# Documentación

- Para los siguientes problemas utilice la biblioteca estática `libstack_fDoble.a`.

  En esta biblioteca la pila está implementada de dat forma que cuando esta se llena, la memoria disponible para la pila se incrementa (mediante un `realloc`) para poder seguir insertando elementos en la pila.

- A menos que expresamente se diga lo contrario, en todos los problemas se debe implementar el oportuno control de errores (CdE). Para ello deben tenerse en cuenta las característica de la biblioteca utilizada.

- Para probar los códigos C puede utilizar la siguiente función `main`. Deberá asignar a la variable `f` (puntero a función) la función apropiada

```c
int main (int argc, char *argv[]){
    int (*f)(int, char *[]);

    // uncomment the appropiate function
    // f = main_balancedExpression;
    // f = main_decimal2base;
    // f = main_stackInvert;
    // f = main_stack_deepCopy;
     f = main_evalPostFix;
    // f = main_string_invert;
    // f = main_reverseWords;

    return f(argc, argv);
}
```

# Problemas de pilas

## Balanceo de paréntesis.

Proporcione el código de C con control de errores del algoritmo de balanceo de paréntesis visto en clase.

```c
Bool balancedExpression (char *str)
```

**Código C:**

```
Bool balancedExpression (char *str) {
  int i = 0;
  Stack *s = NULL;
  char *it;
  Bool bol = TRUE;
  Status st = OK;

  if (!str) return FALSE;

  s = stack_init ();
  if (!s) return FALSE;

  while (str[i] != '\0' && st == OK) {
      if (str[i] == '(') {
          st = stack_push (s, str + i);
      }
      else if (str[i] == ')') {
         if ((it = (char*) stack_pop (s)) == NULL)
             st = ERROR;
      }
      i++;
  }

  if (st == ERROR || stack_isEmpty (s) == FALSE)
      bol = FALSE;

  stack_free (s);

  return bol;
}


int main_balancedExpression (int argc, char *argv[]) {
    Bool bol;
    char str[] = "( (a+b)*(c+d) )" ;

    bol = balancedExpression (str);

    if (bol == TRUE)
        return EXIT_SUCCESS;
    return EXIT_FAILURE;

}
```

## Decimal converter.

Proporcione el pseudocódigo sin control de errores y el código C con control de errores de una función `char * decimal2base (int number, int base)` que, mediante el algoritmo de división repetida por una base y acumulación de restos y haciendo uso de una pila, devuelva la representación en una base menor de 10 (como cadena de caracteres) de un numero entero positivo pasado como argumento de entrada.

*Ejemplo*:

```
Input  : 8    (integer)
Output : 1000 (base = 2)
```

### Pseudocódigo:

**Algorithm 1:** Decimal to base, no error control

```
   input : Integer: num
           String: base
   output: String: str
1  s = stack_new ()
2  while num ≥ base do
3  |   remainder = num mod base
4  |   num = num / base
5  |   stack_push (s, remainder)
6  end
7  str = string_new ()
8  concat (str, int2char (num))
9  while stack_isEmpty (s) ≡ FALSE do
10 |   e = stack_pop (s)
11 |   concat (str, int2char (e) )
12 end
13 stack_free (s)
14 return str
```

### Código C:

```c
char *decimal2base(int number, int base) {
    Stack *s = NULL;
    int *remainder;
    int i = 0;
    char *str = NULL;  // base-expression
    int *tmp = NULL;
    int size;
    Status st = OK;
```

```
    // init the stack for integers
    s = stack_init ();
    if (!s) return NULL;

    while (number >= base && st == OK) {
        remainder = (int *) malloc (sizeof (int));
        if (remainder) {
            *remainder = number % base;
            number = number / base;
            st = stack_push (s, remainder);
        }
        else
            st = ERROR;
    }

    if (st == OK) {
        size = stack_size (s);
        str = (char *) malloc (size + 2);
        if (!str)
            st = ERROR;
        else
            str[i++] = number + '0';
    }

    if (st == OK) {
        while (stack_isEmpty (s) != TRUE ) {
            tmp =  (int *) stack_pop(s);
            str[i++] = *tmp + '0';   // convert integer --> char
            free (tmp);
         }
        str[i] = '\0';
    }

    if (st == ERROR) {
        while (stack_isEmpty (s) != TRUE ) {
            tmp =  (int *) stack_pop(s);
            free (tmp);
        }
    }

    stack_free (s);
    return str;
}

int main_decimal2base (int argc, char *argv[]){
    char *exp = NULL;
```

```
    int number = 9;
    int base = 2;

    exp = decimal2base (number, base);

    if (exp) {
        fprintf (stdout,"Number: %d, Conversion: %s", number, exp);
        free (exp);
        return EXIT_SUCCESS;
    }

    return EXIT_FAILURE;

}
```

## Inverting the stack

Se desea crear un función `Status stackInvert (Stack **s)` que invierta el contenido de una pila. La pila **no** debe modificarse cuando la función devuelva ERROR.

