Programación II

Tema 2. Pilas

Iván Cantador y Rosa Mª Carro

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Contenidos

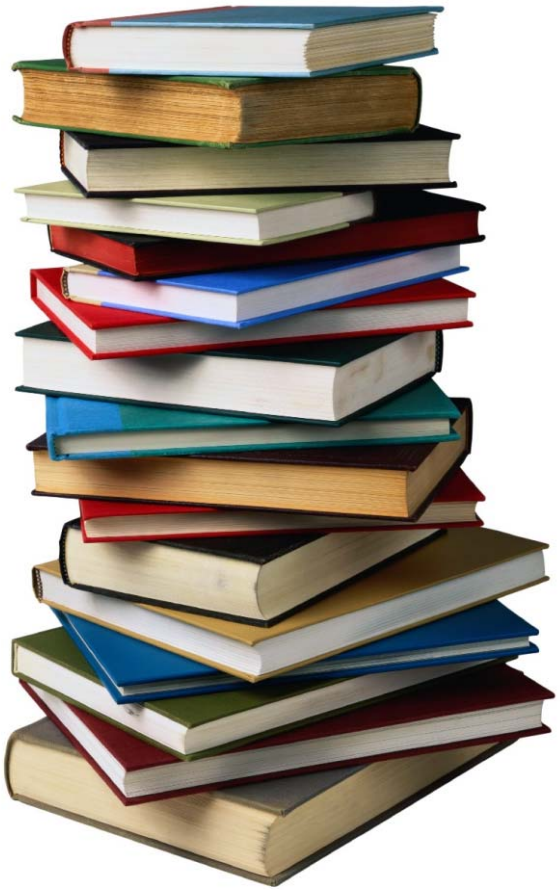
- El TAD Pila
- Estructura de datos y primitivas de Pila
- Implementación en C de Pila
  - Implementación con top de tipo entero
- Ejemplos de aplicación de Pila
  - Balanceo de paréntesis
  - Evaluación de expresiones posfijo
  - Conversión entre notaciones infijo, posfijo y prefijo
- Anexo
  - Implementación con top de tipo puntero

Contenidos

- **El TAD Pila**
- Estructura de datos y primitivas de Pila
- Implementación en C de Pila
  - Implementación con top de tipo entero
- Ejemplos de aplicación de Pila
  - Balanceo de paréntesis
  - Evaluación de expresiones posfijo
  - Conversión entre notaciones infijo, posfijo y prefijo
- Anexo
  - Implementación con top de tipo puntero

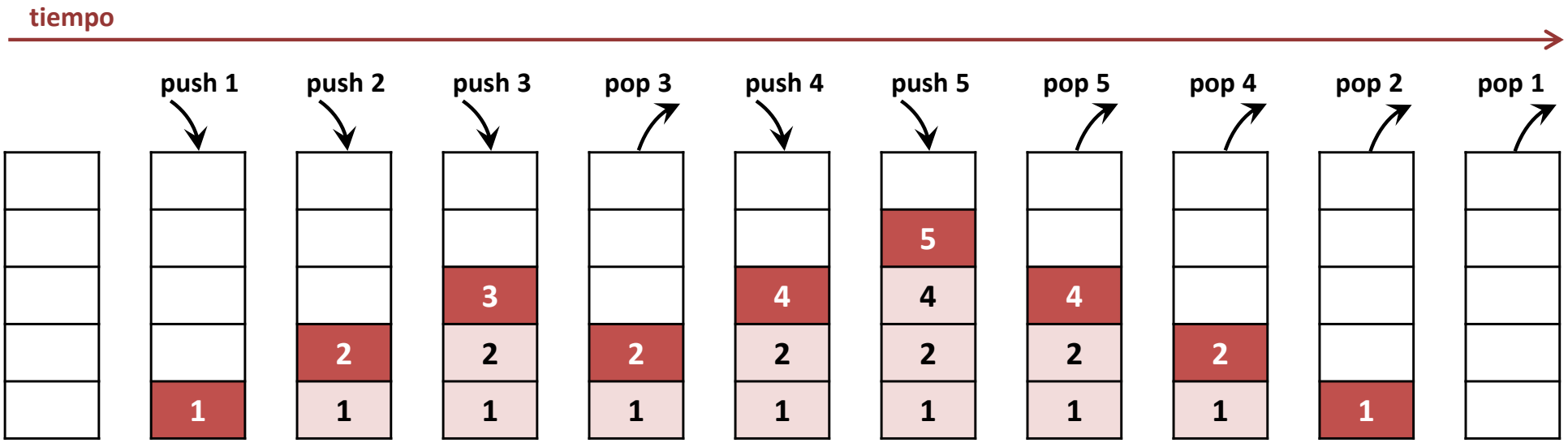
El TAD Pila

- **Pila** (***stack*** en inglés)
  - Colección de elementos LIFO - *Last In, First Out*: “el último que entra, el primero que sale”



El TAD Pila

- **Definición de Pila**
  - Contenedor de elementos que son insertados y extraídos siguiendo el principio de que el último que fue insertado será el primero en ser extraído (*LIFO – Last In, First Out*)
    - Los elementos se insertan de uno en uno: **push** (*apilar*)
    - Los elementos se extraen de uno en uno: **pop** (*desapilar*)
    - El último elemento insertado (que será el primero en ser extraído) es el único que se puede "observar" de la pila: **top** (*tope*, *cima*)



El TAD Pila

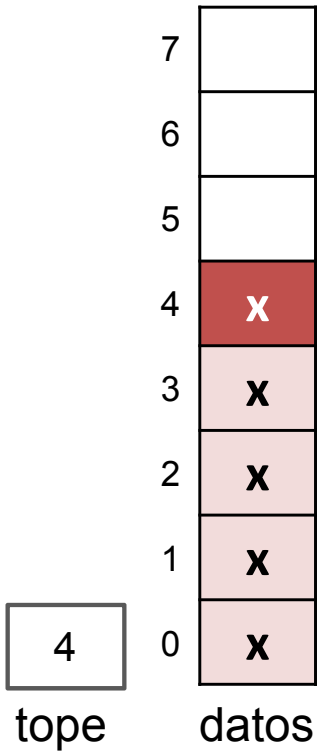
- **Aplicaciones reales de las pilas**
  - En general, todas aquellas aplicaciones que conlleven:
    - **estrategias "vuelta atrás"** (*back tracking*): la acción deshacer/*undo*
    - **algoritmos recursivos**
  - Ejemplos:
    - **Editores de texto**: pila con los últimos cambios realizados sobre un documento
    - **Navegadores web**: pila de direcciones con los sitios web más recientemente visitados
    - **Pila de programa**: zona de memoria de un programa donde se guardan temporalmente los argumentos de entrada de funciones
    - **Comprobación del balanceo de ()**, **{}**, **[]** en compiladores
    - ***Parsing* de código XML/HTML**, comprobando la existencia de etiquetas de comienzo <tag> y finalización </tag> de elementos en un documento XML
    - **Calculadoras con notación polaca inversa** (posfijo): se convierten expresiones "infijo" a "posfijo" soportando complejidad superior a la de calculadoras algebraicas (p.e., realizan cálculos parciales sin tener que pulsar "=")

Contenidos

- El TAD Pila
- **Estructura de datos y primitivas de Pila**
- Implementación en C de Pila
  - Implementación con top de tipo entero
- Ejemplos de aplicación de Pila
  - Balanceo de paréntesis
  - Evaluación de expresiones posfijo
  - Conversión entre notaciones infijo, posfijo y prefijo
- Anexo
  - Implementación con top de tipo puntero

Estructura de datos y primitivas de Pila

- **Una pila está formada por:**
  - **datos:** conjunto de elementos, en general del mismo tipo, ordenados implícitamente y accesibles desde un único punto: el *tope*
  - **tope**: indicador de la posición del último elemento insertado; da lugar a una ordenación LIFO (*last in, first out*)



(en este dibujo asumimos que la pila tiene tamaño máximo de 8, pero no tiene por qué ser siempre ese valor)

Estructura de datos y primitivas de Pila

- **Primitivas**

`Pila` **`pila_crear`**`()`: crea, inicializa y devuelve una pila

`pila_liberar``(Pila s)`: libera (la memoria ocupada por) la pila

`boolean` **`pila_vacia`**`(Pila s)`: devuelve *true* si la pila está vacía y *false* si no

`boolean` **`pila_llena`**`(Pila s)`: devuelve *true* si la pila está llena y *false* si no

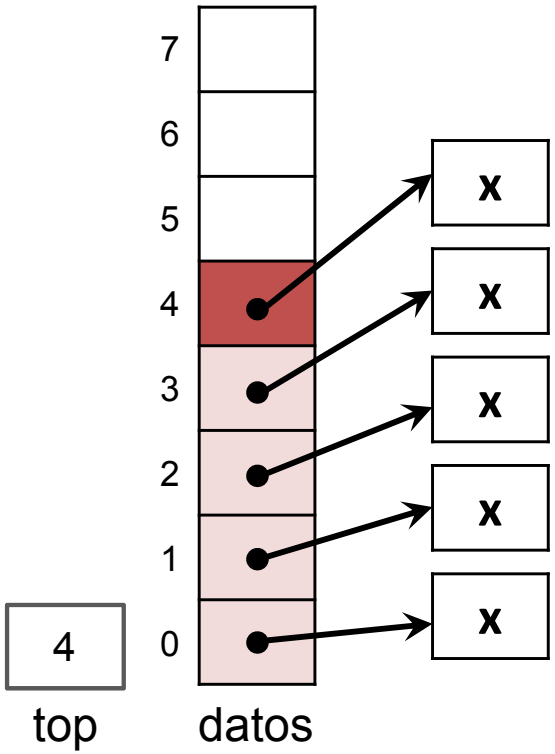
`status` **`pila_push`**`(Pila s, Elemento e)`: inserta un dato en una pila

`Elemento` **`pila_pop`**`(Pila s)`: extrae el dato que ocupa el top de la pila

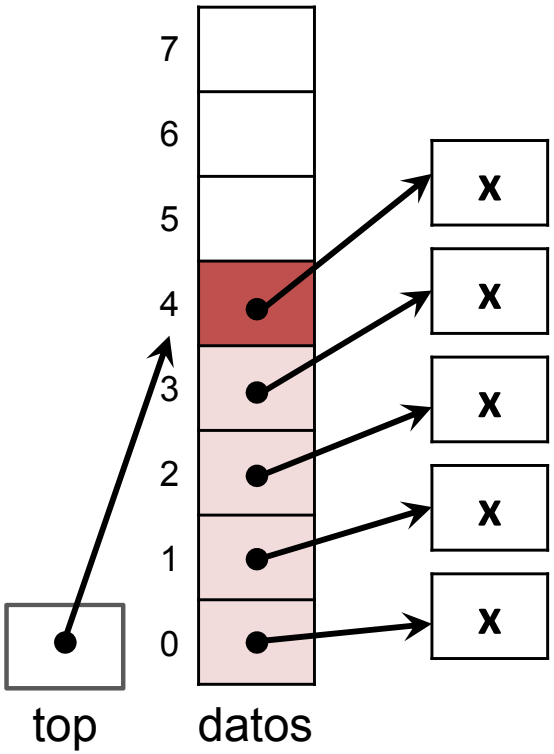
`Elemento` **`pila_getTop`**`(Pila s)`: devuelve el dato que ocupa el top de la pila sin extraerlo de ella

Estructura de datos y primitivas de Pila

- **EdD en C**
  - **datos**: en este tema será un array de punteros: `Elemento *datos[];`
  - **top**: en este tema se declarará de 2 maneras (versiones) distintas
    - V1: Como un entero: `int top;`
    - V2: Como un puntero a un elemento del array: `Elemento **top;`



Versión 1



Versión 2 (en anexo)

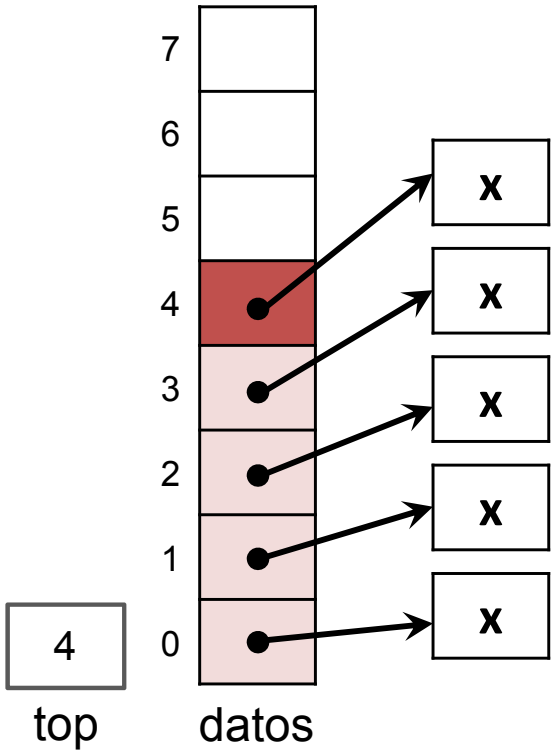
Contenidos

- El TAD Pila
- Estructura de datos y primitivas de Pila
- **Implementación en C de Pila**
  - **Implementación con top de tipo entero**
- Ejemplos de aplicación de Pila
  - Balanceo de paréntesis
  - Evaluación de expresiones posfijo
  - Conversión entre notaciones infijo, posfijo y prefijo
- Anexo
  - Implementación con top de tipo puntero

Implementación en C de Pila

- **Implementación con top de tipo entero**
  - Asumimos la existencia del TAD Elemento
  - **EdD de Pila mediante un array**

```
// En pila.h
typedef struct _Pila Pila;

// En pila.c
#define PILA_MAX 8
struct _Pila {
   Elemento *datos[PILA_MAX];
   int      top;
};
```



Implementación en C de Pila

- **Implementación con top de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

