1-Introducción

En este capitulo se presentan los conceptos y definiciones fundamentales que acompañarán al alumno a lo largo de toda la asignatura. Es muy importante fijar estos conceptos ya que de ello depende que la comprensión de esta asignatura sea más o menos compleja.

- **Abstracción**
  - Operación mediante la cual la inteligencia llega a formar conocimiento conceptual común a un conjunto de entidades, separando de ellos los datos contingentes e individuales para atender a lo que los constituye esencialmente. Es decir, aislar mentalmente o considerar por separado las cualidades de un objeto.

1.2-Estructuras de datos y Tipos de Datos Abstractos

- **Tipo de datos**
  - Un tipo de datos es una colección de valores
- **Tipo de Datos Abstracto (TDA)**
  - Es un tipo de datos definido de forma única mediante un tipo y un conjunto dado de operaciones definidas sobre el tipo.
- **Estructura de datos**
  - Implementación física de un tipo de datos abstracto.

Un ejemplos de esto conceptos es el siguiente:

En algunos lenguajes de programación que implementan el tipo **Booleano,** éste está formado por los valores {True, False}, pero no es un tipo de datos abstracto ya que sobre él no se han definido las operaciones para manipularlo. Sin embargo, el tipo Booleano junto con los operadores booleanos (AND, OR, NOT) constituyen un tipo de datos abstracto.

- **Encapsulado**
  - Es consecuencia misma del proceso de abstracción y consiste en “ocultar” la implementación de las operaciones que manipulan los objetos, ofreciendo únicamente una “interfaz” que permita realizar operaciones, funciones de acceso y variables de entrada salida. (Ej. representación enteros en diferentes notaciones, little endian o big endian).

Normalmente existe más de un TDA para resolver un problema concreto. La elección de uno u otro dependerá de algunas características del problema a resolver (disponibilidad de memoria, velocidad de resolución, etc). Ello implica que cada implementación de un TDA presentará un **coste** diferente.
El coste de un TDA no viene determinado por un sólo factor. En primer lugar se deberá analizar el almacenamiento requerido por los datos y la cantidad de ellos que serán manejados. El segundo aspecto a considerar será el coste de las operaciones básicas. En el caso de los árboles, por ejemplo, serán los costes de las operaciones de inserción, búsqueda y eliminación. Para calcular el coste de esas operaciones deberán emplearse técnicas de análisis de algoritmos.

Por tanto, el diseño de un TDA (debido al proceso de abstracción) deberá realizarse siguiendo tres pasos fundamentales:

1. Análisis de datos y operaciones.
2. Elección del Tipo de datos Abstracto.
3. Elección de la Implementación.

En el contexto global de la programación, independientemente del lenguaje utilizado, no suele atenderse a la diferencia entre tipo de dato y TDA. Sin embargo, todos ellos son de hecho TDAs. De este modo la definición de los tipos de datos estándar son las siguientes:

- **Entero**
  - Tiene como tipo el conjunto de números enteros definido por las matemáticas $\{-1,-2,...,\infty\}\cup\{1,2,...,\infty\}$ y como operaciones la suma, resta, multiplicación y división entra.
- **Real**
  - Tiene como tipo el conjunto de números reales definido por las matemáticas y como operaciones la suma, resta, multiplicación y división.
- **Booleano**
  - Tiene como tipo el conjunto de valores booleanos {True, False} y como operaciones las definidas por el álgebra de Boole (AND, OR, NOT).
- **Carácter**
  - Tiene como tipo el conjunto de caracteres definido por un alfabeto dado y como operaciones todos los operadores relacionales (tomando como ejemplo dos variables arbitrarias (a, b) $\{a<b, a>b, a=b, a\leq b, a\geq b, a\neq b\}$