**Código C:**

```
Status stackInvert (Stack **s) {

    Stack *aux = NULL;
    Status st = OK;
    void *e = NULL;

    aux = stack_init ();
    if (!aux)
        return ERROR;

    while (stack_isEmpty (*s) == FALSE && st == OK) {
        e = stack_pop (*s);
        st = stack_push (aux, e);
    }

     // The auxiliar stack is assigned to the original stack
    if (st == OK) {
        stack_free (*s);
        *s = aux;
    }

    // Recover the original stack if st == ERROR
```

```
    else {
        stack_push (*s, e);          // No Error
        while (stack_isEmpty (aux) == FALSE) {
            e = stack_pop (aux);
            stack_push (*s, e);     // No Error
        }
        stack_free (aux);
    }

    return st;
}

int main_stackInvert (int argc, char *argv[]){
    Stack *s = NULL;
    int i;
    int nelem;
    int elem[] = {1, 2, 3, 4, 5};
    Status st = OK;

    s = stack_init ();
    if (!s) return EXIT_FAILURE;

    nelem = sizeof (elem) /sizeof (int);
    for (i=0; i<nelem && st == OK; i++) {
        st = stack_push (s, elem+i);
    }

     if (st == OK) {
         stack_print (stdout, s, int_print);
         // invert the satck
         st = stackInvert (&s);
         stack_print (stdout, s, int_print);
     }

    stack_free (s);
    if ( st == OK)
        return EXIT_SUCCESS;

    return EXIT_FAILURE;
}
```

## Merge stacks

Suponed que se disponen de dos pilas en los que sus elementos se encuentran ordenados en **orden ascendente**. Proporcione el pseudocódigo **sin control de errores** y el código C

de una función `Status mergeStacks (Stack *s1, Stack *s2, Stack *s3, int (*ele_cmp)(void *, void *))` que devuelve una pila ordenada en **orden descendiente**.

### Pseudocódigo

**Algorithm 2:** Merge stacks, no error control

```
input : Stack: s1
        Stack: s2
        Stack: s
output: Status: st
1  while stack_isEmpty (s1) ≡ False & stack_isEmpty (s2) ≡ False do
2      if stack_top (s1) > stack_top (s2) then
3          e = stack_pop (s1)
4      else
5          e = stack_pop (s2)
6      end
7      stack_push (s, e)
8  end
9  // Detect the non-empty stack
10 if stack_isEmpty (s1) ≡ True then
11     ps = s2
12 else
13     ps = s1
14 end
15 // Pop from the non-empty stack
16 while stack_isEmpty (ps) ≡ False do
17     e = stack_pop (ps)
18     stack_push (s, e)
19 end
20 return st
```

## Inverting String

Proporcione el pseudocódigo sin control de errores y el código C con control de errores de una función `char * string_invert (char *str)` que, usando una pila, invierta una cadena de caracteres

*Ejemplo de ejecución:*

```
$ main "hola don pepe"
String: hola don pepe
Inverted string: epep nod aloh
```

## Stack deep copy

Crear una función de prototipo

```
Stack * stack_deepCopy (Stack *s, void * (*ele_cpy)(const void *),
    void (*ele_free)(void *))
```

que realice una copia **profunda** (un duplicado) de la pila apuntada por `s`. La función no debe modificar la pila original. La función devolverá un puntero a la copia o null sino hubiese memoria suficiente para realizar la copia.

```
/**
  * @brief The ele_cpy function copies the element pointed to by a.
  *
  * This function allocates memory for the copy.
  *
  * @param A pointer to the element
  * @return  This function returns a pointer to the new element or
     null
  * if insufficient memory is available to create the copy.
**/
  void * ele_cpy (const void *a);

/**
  * @brief  This function frees all the memory used by th element.
  *
  * @param a A pointer to the integer
**/
  void int_free (void *a);
```