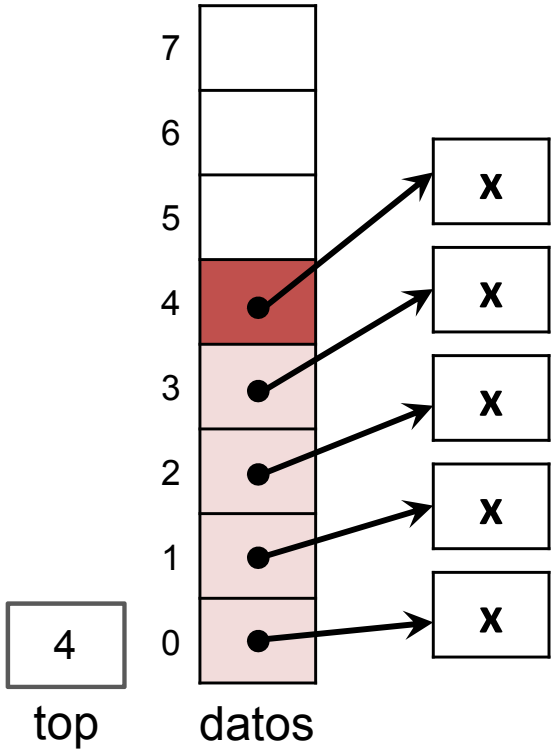
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```

  - **Estructura de datos (en pila.c)**

```
struct _Pila {
   Elemento *datos[PILA_MAX];
   int      top;
};
```



Implementación en C de Pila

- **Implementación con top de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

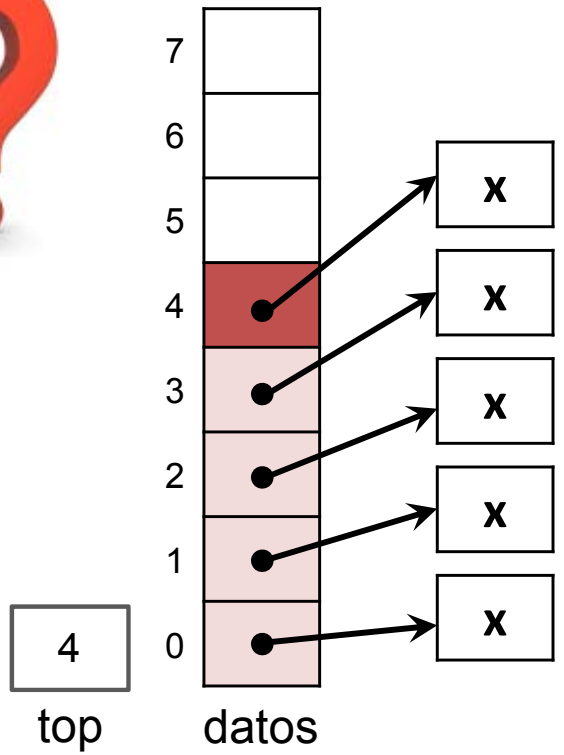
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```

  - **Estructura de datos (en pila.c)**

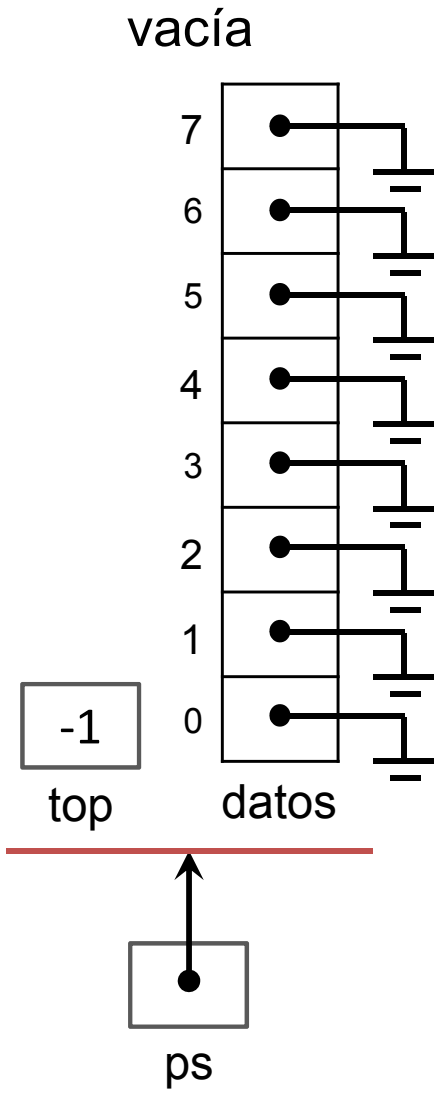
```
struct _Pila {
   Elemento *datos[PILA_MAX];
   int      top;
};
```



Implementación en C de Pila

- **Implementación con top de tipo entero**

```
Pila *pila_crear() {
    Pila *ps = NULL;
    int i;

    ps = (Pila *) malloc(sizeof(Pila));
    if (ps == NULL) {
        return NULL;
    }

    for (i=0; i<PILA_MAX; i++) {   // Bucle opcional
        ps->datos[i] = NULL;
    }

    ps->top = -1;

    return ps;
}
```

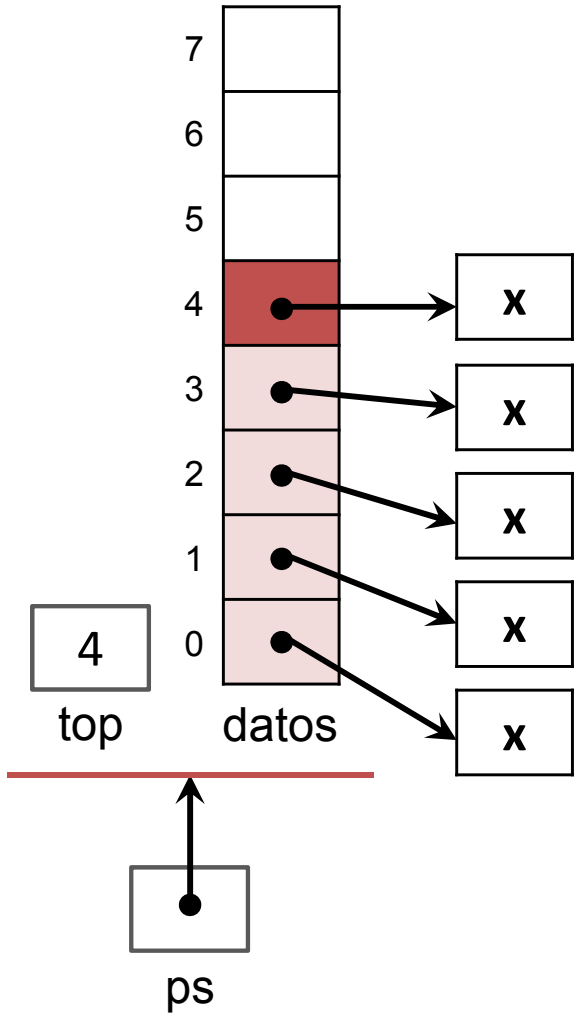


Implementación en C de Pila

- **Implementación con top de tipo entero**

Existe: `void` **`elemento_liberar`**`(Elemento *pe);`

```
void pila_liberar(Pila *ps) {




}
```

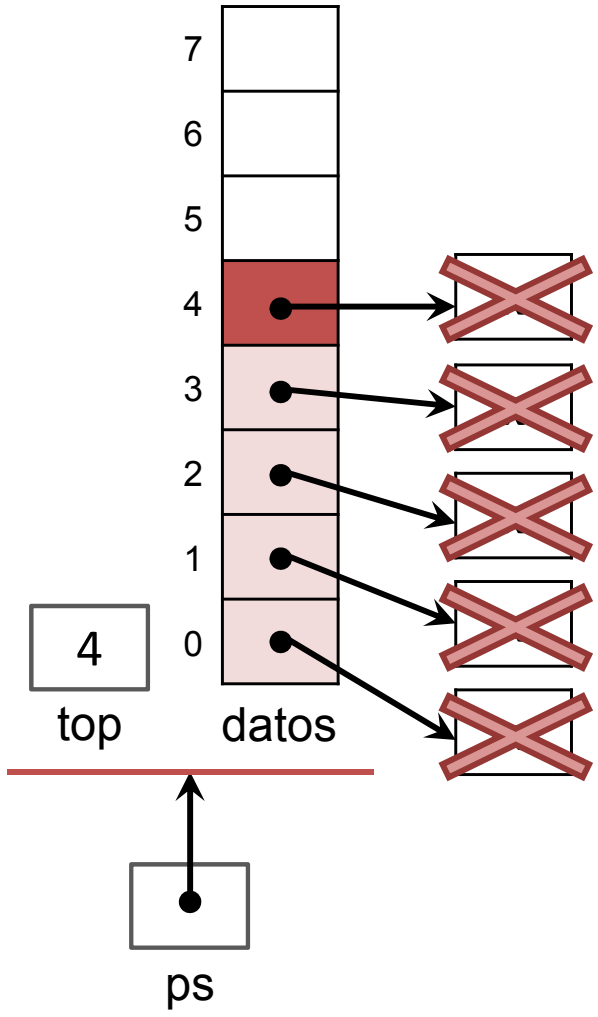


Implementación en C de Pila

- **Implementación con top de tipo entero**

Existe: `void` **`elemento_liberar`**`(Elemento *pe);`

```
void pila_liberar(Pila *ps) {
    int i;

    if (ps != NULL) {
        for (i=0; i<=ps->top; i++) {
            elemento_liberar(ps->datos[i]);
            ps->datos[i] = NULL;  // Opcional
        }



    }
}
```

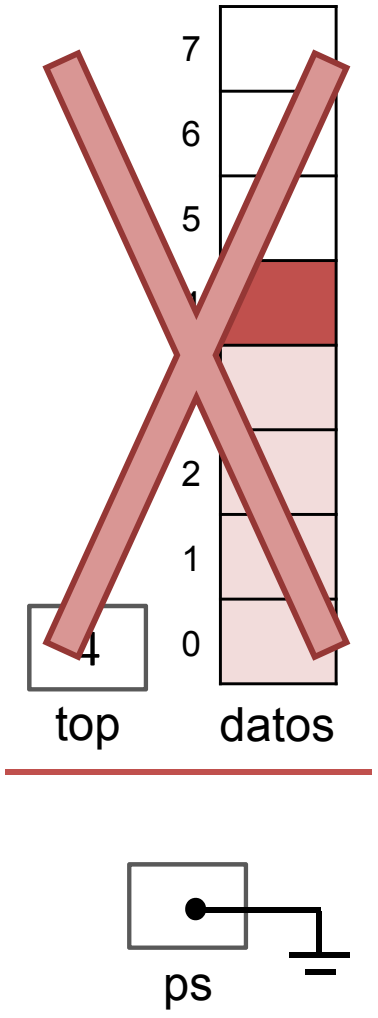


Implementación en C de Pila

- **Implementación con top de tipo entero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void pila_liberar(Pila *ps) {
    int i;

    if (ps != NULL) {
        for (i=0; i<=ps->top; i++) {
            elemento_liberar(ps->datos[i]);
            ps->datos[i] = NULL;  // Opcional
        }
        free(ps);
        // ps = NULL; se hace fuera, tras llamar
        // a pila_liberar
    }
}
```



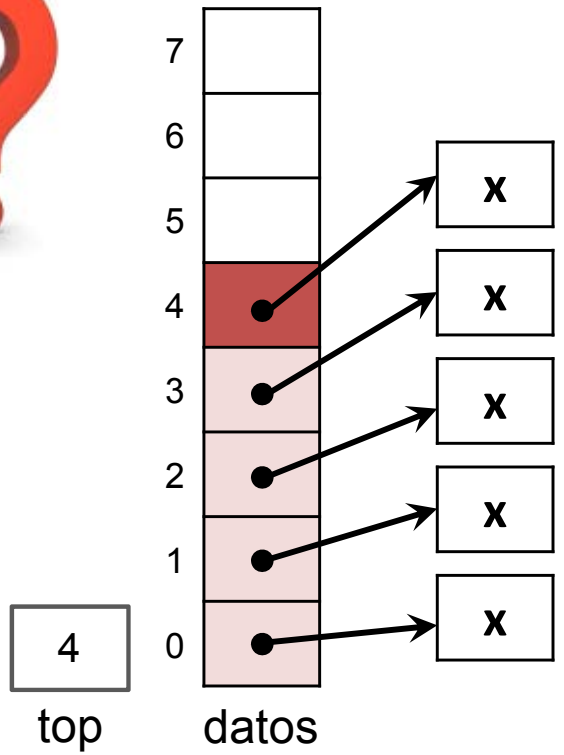
Implementación en C de Pila

- **Implementación con top de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```



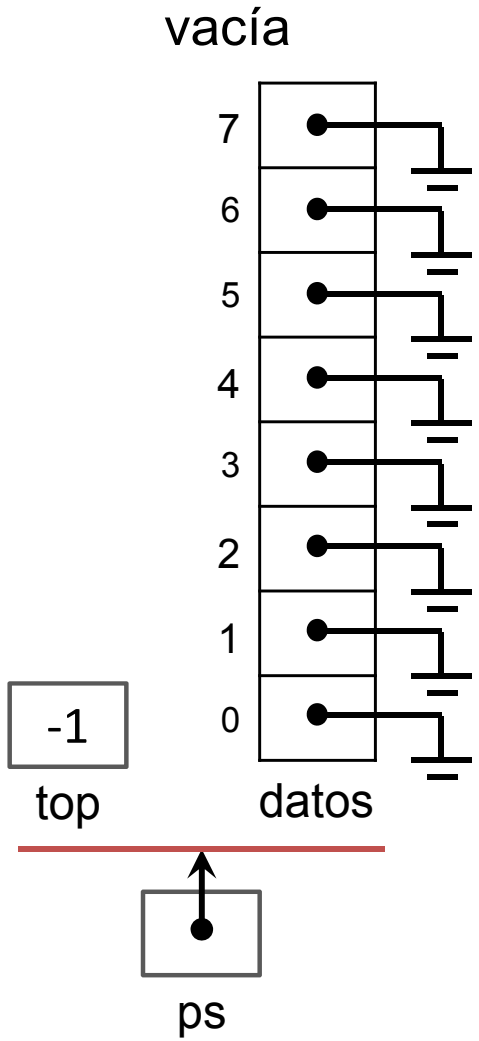
  - **Estructura de datos (en pila.c)**

