

Universidad Autónoma de Madrid
Escuela Politécnica Superior
Grado en Ingeniería Informática
Programación II
Hoja de ejercicios - Pilas

Sobre las siguientes sentencias determinar la corrección del equilibrado de símbolos (), [] y {} mediante la evolución de una pila.

Ejercicio 1

$2 + \{ 3 * (4 + 5) - (7 + 9) / (5 * 4) \} - 8 * \{ 7 / [6 + 2 * (1 + 3)] \};$

{	()	()	()	}	{	[()]	}	;
										(
	((([[[
{	{	{	{	{	{	{		{	{	{	{	{		

Al llegar al fin de expresión ;, como la pila está vacía la expresión es correcta.

Ejercicio 2

$5 - \{ [(3 - 5) * 2 - 8 / (4 + 7 * (3 - 2))] + 6 * (1 - 9[]) \};$

{	[()	(())	((]	}
					((ES	
		(((((((
	[[[[[[[[[[
{	{	{	{	{	{	{	{	{	{	{	

ES = Error Sintáctico

Ejercicio 3

$(([a + ((b + c) - (d * e) + \{h - g\}] + \{[b - a] * (f - i)\}));$

({	[(()	()	{	}]	{	[]	()	}	})
			((({										
			(((((((ES								
		[[[[[[[[[
	{	{	{	{	{	{	{	{	{	{								
(((((((((((

ES = Error Sintáctico

Ejercicio 4

$(a + [b * \{c - \{d + e / (f)\} - g\}] + (v - \{w * (x + a)\}));$

([{	{	()	}	}]	({	()	}))	;
				(
		{	{	{	{					(
		{	{	{	{	{				{	{	{				
	[[[[[[[(((((
(((((((((((((((

Al llegar al fin de expresión ;, como la pila está vacía la expresión es correcta.

Evaluar las siguientes expresiones posfijo mediante una pila.

Ejercicio 5

1 2 * 4 2 / 3 4 + 5 * - +;

1	2	*	4	2	/	3	4	+	5	*	-	+	;
							4		5				
				2		3	3	7	7	35			
	2		4	4	2	2	2	2	2	2	-33		
1	1	2	2	2	2	2	2	2	2	2	2	-31	

Al llegar el fin de expresión ;, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

Ejercicio 6

5 4 3 2 1 * + 1 2 3 * / 2 * 1 / 2 + - + *;

5	4	3	2	1	*	+	1	2	3	*	/	2	*	1	/	2	+	-	+	*	;
									3												
				1				2	2	6		2		1		2					
			2	2	2		1	1	1	1	1/6	1/6	1/3	1/3	1/3	1/3	7/3				
		3	3	3	3	5	5	5	5	5	5	5	5	5	5	5	5	8/3			
	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4/3		
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	20/3	

Al llegar el fin de expresión ;, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

Ejercicio 7

A B C + D E F G - / * - +;

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

Al llegar el fin de expresión ;, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

Ejercicio 8

A B / C A * + E F - G B / + + ;

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

Al llegar el fin de expresión ;, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

Convertir a posfijo las siguientes expresiones infijo según las reglas de precedencia y asociatividad de operadores discutida en clase.

Ejercicio 9

$$3 * 6 + 7 * 2 - 1 + 6 * 4 / 3;$$

Símbolo	Traducción parcial	Pila de operadores
3	3	
*	3	*
6	3 6	*
+	3 6 *	+
7	3 6 * 7	+
*	3 6 * 7	+ *
2	3 6 * 7 2	+ *
-	3 6 * 7 2 * +	-
1	3 6 * 7 2 * + 1	-
+	3 6 * 7 2 * + 1 -	+
6	3 6 * 7 2 * + 1 - 6	+
*	3 6 * 7 2 * + 1 - 6	+ *
4	3 6 * 7 2 * + 1 - 6 4	+ *
/	3 6 * 7 2 * + 1 - 6 4 *	+ /
3	3 6 * 7 2 * + 1 - 6 4 * 3	+ /
;	3 6 * 7 2 * + 1 - 6 4 * 3 / +	

Ejercicio 10

$$2 * (3 + 5) - (4 + 1) / 7 + 8;$$

Símbolo	Traducción parcial	Pila de operadores
2	2	
*	2	*
(2	* (
3	2 3	* (
+	2 3	* (+
5	2 3 5	* (+
)	2 3 5 +	*
-	2 3 5 + *	-
(2 3 5 + *	- (
4	2 3 5 + * 4	- (
+	2 3 5 + * 4	- (+
1	2 3 5 + * 4 1	- (+
)	2 3 5 + * 4 1 +	-
/	2 3 5 + * 4 1 +	- /
7	2 3 5 + * 4 1 + 7	- /
+	2 3 5 + * 4 1 + 7 / -	+
8	2 3 5 + * 4 1 + 7 / - 8	+
;	2 3 5 + * 4 1 + 7 / - 8 +	

Ejercicio 11

$A * B * C / D / E - F * G + H - I ;$

Símbolo	Traducción parcial	Pila de operadores
A	A	
*	A	*
B	A B	*
*	A B *	*
C	A B * C	*
/	A B * C *	/
D	A B * C * D	/
/	A B * C * D /	/
E	A B * C * D / E	/
-	A B * C * D / E /	-
F	A B * C * D / E / F	-
*	A B * C * D / E / F	- *
G	A B * C * D / E / F G	- *
+	A B * C * D / E / F G * -	+
H	A B * C * D / E / F G * - H	+
-	A B * C * D / E / F G * - H +	-
I	A B * C * D / E / F G * - H + I	-
;	A B * C * D / E / F G * - H + I -	

