Programación II

Tema 2. Pilas (parte 2)

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
- **Anexos**

ANEXOS

- **Comprobamos que la interfaz sigue siendo válida aunque cambiemos la implementación del TAD:**
  - **ANEXO 1:**
    https://moodle.uam.es/mod/resource/view.php?id=1330528
    Implementación con memoria dinámica para el array (para permitir variaciones de tamaño de la pila)
  - **ANEXO 2:**
    Implementación con tope de tipo puntero (para ver otra posibilidad)

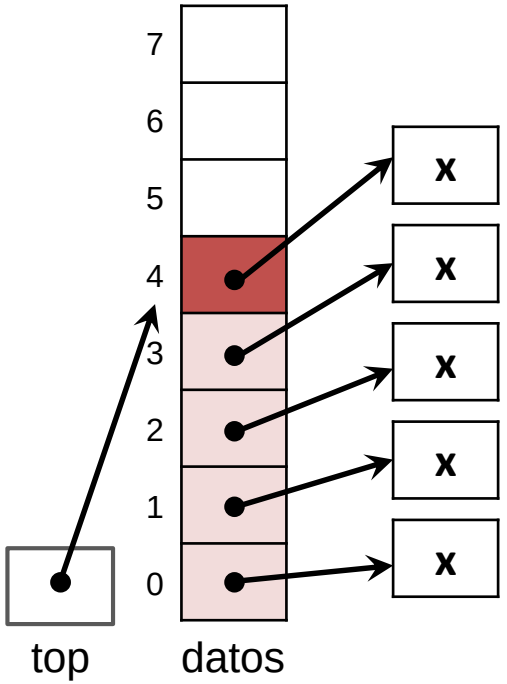
ANEXO 2

Implementación con tope de tipo puntero (para ver otra posibilidad)

Implementación en C de Pila

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

```
// En stack_h
typedef struct _Stack Stack;

// En stack_c
#define STACK_MAX 8
struct _Stack {
   Element *datos[STACK_MAX];
   Element **top;
};
```



Implementación en C de Pila

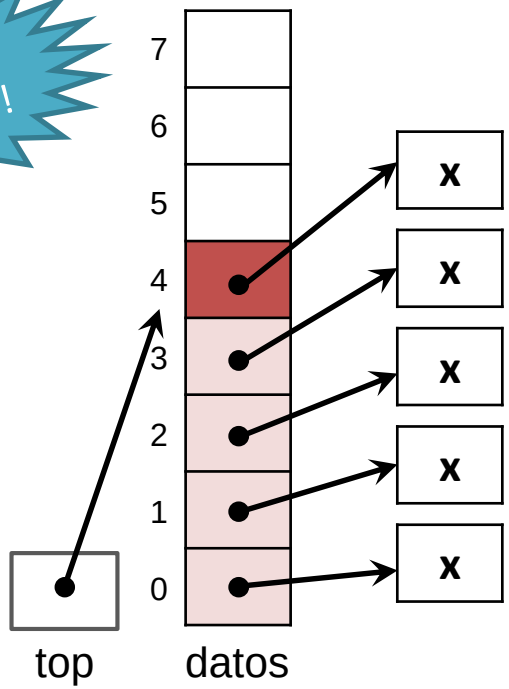
- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Element que, entre otras, incluye en su interfaz las funciones *free* y *copy*:

```
void element_free(Element *pe);
Element *element_copy(const Element *pe);
```

  - **Primitivas (prototipos en stack_h)**

```
Stack *stack_init();
void stack_free(Stack *ps);
Boolean stack_isEmpty(const Stack *ps);
Boolean stack_isFull(const Stack *ps);
Status stack_push(Stack *ps, const Element *pe);
Element *stack_pop(Stack *ps);
Element *stack_top(Stack *ps);
```

¡La interfaz NO cambia!