```
struct _Pila {
   Elemento *datos[PILA_MAX];
   int      top;
};
```

Implementación en C de Pila

- **Implementación con top de tipo entero**

```
boolean pila_vacia(const Pila *ps) {
    if (ps == NULL) {
        return TRUE;
    }

    if (ps->top == -1) {
        return TRUE;
    }
    return FALSE;
}
```



Nota: 

Llamaremos a pila_vacia antes de hacer pop, comprobando si hay elementos en la pila. Si la pila está vacía, no llamaremos a pop.

Por eso, si ps es NULL, aquí detectamos el error y devolvemos TRUE indicando que la pila "está" vacía, con el fin de que luego no se haga pop.

Implementación en C de Pila

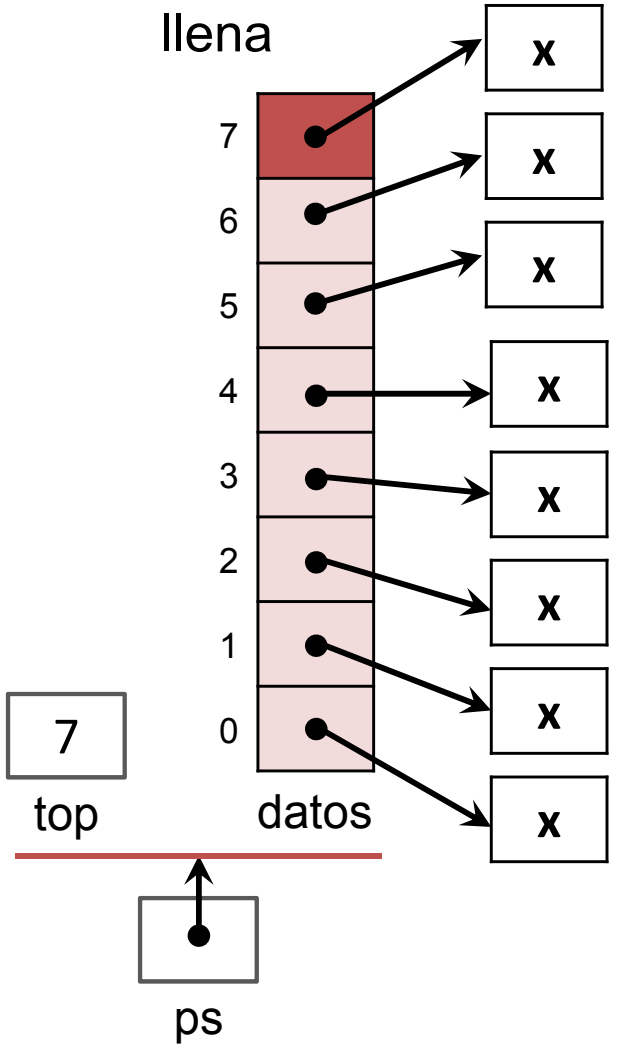
- **Implementación con top de tipo entero**

```
boolean pila_llena(const Pila *ps) {
    if (ps == NULL) {
        return TRUE;
    }

    if (ps->top == PILA_MAX-1) {
        return TRUE;
    }
    return FALSE;
}
```

Nota:

Llamaremos a pila_llena antes de hacer push, comprobando si la pila tiene o no capacidad. Si la pila está llena, no llamaremos a push.

Por eso, si ps es NULL, aquí detectamos el error y devolvemos TRUE indicando que la pila "está" llena, con el fin de que luego no se haga push.



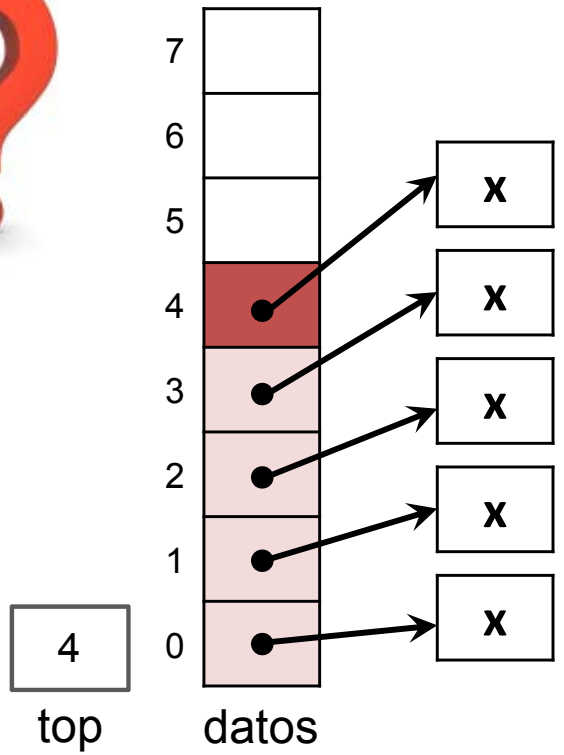
Implementación en C de Pila

- **Implementación con top de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```



  - **Estructura de datos (en pila.c)**

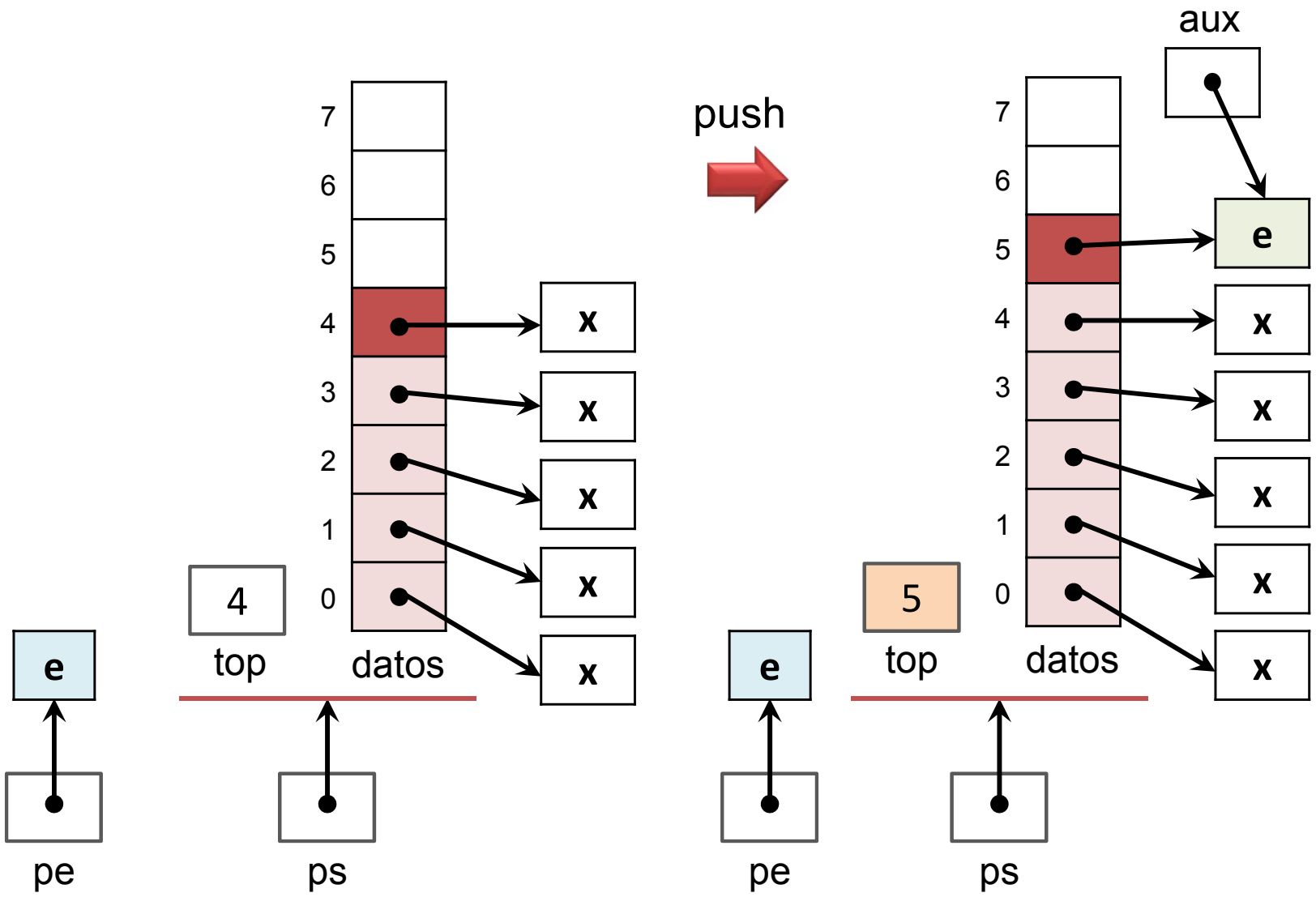
```
struct _Pila {
   Elemento *datos[PILA_MAX];
   int      top;
};
```

Implementación en C de Pila

- **Implementación con top de tipo entero**

```
status pila_push(Pila *ps, const Elemento *pe) {
```

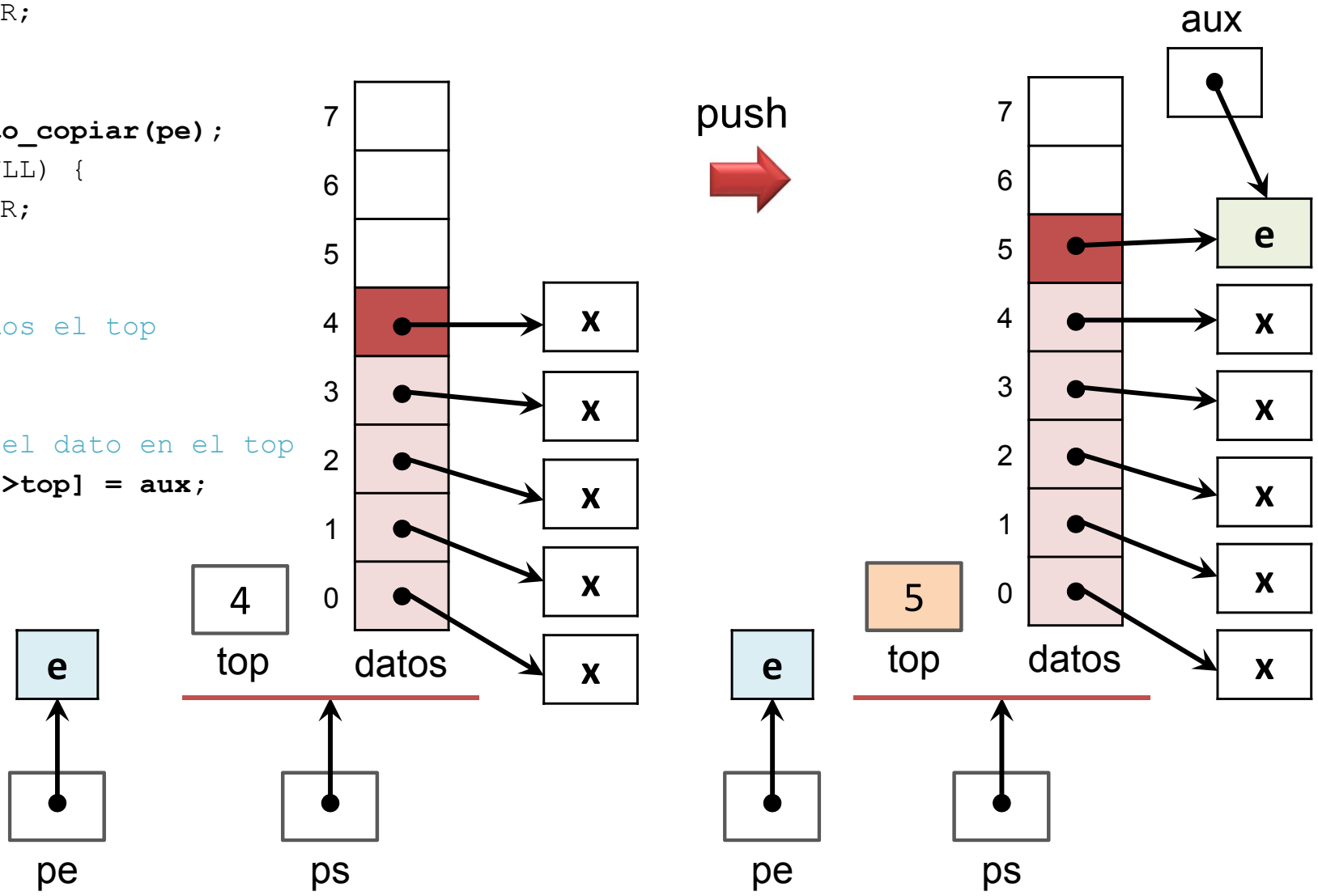
```
}
```



Implementación en C de Pila

- **Implementación con top de tipo entero**

```
status pila_push(Pila *ps, const Elemento *pe) {
    Elemento *aux = NULL;

    if (ps == NULL || pe == NULL || pila_llena(ps) == TRUE) {
        return ERR;
    }

    aux = elemento_copiar(pe);
    if (aux == NULL) {
        return ERR;
    }

    // Actualizamos el top
    ps->top++;