Ejercicio 12

$(A + B) * C / D - (F + G) - H * (I + J) ;$

Símbolo	Traducción parcial	Pila de operadores
((
A	A	(
+	A	(+
B	A B	(+
)	A B +	
*	A B +	*
C	A B + C	*
/	A B + C *	/
D	A B + C * D	/
-	A B + C * D /	-
(A B + C * D /	- (
F	A B + C * D / F	- (
+	A B + C * D / F	- (+
G	A B + C * D / F G	- (+
)	A B + C * D / F G +	-
-	A B + C * D / F G + -	-
H	A B + C * D / F G + - H	-
*	A B + C * D / F G + - H	- *
(A B + C * D / F G + - H	- * (
I	A B + C * D / F G + - H I	- * (
+	A B + C * D / F G + - H I	- * (+
J	A B + C * D / F G + - H I J	- * (+
)	A B + C * D / F G + - H I J +	- *
;	A B + C * D / F G + - H I J + * -	

Convertir a prefijo las siguientes expresiones infijo mediante la evolución de una pila.

Ejercicio 13

$A + B - C;$

Símbolo	Traducción parcial	Pila de operadores
A	A	
+	A	+
B	A B	+
-	A B +	-
C	A B + C	-
;	A B + C -	

A	B	+	C	-	;
	B		C		
A	A	+ A B	+ A B	- + A B C	

Al llegar el fin de expresión **;**, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

Ejercicio 14

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

Símbolo	Traducción parcial	Pila de operadores
((
A	A	(
+	A	(+
B	A B	(+
)	A B +	
*	A B +	*
(A B +	* (
C	A B + C	* (
-	A B + C	* (-
D	A B + C D	* (-
)	A B + C D -	*
;	A B + C D - *	

A	B	+	C	D	-	*	;
				D			
	B		C	C	- C D		
A	A	+ A B	+ A B	+ A B	+ A B	* + A B - C D	

Al llegar el fin de expresión **;**, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

Ejercicio 15

$A - B / (C * D ^ E);$

Símbolo	Traducción parcial	Pila de operadores
A	A	
-	A	-
B	A B	-
/	A B	- /
(A B	- / (
C	A B C	- / (
*	A B C	- / (*

D	A B C D	- / (*
^	A B C D	- / (* ^
E	A B C D E	- / (* ^
)	A B C D E ^ *	- /
;	A B C D E ^ * / -	

A	B	C	D	E	^	*	/	-	;
				E					
			D	D	^DE				
		C	C	C	C	*C^DE			
	B	B	B	B	B	B	/B*C^DE		
A	A	A	A	A	A	A	A	- A/B*C^DE	

Al llegar el fin de expresión ;, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

Ejercicio 16

$A \wedge B * C - D + E / (G + H); \quad +-*^{\wedge}ABCD//EF+GH$

Símbolo	Traducción parcial	Pila de operadores
A	A	
^	A	^
B	A B	^
*	A B ^	*
C	A B ^ C	*
-	A B ^ C *	-
D	A B ^ C * D	-
+	A B ^ C * D -	+
E	A B ^ C * D - E	+
/	A B ^ C * D - E	+ /
F	A B ^ C * D - E F	+ /
/	A B ^ C * D - E F /	+ /
(A B ^ C * D - E F /	+ / (
G	A B ^ C * D - E F / G	+ / (
+	A B ^ C * D - E F / G	+ / (+
H	A B ^ C * D - E F / G H	+ / (+
)	A B ^ C * D - E F / G H +	+ /
;	A B ^ C * D - E F / G H + / +	

A	B	^	C	*	D	-	E	F	/
								F	
	B		C		D		E	E	/EF
A	A	^AB	^AB	*^ABC	*^ABC	-*^AB CD	-*^ABC D	-*^ABCD	-*^ABCD

G	H	+	/	+	;
	H				
G	G	+GH			
/EF	/EF	/EF	//EF+GH		
-*^ABCD	-*^ABCD	-*^ABCD	-*^ABCD	+-*^ABCD//EF+GH	

Al llegar el fin de expresión ;, hacemos 1 pop. Como la pila queda vacía, lo extraído en el pop es la expresión final.

[illegible]

Una implementación en C del TAD Pila tiene como único interfaz las primitivas:

- `Pila *pila_crear()`, que crea (reservando memoria), inicializa y devuelve una pila.
- `void pila_liberar(Pila *ps)`, que libera la memoria de una pila y sus elementos.
- `boolean pila_vacia(Pila *ps)`, que comprueba si una pila está vacía.
- `boolean pila_llena(Pila *ps)`, que comprueba si una pila está llena.
- `status pila_push(Pila *ps, Elemento *pe)`, que inserta un elemento en la pila.
- `Elemento *pila_pop(Pila *ps)`, que extrae y devuelve un elemento en la pila.

Asumir que un `push` precedido de un `pop` sobre una pila no causa error.

Ejercicio 21

Dar el código C de una función `Elemento *pila_getPrimero(Pila *ps)` que devuelve el primer elemento insertado en una pila, sin que la pila quede modificada.

