# PRÁCTICA 1: Tipos Abstractos de Datos y Estructuras de datos

## OBJETIVOS

- Profundizar en el concepto de TAD (Tipo Abstracto de Dato).
- Aprender a elegir la estructura de datos apropiada para implementar un TAD.
- Codificar sus primitivas y utilizarlo en un programa principal.
- Entender el tipo de dato `void *` del lenguaje 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.
- **Semana 2:** Ejercicio 2.
- **Semana 3:** Ejercicio 3.

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: `P1_Prog2_G2161_P05.zip`).

El fichero comprimido debe contener la siguiente organización de ficheros:

```
--- P1_Prog2_Gy_Pz/
   |--- point.c
   |--- point.h
   |--- map.c
   |--- map.h
   |--- types.h
   |--- p1_e1.c
   |--- p1_e2.c
   |--- p1_e3.c
   |--- Makefile
   |--- laberinto_1.txt
   |--- laberinto_2.txt
```

Las fechas de la entrega final son las siguientes:

- Los alumnos de Evaluación Continua, la semana del **28 de febrero** (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.

En este ejercicio definiremos e implentaremos el Tipo Abstracto de Dato (TAD) `Point`. El TAD `Point` tiene tres atributos: las coordenadas **x** e **y** del punto (ambas de tipo **`int`**) y un **symbolo** (de tipo **`char`**). Los posibles valores del símbolo están definidos en el fichero `point.h` (ver Apéndice 2).

Otros tipos de datos útiles están definidos en el fichero `types.h` (ver Apéndice 1).

### Definición del tipo abstracto de dato `Point`

Para definir el TAD `Point` de acuerdo con la metodología de encapsulamiento vista en clase deberás crear el fichero `point.h`. Este fichero debrá incluir:

- Las directivas `#include` que sean necesarias.
- La declaración del tipo `Point`,

```
typedef struct _Point Point;
```

- Los prototipos de las funciones de la interfaz pública del TAD `Point`

### Implementación del TAD `Point`

Para implementar el TAD `Point` deberás crear el fichero `point.c`. En este este fichero deberás:

- Incluir la estructura de datos `_Point` adecuada. En este caso utilizaremos:

```
struct _Point {
    int x, y;
    char symbol;

    Bool visited;  // for DFS
};
```

- Implementar funciones de la interfaz pública declaradas en el fichero `point.h`.
- Las funciones privadas que consideres oportunas.

**Comprobación de la corrección de la definición del tipo `Point` y de su interfaz.**

Crea un programa en un fichero de nombre `p1_e1.c` cuyo ejecutable se llame `p1_e1`, y que realice las siguientes operaciones:

- Declarar un array de 4 punteros a punto de nombre `p`.
- Inicializar los elementos del array `p[0]` y `p[1]`. Al primero se le asignará un punto de coordenadas `(0, 0)` y *symbol* `BARRIER`. y al segundo el punto de coordenadas `(0, 1)` y *symbol* `BARRIER`.
- Imprimir ambos puntos en la salida estándar y después un salto de línea.
- Comparar ambos puntos y mostrar un mensaje que muestre el resultado.
- Asignar al tercer elemento del array `p` una copia del primer punto.
- Imprimir el tercer punto en la salida estándar.
- Comparar el primer y tercer puntos y mostrar un mensaje con el resultado.
- Modificar el punto referenciado por el puntero `p[2]` asignándole al campo *symbol* el valor `SPACE`.
- Comparar nuevamente el primero y tercer punto.
- **Asignar** a `p[3]` el primer punto.
- Modificar el campo *symbol* del punto referenciado por `p[3]` asignándole el valor `OUTPUT`
- Imprimir los cuatro puntos.

Ten en cuenta la diferencia en como se gestiona la memoria cuando **creas** un punto, haces una **copia** de un punto o **asignas** un elemento del array de punteros a un punto ya creado.

**IMPORTANTE:** El programa deberá gestionar correctamente la memoria y efectuará el oportuno control de errores.

**Salida esperada:**

```
[(0, 0): +][(0, 1): +]
Equal points p[0] and p[1]? No
Creating p[2]: [(0, 0): +]
Equal points p[0] and p[2]? Yes
Modifying p[2]: [(0, 0): .]
Equal points p[0] and p[2]? No
Assign p[3] = p[0]
Modifying p[3]:[(0, 0): o]
[(0, 0): o][(0, 1): +][(0, 0): .][(0, 0): o]
```

## EJERCICIO 2.

En este ejercicio se definirá el Tipo Abstracto de Datos (TAD) `Map` como un conjunto de *puntos*. Un *mapa* representa la abstracción de un *laberinto*.