    // Guardamos el dato en el top
    ps->datos[ps->top] = aux;

    return OK;
}
```



Implementación en C de Pila

- **Implementación con top de tipo entero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

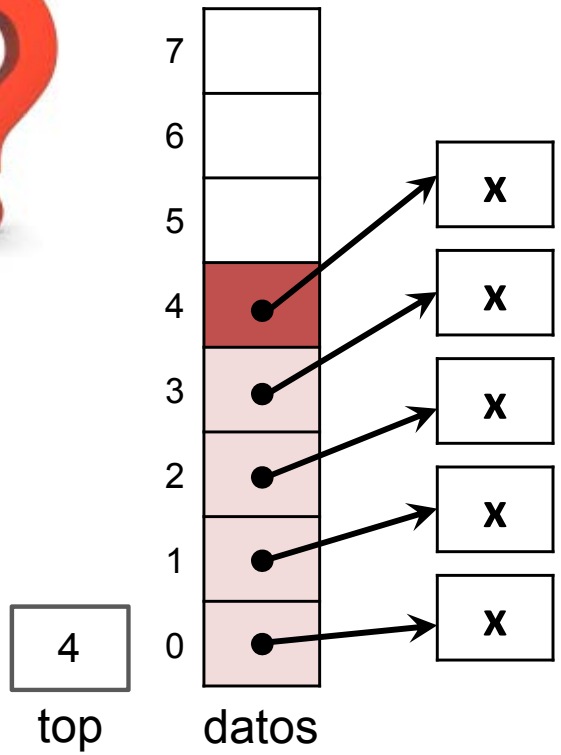
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```

  - **Estructura de datos (en pila.c)**

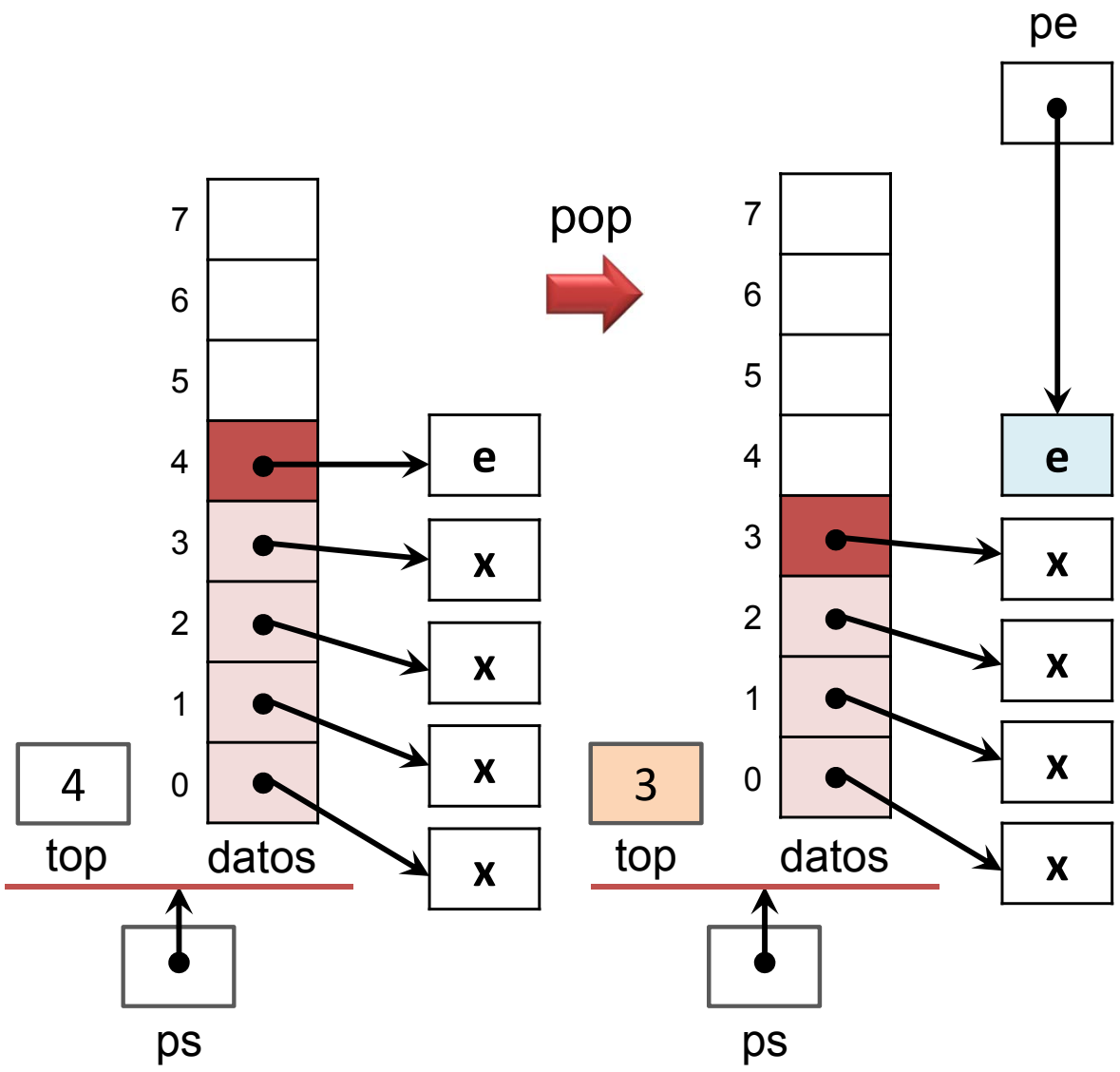
```
struct _Pila {
   Elemento *datos[PILA_MAX];
   int      top;
};
```



Implementación en C de Pila

- **Implementación con top de tipo entero**

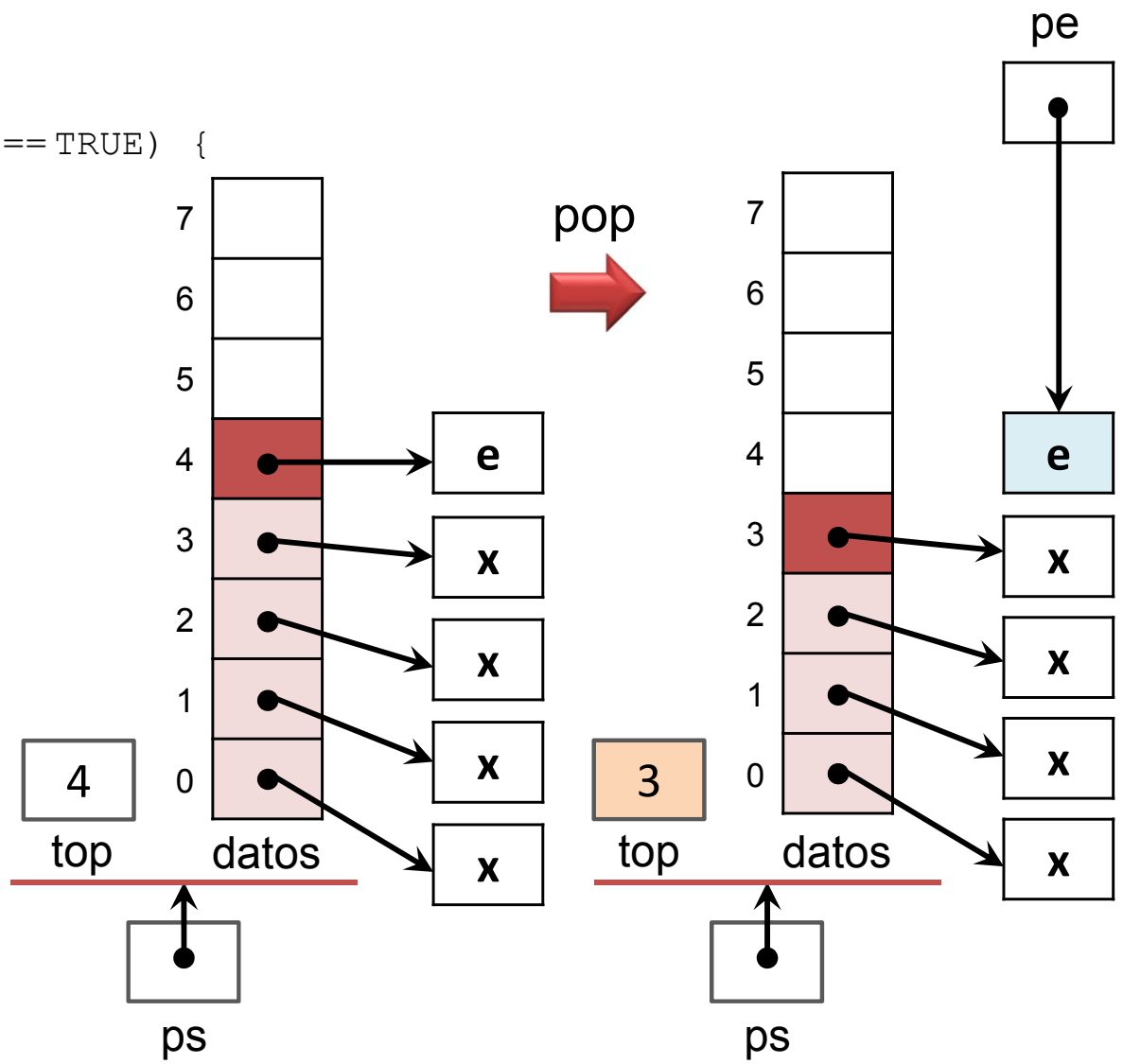
```
Elemento *pila_pop(Pila *ps){

}
```



Implementación en C de Pila

- **Implementación con top de tipo entero**

```
Elemento *pila_pop(Pila *ps){
    Elemento *pe = NULL;

    if (ps==NULL|| pila_vacia(ps)==TRUE) {
        return NULL;
    }

    // Recuperamos el dato del top
    pe = ps->datos[ps->top];

    // Ponemos el Elemento* a NULL
    ps->datos [ps->top] = NULL;

    // Actualizamos el top
    ps->top--;

    return pe;
}
```



Nota: el elemento cuyo puntero se devuelve ha de liberarse fuera, tras la llamada a pop

Contenidos

- El TAD Pila
- Estructura de datos y primitivas de Pila
- Implementación en C de Pila
  - Implementación con top de tipo entero
- **Ejemplos de aplicación de Pila**
  - **Balanceo de paréntesis**
  - Evaluación de expresiones posfijo
  - Conversión entre notaciones infijo, posfijo y prefijo
- Anexo
  - Implementación con top de tipo puntero

Balanceo de paréntesis

- Determinar si una expresión (p.e., una operación aritmética) contiene el mismo número de paréntesis de apertura que de cierre

  `( 3 * 4 / ( 6 - 1 ) )` → CORRECTA

  `2 + ( 5 / ( 4 + 7 )` → INCORRECTA

  `6 * 4 + 9 )` → INCORRECTA

Balanceo de paréntesis

- Pseudocódigo (sin CdE)

```
boolean balanceoParentesis(CC S) {
    p = pila_crear()

    mientras (it=leerSimbolo(S)) ≠ FIN:     // leer siguiente símbolo de S
        si esParentesisApertura(it)         // si it = (
            pila_push(p,it)
        si no, si esParentesisCierre(it)    // si it = )
            pila_pop(p)


    si pila_vacia(p)=FALSE
        pila_liberar(p)
        devolver FALSE

    pila_liberar(p)
    devolver TRUE
}
```

¿Detección de errores de balanceo?

- Ejemplo 1: **(3 * 4 / (6 - 1))**

Símbolo	Pila de paréntesis
(	(
3	(
*	(
4	(
/	(
(	((
6	((
-	((
1	((
)	(
)	

Balanceo de paréntesis

- Pseudocódigo (sin CdE)

```
boolean balanceoParentesis(CC S) {
    p = pila_crear()

    mientras (it=leerSimbolo(S)) ≠ FIN:     // leer siguiente símbolo de S
        si esParentesisApertura(it)         // si it = (
            pila_push(p,it)
        si no, si esParentesisCierre(it)    // si it = )
            si pila_pop(p)=NULL  (A)
                pila_liberar(p)
                devolver FALSE

    si pila_vacia(p)=FALSE  (B)
        pila_liberar(p)
        devolver FALSE
    pila_liberar(p)
    devolver TRUE
}
```

Errores de balanceo se detectan en: (A) (B)

¿Posibles fuentes de error?

- Ejemplo 1: **(3 * 4 / (6 - 1))**

Símbolo	Pila de paréntesis
(	(
3	(
*	(
4	(
/	(
(	((
6	((
-	((
1	((
)	(
)	

Balanceo de paréntesis

- Pseudocódigo (sin CdE)

```
boolean balanceoParentesis(CC S) {
1  p = pila_crear()

   mientras (it=leerSimbolo(S)) ≠ FIN:     // leer siguiente símbolo de S
      si esParentesisApertura(it)          // si it = (
2     pila_push(p,it)
      si no, si esParentesisCierre(it)     // si it = )
         si pila_pop(p)=NULL               A
            pila_liberar(p)
            devolver FALSE

   si pila_vacia(p)=FALSE                  B
      pila_liberar(p)
      devolver FALSE
   pila_liberar(p)
   devolver TRUE
}
```

Posibles fuentes de error (funciones que pueden devolver ERROR): 1 2

Errores de balanceo se detectan en: A B

- Ejemplo 2: `2 + ( 5 / ( 4 + 7 )`
- Ejemplo 3: `6 * 4 + 9 )`

- Ejemplo 1: `( 3 * 4 / ( 6 - 1 ) )`

Símbolo	Pila de paréntesis
(	(
3	(
*	(
4	(
/	(
(	((
6	((
-	((
1	((
)	(
)	

Balanceo de paréntesis

- Determinar si una expresión (p.e. una operación aritmética) contiene el mismo número de paréntesis de apertura que de cierre, **distinguiendo '()', '{}', '[]'**

```
{ 3 * 4 / ( 6 - 1 ) }
```
→ CORRECTA

```
2 + ( 5 / [ 4 + 7 ) )
```
→ INCORRECTA

```
6 * 4 + 9 }
```
→ INCORRECTA

Balanceo de paréntesis

- Pseudocódigo incompleto (solo mira paréntesis, sin CdE)

```
boolean balanceoParentesis(CC S) {
    p = pila_crear()

    mientras (it=leerSimbolo(S)) ≠ FIN:   // leer siguiente símbolo de S
        si esParentesisApertura(it)       // si it = '('
            pila_push(p,it)
        si no, si esParentesisCierre(it)  // si it = ')'
            si pila_pop(p)=NULL
                pila_liberar(p)
                devolver FALSE

    si pila_vacia(p)=FALSE
        pila_liberar(p)
        devolver FALSE
    pila_liberar(p)
    devolver TRUE
}
```

Balanceo de paréntesis

- Pseudocódigo con distintos símbolos (sin CdE)