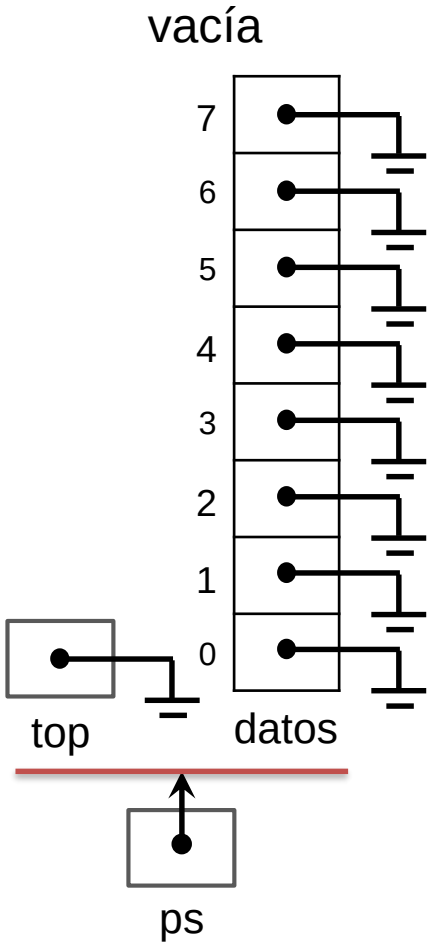
  - **Estructura de datos (en stack_c)**

```
struct _Stack {
   Element  *datos[STACK_MAX];
   Element **top;
};
```

Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
Stack *stack_init() {
   Stack *ps = NULL;
   int i;

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

   for (int i=0; i<STACK_MAX; i++){
      ps->datos[i] = NULL;
    }
   ps->top = NULL;

   return ps;
}
```

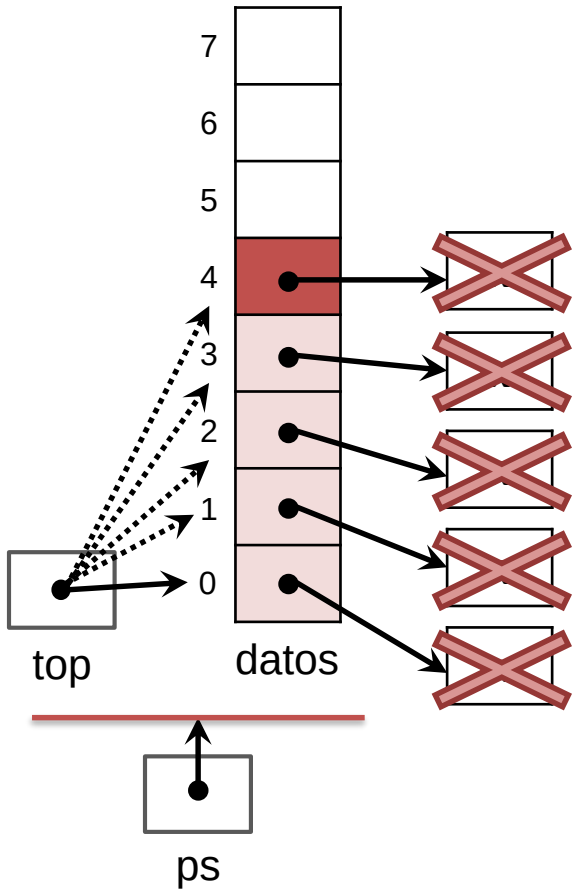


Implementación en C de Pila

- **Implementación con top de tipo puntero**

Existe: `void element_free(Element *pe);`