El siguiente pictograma es un ejemplo de un laberinto de 20 puntos distribuidos en 4 filas y 5 columnas. Todo laberinto tendrá un punto de entrada y uno de salida. Para todo punto del laberinto, la coordenada *x* corresponde a la columna y la coordenada *y* a la fila que ocupa en el laberinto. En este ejemplo las coordenadas del punto de entrada son (3,1) y las del de salida (1, 2). Se asume que las coordenadas del punto de la esquina superior izquierda son (0,0)

```
+++++
+..i+
+o..+
+++++
```

### 2.1. Definición e implementación del TAD Map: Estructura de datos e interfaz.

Se desea definir una estructura de datos para representar el TAD `Map`. Asumiremos que los datos que almacena el laberinto son de tipo `Point` y que su capacidad máxima es 64 filas y 64 columnas.

Para diseñar e implementar el TAD `Map` deberás:

- Crear el fichero `map.h` con la definición del tipo `Map`, los prototipos de las funciones de la interfaz (ver Apéndice 3) y las directivas `#include` necesarias.
- Definir en el fichero `map.c` la estructura de datos `_Map`:

```c
#define MAX_NCOLS 64        // Maximum map cols
#define MAX_NROWS 64        // Maximum map rows
#define MAX_BUFFER 64       // Maximum file line size

struct _Map {
    unsigned int nrows, ncols;

    Point *array[MAX_NROWS][MAX_NCOLS]; // array with the Map points
    Point *input, *output;              // points input/output
};
```

- Implementar en el fichero `map.c` las funciones de la interfaz declaradas en el fichero `map.h`.

- Incluir en el fichero `map.c` las funciones privadas que consideres oportunas.

### 2.2. Comprobación de la corrección de la definición del tipo Map y de su interfaz.

Crea un programa en un fichero de nombre `p1_e2.c` cuyo ejecutable se llame `p1_e2`, y que realice las siguientes operaciones:

- Inicializa un `Map` de 4 columnas y 3 filas.
- Inserta en el `Map` el siguiente laberinto punto a punto

```
++++
+io+
++++
```

- Imprime el mapa.
- Devuelve los vecinos en el `Map` del punto correspondiente al output del laberinto.

El programa debe gestionar adecuadamente los recursos y gestionar correctamente los errores

**Salida esperada:**

```
Map:
3, 4
[(0, 0): +][(1, 0): +][(2, 0): +][(3, 0): +][(0, 1): +][(1, 1): i][(2,
    1): o][(3, 1): +][(0, 2): +][(1, 2): +][(2, 2): +][(3, 2): +]
Get output neighboors:
[(3, 1): +][(2, 0): +][(1, 1): i][(2, 2): +]
```

## Ejercicio 3: Lectura de un laberinto desde fichero.

Implementa la siguiente función de la interfaz de `Map`, que permite cargar un laberinto a partir de la información leída de un fichero de texto. El formato del fichero está definido en la documentación de la función.

```
/**
 * @brief Reads a map definition from a text file.
 *
 * Reads a map description from the text file pointed to by pf,
 * and fills the map.
 * This function allocates memory for the map and its points.
 *
 * The file contains a header with the map number of rows and cols.
 * The next file lines corresponds with the points of the map.
 *.
 * For example the file:
 *
 * 4 3
 * ++++
 * +io+
 * ++++
 *
 * Represents a map with 4 rows, 3 columns and the points:
 * [(0, 0): +][(1, 0): +][(2, 0): +][(3, 0): +][(0, 1): +][(1, 1): i]
 * [(2, 1): o][(3, 1): +][(0, 2): +][(1, 2): +][(2, 2): +][(3, 2): +]
 *
 * @param pf, Pointer to the input stream.
 *
 * @return the map or NULL if there is any error
 */
Map * map_readFromFile (FILE *pf);
```

Para probar la función crea el programa `p1_e3.c` que reciba como argumento el nombre de un fichero de texto, cree un `Map` de acuerdo a la descripción contenida en ese fichero, imprima el mapa por pantalla e imprima por pantalla los vecinos del punto output y los de la esquina inferior derecha del mapa.

La ejecución de este programa debe funcionar sin problemas, incluyendo una gestión adecuada de la memoria (*valgrind* no debería mostrar fugas ni errores).