**Código C:**

```
Stack * stack_deepCopy (Stack *s, void * (*ele_cpy)(const void *),
        void (*ele_free)(void *)){

    Status st = OK;
    Stack *scp = NULL, *saux = NULL;
    void *item , *newitem = NULL;

    if (!s || !ele_cpy || !ele_free) return NULL;

    scp = stack_init ();
    if (!scp)  return NULL;

    saux = stack_init ();
    if (!saux) {
```

```
        stack_free (scp);
        return NULL;
    }


    while (stack_isEmpty (s) == FALSE && st == OK){
        item = stack_pop (s);
        st = stack_push (saux, item);
    }

    if (st == ERROR)
        stack_push (s, item);

    while (stack_isEmpty (saux) == FALSE){      //recover the
       original stack
        item = stack_pop (saux);
        stack_push (s, item);                   // No error
        // copy the item and push it in stack_copy
        if (st == OK) {
            newitem = ele_cpy (item);
            st = stack_push (scp, newitem);
            if (st == ERROR)  ele_free (newitem);
        }
    }

    // if ERROR empty the stack_copy
    if (st == ERROR) {
        while (stack_isEmpty (scp) == FALSE) {
            newitem = stack_pop (scp);
            ele_free (newitem);
        }
        stack_free (scp);
        scp = NULL;
    }

    stack_free (saux);
    return scp;
 }


int main_stack_deepCopy (int argc, char *argv[]) {
    Stack *s = NULL, *scp = NULL;
    int arr[] = {5, 10, 15, 20};
    int size, i;
    void *item;
    Status st = OK;
```

```
    // create the stack
    s = stack_init ();
    if (s == NULL)
        return EXIT_FAILURE;

    // push elements in the stack
    size = sizeof(arr)/sizeof(int);
    for (i = 0;  i < size && st == OK; i++) {
        st = stack_push (s, arr+i);
    }

    // do the deep copy
    if (st == OK) {
        scp =  stack_deepCopy (s, int_copy, int_free);
        if (!scp)
            st = ERROR;
    }

    stack_print (stdout, scp, int_print);

    stack_free (s);

    // Empty the deep_copy_stack
    while (stack_isEmpty (scp) == FALSE) {
        item = stack_pop (scp);
        int_free (item);
    }

    stack_free (scp);

    if (st == ERROR)
        return EXIT_FAILURE;
    return EXIT_SUCCESS;
}
```

## Palíndromo

Se dice que una secuencia de caracteres es un palíndromo si la secuencia de caracteres es igual leída desde el principio que desde el final.

Escriba el pseudocódigo y código C con control de errores de un algoritmo que determine si una cadena de caracteres es un palíndromo utilizando una pila.

*Ejemplo:*

```
Input  : esAse
Output : The string: esAse is a palindrome

Input  : esbbse
Output : The string: esbbse is a palindrome
```

**Código C:**

```c
 Bool isPalindrome (char *str) {
     int mid, size, i =0;
     Stack *s = NULL;
     char *ele = NULL;
     Status st = OK;

     if (!str) return FALSE;

     s = stack_init ();
     if (!s) return ERROR;

     size = strlen (str);
     mid = size / 2;

     while (i < mid && st == OK) {
         st = stack_push (s, str + i);
         i = i + 1;
     }

     // if strlen (str) is odd skip the miidle character in the string
     if (st == OK && size % 2 != 0) {
         i++;
     }

     while (stack_isEmpty (s) == FALSE && st == OK) {
         ele = (char*) stack_pop (s);
         if (*ele != str[i++])
             st = ERROR;
     }

     //ATENTION: Even st == ERROR for psshing......

     stack_free (s);

     if (st == OK)
         return TRUE;
     return FALSE;
```

```
}

int main_isPalindrome (int argc, char *argv[]) {
    char str[]="abcHba";
    Bool bol = FALSE;

    bol = isPalindrome (str);
    if (bol == TRUE)
        fprintf (stdout, "The string: %s is a palindrome", str);
    else
        fprintf (stdout, "The string: %s no es palindrome", str);

    return EXIT_SUCCESS;
}
```

## Reverse words

Crear una función de prototipo `Status reverseWords (char *strout, char *strin)` que invierta las palabras de una cadena de caracteres. Las palabras están separadas por uno o varios caracteres en blanco. La función **no** debe modificar la cadena de caracteres original.