```
void pila_liberar(Stack *ps) {

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


   }
}
```

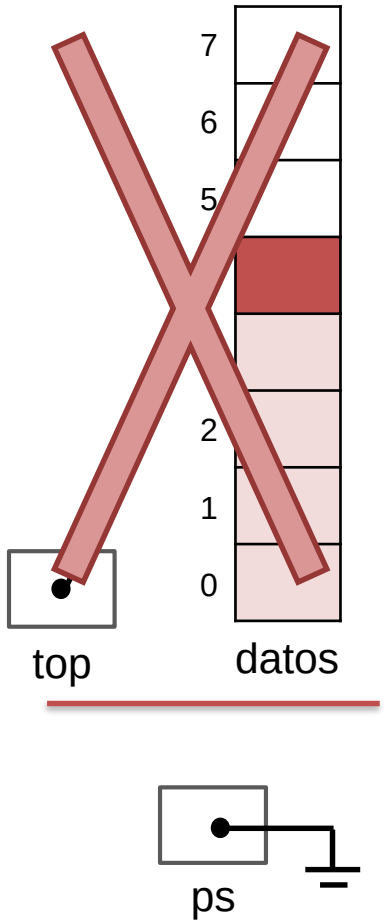


Implementación en C de Pila

- **Implementación con top de tipo puntero**

Existe: `void element_free(Element *pe);`

```
void stack_free(Stack *ps) {

   if (ps == NULL) return;

   while (ps->top >= ps->datos) {
      element_free(ps->top);
      ps->top--;
   }

   free(ps);
   // ps = NULL; se hace fuera,
   // tras llamar a pila_liberar
}
```



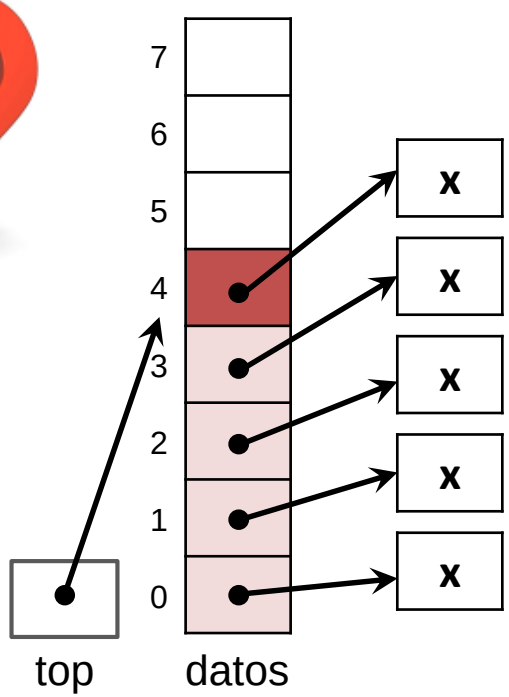
Implementación en C de Pila

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

```
void element_free(Element *pe);
Element *element_copy(const Element *pe);
```

  - **Primitivas (prototipos en stack_h)**

```
Stack *stack_init();
void stack_free(Stack *ps);
Boolean stack_isEmpty(const Stack *ps);
Boolean stack_isFull(const Stack *ps);
Status stack_push(Stack *ps, const Element *pe);
Element *stack_pop(Stack *ps);
```



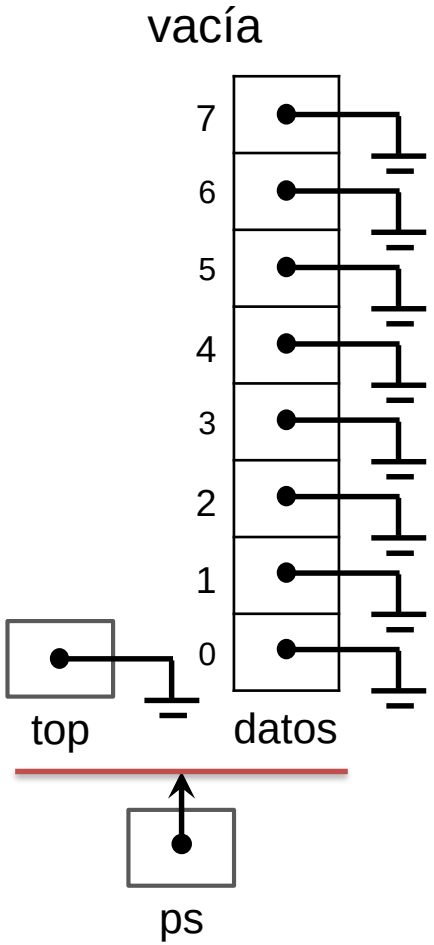
  - **Estructura de datos (en pila.c)**