A continuación se muestra la salida del programa cuando se ejecuta en la terminal el comando:

```
./p1_e3 laberinto_1.txt
```

```
Maze:
4, 5
[(0, 0): +][(1, 0): +][(2, 0): +][(3, 0): +][(4, 0): +][(0, 1): +][(1,
    1): .][(2, 1): .][(3, 1): i][(4, 1): +][(0, 2): +][(1, 2): o][(2,
   2): .][(3, 2): .][(4, 2): +][(0, 3): +][(1, 3): +][(2, 3): +][(3,
   3): +][(4, 3): +]
Get output neighboors:
[(2, 2): .][(1, 1): .][(0, 2): +][(1, 3): +]
Get right inferior corner neighboors:
[(4, 2): +][(3, 3): +]
```

## APÉNDICES

### Apéndice 1.

Contenido del fichero `types.h`:

```
/**
 * @file types.h
 * @author Profesores Programación 2
 * @date 2 February 2020
 * @brief ADT Boolean and Status
 *
 * @details Here typically goes a more extensive explanation of what
    the header
 * defines. Doxygens tags are words preceeded by @.
 *
 * @see
 */

#ifndef TYPES_H_
#define TYPES_H_

/**
 * @brief ADT Boolean
 */
typedef enum {
    FALSE=0, /*!< False value */
    TRUE=1  /*!< True value  */
} Bool;

/**
 * @brief ADT Status
 */
typedef enum {
    ERROR=0, /*!< To codify an ERROR output */
    OK=1     /*!< OK output */
} Status;

#endif /* TYPES_H_ */
```

**Apéndice 2.**

Contenido del fichero `point.h`:

```
/*
 * File:   point.h
 * Author: profesores
 *
 * Created on 21 de enero de 2022, 15:50
 */

#ifndef POINT_H
#define POINT_H

#include <stdlib.h>
#include <stdio.h>

#include "types.h"

#define ERRORCHAR 'E'
#define INPUT 'i'
#define OUTPUT 'o'
#define BARRIER '+'
#define SPACE '.'

typedef struct _Point Point;


/**
 * @brief Constructor. Initialize a point.
 *
 * This function allocates memory for a point and sets its fields
    acoording
 * its input parameter values.
 *
 * @code
 * // Example of use
 * Point * v;
 * v = point_new (0, 0, BARRIER);
 * @endcode
 *
 * @param x Point x coordinate
 * @param y Point y coordinate
 * @param symbol Point symbol
 *
 * @return Return the initialized point if it was done correctly,
 * otherwise return NULL.
```

```
 */
Point * point_new (int x, int y, char symbol);

/**
 * @brief Destructor. Free the dynamic memory reserved for a point .
 *
 * @param p Point to free
 */
void point_free (Point *p);


/**
 * @brief Gets the x coordinate of a given point.
 *
 * @param Point pointer
 *
 * @return Returns the x coordinate of a given point, or INT_MAX in
 * case of error.
 */
int point_getCoordinateX (const Point *p) ;


/**
 * @brief Gets the y coordinate of a given point.
 *
 * @param Point pointer
 *
 * @return Returns the y coordinate of a given point, or INT_MAX in
 * case of error.
 */
int point_getCoordinateY (const Point *p) ;


/**
 * @brief Gets the symbol of a given point.
 *
 * @param Point pointer
 *
 * @return Returns the x coordinate of a given point, or ERRORCHAR in
 * case of error.
 */
char point_getSymbol (const Point *p) ;


/**
 * @brief Modifies the x coordinate  of a given point.
 *
```

```
 * @param p Point pointer
 * @param x New x coordinate, must be equal or greater than 0
 *
 * @return Returns OK or ERROR in case of error
 */
Status point_setCoordinateX (Point *p, int x) ;


/**
 * @brief Modifies the y coordinate  of a given point.
 *
 * @param p Point pointer
 * @param y New y coordinate, must be equal or greater than 0
 *
 * @return Returns OK or ERROR in case of error
 */
Status point_setCoordinateY (Point *p, int y) ;


/**
 * @brief Modifies the symbol of a given point.
 *
 * @param p Point pointer
 * @param c New symbol, must be a valid symbol
 *
 * @return Returns OK or ERRORCHAR in case of error
 */
Status  point_setSymbol (Point *p, char c) ;

Bool point_getVisited (const Point *p); // DFS (P2)
Status point_setVisited (Point *p, Bool bol);    // DFS (P2)


/**
 * @brief Reserves memory for a point where it copies the data from
 * the point src.
 *
 * @code
 * // Example of use
 * Point *trg, *src;
 * src = point_new();
 * trg = point_hardcpy(src);
 * // .... aditional code ...
 * // free points
 * point_free(src);
 * point_free(trg);
 * @endcode
```

```
 *
 * @param src Original Point pointer
 *
 * @return Returns the pointer of the copied Point if everything
 * went well, or NULL otherwise.
 */
Point *point_hardcpy (const Point *src);


/**
 * @brief Compares two points.
 *
 * @param p1,p2 Points to compare.
 *
 * @return Returns True or False.
 * In case of error, returns FALSE.
 */
Bool point_equal (const void *p1, const void *p2);



/**
 * @brief Prints in pf the data of a point.
 *
 * The format will be: [(x, y): symbol], without line break at the
 * end.
 *
 * @code
 * Point * v;
 * v = point_new ();
 * point_print (stdout, v);
 * @endcode
 *
 * @param pf File descriptor
 * @param v Point to be printed
 *
 * @return Returns the number of characters that have been written
 * successfully. If there have been errors returns -1.
 */
int point_print (FILE *pf, const void *p); // Print Stack

#endif /* POINT_H */
```