Asumir que existe una función `Elemento *elemento_copiar(Elemento *pe)` que devuelve una copia del elemento apuntado por `pe`. La función deber hacer el control de errores pertinente.

```
Elemento *pila_getPrimero(Pila *ps) {
    Pila *pilaAux = NULL;
    Elemento *pe = NULL, *ret = NULL;

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

    // Creamos pila auxiliar
    pilaAux = pila_crear();
    if (pilaAux==NULL) {
        return NULL;
    }

    // Copiamos los elementos de ps en orden inverso en pilaAux
    while (pila_vacia(ps)==FALSE) {
        pe = pila_pop(ps);
        pila_push(pilaAux, pe);
        elemento_liberar(pe);
    }

    // El último elemento extraído de ps (e insertado en pilaAux)
    // es el primer elemento que se introdujo en ps
    ret = pila_pop(pilaAux);

    // Restauramos ps vaciando pilaAux
    pila_push(ps, ret);

    while (pila_vacia(pilaAux)==FALSE) {
        pe = pila_pop(pilaAux);
        pila_push(ps, pe);
        elemento_liberar(pe);
    }
    pila_liberar(pilaAux);

    return ret;
}
```

Ejercicio 22

Dar el código C de una función de prototipo `int pila_tamano(Pila *ps)` que reciba como parámetro una pila y devuelva su número de elementos.

La función ni puede acceder directamente a la estructura de la pila, ni debe dejar alterada la misma tras su ejecución. Además debe hacer el control de errores pertinente.

Da primero el pseudocódigo sin control de errores, después el pseudocódigo con control de errores y finalmente el código C

Pseudocódigo:

*/*Sin Control de Errores */*

```
int pila_tamano (Pila p)
    paux = pila_crear()
    tam=0

    mientras pila_vacia (p) = FALSE:
        elem=pila_pop (p)
        pila_push (paux, elem)
        incr tam

    mientras pilaVacia (paux) = FALSE:
        elem=pila_pop(paux)
        pila_push(p, elem)

    pila_liberar (paux)
    devolver tam
```

*C) /*Con Control de Errores */*

```
int pila_tamano (Pila p)

    si (paux=pila_crear()) = ERROR:
        devolver -1

    tam=0

    mientras pila_vacia (p) = FALSE:
        elem=pila_pop (p)
        pila_push (paux, elem)
        incr tam

    mientras pilaVacia (paux) = FALSE:
        elem=pila_pop(paux)
        pila_push(p, elem)

    pila_liberar (paux)
    devolver tam
```

Código C:

```
int pila_tamano(Pila *ps) {
    Pila *pilaAux = NULL;
    Elemento *pe = NULL;
    int tam = 0;

    if (ps==NULL) {
        return -1;
    }

    // Creamos pila auxiliar
    pilaAux = pila_crear();
    if (pilaAux==NULL) {
        return NULL;
    }

    // Copiamos los elementos de ps en orden inverso en pilaAux
    // y vamos incrementando el contador de tamaño
    while (pila_vacia(ps)==FALSE) {
        pe = pila_pop(ps); // pe apunta al elemento que estaba en el tope de la pila
        pila_push(pilaAux, pe); //Hace copia de elemento y lo introduce en pilaAux
        elemento_liberar(pe); //Libera elemento extraído
        tam++;
    }

    // Restauramos ps vaciando pilaAux
    while (pila_vacia(pilaAux)==FALSE) {
        pe = pila_pop(pilaAux);
        pila_push(ps, pe);
        elemento_liberar(pe);
    }

    pila_liberar(pilaAux);

    return tam;
}
```

Ejercicio 23

(TAD STRING = cadena de caracteres) Escribir el psc de una función que reciba una cadena de caracteres y la devuelva (en el argumento inver) en forma invertida

STATUS invCadena (STRING org, STRING inver)

Se dispone de las primitivas del TAD cadena de caracteres:

STATUS stIni (STRING s), inicializa una cadena de caracteres,

BOOLEAN stEmpty (STRING s), comprueba si una cadena está vacía

STATUS stInsert (STRING s, int i, char c), inserta el carácter c en la posición i de la cadena s, desplaza los caracteres i+1 y siguientes una posición a la derecha.

char stExtract (STRING s, int i), extrae el carácter que ocupa la posición i de la cadena s, desplaza los caracteres i+1 y siguientes una posición a la izquierda.

void stDestroy (STRING s), libera los recursos asociados a la cadena s

- Dar una primera versión sin control de errores (CdE).
- Dar una segunda versión con control de errores. En este caso la cadena org no debe ser modificada cuando la función devuelva ERROR

<pre>/* Sin Control de Errores */ STATUS invCadena (STRING org, STRING inver) stIni (inver) while stEmpty (org) == FALSE: stExtract (org, 0, ch) stInsert (inver, 0, ch) return OK</pre>	<pre>/* Con Control de Errores */ STATUS invCadena (STRING org, STRING inver) if stIni (inver) == ERROR: return ERROR while stEmpty (org) == FALSE: stExtract (org, 0, ch) /* si error al insertar, poner el extraído en su sitio y volver a pasar todos los pasados a la cadena invertida a la original */ if stInsert(inver, i, ch) == ERROR: stInsert (org, 0, ch) while stEmpty (inver) == FALSE: stExtract (inver, 0, ch) stInsert (org, 0, ch) return ERROR return OK</pre>
--	---

Ejercicio 24

Dar el pseudocódigo sin control de errores y el código C de una función char *aBinario(int n) que, mediante el algoritmo de división repetida por 2 y acumulación de restos, y haciendo uso de una pila, devuelva la representación binaria (como cadena de caracteres) de un numero entero positivo pasado como argumento de entrada.