*Ejemplo de ejecución:*

```
$ main "Hello World"
Input: Hello World
Output: olleH dlroW

$ main "Programming is  nice!!"
Input : Programming is  nice!!
Output: gnimmargorP si  !!ecin
```

```

/**
  * @brief: Reverse the words of an input string to an output string.
  *
  * The words in the input string are separated by a space character.
  * The strings may not overlap, and the destination string strout
  * must be large enough to receive the inverted expression.
  *
  * @param strout, Pointer to the output buffer
  * @param strin, Pointer to the input expression
  * @return The function returns OK or ERROR
 **/
    Satus reverseWords (char *strout, char *strin);
```

**Pseudocódigo:**

**Algorithm 3:** Reverse words, no error control

```
   input  : String: in_str = {c_1 c_2 ... c },
            String: rev_str = ∅
   output: Status:
1  s = stack_new ()
2  foreach c in in_str do
3      if c ≠ ' ' then
4          stack_push (s, c)
5      else
6          while stack_isEmpty (s) ≡ False do
7              e = stack_pop (s)
8              concat (rev_str, e)
9          end
10         concat (rev_str, ' ')
11     end
12 end
13 while stack_isEmpty (s) ≡ FALSE do
14     e = stack_pop (s)
15     concat (rev_str, e)
16 end
17 stack_free (s)
18 return OK
```

## Pila genérica con *seguimiento del máximo*

Suponga que se dispone de la biblioteca estática `libstack_fDoble.a` con la interfaz del TAD STACK.

Se desea crear el TAD MAX_IN_STACK para manejar una **pila** que además de incluir las funciones de la interfaz del TAD STACK, incluya la función **`char`** `*maxinstack_getMax (Maxinstack *s)` que permita, con coste O(1), consultar el máximo de los elementos almacenados en la pila (sin extraerlo).

(a) Proporcione el fichero `maxinstack.h` con la especificación del TAD `MAX_IN_STAK`

(b) Propocione el código C del fichero `maxinstack.c` con la estructura de datos y la implementación de las funciones de la interfaz del TAD MAX_IN_STACK

**Código C:**

```c
typedef struct _Maxinstack Maxinstack;
```

```
struct _Maxinstack {
    Stack *data;
    Stack *max;
    int (*f_compare)(const void *, const void *);   //func to compare
        stacks elements
};


Maxinstack *maxinstack_init (int (*fcmp)(const void *, const void *)
   ) {
    Maxinstack *s;

    s = (Maxinstack *) malloc (sizeof(Maxinstack));
    if (!s) {
   return NULL;
    }

    // init the data stack
    s->data = stack_init ();
    if (!s) {
        free (s);
        return NULL;
    }

    //init the maxima stack
    s->max = stack_init ();
    if (!s) {
        stack_free (s->data);
   free (s);
        return NULL;
    }

    s->f_compare = fcmp;
    return s;
}

void maxinstack_free (Maxinstack* stc) {
   if (!stc) return;

   stack_free (stc->data);
   stack_free (stc->max);
   free (stc);
}


Bool maxinstack_isEmpty (const Maxinstack* stc) {
```

```
    if (!stc) return TRUE;

    return stack_isEmpty (stc->data);
}

Status maxinstack_push (Maxinstack *stc, const void* ele) {
   const void *top = NULL;
   Status ret = OK;

   if (!stc || !ele) return ERROR;

   // push in the data stack
   if (stack_push (stc->data, ele) == ERROR)
       return ERROR;

   // push in the maxima stack
   top = (void *) stack_top (stc->max);
   if (stack_isEmpty (stc->max) == TRUE || stc->f_compare (ele, top)
      > 0)
       ret = stack_push (stc->max, ele);
   else
       ret = stack_push (stc->max, top);

   if (ret == ERROR)
       stack_pop (stc->data);

   return ret;
}


void * maxinstack_pop (Maxinstack* stc) {
    if (!stc || maxinstack_isEmpty (stc) == TRUE) return NULL;

    stack_pop (stc->max);
    return stack_pop (stc->data);
}


void * maxinstack_top (Maxinstack* stc) {
    if (stc == NULL) return NULL;

    return (stack_top (stc->data) );
}


void * maxinstack_getMax (Maxinstack *s) {
   if (!s) return NULL;
```

```
   return (stack_top(s->max));
}


int maxinstack_print (FILE *pf, const Maxinstack* stc,
        int (*f_print)(FILE *, const void *) ) {
    if(!pf || !stc || !f_print) return -1;

    return (stack_print (pf, stc->data, f_print));
}


int main_maxinstack (int argc, char *argv[]) {
    Maxinstack *s = NULL;
    char data[] = {'c', 'a', 'b', 'z'};
    int size;
    Status st = OK;
    int i;
    void *ele;

    size = sizeof(data) / sizeof(char);

    s = (Maxinstack *) maxinstack_init (char_cmp);
    if (s == ERROR) return EXIT_FAILURE;

    for (i = 0; i < size && st == OK; i++) {
        st = maxinstack_push (s, data + i);
        ele = maxinstack_getMax (s);
        char_print (stdout, ele);
    }

    if (st == ERROR) {
        maxinstack_free (s);
        return EXIT_FAILURE;
    }

    maxinstack_print (stdout, s, char_print);

    while (maxinstack_isEmpty (s) == FALSE )
        maxinstack_pop (s);

    maxinstack_print (stdout, s, char_print);

    maxinstack_free (s);
    return EXIT_SUCCESS;
}
```