**Apéndice 3.**

Contenido del fichero `map.h`:

```
/*
 * File:   map.h
 * Author: profesores
 *
 * Created on 21 de enero de 2022, 15:44
 */

#ifndef MAP_H
#define MAP_H

#include "point.h"

typedef enum {
    RIGHT = 0,
    UP = 1,
    LEFT = 2,
    DOWN = 3,
    STAY = 4,
} Position;

typedef struct _Map Map;


/**
 * @brief  Creates a new empty Map with nrows and ncols.
 *
 * Allocates memory for a new map and initializes it to be empty
 * (no points).
 *
 * @param nrows, ncols Dimension of the map
 *
 * @return A pointer to the graph if it was correctly allocated,
 * NULL otherwise.
 **/
Map * map_new (unsigned int nrows,  unsigned int ncols);


/**
 * @brief Frees a graph.
 *
 * Frees all the memory allocated for the graph, including its points
 *
 * @param g Pointer to graph to be freed.
```

```
 **/
void map_free (Map *);

/**
 * @brief Inserts a point in a map.
 *
 * Insert a point in the map at the coordinates indicated by the
    point.
 * The upper left corner point of the map has (0,0) coordinates.
 *
 * @param mp Pointer to the map.
 * @param p Pointer to the point to be inserted.
 *
 * @return Returns a pointer to the inserted point, NULL otherwise.
 **/
Point *map_insertPoint (Map *mp, Point *p);



/**
 * @brief  Returns the total number of columns
 * in the map.
 *
 * @param mp Pointer to the map.
 *
 * @return Returns The number of columns in the map, or -1 if
 * there is any error.
 **/
int map_getNcols (const Map *mp);

/**
 * @brief  Returns the total number of rows
 * in the map.
 *
 * @param mp Pointer to the map.
 *
 * @return Returns The number of rows in the map, or -1 if
 * there is any error.
 **/
int map_getNrows (const Map *mp);

Point * map_getInput(const Map *mp);
Point * map_getOutput (const Map *mp);

/**
 * @brief  Returns the map point with the same p coordinates.
 *
```

```
 * @param mp Pointer to the map.
 * @param p Pointer to the point
 *
 *
 * @return Returns pointer to the map point , or NULL if
 * there is any error.
 **/
Point *map_getPoint (const Map *mp, const Point *p);


/**
 * @brief  Returns the neighboor of the point p at the position pos
 * in the map.
 *
 * @param mp Pointer to the map.
 * @param p Pointer to the point
 * @param pos Neighboor position relative to the point being
 *  RIGHT = 0, UP = 1, LEFT = 2, DOWN = 3,STAY = 4,
 *
 * @return Returns pointer to the neighboor, or NULL if
 * there is any error.
 **/
Point *map_getNeighboor(const Map *mp, const Point *p, Position pos);

// setters
Status map_setInput(Map *mp, Point *p);
Status map_setOutput (Map *mp,Point *p);

/**
 * @brief Reads a map definition from a text file.
 *
 * Reads a map description from the text file pointed to by pf,
 * and fills the map.
 * This function allocates memory for the map and its points.
 *
 * The file contains a header with the map number of rows and cols.
 * The next file lines corresponds with the points of the map.
 *.
 * For example the file:
 *
 * 4 3
 * ++++
 * +io+
 * ++++
 *
 * Represents a map with 4 rows, 3 columns and the points:
 * [(0, 0): +][(1, 0): +][(2, 0): +][(3, 0): +][(0, 1): +][(1, 1): i]
```

```
 * [(2, 1): o][(3, 1): +][(0, 2): +][(1, 2): +][(2, 2): +][(3, 2): +]
 *
 * @param pf, Pointer to the input stream.
 *
 * @return the map or NULL if there is any error
 */
Map * map_readFromFile (FILE *pf);


/**
 * @brief Compares two maps.
 *
 * @param p1,p2 Pointers to maps to compare.
 *
 * @return Returns True or False.
 * In case of error, returns FALSE.
 */
Bool map_equal (const void *_mp1, const void *_mp2);

/**
 * @brief Prints in pf a map
 *
 * Prints the total number rows and columns and after a new line
 * the map points
 *
 * @code
 * Map * mp;
 *..... //  additional code
 * map_print (stdout, mp);
 * @endcode
 *
 * @param pf File descriptor
 * @param mp map to be printed
 *
 * @return Returns the number of characters that have been written
 * successfully. If there have been errors returns -1.
 */
int map_print (FILE*pf, Map *mp);


Point * map_dfs (Map *mp);  // DFS (P2)

#endif /* MAP_H */
```