```
struct _Stack {
   Element  *datos[PILA_MAX];
   Element **top;
};
```

Implementación en C de Pila

- **Implementación con top de tipo puntero**

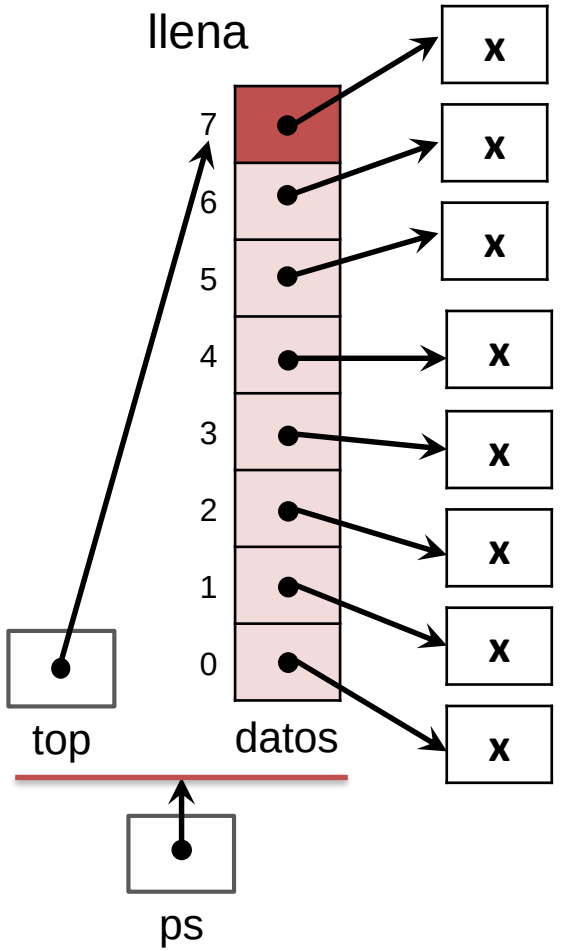
```
Boolean stack_isEmpty(const Stack *ps) {

   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 stack_isFull(const Stack *ps) {

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

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



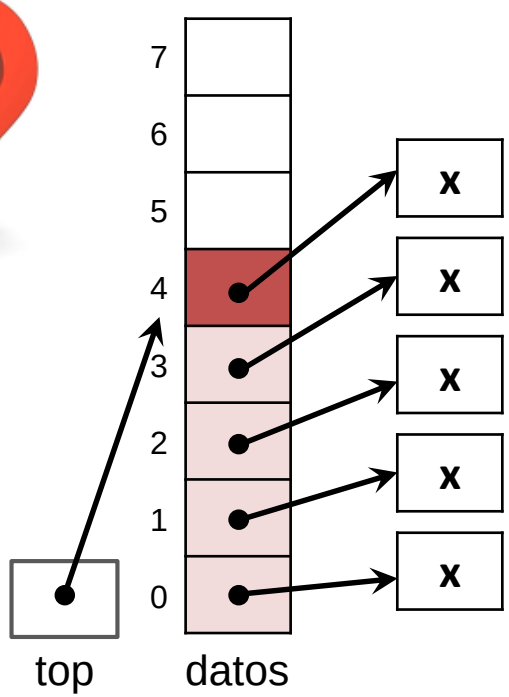
Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Element que, entre otras, incluye las funciones de la interfaz *free* y *copy*:

```
void element_free(Element *pe);
Element *element_copy(const Element *pe);
```

  - **Interfaz (prototipos en stack_h)**

```
Stack *stack_init();
void stack_free(Stack *ps);
Boolean stack_isEmpty(const Stack *ps);
Boolean stack_isEmpty(const Stack *ps);
Status stack_push(Stack *ps, const Element *pe);
Element *stack_pop(Stack *ps);
Element *stack_top(Stack *ps);
```