```
boolean balanceoParentesis(CC S) {
    p = pila_crear()

    mientras (it=leerSimbolo(S)) ≠ FIN:  // leer siguiente símbolo de S
        si esSimboloApertura(it)         // si es (, [ o {
            pila_push(p,it)
        si no, si esSimboloCierre(it)
            e = pila_pop(p)
            si e=NULL O (it=')' Y e≠'(')
                       O (it=']' Y e≠'[')
                       O (it='}' Y e≠'{')
                pila_liberar(p)
                devolver FALSE

    si pila_vacia(p) = FALSE
        pila_liberar(p)
        devolver FALSE
    pila_liberar(p)
    devolver TRUE
}
```

Balanceo de paréntesis

- Pseudocódigo (sin CdE)

```
boolean balanceoParentesis(CC S) {
    p = pila_crear()

    mientras (it=leerSimbolo(S)) ≠ FIN:  // leer siguiente símbolo de S
        si esSimboloApertura(it)         // si es (, [ o {
            pila_push(p,it)
        si no, si esSimboloCierre(it)
            e = pila_pop(p)
            si e=NULL O sonParentesisPareja(it,e)=FALSE
                pila_liberar(p)
                devolver FALSE

    si pila_vacia(p) = FALSE
        pila_liberar(p)
        devolver FALSE
    pila_liberar(p)
    devolver TRUE
}
```

Balanceo de paréntesis

- Pseudocódigo (sin CdE)

```
boolean balanceoParentesis(CC S) {
    p = pila_crear()

    mientras (it=leerSimbolo(S)) ≠ FIN:   // leer siguiente símbolo de S
        si esSimboloApertura(it)          // si es (, [ o {
            pila_push(p,it)
        si no, si esSimboloCierre(it)
            e = pila_pop(p)
        A si e=NULL O sonParentesisPareja(it,e)=FALSE B
            pila_liberar(p)
            devolver FALSE

    si pila_vacia(p) = FALSE C
        pila_liberar(p)
        devolver FALSE
    pila_liberar(p)
    devolver TRUE
}
```

Errores de balanceo se detectan en: A B C

Balanceo de paréntesis

- Pseudocódigo (sin CdE)

```
boolean balanceoParentesis(CC S) {
1  p = pila_crear()

   mientras (it=leerSimbolo(S)) ≠ FIN:   // leer siguiente símbolo de S
      si esSimboloApertura(it)           // si es (, [ o {
2     pila_push(p,it)
      si no, si esSimboloCierre(it)
         e = pila_pop(p)
A     si e=NULL O sonParentesisPareja(it,e)=FALSE B
         pila_liberar(p)
         devolver FALSE

   si pila_vacia(p) = FALSE C
      pila_liberar(p)
      devolver FALSE
   pila_liberar(p)
   devolver TRUE
}
```

Posibles fuentes de error (funciones que pueden devolver ERROR): 1 2

Errores de balanceo se detectan en: A B C

Balanceo de paréntesis

- Pseudocódigo (sin CdE)

```
boolean balanceoParentesis(CC S) {
    p = pila_crear()

    mientras (it=leerSimbolo(S)) ≠ FIN:   // leer siguiente símbolo de S
        si esSimboloApertura(it)          // si es (, [ o {
            pila_push(p,it)
        si no, si esSimboloCierre(it)
            e = pila_pop(p)
            si e=NULL O sonParentesisPareja(it,e)=FALSE
                pila_liberar(p)
                devolver FALSE

    si pila_vacia(p) = FALSE
        pila_liberar(p)
        devolver FALSE
    pila_liberar(p)
    devolver TRUE
}
```

- Ejemplo 4: `{ 3 * 4 / ( 6 - 1 ) }`
- Ejemplo 5: `2 + ( 5 / [ 4 + 7 ) )`

Contenidos

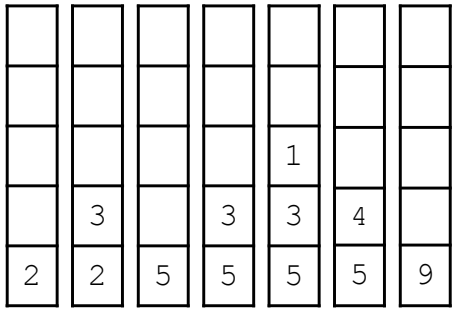
- El TAD Pila
- Estructura de datos y primitivas de Pila
- Implementación en C de Pila
  - Implementación con top de tipo entero
- **Ejemplos de aplicación de Pila**
  - Balanceo de paréntesis
  - **Evaluación de expresiones posfijo**
  - Conversión entre notaciones infijo, posfijo y prefijo
- Anexo
  - Implementación con top de tipo puntero

Evaluación de expresiones posfijo

- **Notación infijo**
  - Los operadores se indican entre operandos: **`A+B`**
  - La habitual
- **Notación posfijo (o sufijo)**
  - Los operadores se indican después de los operandos: **`AB+`**
  - La más "práctica" para un ordenador
- **Notación prefijo**
  - Los operadores se indican antes de los operandos: **`+AB`**
- Ejemplo:
  - En infijo: `2 - 3 * (4 + 1)`
  - En posfijo: `2 3 4 1 + * -`
  - En prefijo: `- 2 * 3 + 4 1`

Evaluación de expresiones posfijo

- **Ejemplos**
  - Ejemplo: `2 3 + 3 1 + + ;`

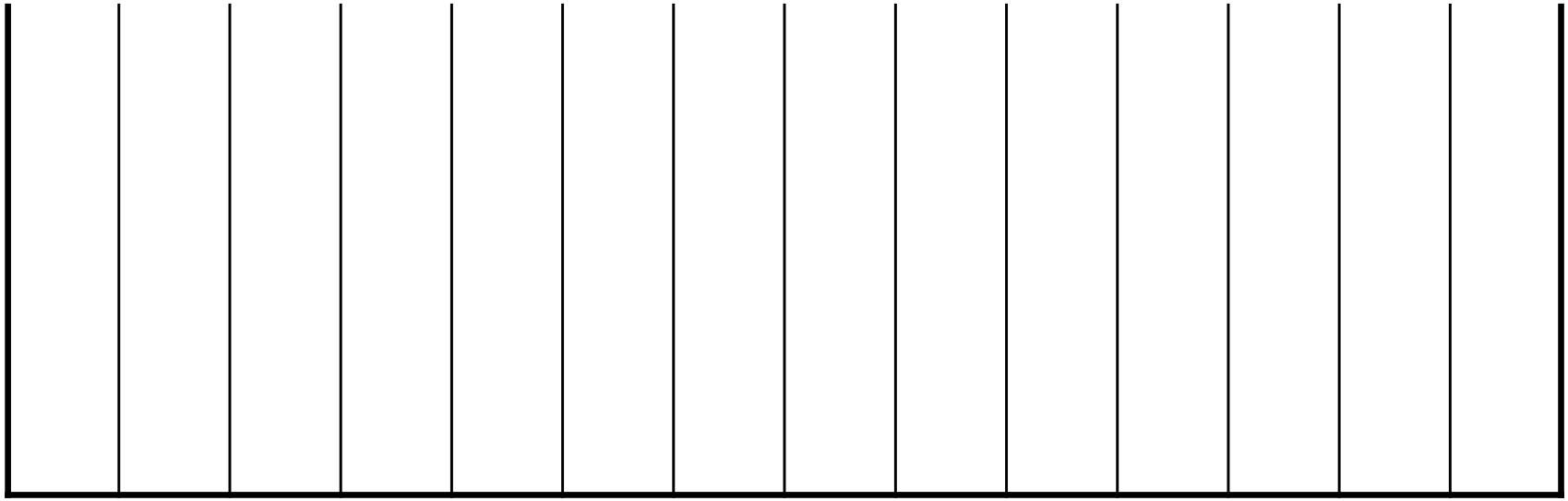


- Ejemplo: `1 2 * 4 2 / 3 4 + 5 * - + ;`

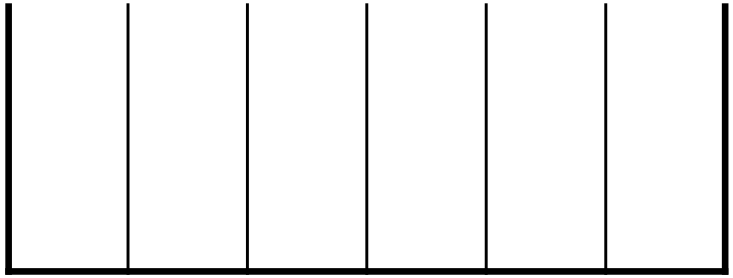
Evaluación de expresiones posfijo

- **Ejemplos**

6 2 / 3 2 * 5 1 - 3 * + - ;



A B * C - ;



- 2 3 + 3 1 + + ; → 9
- 1 2 * 4 2 / 3 4 + 5 * - + ; → – 31

Evaluación de expresiones posfijo

- **Pseudocódigo (sin CdE)**

```
status evaluarPosfijo(CC S, Entero r) {
    p = pila_crear()
    mientras (it=leerSimbolo(S)) ≠ FIN:     // leer siguiente símbolo de S
          si esOperando(it)
             pila_push(p,it)
          else si esOperador(it)
             op2 = pila_pop(p)
             op1 = pila_pop(p)
             e = evaluar(op1,op2,it)
             pila_push(p,e)

    r = pila_pop(p)
    si pila_vacia(p)=FALSE     // si la pila no queda vacía, incorrecto
       pila_liberar(p)
       devolver ERROR

    // Si queda vacía, correcto; en r está el resultado (el ultimo pop)
    pila_liberar(p)
    devolver OK
}
```

- Ejemplo: **1 6 * 2 1 / 3 4 - + ;**

¿Control de errores?

Evaluación de expresiones posfijo

- **Pseudocódigo (con CdE)**

```
status evaluarPosfijo(CC S, Entero r) {
    si (p=pila_crear())=NULL devolver ERROR
    mientras (it = leerItem(S)) ≠ FIN:
        si esOperando(it)
            si pila_push(p,it = ERROR) // ERROR 1: la pila está llena
                pila_liberar(p)
                devolver ERROR
        si no, si esOperador(it)
            op2 = pila_pop(p)
            op1 = pila_pop(p)
            si (op2 ≠ ERROR) Y (op1 ≠ ERROR)
                e = evaluar(op1,op2,it)
                pila_push(p,e) //Como se extraen 2, se assume que cabe 1
            si no // ERROR 2: no hay suficientes operandos en la pila
                pila_liberar(p)
                devolver ERROR
        r = pila_pop(p)
        si r=NULL //ERROR 3: imposible extraer resultado
            pila_liberar(p)
            devolver ERROR
        si (pila_vacia(p)=FALSE) //ERROR 4: la pila al final no queda vacía
            pila_liberar(p)
            devolver ERROR
     devolver OK
}
```

Evaluación de expresiones posfijo

- **Pseudocódigo (con CdE)**

```
status evaluarPosfijo(CC S, Entero r) {
    si (p=pila_crear())=NULL devolver ERROR
    mientras (it = leerItem(S)) ≠ FIN:
        si esOperando(it)
            si pila_push(p,it = ERROR) // ERROR 1: la pila está llena
                pila_liberar(p)
                devolver ERROR
        si no, si esOperador(it)
            op2 = pila_pop(p)
            op1 = pila_pop(p)
            si (op2 ≠ ERROR) Y (op1 ≠ ERROR)
                e = evaluar(op1,op2,it)
                pila_push(p,e) //Como se extraen 2, se assume que cabe 1
            si no // ERROR 2: no hay suficientes operandos en la pila
                pila_liberar(p)
                devolver ERROR
        r = pila_pop(p)
        si r=NULL //ERROR 3: imposible extraer resultado
            pila_liberar(p)
            devolver ERROR
        si (pila_vacia(p)=FALSE) //ERROR 4: la pila al final no queda vacía
            pila_liberar(p)
            devolver ERROR
     devolver OK
}
```

- Ejemplo: `1 6 * 2 + / ;`
- Ejemplo: `1 6 * 2 1 ;`
- Ejemplo: `1 3 * 4 2 / + 4 3 - 5 * + ;`

Contenidos

- El TAD Pila
- Estructura de datos y primitivas de Pila
- Implementación en C de Pila
  - Implementación con top de tipo entero
- **Ejemplos de aplicación de Pila**
  - Balanceo de paréntesis
  - Evaluación de expresiones posfijo
  - **Conversión entre notaciones infijo, posfijo y prefijo**
- Anexo
  - Implementación con top de tipo puntero

Conversión de expresiones: infijo-posfijo

- **Infijo → Posfijo**
- Conversión de A + B * C
  - A +(B * C)=> ABC*+
  - A + B * C => ABC*+
- Hay que tener en cuenta:
  - Precedencia de operadores
  - Asociatividad de operaciones (a igual precedencia, de izquierda a derecha)

precedencia

\+

\-

operador	asociatividad
() [] . ->	izquierda-derecha
* / %	izquierda-derecha
+ -	
&&	
||	
=	derecha-izquierda

Conversión de expresiones: infijo-posfijo

- **Idea inicial (¿correcta o errónea?)**

`A + B * C ; => ABC*+`

Símbolo	Traducción parcial	Pila de OPERADORES
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
;	ABC*+	

- **Posible algoritmo** ¿Funciona para `A * B + C / E ;` ?

1. Si el símbolo actual it es operando: "imprimir" (añadir it a cadena de traducción parcial)
2. Si it es operador: pila_push(pila, it)
3. Al final: vaciar pila e imprimiendo elementos en cadena traducción en el orden en que se van extrayendo

Conversión de expresiones: infijo-posfijo

- **Otro ejemplo**

```
A * B + C / E ; => AB*CE/+
```

Símbolo	Traducción parcial	Pila de OPERADORES
A	A	
*	A	*
B	AB	*
+	AB	* +
C	ABC	* +
/	ABC	* + /
E	ABCE	* + /
;	ABCE/+* MAL	

- **Ajustes al algoritmo**
  - Los operadores de mayor precedencia salen antes de pila
  - A igual precedencia, sale antes el que se leyó antes

Conversión de expresiones: infijo-posfijo

- Teniendo en cuenta precedencia de operadores