## Evaluación expresión postfija

Proporcione el código C con control de errores del algoritmo que evalúa una expresión postfijo. Suponga que la expresión postfijo solo contiene números naturales de un única cifra y operadores sin delimitador. Por ejemplo 23+75-/. Pruebe el algoritmo con ayuda de una función main que tenga como argumento una expresión postifijo.

*Ejemplo:*

```
$ .\main 23+75-*
Input postfix expression: 23+31-* Evaluation:10

$ .\main 23+31-
Invalid expression: 23+31-
```

Pare ello considere los operadores y el siguiente prototipo de la función

```
#define  OPERATORS {'+', '-','*', '/', '%'}
/**
  * @brief: Evaluate a postfix expression.
  *
  * The input string only contain operators or natural numbers of one
     digit. There are not delimiters.
  *
  * @param exp, Pointer to the input expression
  * @param r, Return the evaluation result or NULL if the expression
     is not valid
  * @return OK or ERROR
 **/
    Status evalPostFix (char *exp, int **r)`
```

**Código C:**

```
#define  OPERATORS "+-*/%"

Bool isOperator (char c) {
    char *oper = OPERATORS;
    char *pc;

    // exclude the char '\0' as operator
    if (c == '\0')
       return FALSE;
```

```
    pc = strchr (oper, c);

    if (!pc) return FALSE;
    return TRUE;
}

Bool isOperand (char c) {
    Bool b;

    b = (isOperator(c) == TRUE)  ? FALSE : TRUE;
    return b;
}

int *evaluate (int *arg1, int *arg2, char oper) {
    int *p = NULL;

    if (!arg2 || !arg1) return NULL;

    p = (int*) malloc (sizeof(int));
    if (!p) return NULL;

    switch (oper) {
        case '+':
            *p = *arg1 + *arg2;
            break;
        case '-':
            *p = *arg1 - *arg2;
            break;
        case '*':
            *p = *arg1 * *arg2;
            break;
        case '/':
            *p = *arg1 / *arg2;
            break;
        case '%':
            *p = *arg1 % *arg2;
            break;
        default :
            free(p);
            p = NULL;
    }

    return p;
}

Status evalPostFix (char *exp, int **r){
```

```
    Stack *s = NULL;
    Status st = OK;
    int i = 0;
    void *e = NULL;
    int *arg1 = NULL, *arg2 = NULL;

    if (!exp) return ERROR;

    s = stack_init ();
    if (!s) return
            ERROR;

    while (exp[i] != '\0' && st == OK) {
        if (isOperand (exp[i]) == TRUE) {
            e = int_init (exp[i] - '0'); // return a pointer to
               integer
            if (e)
                st = stack_push (s, e);
            else
                st = ERROR;
        }
        else if (isOperator (exp[i]) == TRUE){
            arg2 = (int*) stack_pop (s);
            arg1 = (int*) stack_pop (s);
            e = evaluate (arg1, arg2, exp[i]);
            free (arg1);
            free (arg2);
            if (e)
                st = stack_push (s, e);
            else
                st = ERROR;
        }
        i++;
    }


    if (st == OK) {
        *r = stack_pop (s);
        if ((*r) == NULL || stack_isEmpty (s) == FALSE)
            st = ERROR;
    }

    // Empty the stack
    while (stack_isEmpty (s) == FALSE) {
        e = stack_pop (s);
        free (e);
    }
```

```
    stack_free (s);

    return st;
}

int main_evalPostFix (int argc, char *argv[]) {
    char *exp = NULL ;
    int *ret = NULL;
    Status st;

    if (argc != 2) return EXIT_FAILURE;

    exp = argv[1];
    st = evalPostFix (exp, &ret);

    if (st == OK)
        fprintf (stdout, "Input postfix expression: %s Evaluation:%d"
           , exp, *ret);
    else
        printf("Invalid postfix expression: %s", exp);

    free (ret);
    return EXIT_SUCCESS;
}

//_EVALPOSTFIX
```