Asumir que existe la función convierteAcaracter, que recibe un puntero a elemento y devuelve el carácter almacenado en el elemento.

La función deber hacer el control de errores pertinente.

PSEUDOCÓDIGO:

```
/*Vale para binario, octal, etc. cambiando la base. Sin Control de Errores */
CC cDecimalToBase (entero num, entero base)
    paux = pila_crear ()
    /*Inserta los restos en la pila */
    mientras num >= base:
        resto = num modulo base
        num = num / base
        pila_push (paux, resto)
    /*Inserta el ultimo cociente: bit mas significativo */
    pila_push(paux, num)
    cadena = cc_crear ()
    mientras pila_vacia (paux) = FALSE:
        pila_pop (paux, num)
        cc_concatenar (cadena, num)
    pila_liberar (paux)
    devolver cadena
```

Código C:

```
char *aBinario(int n) {
    Pila *pilaAux = NULL;
    Elemento *pe = NULL, *uno = NULL;
    char *cadena = NULL;
    int resto, cociente, i;

    // Control error base <0
    if (n<=0) {
        return NULL;
    }

    // Creamos pila auxiliar
    pilaAux = pila_crear();
    if (pilaAux==NULL) {
        return NULL;
    }

    // Guardamos los restos de ir dividiendo entre 2 en la pila

    i = 1;
    while (num >= base) {
        resto = num % base;
        num = num / base;
        pe = elemento_crear(resto);
        pila_push (pilaAux, pe);
        elemento_liberar(pe);
        i++; /* Para saber cuántos se guardan (determina tamaño de cadena a devolver)*/
    }

    // Guardamos el cociente final en la pila
    pe = elemento_crear(cociente);
    pila_push(pilaAux, pe);
    elemento_liberar(pe);

    // Reservamos memoria para la cadena de caracteres
    cadena = malloc((i+1)*sizeof(char)); /* +1 para el \0 */

    if (cadena==NULL) {
        pila_vaciar(pilaAux);
        return NULL;
    }

    // Guardamos el cociente y los restos en orden inverso a como se introdujeron en la pila
    i = 0;

    while (pila_vacia(pilaAux)==FALSE) {
        pe = pila_pop(pilaAux);
        cadena[i] = convierteAcaracter (pe);
        elemento_liberar(pe);
        i++;
    }
    cadena[i] = '\0' // fin de cadena

    pila_liberar(pilaAux);

    return cadena;
}
```

Ejercicio 25

Dar el pseudocódigo y el código C de una función de prototipo `boolean esCapicua(char *cadena)` que reciba como entrada una cadena de caracteres y que, haciendo uso de una pila, determine si esa cadena es de la forma:

$x\ C\ y$

en donde x es una cadena que consta de las letras 'A' y 'B', y es la inversa de x (es decir, si x es "ABABBA", y debe ser igual a "ABBABA") y C se refiere al carácter 'C'. En cada punto sólo puede leerse el siguiente carácter de la cadena. No hace falta hacer el control de errores. Asumir que Elemento es un TAD para almacenar un carácter con funciones primitivas acordes y que existe la función `convierteAcaracter`, que recibe un puntero a Elemento y lo convierte en un carácter.

Pseudocódigo:

```
/* Sin Control de Errores */
boolean esCapicua (CC exp)
    p = pila_crear()

    /* Lee la parte X de la expresion */
    car = readItem (exp)
    mientras (car) ≠ C:                // mientras (car = readItem (exp)) ≠ C:
        pila_push(p, car)              //      pila_push(p, car)
        car = readItem (exp)

    /* Lee la parte Y de la expresion y compara con X */
    mientras (car2=readItem (exp)) ≠ FIN:    /* en car2 tengo el que leo de la cadena ... */
        car = pila_pop(p)                  /* ... y en car el que saco de la pila */

        si car = ERROR: // no se ha podido extraer de la pila → X tiene menos caracteres que Y
            pila_liberar (p)
            dev FALSE

        si car ≠ car2: // X e Y tienen caracteres diferentes → no cumple la condición buscada
            pila_liberar (p)
            dev FALSE

    si pila_vacia (p) = FALSE: // X tiene mas caracteres que Y
        dev FALSE

    pila_liberar (p)
    dev TRUE
```

Código C:

```
boolean esCapicua(char *cadena) {
    Pila *pilaAux = NULL;
    Elemento *pe = NULL;
    int i,longitud;
    char car;

    if (cadena==NULL) {
        return NULL;
    }
    // Creamos pila auxiliar
    pilaAux = pila_crear();
    if (pilaAux==NULL) {
        return NULL;
    }

    // Recorremos la cadena hasta encontrar 'C'
    // y vamos insertando los caracteres en la pila
    longitud= strlen(cadena);
    for (i=0; i<longitud && cadena[i]!='C'; i++) {
        pe = elemento_crear(cadena[i]);
        pila_push(pilaAux, pe);
        elemento_liberar(pe);
    }

    // En caso de que hayamos llegado a fin de cadena
    // devolvemos error: la cadena no sigue el patron x C y
    if (i==longitud) {
        pila_liberar(pilaAux);
        return FALSE;
    }
}
```

```

// Recorremos la cadena despues de 'C'
for (; i<longitud; i++) {
    pe = pila_pop(pilaAux);

    if (pe == NULL) {
        pila_liberar(pilaAux);
        return FALSE;
    }
    car = convierteAcaracter(pe);
    if (cadena[i] !=car) {
        pila_liberar(pilaAux);
        elemento_liberar(pe);
        return FALSE;
    }

    elemento_liberar(pe);
}

if (pila_vacia(pilaAux)==FALSE) {
    pila_liberar(pilaAux);
    return FALSE;
}

pila_liberar(pilaAux);
return TRUE;
}