```
A * B + C / E ; => AB*CE/+
```

Símbolo	Traducción parcial	Pila de OPERADORES
A	A	
*	A	*
B	AB	*
+	AB*	+
C	AB*C	+
/	AB*C	+ /
E	AB*CE	+ /
;	AB*CE/+	

- **Idea para el algoritmo**

1. Si it es operando, imprimir it
2. Un operador leído saca de la pila todos los operadores de mayor o igual precedencia, en el orden correspondiente (pop)

   >: el más preferente sale antes

   =: por la regla de asociatividad izquierda-derecha, también sale antes
3. Al final, vaciar la pila de operandos e imprimir

Conversión de expresiones: infijo-posfijo

- **Algoritmo**
  1. Si it es operando, añadir a la expresión
  2. Si es operador:
     mientras prec (operador) ≤ prec (tope(pila))
     extraer elemento de la pila y añadir a la expresión
     meter operador leído en la pila
  3. Si es fin de cadena:
     mientras pila no vacía:
     extraer elemento de la pila y añadirlo a la expresión

- Teniendo en cuenta precedencia de operadores

`A + B * C – D ; => ABC*+D-`

Símbolo	Traducción parcial	Pila de OPERADORES
A	A	
+	A	+
B	AB	+
*	AB	+ *
C	ABC	+ *
-	ABC*+	-
D	ABC*+D	-
;	ABC*+D-	

Conversión de expresiones: infijo-posfijo

- **Pseudocódigo (sin CdE)**

```
status infijoAPosfijo(CC S1, CC S2)
    p = pila_crear()
    mientras (it=leerItem(S1)) ≠ FIN:
        si esOperando(it)=TRUE
            print(S2,it)
        si no, si esOperador(it)=TRUE
            mientras (pila_vacia(p)=FALSE) Y (prec(it)≤ prec(top(p))
                e = pila_pop(p)
                print(S2,e)
            pila_push(p,it)
    mientras pila_vacia(p)=FALSE:
        it = pila_pop(p)
        print(S2,it)
    pila_liberar(p)
    devolver OK
```

prec(op): Devuelve el nivel de precedencia de un operador
top(p): Devuelve el valor en el top de la pila, sin extraerlo
print(S,it): Concatena el carácter it al final de la cadena S

- ¿CdE?
  - pila_crear: comprobarlo
  - pila_pop: no hay que controlar error, porque justo antes se mira si está vacía.
  - pila_push: si pila llena, error
  - operadores correctos: leerItem, esOperando, esOperador

Conversión de expresiones: infijo-posfijo

- **Pseudocódigo (sin CdE)**

```
status infijoAPosfijo(CC S1, CC S2)
1   p = pila_crear()
    mientras (it=leerItem(S1)) ≠ FIN:
        si esOperando(it)=TRUE
            print(S2,it)
        si no, si esOperador(it)=TRUE
            mientras (pila_vacia(p)=FALSE) Y (prec(it)≤ prec(top(p))
                e = pila_pop(p)
                print(S2,e)
        2 pila_push(p,it)
    mientras pila_vacia(p)=FALSE:
        it = pila_pop(p)
        print(S2,it)
    pila_liberar(p)
    devolver OK
```

Línea 1 cambia a:

```
si (p=pila_crear())=ERROR
    devolver ERROR
```

Línea 2 cambia a:

```
si pila_push(p,it)=ERROR
    pila_liberar(p)
    devolver ERROR
```

Conversión de expresiones: infijo-posfijo

- ¡Ojo! A diferencia de la evaluación, el algoritmo de conversión no detecta errores sintácticos
  - A+B* → AB*+
  - ABC* → ABC*
- **Función de precedencia**

```
int prec(char op)
   switch(op) {
      case '+': case '-': return 1;
      case '*': case '/': return 2;
      case '^': return 3;
   }
   return 0;  // Error
}
```

Conversión de expresiones: infijo-posfijo

- **Con paréntesis…**

```
A * B / (C + D) * (E – F);
```

- Clave: tratar las expresiones entre paréntesis como expresiones en si misma → se traducen antes de seguir la traducción general
  - Ejemplo: `A * B / [CD+] * [EF-] => AB*CD+/EF-*`
- **En la práctica:**
  - ‘(‘ se introduce en la pila
  - cuando se encuentra ‘)’ se sacan todos los operadores hasta ‘(‘
- ¿Hay que tener en cuenta el balanceado de paréntesis?
  - El algoritmo de conversión lo tendrá en cuenta

Conversión de expresiones: infijo-posfijo

- Con paréntesis…

```
A * B / (C + D) * (E – F) ;
```

Símbolo	Traducción parcial	Pila de OPERADORES

Conversión de expresiones: infijo-posfijo

- Con paréntesis…

```
A * B / (C + D) * (E – F) ;
```

Símbolo	Traducción parcial	Pila de OPERADORES
A	A	
*	A	*
B	AB	*
/	AB*	/
(	AB*	/ (
C	AB*C	/ (
+	AB*C	/ (+
D	AB*CD	/ (+
)	AB*CD+	/
*	AB*CD+/	*
(	AB*CD+/	* (
E	AB*CD+/E	* (
–	AB*CD+/E	* (–
F	AB*CD+/EF	* (–
)	AB*CD+/EF–	*
;	AB*CD+/EF–*	

Conversión de expresiones: posfijo-prefijo

- Se puede seguir la misma estrategia de evaluación para otras conversiones
- **Posfijo → Prefijo**
  1) Algoritmo de evaluación de expresiones posfijo
  2) En vez de "operar", crear expresiones parciales de tipo prefijo (e introducirlas en la pila, como se hacía al evaluar con el resultado)
- Ejemplo: **`AB*CD+/`**

Conversión de expresiones: posfijo-infijo

- **Posfijo → Infijo**
  1) Algoritmo de evaluación de expresiones posfijo
  2) En vez de "operar", crear expresiones parciales de tipo infijo e introducirlas, entre paréntesis, en la pila
- Ejemplo: **`AB/CD+/EF-*`**

Conversión de expresiones: infijo-prefijo

- **Infijo→ prefijo**

  Algoritmo: infijo → sufijo → prefijo

- Ejemplos:

```
A+B-C;                    (A+B)*(C-D);
```

Repaso

- Evaluar la siguiente expresión sufijo

```
2 1 / 4 2 * + 6 5 - 8 2 / + + ;
```

- Convertir la siguiente expresión infijo en sufijo

```
A + B / C - D * F ;
```

- Convertir la siguiente expresión infijo en sufijo

```
(A + B) / C * (D - E) + F ;
```

- Convertir la siguiente expresión sufijo en infijo

```
A B C / + D F * - ;
```

- Convertir la siguiente expresión sufijo en prefijo

```
A B C / + D F * - ;
```

- Evaluar la siguiente expresión prefijo

```
/ / 6 5 - 9 3 ;
```

Contenidos

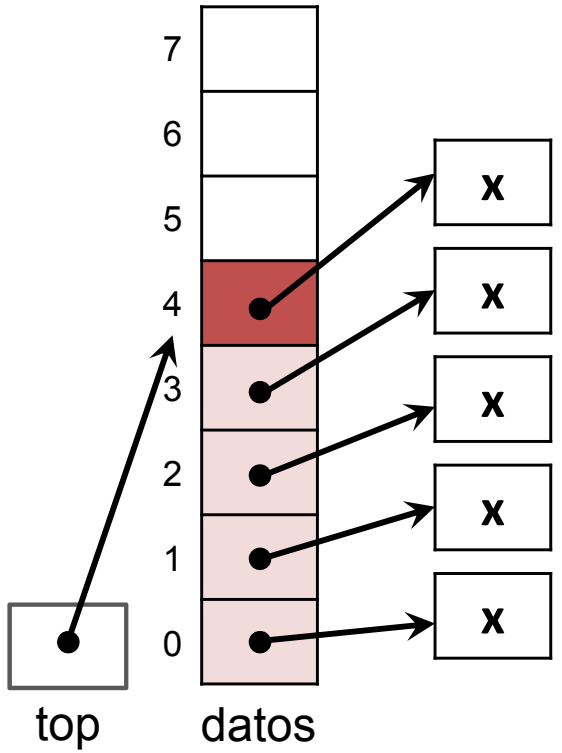
- El TAD Pila
- Estructura de datos y primitivas de Pila
- Implementación en C de Pila
  - Implementación con top de tipo entero
- Ejemplos de aplicación de Pila
  - Balanceo de paréntesis
  - Evaluación de expresiones posfijo
  - Conversión entre notaciones infijo, posfijo y prefijo

- **Anexo**
  - **Implementación con top de tipo puntero**

Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Elemento
  - **EdD de Pila mediante un array**

```
// En pila.h
typedef struct _Pila Pila;

// En pila.c
#define PILA_MAX 8
struct _Pila {
   Elemento *datos[PILA_MAX];
   Elemento **top;
};
```



Implementación en C de Pila

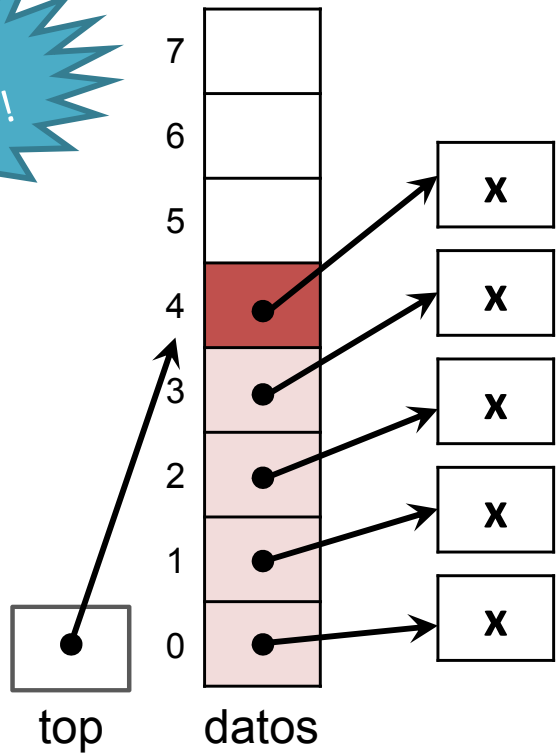
- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```

¡La interfaz NO cambia!



  - **Estructura de datos (en pila.c)**

```
struct _Pila {
   Elemento *datos[PILA_MAX];
   Elemento **top;
};
```

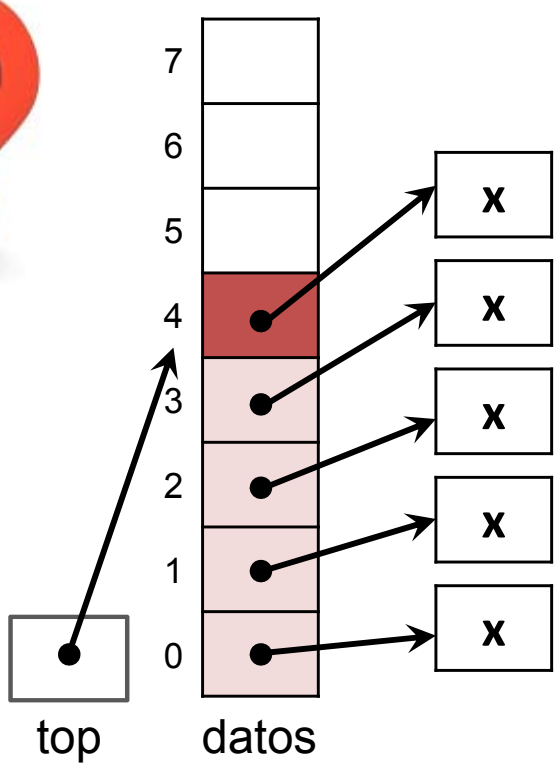
Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```



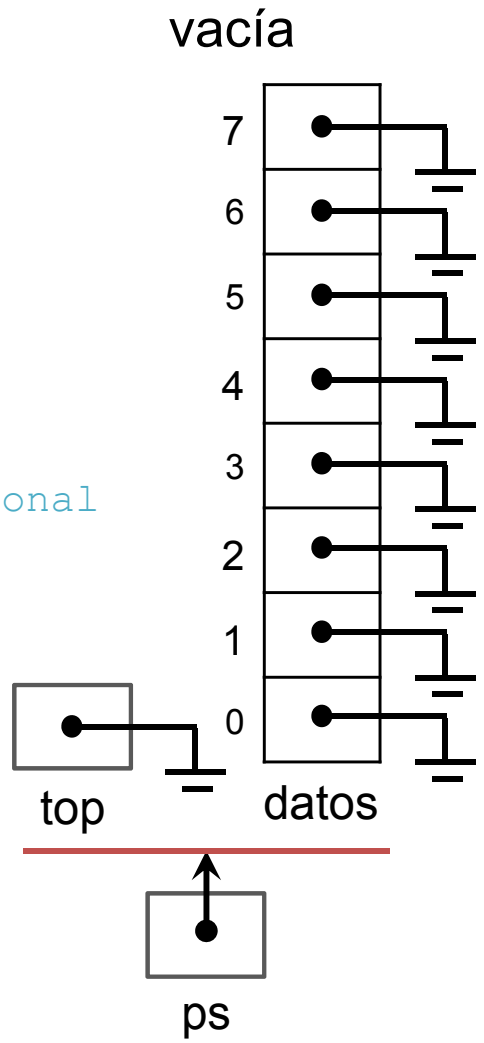
  - **Estructura de datos (en pila.c)**

```
struct _Pila {
   Elemento *datos[PILA_MAX];
   Elemento **top;
};
```

Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
Pila *pila_crear() {
    Pila *ps = NULL;
    int i;

    ps = (Pila *) malloc(sizeof(Pila));
    if (ps == NULL) {
        return NULL;
    }

    for (int i=0; i<PILA_MAX; i++) {  // Bucle opcional
        ps->datos[i] = NULL;
    }

    ps->top = NULL;

    return ps;
}
```

