PRÁCTICA 2: Tipo Abstracto de Dato Pila

OBJETIVOS

- Aprender a utilizar el TAD Pila para resolver problemas.
- Aprender a elegir la estructura de datos apropiada para implementar el TAD Pila.
- Codificar sus primitivas y utilizarlo en un programa principal.
- Aprender a utilizar una librería estática en C.

NORMAS

Los programas que se entreguen deben:

- Estar escritos en C, siguiendo las normas de programación establecidas.
- Compilar sin errores ni warnings incluyendo las banderas `-Wall` y `-pedantic` al compilar.
- Ejecutarse sin problema en una consola de comandos.
- Incorporar un adecuado control de errores. Es justificable que un programa no admita valores inadecuados, pero no que se comporte de forma anómala con dichos valores.
- No producir fugas de memoria al ejecutarse.

PLAN DE TRABAJO

- **Semana 1:** Ejercicio 1. Prueba de la biblioteca de pila e implementación y uso del algoritmo de ordenación de pila.
- **Semana 2:** Ejercicio 2. Algoritmo de recorrido del laberinto en profundidad.
- **Semana 3:** Ejercicio 3. Implementación y prueba de una biblioteca para el TAD Pila.

Cada profesor indicará en clase cómo realizar las **entregas parciales semanales**: papel, e-mail, Moodle, etc.

La entrega final se realizará a través de Moodle, siguiendo escrupulosamente las instrucciones indicadas en el enunciado referentes a la organización y nomenclatura de ficheros y proyectos. Se recuerda que el fichero comprimido que se debe entregar debe llamarse `Px_Prog2_Gy_Pz`, siendo `x` el número de la práctica, `y` el grupo de prácticas y `z` el número de pareja (ejemplo de entrega de la pareja 5 del grupo 2161: `P2_Prog2_G2161_P05.zip`).

El fichero comprimido debe contener la siguiente organización de ficheros:

```
--- P2_Prog2_Gy_Pz/
 |--- point.c
 |--- point.h
 |--- map.c
 |--- map.h
 |--- p2_e1a.c
 |--- p2_e1b.c
 |--- p2_e1c.c
 |--- p2_e2.c
 |--- p2_e3.c
 |--- Makefile
 |--- libstack_fDoble.a
 |--- stack_fDoble.c
 |--- stack_fDoble.h
 |--- laberinto_1.txt
 |--- laberinto_2.txt
```

Las fechas de subida a Moodle del fichero son las siguientes:

- Los alumnos de Evaluación Continua, la semana del **21 de marzo** (cada grupo puede realizar la entrega hasta las **23:55 h**. de la noche anterior a su clase de prácticas).
- Los alumnos de Evaluación Final, según lo especificado en la normativa.

Ejercicio 1.

El objetivo de este ejercicio consiste en implementar un algoritmo para ordenar los elementos de una pila.

Algoritmo

El algoritmo recibe una pila y devuelve otra pila con los elementos originales ordenados de menor (bottom) a mayor (top). La pila original se vacía. En el pseudo-código no se ha incluido el oportuno control de errores.

Algorithm 1: Order stack, no error control

```
   input  : Stack s
   output: Stack s_o
1  s_o = stack_init ()
2  while stack_isEmpty (s) ≡ False do
3  |   e = stack_pop (s)
4  |   while stack_isEmpty (s_o) ≡ False & e < stack_top (s_o) do
5  |   |   e_a = stack_pop (s_o)
6  |   |   stack_push (s, e_a)
7  |   end
8  |   stack_push (s_0, e)
9  end
10 return s_o
```

En el primer apartado del ejercicio, el algoritmo anterior se utilizará para ordenar una pila de puntos. En el segundo apartado se ordenará una pila de enteros.

Recursos adicionales

Debes hacer uso de la librería del TAD Stack `libstack_fDoble.a` y la interfaz pública definida en el archivo de cabecera `stack_fDoble.h` (puedes ver el contenido de este fichero en el apéndice 1.1 de este documento)

Ejercicio 1a.

Funciones adicionales de punto:

Añade a la interfaz del TAD Point las funciones `point_euDistance()` que calcula la distancia euclidea entre dos puntos cualesquiera y `point_cmpEuDistance()` que compara dos puntos utilizando sus distancias euclideas al origen de coordenadas $(0, 0)$.

```
/**
 * @brief Calculate the euclidean distance betweeen two points.
 *
 * The euclidean distance is defined as sqrt ((x1-x2)^2 + (y1-y2)^2)
 * where (x1, y1) and (x2, y2) are the coordinate of both points
 *
 * @code
 * // Example of use
 * const Point *p1, *p2;
 * double d;
 * p1 = point_new (x1, y1, BARRIER);
 * p2 = point_new (x2, y2, SPACE);
 * point_euDistance (p1, p2, &d);
 * printf ("%lf", d);
 * // .... additional code ....
 * @endcode
 *
 * @param p1 pointer to point
 * @param p2 pointer to point
 * @param distance addresss
 *
 * @return Returns OK or ERROR in case of invalid parameters
 */
Status point_euDistance (const Point *p1, const Point *p2, double *
   distance);

/**
 * @brief Compares two points using their euclidean distances to the
    point (0,0).
 *
 *
 * @param p1,p2 Points to compare.
 *
 * @return It returns an integer less than, equal to, or greater than
    zero if
 * the euclidean distance of p1 to the origin of coordinates is found
   ,
 * respectively, to be less than, to match or be greater
 * than the euclidean distance of p2. In case of error, returns
    INT_MIN.
 */
int point_cmpEuDistance (const void *p1, const void *p2);
```

Comprobación de la corrección de las funciones

Crea un programa **p2_e1a.c** que realice las siguientes acciones:

- Generar n puntos de coordenadas aleatorias entre 0 y 100 donde n es un parámetro de entrada del programa.
- Imprimir para cada uno de los puntos anteriores la distancia euclidea al origen de coordenadas.
- Comparar las distancias euclideas para todos los pares de puntos.

Ejemplo de salida:

El siguiente ejemplo corresponde a la ejecución del programa para 5 puntos cuando **no** se elige una semilla aleatoria (ver la siguiente ayuda)

```
$ ./p2_e1a 5
Point p[0]=[(6, 3): +] distance: 6.708204
Point p[1]=[(5, 7): +] distance: 8.602325
Point p[2]=[(5, 3): +] distance: 5.830952
Point p[3]=[(2, 6): +] distance: 6.324555
Point p[4]=[(1, 9): +] distance: 9.055385
p[0] <  p[0]: False
p[0] <  p[1]: True
p[0] <  p[2]: False
p[0] <  p[3]: False
p[0] <  p[4]: True
p[1] <  p[1]: False
p[1] <  p[2]: False
p[1] <  p[3]: False
p[1] <  p[4]: True
p[2] <  p[2]: False
p[2] <  p[3]: True
p[2] <  p[4]: True
p[3] <  p[3]: False
p[3] <  p[4]: True
p[4] <  p[4]: False
```

Ayuda: Como generar puntos con coordenads aleatorias

El siguiente extracto de código (**incompleto y sin control de errores**) genera `NPOINTS` puntos con coordenadas aleatorias entre `0` y `MAX_RAND` y calcula sus distancias al origen de coordenadas.

```
#include <math.h>
#include <time.h>
```