```

Ejercicio 26

Asumiendo la existencia de un TAD Elemento que guarda información asociada a una acción realizada en un editor de texto (e.g., cortar un texto, poner en negrita una frase, cambiar el tamaño de fuente de una palabra, etc.), dar el código C de funciones “deshacer” y “rehacer” acciones, mediante el uso del TAD Pila.

```

Elemento *deshacer(Pila *pilaDeshacer, Pila *pilaRehacer) {
    Elemento *accion = NULL;

    if (pilaDeshacer==NULL || pilaRehacer==NULL) {
        return NULL;
    }

    if (pila_vacia(pilaDeshacer)==TRUE || pila_llena(pilaRehacer)==TRUE) {
        return NULL;
    }

    accion = pila_pop(pilaDeshacer);
    pila_push(pilaRehacer, accion);

    return accion;
}

Elemento *rehacer(Pila *pilaDeshacer, Pila *pilaRehacer) {
    Elemento *accion = NULL;

    if (pilaDeshacer==NULL || pilaRehacer==NULL) {
        return NULL;
    }

    if (pila_vacia(pilaRehacer)==TRUE || pila_llena(pilaDeshacer)==TRUE) {
        return NULL;
    }

    accion = pila_pop(pilaRehacer);
    pila_push(pilaDeshacer, accion);

    return accion;
}

```

Ejercicio 27. Escribir el pseudocódigo de las siguientes rutinas:

- `status compruebaParentesis(Cadena str)`: comprueba si los paréntesis de la cadena de caracteres `str` están bien equilibrados utilizando para ello un “contador” de paréntesis.
- `status compruebaSimbolos(Cadena str)`: mediante “contadores” comprueba el buen equilibrio de paréntesis, corchetes y llaves.
- `status compruebaParentesisConPila(Cadena str)`: mediante una pila comprueba el buen equilibrio de paréntesis.
- `status compruebaSimbolosConPila(Cadena str)`: mediante una pila comprueba el buen equilibrio de paréntesis, corchetes y llaves.

Nota: para leer el carácter ‘c’ que contiene una cadena ‘srt’ en la posición ‘i’ se puede utilizar la función `cadena_leer(Cadena srt, pos i, car c)`

a)

```
status compruebaParentesis(Cadena srt)
    Entero contador = 0;
    para i de 1 a cadena_longitud(srt):
        cadena_leer(c, i, srt);
        si c == '(' :
            incrementar contador;
        sino:
            si c == ')' :
                decrementar contador;
    si contador == 0:
        devolver OK;
    sino:
        devolver ERROR;
```

b) OJO: COMO ES NECESARIO COMPROBAR QUE ESTÉN BIEN SITUADOS LOS PARES, NO SE PUEDE HACER CON CONTADORES

EJEMPLO: { () } , COMPROBADO SOLO CON CONTADORES, DARÍA OK. ES NECESARIO USAR UNA PILA PARA COMPROBAR EL ORDEN

c)

```
status compruebaParentesisConPila(Cadena str)
    si pila_crear(p) == ERROR:
        devolver ERROR;
    para i de 1 a cadena_longitud(srt):
        cadena_leer(c, i, srt);
        si c == '(' :
            si push(c, p) == ERROR:
                mientras pila_vacia(p) == FALSE:
                    pop(c, p);
            sino:
                si c == ')' :
                    si pop(c, p) == ERROR:
                        devolver ERROR;
        si pila_vacia(p) == V:
            devolver OK;
        sino:
            devolver ERROR;
```

d)

```
status compruebaTodosConPila(Cadena str)
    si pila_crear(p) == ERROR:
        devolver ERROR;
    para i de 1 a cadena_longitud(srt):
        cadena_leer(c, i, srt);
        si ( c == '(' ) || ( c == '[' ) || ( c == '{' ) :
            si push(c, p) == ERROR:
                mientras pila_vacia(p) == FALSE:
                    pop(c, p);
            sino:
                si pop(c, p) == ERROR:
                    devolver ERROR;
                en caso de:
                    c == ')' :
```

```

        si c != '(' :
            mientras pila_vacia(p) == FALSE:
                pop(c, p);
devolver ERROR;
c == ']' :
    si c != '[' :
        mientras pila_vacia(p) == FALSE:
            pop(c, p);
devolver ERROR;
c == '}' :
    si c != '{' :
        mientras pila_vacia(p) == FALSE:
            pop(c, p);
devolver ERROR;
si pila_vacia(p) == V:
    devolver OK;
sino:
    devolver ERROR;

```

Ejercicio 28. Escribir el código C de:

- Una estructura Pila en la que el top es de tipo entero y en la que los datos son un array de punteros a Elemento con un tamaño máximo de PILA_MAX elementos.
- Una función de nombre `getTop` que recibe un puntero a Pila y devuelva el dato (puntero a Elemento con memoria reservada) en su top sin efectuar un pop del mismo. Asumir que existe una función `Elemento *elemento_copiar(Elemento *pe)` que devuelve una copia del elemento apuntado por `pe`.
- Una función de nombre `pila_copiar` que recibe un puntero a Pila y devuelva una copia de la misma (puntero a Pila) dejando la pila original como se encontraba en un principio. Escribir primero el pseudocódigo y después el código C utilizando las siguientes funciones
 - `status pila_crear(Pila *ps) : devuelve OK/ERROR`
 - `boolean pila_vacia(Pila *ps) : devuelve True/FALSE`
 - `status pila_push(Pila *ps, Elemento *pe) : devuelve OK/ERROR`
 - `Elemento *pila_pop(Pila *ps) : devuelve puntero a elemento o NULL`

