Tema 1. Tipos Abstractos de Datos (TAD)

1.0. Contenido y documentación

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/22b28996-7a63-4f43-8c43-bbab6a172b82/U0_RepasoC.pdf

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c25edc87-c9b4-4bd0-9603-b0b9a9ae923c/U1_EstructurasDeDatos.pdf

1.1. Tipos Abstractos de Datos (TAD)

*Definición. Un **Tipo Abstracto de Datos (TAD)** es un conjunto de datos con entidad propia (identidad definida) y un conjunto de operaciones primitivas aplicables sobre esos datos.*

*Definición. Una **Estructura de Datos (EdD)** es un tipo usado para representar los datos en el TAD.*

```
Real real_part;
Real imag_part;
```

Propiedades de un TAD:

- Toda interacción con el TAD debe hacerse a través de las primitivas definidas.
- Cualquier función que opera obre el TAD debe construirse haciendo llamadas a las primitivas.

*Definición. La **interfaz pública** del TAD es el conjunto de funciones primitivas del mismo.*

1.1.1. Diseño e implementación de un TAD

1. **Especificaciones del TAD**: nombre, datos, funciones primitivas y otras funciones (derivadas).
2. **Definición de la EdD**: cómo organizar los datos.
3. **Implementación de las primitivas y funciones derivadas**: dependerá de la EdD elegida.
4. **Documentación**: debe tenerse especial cuidado con la interfaz pública.

```
/* Declaración de EdD */
typedef struct _ComplexNumber ComplexNumber;

/* Cabeceras de funciones de la interfaz del Tad */
ComplexNumber *cn_create (float re, float im);
```

```
void cn_free (ComplexNumber *pc);
float cn_get_real_part (const ComplexNumber *pc);
float cn_get_imag_part (const ComplexNumber *pc);
```

```
#include "complex_number.h"

/* Definición de EdD */
struct _ComplexNumber{
  float re, im;
};

/* Implementación de funciones públicas y privadas */
ComplexNumber *cn_create (float re, float im){
  ComplexNumber *pc = (ComplexNumber *)malloc(sizeof(ComplexNumber));
  if (!pc) return NULL;
  pc->re = re;
  pc->im = im;
  return pc;
}

void cn_free (ComplexNumber *pc){
  free(pc);
}

float cn_get_real_part (const ComplexNumber *pc){
  // Comprobar que PC no es NULL
  return pc->re;
}

float cn_get_real_part (const ComplexNumber *pc){
  // Comprobar que PC no es NULL
  return pc->im;
}
```