

ED Tema 1

pemaga12

June 2020

1. Tipos abstractos de datos

- Tipo de datos = grupo de valores + operaciones definidas sobre valores
- Tipo abstracto de datos (TAD) = tipo de datos donde la representación de valores y la implementación de operaciones son opacas y están ocultas al usuario.
Existen datos predefinidos como por ejemplo: bool, char, int, float, double, string...

Una estructura de datos es una estrategia de almacenamiento en memoria de la información que se quiere guardar. Por ejemplo, un array es una estructura de datos.

2. Conceptos relacionados con TADs

- Tipos representantes: Tipos de implementación concretos que se van a usar para representar el TAD.
- Funcion de abstracción: correspondencia entre la representación y los valores del TAD.
- Invariante de la representación: Condiciones que se deben cumplir para que una representación sea válida.
TAD rectángulo representado por un punto en un plano, su altura y su anchura:
La anchura y la altura han de ser ≥ 0 .
- Tipos de operaciones de los TADs
 - Constructoras: Crean valores del tipo.
 - Observadoras: Devuelven información de los valores del tipo al que pertenece.
 - Mutadoras: Modifican un valor del tipo. Su resultado es un valor del tipo.

- Destructoras: Establecen las condiciones necesarias para que un valor del tipo pueda "desaparecer"
- Parcialidad de las operaciones de un TAD:
Una operación es parcial cuando hay situaciones en las que no está definida y conduce a situaciones de error.
Es por ello que cualquier operación parcial conlleva a precondiciones adicionales que garanticen un comportamiento predecible, y debe estar debidamente comentadas.
También será necesario el manejo de errores.

3. Definición e implementación de un TAD

Pasos para implementar un tad:

- Elegir los tipos de implementación concretos que se usarán para implementar el TAD.
- Dar la interpretación que se va a hacer de sus valores.
- Definir las condiciones que se deben cumplir para que los valores se consideren válidos.
- Decidir cuando dos valores del tipo representante corresponden a un mismo valor del TAD.
- Implementar las operaciones especificadas.

Ejemplo: Tad Rectángulo alineado con los ejes:

- Tipos representantes: Un punto y un par de números reales.
- Funcion de abstracción: El punto representa el extremo inferior izquierdo del rectangulo. Los valores reales: El ancho y el alto. Un rectángulo es vacío si $\text{alto} = \text{ancho} = 0$.
- Invariante de la representación: El ancho y el alto deben de ser valores mayores o iguales que 0.
- Función de equivalencia: Dos rectángulos son iguales cuando tienen el mismo ancho, alto y origen.

4. Constructoras de tipos disponibles en los lenguajes de programación

Las constructoras de tipos (como struct en C) no permiten ocultar los detalles al usuario

```
struct Fecha {  
    int dia; int mes; int anyo;  
};
```


Esto no es positivo, ya que permite un libre acceso a la representación interna del tipo de datos.

Además se permite crear datos que no tengan sentido, como por ejemplo dia = 30; mes = 2;

5. Herramientas de implementación de TADs. Las clases.

- Para que un lenguaje soporte la implementación de TADs, el lenguaje debe contar con mecanismos que permitan separar la especificación de la implementación.
- Las clases de los lenguajes OO están pensadas para permitir que los tipos definidos por el programador reciban el mismo tratamiento que los tipos predefinidos.
- En C++ las clases son las herramientas que se usan para implementar los TADs.
- Declaraciones necesarias para que funcione.
 - Declaración de la clase en el archivo .h (fecha.h en este ejemplo)

```
// includes imprescindibles para que compile el .h  
#include <libreria_sistema>  
#include "modulo_propio.h"  
// protección contra inclusión múltiple  
#ifndef _FECHA_H_  
#define _FECHA_H_  
// ... declaración de la clase Fecha aquí...  
// fin de la protección contra inclusión múltiple  
#endif
```



Documentación del módulo:
en el .h

- Implementación de la clase en el .cpp asociado al .h (fecha.cpp)

```
#include "fecha.h"
// includes adicionales para que compile el .cpp
#include <libreria_sistema>
#include "modulo_propio.h"
// implementación de las operaciones de Fecha ...
```

6. Implementaciones estáticas y dinámicas de TADs

- A veces los valores de los TADs pueden llegar a requerir mucho espacio, o su tamaño puede variar con respecto a la ejecución del programa. En estos casos conviene realizar una implementación dinámica del TAD.
- Memoria dinámica:
 - Se gestiona mediante punteros
 - Reserva y liberación de memoria mediante los operadores new y delete si se trata de reservar y liberar arrays.
- Consejos:
 - Inicializar los punteros nada mas declararlos.
 - Nada más liberar un puntero asignarle el valor NULL.
- Implementación dinámica del TAD punto:

```
class Punto {
    float * _x, * _y;
public:
    // constructor
    Punto(float xx = 0, float yy = 0){
        _x = new float;    *_x = xx;
        _y = new float;    *_y = yy;
    }
    // destructor
    ~Punto() { delete _x; delete _y; }

    // constructor de copia
    Punto (const Punto& p) {
        _x = new float;    *_x = *p._x;
        _y = new float;    *_y = *p._y;
    }
    // sobrecarga del operador de asignación: copia los contenidos
    // (apuntados por atributos)
    Punto& operator= (const Punto& p) {
        *_x = *p._x;
        *_y = *p._y;
        return *this;
    }
    //... resto de operaciones
};
```

7. TADs genéricos

Uno o más de los tipos que se usan en la implementación del TAD se dejan sin especificar.

C++ soporta el desarrollo de código genérico mediante el mecanismo de plantillas ([template](#)).

8. Operaciones parciales: Tratamiento de errores

Existen distintas estrategias de tratamiento de errores:

- Mostrar un mensaje en pantalla y devolver cualquier valor.
- Devolución de valor de error.
 - Requiere que haya un valor de error disponible que no pueda ser confundido con una respuesta de no error, o devolver un booleano indicando el estado de error.
 - El código cliente debe verificar mediante algún condicional si ha habido o no error.
- Lanzar una excepción.
 - Requiere soporte del lenguaje de programación.
 - El código cliente debe decidir si quiere manejar la excepción lanzada.
 - El código cliente que maneja excepciones es más limpio que en el caso anterior.
- Para que las excepciones se recojan se debe llamar a la función desde un bloque try, y tras esto intentar capturar las excepciones (en caso de que se lance alguna) en un bloque catch.