Nota: no es necesario implementar control de errores.

a)

```

typedef struct Pila {
    int top;
    Elemento *elementos[PILA_MAX];
} Pila;

```

b)

```

Elemento *getTop(Pila *ps) {
    return elemento_copiar(ps->elementos[ps->top]);
}

```

c)

Pseudocódigo (sin control de errores):

```

Pila pila_copiar(Pila porig) {
    paux = pila_crear ()
    pdest = pila_crear ()

    mientras pila_vacia (porig) = FALSE:
        item=pila_pop (porig)
        pila_push (paux, item)

    mientras pilaVacia (paux) = FALSE:
        item=pila_pop(paux)
        pila_push(porig, item)
        pila_push(pdest, item)

    pila_liberar (paux)
    devolver pdest

```


Código C:

```
Pila *pila_copiar(Pila *ps) {
    Pila *pilaCopia=NULL, *pilaAux = NULL;
    Elemento *pe = NULL;

    // Creamos pila auxiliar
    pilaAux = pila_crear();
    if (pilaAux==NULL) {
        return NULL;
    }

    // Creamos pila copia
    pilaCopia = pila_crear();
    if (pilaCopia==NULL) {
        pila_liberar(pilaAux);
        return NULL;
    }

    // Copiamos los elementos de ps en pilaAux en orden inverso
    while (pila_vacia(ps)==FALSE) {
        pe = pila_pop(ps);
        pila_push(pAux, pe);
        elemento_liberar(pe);
    }

    // Extraemos los elementos de pilaAux y los insertamos en ps y pilaCopia
    while (pila_vacia(pilaAux)==FALSE) {
        pe = pila_pop(pilaAux);
        pila_push(ps, pe);
        pila_push(pilaCopia, pe);
        elemento_liberar(pe);
    }

    pila_liberar(pilaAux);

    return pilaCopia;
}
```

Ejercicio 29. Una implementación en C de Pila tiene como único interfaz las funciones:

- `Pila *pila_crear()`, que crea (reservando memoria), inicializa y devuelve una pila.
- `boolean pila_vacia(Pila *ps)`, que comprueba si una pila está vacía.
- `status pila_push(Pila *ps, Elemento *pe)`, que inserta un elemento en la pila.
- `Elemento *pila_pop(Pila *ps)`, que extrae y devuelve un elemento de la pila.

Con esta interfaz se quiere desarrollar una función:

- `Elemento *pila_getPrimero(Pila *ps)`, que devuelve el primer elemento insertado en una pila, sin que la pila quede modificada. Asumir que existe una función `Elemento *elemento_copiar(Elemento *pe)` que devuelve una copia del elemento apuntado por `pe`.

Para ello:

- a. Dar el pseudocódigo de `pila_getPrimero`.
- b. Dar el código C de `pila_getPrimero`, de forma simple sin controlar errores.
- c. Dar el código C de `pila_getPrimero`, controlando errores. Para ello, asumir que un push precedido de un pop sobre una pila no causa error.

a)

```
Elemento pila_getPrimero(Pila ps)
si pila_vacia(pila) = T:
    devolver NULL
p2 = pila_crear();
mientras pila_vacia(ps) = F:
    elem = pila_pop(ps);
    pila_push(p2, elem);

primero= pila_pop(p2);
pila_push(ps,primero);
mientras pila_vacia(p2) = F:
    elem = pila_pop(p2);
    pila_push(ps, elem);
pila_liberar(p2);
devolver primero;
```

b) En C, sin control de errores:

```
Elemento *pila_getPrimero(Pila *ps)
Elemento *pe, *primero;
Pila *p2;
if (pila_vacia(ps)==TRUE)
    return NULL;
p2 = pila_crear();
while (pila_vacia(ps) == F) {
    pe = pila_pop(ps);
    pila_push(p2, pe);
    elemento_liberar (pe);
}
primero = pila_pop(p2);

pila_push(ps, primero);
while (pila_vacia(p2) == F) {
    pe = pila_pop(p2);
    pila_push(ps, pe);
    elemento_liberar (pe);
}
return primero;
}
```

c) En C, con control de errores:

```
Elemento *PilaPrimero(Pila *ps)
Elemento *pe, *primero;
Pila *p2;

if (pila_vacia(ps) == TRUE)
    return NULL;

p2 = pila_crear();
if (p2 == NULL) {
    return NULL;
}
while (pila_vacia(ps) == F) {
    pe = pila_pop(ps);
    if (pila_push(p2, pe) == ERROR) {
        pila_push(ps, pe);
        while (pila_vacia(p2) == F) {
            pe = pila_pop(p2);
            pila_push(ps, pe);
        }
    }
    elemento_liberar (pe);
}
primero = pila_pop(p2);
pila_push(ps, primero);
while (pila_vacia(p2) == F) {
    pe = pila_pop(p2);
    pila_push(ps, pe);
    elemento_liberar (pe);
}
pila_liberar (p2);
return primero;
}
```

Ejercicio 30. Las primitivas de un TAD Cadena para cadenas de caracteres tienen las siguientes especificaciones:

- `boolean cadena_vacia(Cadena A):` devuelve V o F según A este o no vacía;
- `status cadena_insertar(Cadena A, pos i, car c):` intenta insertar el carácter c en la posición i de la cadena A, y devuelve OK o ERROR;
- `status cadena_extraer(Cadena A, pos i, car c):` intenta extraer el carácter c el situado en la posición i de la cadena A, y devuelve OK o ERROR;
- `status cadena_inicializar(Cadena A):` intenta crear una cadena vacía A y devuelve OK o ERROR.