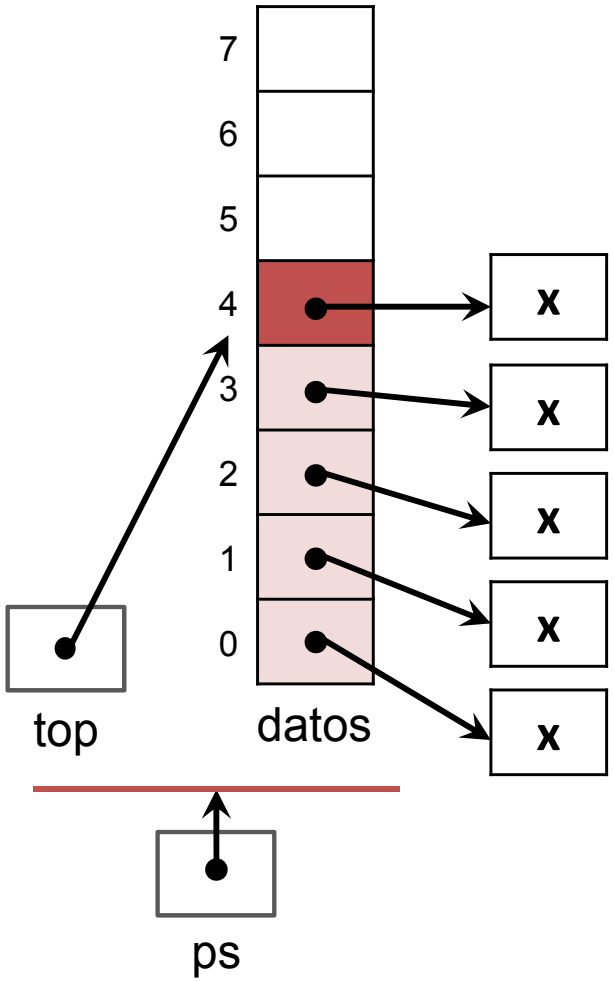


Implementación en C de Pila

- **Implementación con top de tipo puntero**

Existe: `void` **`elemento_liberar`**`(Elemento *pe);`

```
void pila_liberar(Pila *ps) {




}
```

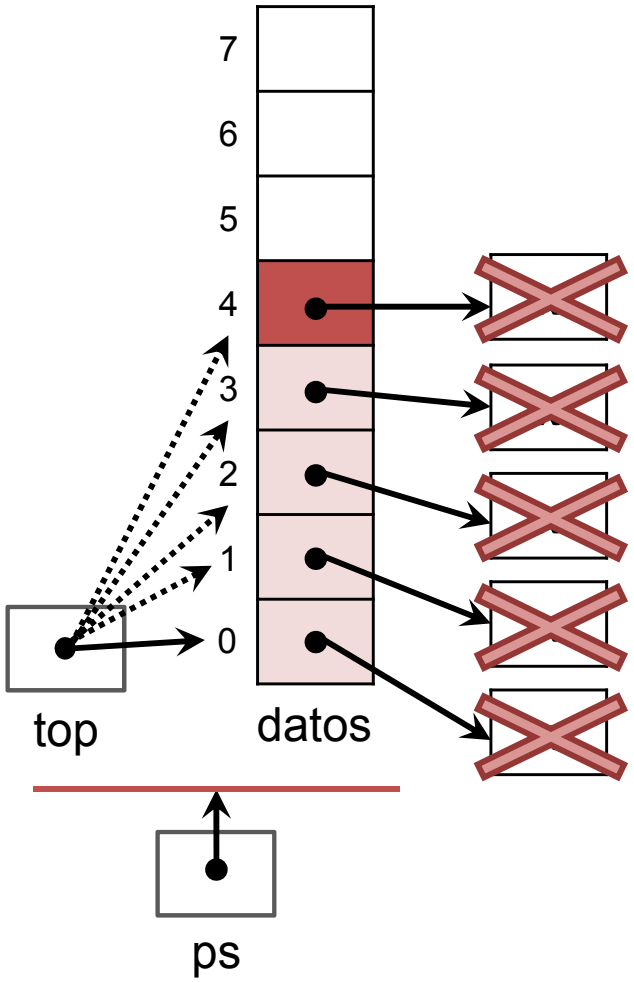


Implementación en C de Pila

- **Implementación con top de tipo puntero**

Existe: `void` **`elemento_liberar`**`(Elemento *pe);`

```
void pila_liberar(Pila *ps) {

    if (ps != NULL) {
        while (ps->top >= ps->datos) {
            elemento_liberar(ps->top);
            ps->top--;
        }


    }
}
```

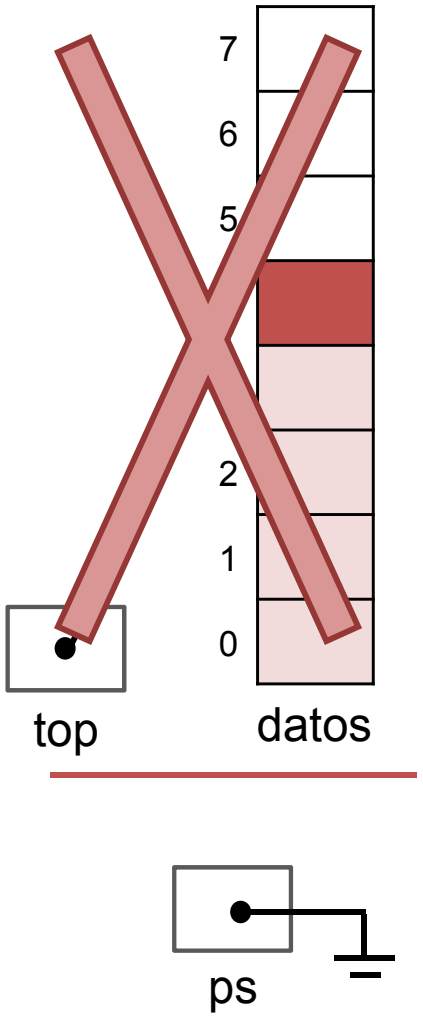


Implementación en C de Pila

- **Implementación con top de tipo puntero**

Existe: `void elemento_liberar(Elemento *pe);`

```
void pila_liberar(Pila *ps) {

    if (ps != NULL) {
        while (ps->top >= ps->datos) {
            elemento_liberar(ps->top);
            ps->top--;
        }
        free(ps);
        // ps = NULL; se hace fuera,
        // tras llamar a pila_liberar
    }
}
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

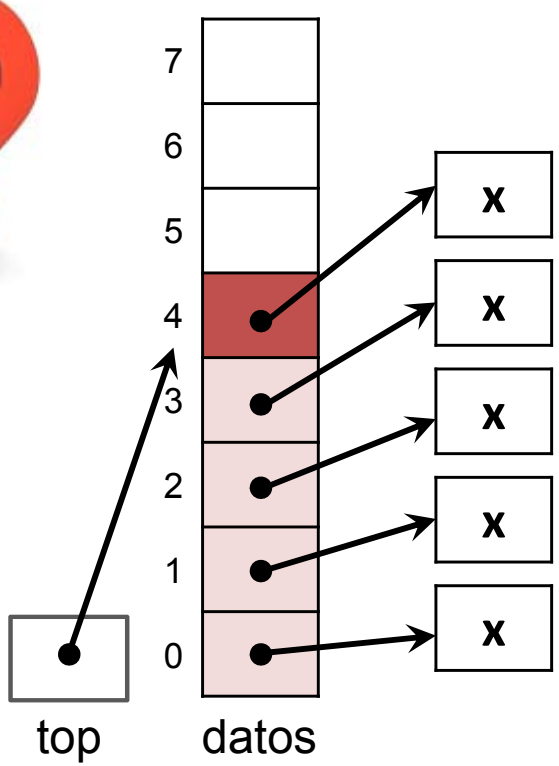
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```

  - **Estructura de datos (en pila.c)**

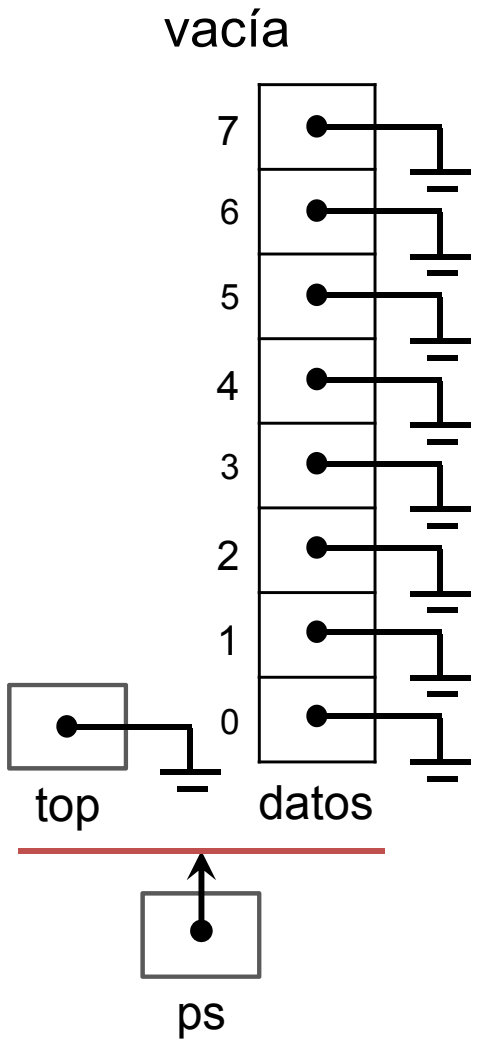
```
struct _Pila {
   Elemento *datos[PILA_MAX];
   Elemento **top;
};
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**

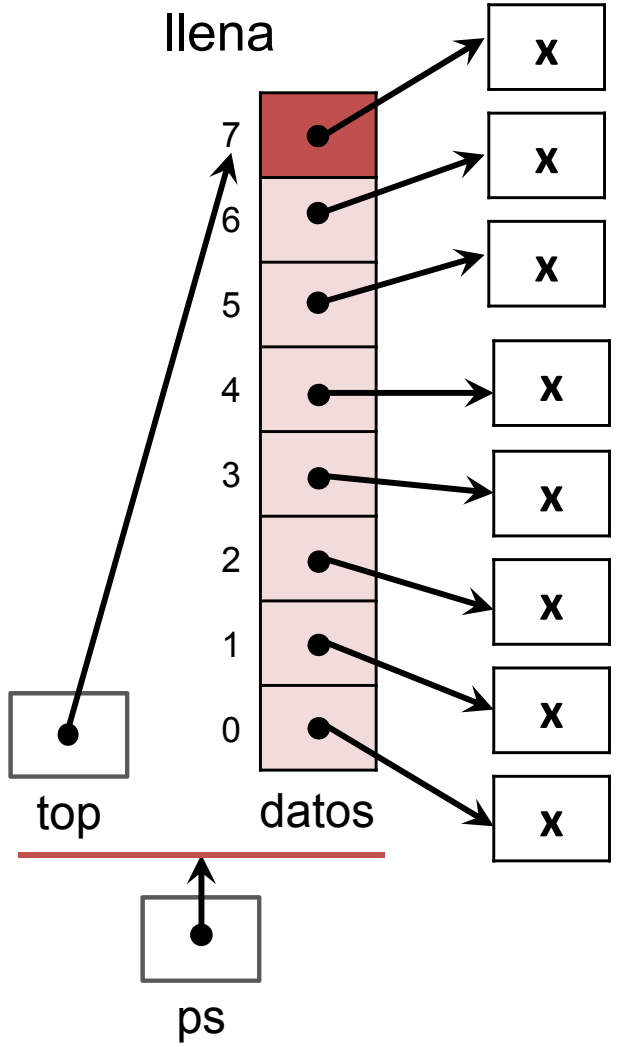
```
boolean pila_vacia(const Pila *ps) {
    if (ps == NULL) {
        return TRUE;
    }

    if (ps->top == NULL) {
        return TRUE;
    }
    return FALSE;
}
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
boolean pila_llena(const Pila *ps) {
    if (ps == NULL) {
        return TRUE;
    }

    // if (ps->top == ps->datos + PILA_MAX - 1)
    if (ps->top == &(ps->datos[PILA_MAX-1])) {
        return TRUE;
    }
    return FALSE;
}
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

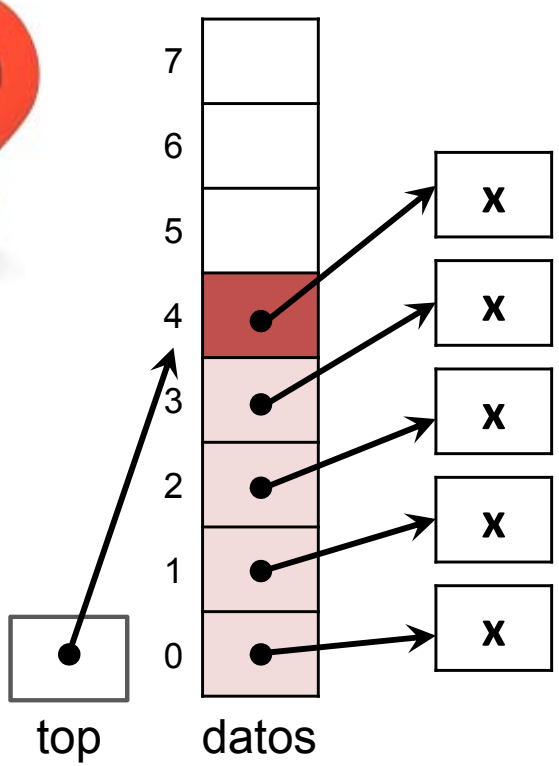
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```

  - **Estructura de datos (en pila.c)**

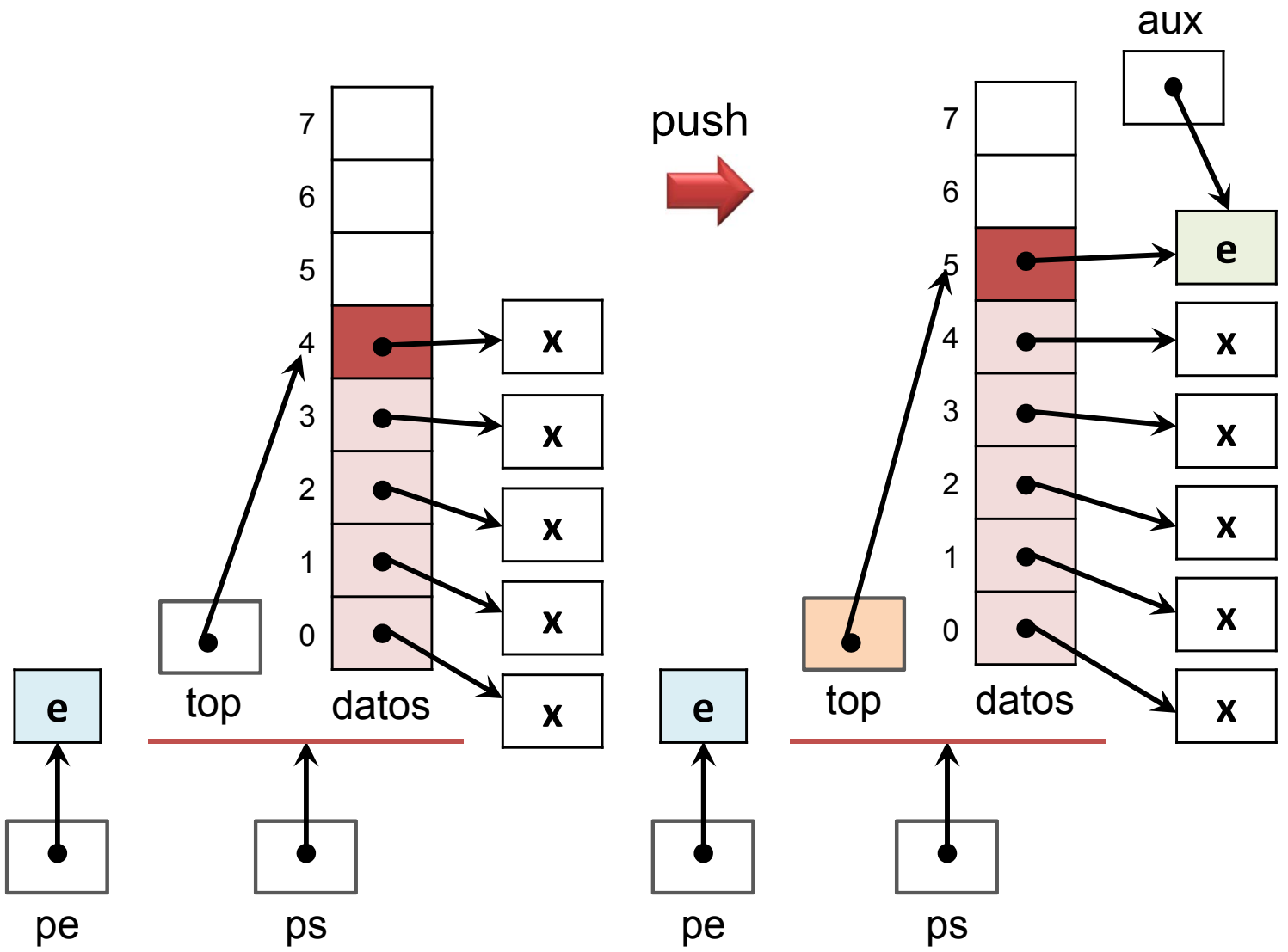
```
struct _Pila {
   Elemento *datos[PILA_MAX];
   Elemento **top;
};
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**