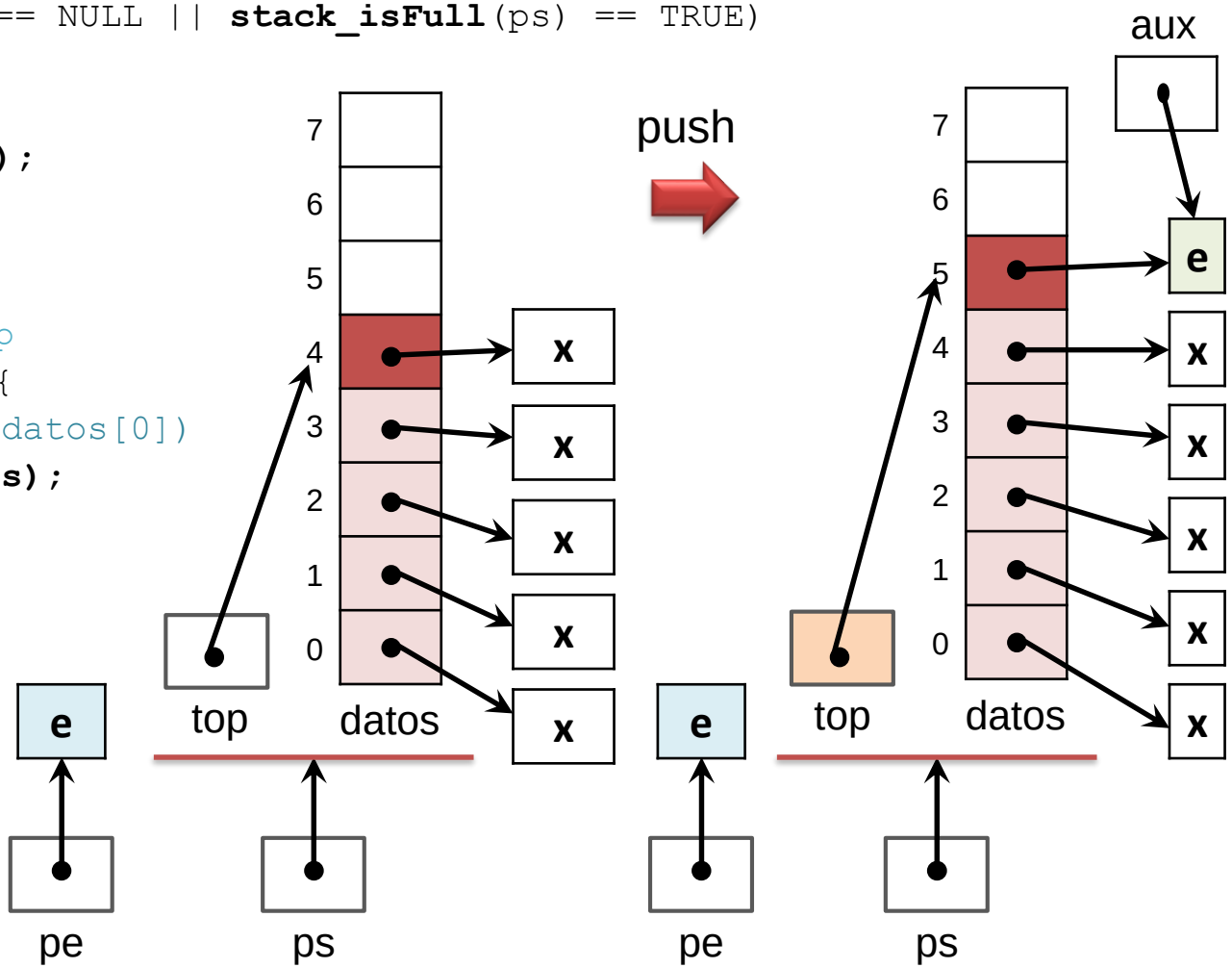
  - **Estructura de datos (en stack_c)**

```
struct _Stack {
   Element  *datos[STACK_MAX];
   Element **top;
};
```

Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
Status stack_push(Stack *ps, const Element *pe){
    Element *aux = NULL;

    if (ps == NULL || pe == NULL || stack_isFull(ps) == TRUE)
        return ERR;

    aux = element_copy(pe);
    if (aux == NULL)
            return ERR;

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

// Asignamos a top
// la dirección guardada
// en el dato
    *(ps->top) = aux;

    return OK;
```



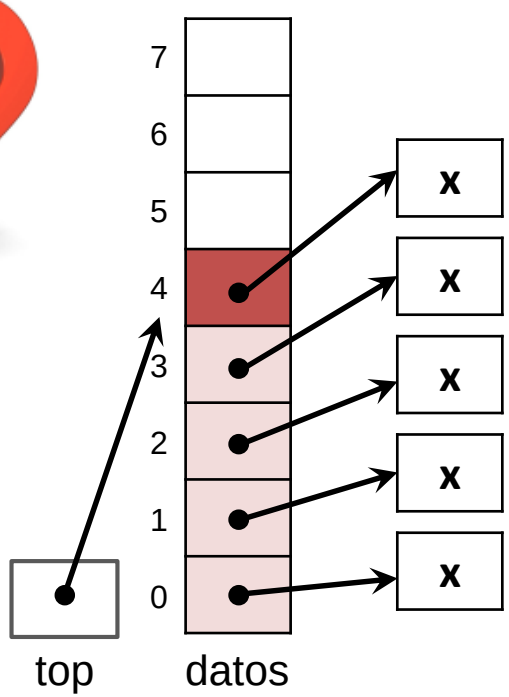
Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Element 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;
};
```

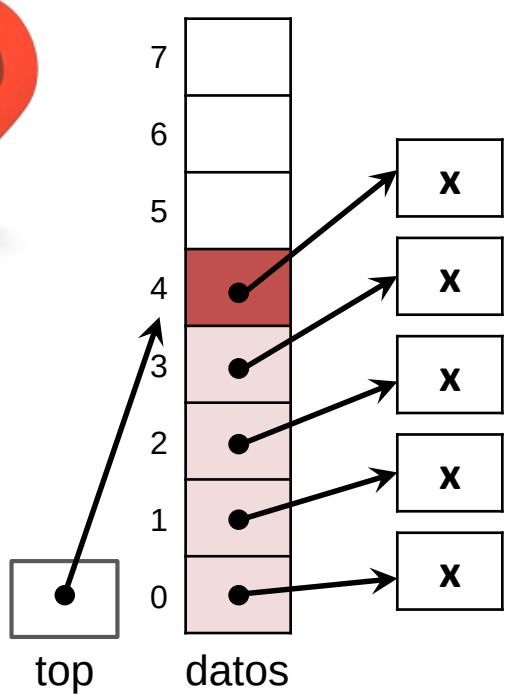
Implementación en C de Pila

- **Implementación con top de tipo puntero**
  - Asumimos la existencia del TAD Element que, entre otras, incluye las funciones de la interfaz *free* y *copy*:

```
void element_free(Element *pe);
Element *element_copy(const Element *pe);
```

  - **Interfaz (prototipos en stack_h)**

```
Stack *stack_init();
void stack_free(Stack *ps);
Boolean stack_isEmpty(const Stack *ps);
Boolean stack_isEmpty(const Stack *ps);
Status stack_push(Stack *ps, const Element *pe);
Element *stack_pop(Stack *ps);
Element *stack_top(Stack *ps);
```



  - **Estructura de datos (en stack_c)**

```
struct _Stack {
   Element  *datos[STACK_MAX];
   Element **top;
};
```

Implementación en C de Pila

- **Implementación con top de tipo puntero**

```
Element *stack_pop(const Stack *ps){
    Element *pe = NULL;

    if (ps == NULL || stack_isEmpty(ps) == TRUE)
        return NULL;

    // Asignamos a pe el dato referenciado
    // por el top
    pe = *(ps->top);

    // Asignamos a Element* NULL
    *(ps->top) = NULL;

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