Especificar en detalle y dar el pseudocódigo de las siguientes rutinas derivadas a partir de las anteriores:

- a. `int cadena_longitud(Cadena A):` devuelve la longitud de la cadena A
- b. `int cadena_localizar(Cadena A, carácter c):` devuelve la posición del carácter c en la cadena o un código de error adecuado
- c. `status cadena_concatenar(Cadena A, Cadena B):` concatena la cadena B tras la cadena A, y devuelve OK o ERROR

a)

```
int cadena_longitud(Cadena A)
i = 0;
si inicializar(S) == ERROR:
    devolver -1;
mientras cadena_vacia(A) == FALSE:
    cadena_extraer(c, i, A);
    si cadena_insertar(c, i, S) == ERROR:
        cadena_insertar(c, i, A);
        mientras cadena_vacia(S) == FALSE:
            decrementar i;
            cadena_extraer(c, i, S);
            cadena_insertar(c, i, A);
        devolver -1;
    si no:
        incrementar i;
j = i;
```

```

mientras cadena_vacia(S) == FALSE:
    decrementar j;
    cadena_extraer(c, j, S);
    cadena_insertar(c, j, A);
devolver i;

```

b)

```

int cadena_localizar(Cadena A, carácter c)
    i = 0;
    si inicializar (S) == ERROR:
        devolver -1;
    cadena_extraer(cExt, i, A);
    cadena_insertar(cExt, i, S);
    mientras (cExt != c) && (cadena_vacia(A) == F):
        incrementar i;
        cadena_extraer(cExt, i, A);
        cadena_insertar(cExt, i, S);
    p = i;
    cAux = cExt;
    para j de i a 0:
        cadena_extraer(cExt, i, S);
        cadena_insertar(cExt, i, A);
        decrementar i;
    si cAux == c:
        devolver p;
    sino:
        devolver -1;

```

c)

```

status cadena_concatenar(Cadena A, Cadena B)
    i = 0;
    mientras cadena_vacia(B) == FALSE:
        si cadena_extraer(c, i, B) == ERROR:
            para j de i-1 a cadena_longitud (A):
                cadena_extraer(c, j, A);
                cadena_insertar(c, j, B);
            devolver ERROR;
        si cadena_insertar(c, cadena_longitud (A)+i, A) == ERROR:
            cadena_insertar(c, i, B);
            para j de i-1 a cadena_longitud (A):
                cadena_extraer(c, j, A); // si pudo insertar podrá extraer
                cadena_insertar(c, j, B); // si pudo extraer podrá insertar
            devolver ERROR;
        incrementar i;
    devolver OK;

```

Ejercicio 31. Escribir el pseudocódigo de

- Una rutina que proporciona la representación binaria de un dato de tipo `int`, mediante el algoritmo de división repetida por 2 y acumulación de restos (pista: introducir los restos en una pila e imprimir su contenido hasta vaciarla).
- La rutina del ejercicio anterior, pero que proporciona la representación en octal (o en hexadecimal si se prefiere) del `int` de entrada.
- Una rutina que resuelve de forma genérica la representación de un dato de tipo `int` en cualquier base.
- Una rutina que proporciona como caracteres los dígitos que componen un `int` de entrada, es decir, 418 se proporcionaría como "418" o '4', '1', '8'.

a)

```
status dec_bin(Entero num)
    pila_crear(p);
    mientras num >= 2:
        resto = num % 2;
        num = num / 2;
        push(resto, p);
    push (num, p); // último cociente = bit más significativo

    mientras pila_vacia(p) == FALSE:
        pop(num, p);
        escribe(num);
    devolver OK;
```

b)

```
status dec_octal(Entero num)
    pila_crear(p);
    mientras num >= 8:
        resto = num % 8;
        num = num / 8;
        push(resto, p);
    push (num, p); // último cociente = bit más significativo
    mientras pila_vacia(p) == FALSE:
        pop(num, p);
        escribe(num);
    devolver OK;
```

```
status dec_hexadecimal(Entero num)
    pila_crear(p);

    mientras num >= 16:
        resto = num % 16;
        num = num / 16;
        en caso de:
            resto == 10 entonces push(A, p);
            resto == 11 entonces push(B, p);
            resto == 12 entonces push(C, p);
            resto == 13 entonces push(D, p);
            resto == 14 entonces push(E, p);
            resto == 15 entonces push(F, p);
        en otro caso:
            push(resto, p);

    // bit más significativo
    en caso de:
        num == 10 entonces escribe(A);
        num == 11 entonces escribe(B);
        num == 12 entonces escribe(C);
        num == 13 entonces escribe(D);
        num == 14 entonces escribe(E);
        num == 15 entonces escribe(F);
    en otro caso:
        escribe(num);
    mientras pila_vacia(p) == FALSE:
        pop(num, p);
        escribe(num);
    devolver OK;
```

c) Versión genérica: Ejercicio 24

d)

```
status dec_caracteres(Entero num)
    pila_crear(p);
    mientras num >= 10:
        resto = num % 10;
        num = num / 10;
        push(resto, p);
    push (num, p); // último cociente = bit más significativo
    mientras pila_vacia(p) == FALSE:
        pop(num, p);
        escribe(num);
        escribe (' '); // escribe un espacio
    devolver OK;
```

Ejercicio 32

Se pide escribir el pseudocódigo y el código C de una función "codificar" con el siguiente prototipo:

```
char *codificar(char *texto, int longitud);
```

La función recibe un texto y su longitud, y devuelve una copia del mismo, pero con sus palabras invertidas.