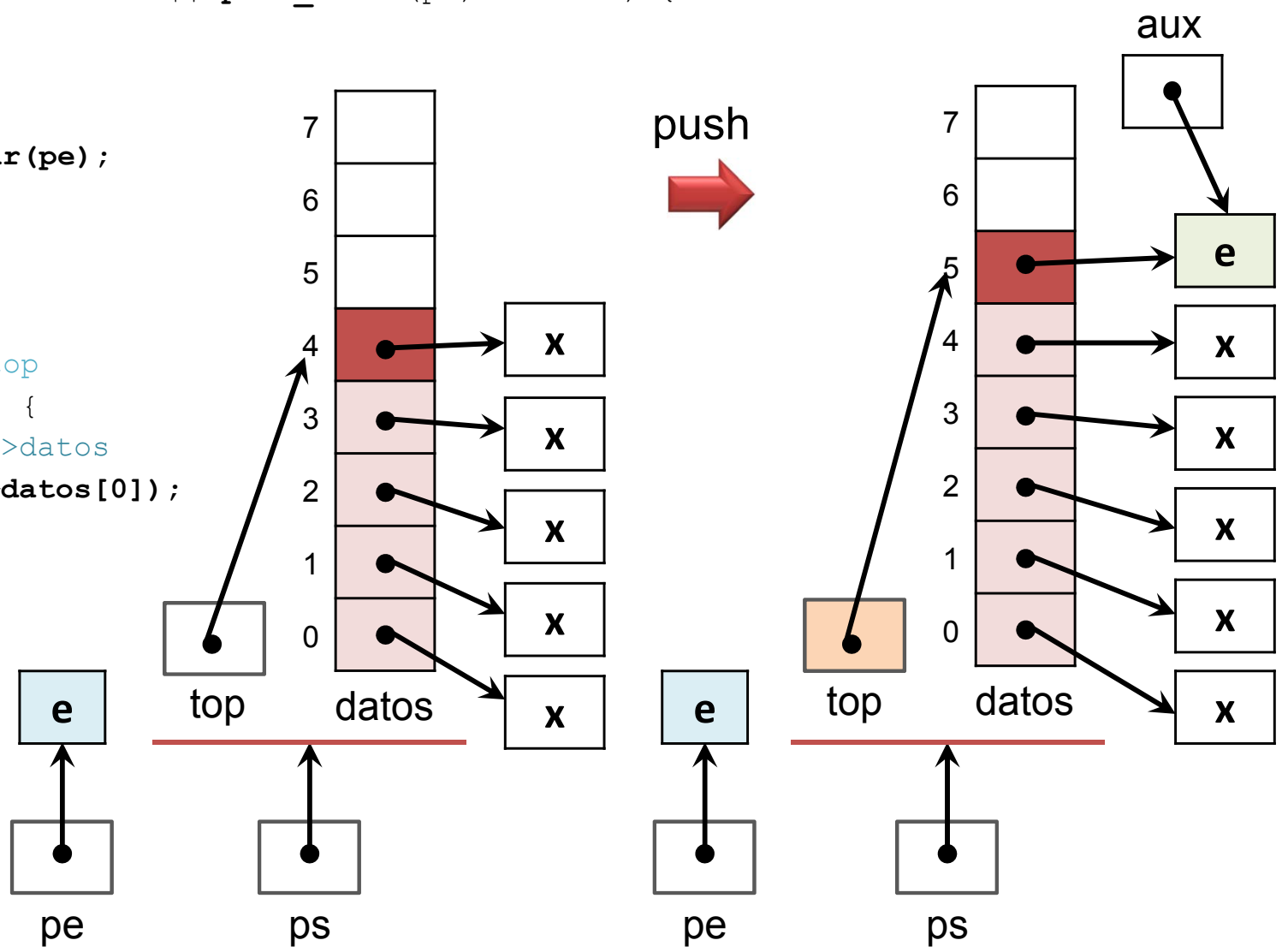
```
status pila_push(Pila *ps, const Elemento *pe){
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
status pila_push(Pila *ps, const Elemento *pe){
    Elemento *aux = NULL;

    if (ps == NULL || pe == NULL || pila_llena(ps) == TRUE) {
        return ERR;
    }

    aux = elemento_copiar(pe);
    if (aux == NULL) {
        return ERR;
    }

    // Actualizamos el top
    if (ps->top == NULL) {
        // ps->top = ps->datos
        ps->top = &(ps->datos[0]);
    }
    else {
        ps->top++;
    }

    // Guardamos el
    // dato en el top
    *(ps->top) = aux;

    return OK;
}
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Elemento que, entre otras, tiene asociadas las primitivas *liberar* y *copiar*:

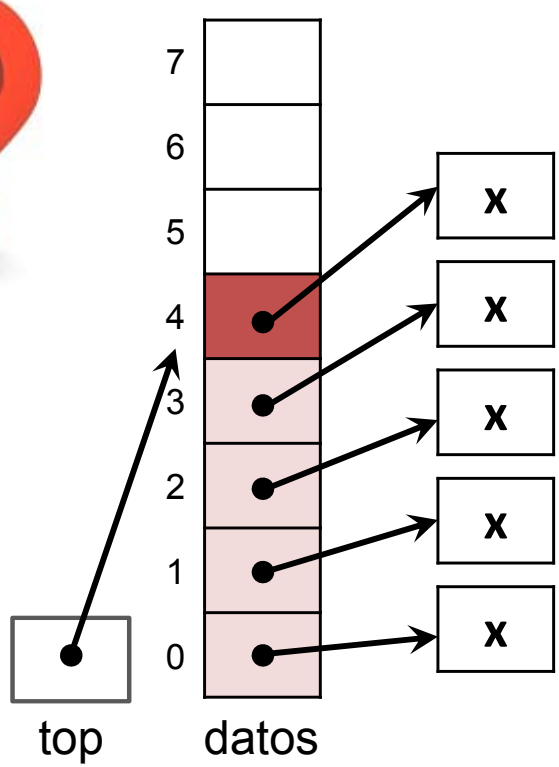
```
void elemento_liberar(Elemento *pe);
Elemento *elemento_copiar(const Elemento *pe);
```

  - **Primitivas (prototipos en pila.h)**

```
Pila *pila_crear();
void pila_liberar(Pila *ps);
boolean pila_vacia(const Pila *ps);
boolean pila_llena(const Pila *ps);
status pila_push(Pila *ps, const Elemento *pe);
Elemento *pila_pop(Pila *ps);
```

  - **Estructura de datos (en pila.c)**

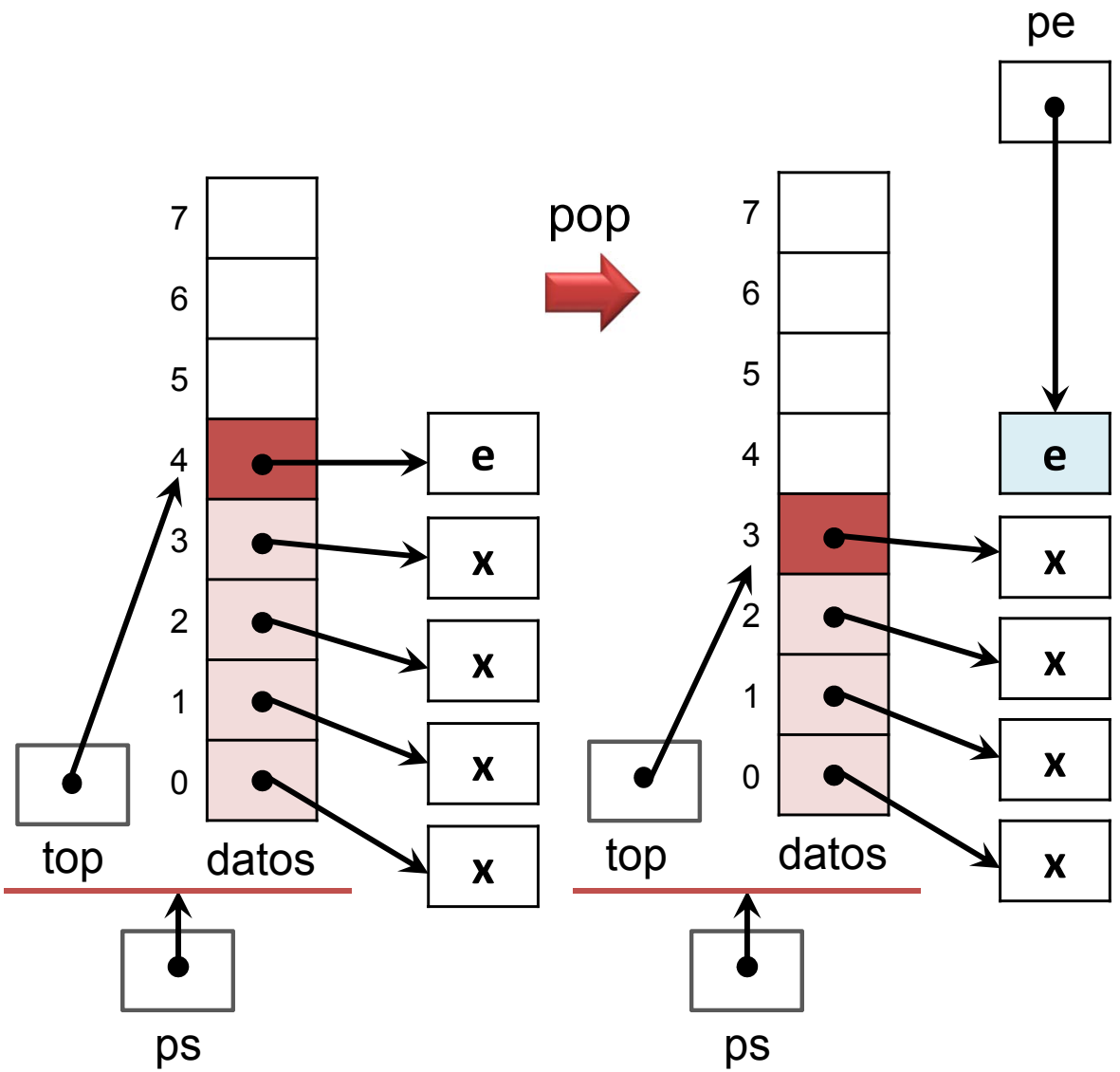
```
struct _Pila {
   Elemento *datos[PILA_MAX];
   Elemento **top;
};
```



Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
Elemento *pila_pop(const Pila *ps){
```




```
}
```

Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
Elemento *pila_pop(const Pila *ps){
    Elemento *pe = NULL;

    if (ps == NULL || pila_vacia(ps) == TRUE) {
        return NULL;
    }

    // Recuperamos el dato del top
    pe = *(ps->top);

    // Ponemos el Elemento* a NULL
    *(ps->top) = NULL;

    // Actualizamos el top
    // if (ps->top != ps->datos)
    if (ps->top != &ps->datos[0]) {
         ps->top--;
    }
    else {
         ps->top = NULL;
    }
    return pe;
}
```