$)^* W / T \wedge Y - V$	Z	(+	XZ
5	$^* W / T \wedge Y - V$	)	(	$XZ+$
		)		$XZ+$
6	$W / T \wedge Y - V$	*	*	$XZ+$
7	$/ T \wedge Y - V$	W	*	$XZ+W$
8	$T \wedge Y - V$	/	/	$XZ+W*$
		/	/	$XZ+W*$
9	$\wedge Y - V$	T	/	$XZ+W*T$
10	$Y - V$	$\wedge$	/^	$XZ+W*T$
11	$- V$	Y	/^	$XZ+W*TY$
		–	/	$XZ+W*TY\wedge$
12	V	–		$XZ+W*TY\wedge/$
		–	–	$XZ+W*TY\wedge/$
13		V	–	$XZ+W*TY\wedge/V$
14				$XZ+W*TY\wedge/V-$

Bibliografía

Cairó, O. y Guardati, S. (2002). Estructuras de Datos, 2da. Edición. McGraw-Hill.

Deitel P.J. y Deitel H.M. (2008) Cómo programar en C++. 6ª edición.Prentice Hall.

Joyanes, L. (2006). Programación en C++: Algoritmos, Estructuras de datos y objetos. McGraw-Hill.