Para la entrada:

ÉSTE ES UN EJEMPLO DE TEXTO A CODIFICAR.

La salida de la función debe ser:

ETSÉ SE NU OLPMEJE ED OTXET A .RACIFIDOC

La función ha de invertir cada palabra (separada por espacios en blanco) del texto mediante una pila.

Asumir que existen las siguientes primitivas de una pila que almacena caracteres, y NO comprobar errores ni liberar memoria al llamar a push y pop.

```
Pila *pila_crear();  
void pila_liberar(Pila *pila);  
status pila_push(Pila *pila, char character);  
char pila_pop(Pila *pila);  
boolean pila_vacia(Pila *pila);
```

Pseudocódigo sin CdE:

CC codificar (CC S)

 P = pila_crear ()

 S' = CC_crear ()

 // Va metiendo caracteres en pila hasta llegar a un espacio (mete los caracteres de una palabra).

 // Cuando lee espacio, da la vuelta a lo que haya en la pila, añadiendo cada carácter extraído a la cadena S'

 // Cuando lee el fin de la cadena, solo queda dar la vuelta a la última palabra introducida en la pila

 mientras (c = CC_leerCaracter (S)) ≠ EOS:

 si (c==' ')

 mientras pila_vacia(P) == FALSE:

 c= pila_pop(P)

 CC_Concatena (S', c)

 CC_Concatena (S', ' ')

 else

 pila_push (c, P)

 /* vacia la pila para invertir la ultima palabra */

 mientras pila_vacia(P) == FALSE:

 c= pila_pop(P)

 CC_Concatena(S', c)

devolver S'

Código C:

```
char *codificar(char *texto, int longitud) {
    Pila *pila = NULL;
    char c, c2, *textoCodificado = NULL;
    int i, j;

    if ( texto == NULL || longitud <= 0 ) {
        return NULL;
    }

    pila = pila_crear();
    if( pila == NULL ) {
        return ERROR;
    }

    textoCodificado = (char *) malloc(longitud*sizeof(char));
    if( textoCodificado == NULL ) {
        pila_liberar(pila);
        return NULL;
    }

    j = 0;
    for ( i=0; i<longitud; i++ ) {
        // Leemos el siguiente caracter del texto
        c = texto[i];

        if ( c == ' ' ) {          // Invertimos la palabra anterior
            while ( pila_vacia(pila) == FALSE ) {
                c2 = pila_pop(pila);
                textoCodificado[j] = c2;
                j++;
            }
            textoCodificado[j] = c;
            j++;
        }
        else {                     // Guardamos el carácter de la palabra actual
            pila_push(pila, c);
        }
    }

    // Invertimos la última palabra del texto
    while ( pila_vacia(pila) == FALSE ) {
        c2 = pila_pop(pila);
        textoCodificado[j] = c2;
        j++;
    }

    pila_liberar(pila);

    return textoCodificado;
}
```


Ejercicio 33.

Supóngase la siguiente implementación de una pila en C de datos de tipo int

```
struct _PILA {
    int s[STACKSIZE];
};
typedef struct _PILA Pila;
```

Donde se usa s[0] para contener el índice entero del tope de la pila y en s[1] a s[STACKSIZE-1] se almacenan los datos de la pila. Escribir el código de las funciones pila_crear, pila_vacia, pila_llena, pila_push y pila_pop. Recordamos que los prototipos son:

```
Pila *pila_crear();
void pila_liberar(Pila *p);
boolean pila_vacia(const Pila *p);
boolean pila_llena(const Pila *p);
status pila_push(Pila *p, const int e);
int pila_pop(Pila *p);
```

<pre>Pila *pila_crear () { Pila * p = (Pila *) malloc (sizeof (Pila)); if (p == NULL) return NULL; p->s[0]= 0; /* =0, pq pila empieza en 1 */ return p; } status pila_push(Pila *p, const int ee) { /* pila llena */ if(p==NULL p->s[0] == STACKSIZE-1) return ERROR; p->s[0]++; p->s[p->s[0]]= e; return OK; }</pre>	<pre>boolean pila_vacia (const Pila *p) { } boolean pila_llena (const Pila *p) { } int pila_pop (Pila *p) { int e; /* pila vacia */ if(p ==NULL p->s[0] == 0) return ERROR; e = p->s[p->s[0]]; p->s[0]--; return OK; }</pre>
---	---

Ejercicio 34.

Supóngase la siguiente implementación de una pila en C de datos de tipo int, con el tope como puntero a la última opción ocupada del array (con un dato válido):

```
struct _PILA {
    int datos[STACKSIZE];
    int *top;
};
typedef struct _PILA Pila;
```

Escribir el código de las funciones pila_crear, pila_vacia, pila_llena, pila_push y pila_pop. Recordamos que los prototipos son los mismos que en el ejercicio anterior.

Solución con pila de punteros a Elemento (Elemento * en vez de int): disponible en el anexo de las diapositivas del tema de Pilas.