Como ya se ha comentado, la implementación de TDA **Real** tiene diferentes implementaciones, aunque la más utilizada es la recomendada por el IEEE en su norma 754-1985. análogamente, para el TDA **carácter**, la implementación dependerá, básicamente, del alfabeto utilizado. En general, todos los alfabetos vendrán caracterizados por una **secuencia de cotejo** (en ASCII el carácter **A** corresponde a la secuencia de cotejo 65), que no es otra cosa que una secuencia ordenada de los caracteres que constituyen su conjunto. Los tres conjuntos de caracteres más conocidos son:

1. CDC-Científico
2. EBCDIC (*Extended Binary Coded Decimal Interchange Code*)
3. ASCII (*American Standard Code for Information Interchange*)

En este contexto se puede definir el tipo de datos **escalares** que son aquellos en los que *el conjunto de valores está ordenado y cada valor es atómico*.
Algunos tipos de datos escalares presentan una propiedad adicional, la *ordinalidad*. Se denominan ***tipos de datos ordinales*** a aquellos en los que cada valor tiene un único predecesor (excepto el primero) y un único sucesor (excepto el último).
Los tipos *carácter, entero y Booleano* son ordinales, pero el tipo *real* no lo es ya que según sabemos por las matemáticas, los números reales no forman un conjunto ordenado al no ser numerables. Se les puede aplicar operadores relacionales pero por ejemplo de, 5.1<5.2 podríamos concluir, erróneamente, que 5.1 es el predecesor de 5.2 y éste último el sucesor de 5.1, cuando entre ambos números existen infinitos números.

A los tipos de datos que se han visto hasta ahora se les denomina **primitivos** o **predefinidos** debido a que la mayoría de lenguajes de alto nivel los incorporan en sus normas básicas. No obstante, estos tipos de datos no suelen ser suficientes para realizar tratamiento de la información. En ese momento aparecen los *tipos de datos definidos por el usuario*, siendo los característicos de la programación estructurada: el conjunto, el arreglo y el registro. Dado son divisibles en componentes se denominarán ***tipos de datos*** compuestos.

La mayoría de estos tipos de datos son colecciones de componentes organizadas mediante la forma de acceso a cada una de sus componentes individuales. A este tipo de datos se le denomina tipo de datos estructurado.

En general las operaciones asociadas a los tipos estructurados son: *almacenar* y *recuperar* sus componentes individuales. Cuanto todos los elementos de la colección son del un mismo tipo denominado *tipo base, se dice que el tipo es homogéneo,* mientras que cuando no lo son se dice que es *heterogéneo*.

Las siguientes definiciones precisan los TDA compuestos básicos.

- **Conjunto**
  - Colección de elementos tratados con las operaciones *unión, intersección, y diferencia de conjuntos.*
- **Arreglo**
  - Colección homogénea de longitud fija tal que cada una de sus componentes pueden ser accedidas individualmente mediante uno (unidimensional) o varios (multidimensional) índices, que serán de tipo ordinal y que indican la posición de la componente dentro de la colección.
- **Registro**
  - Tipo de datos heterogéneo compuesto por un número fijo de componentes denominadas *campos* a las que se accede mediante un *selector de campo* ***(.)***
- **Matriz**
  - Tiene como tipo el conjunto de matrices definido por las matemáticas y los operadores asociados a las mismas: *Obtener_elemento, Asignar_elemento, Sumar, Restar, Negar, Producto_escalar, Producto_matricial, Determinante, Inversa y Transponer.*
- **Lista o secuencia**
  - Colección homogénea de datos, ordenados según su posición en ella, tal que cada elemento tiene un predecesor (excepto el primero) y un sucesor (excepto el último) y sus operadores asociados son:
    - Insertar, Localizar, Recuperar, Suprimir (elemento p), Suprimir_dato (todas las apariciones de x), Anula (vacía lista), Primero (Fin), Imprimir.
  - En general, el término 'lista' se usa cuando el TDA se implementa sobre memoria principal, mientras que 'secuencia' suele reservarse para implementaciones en memoria secundaria.
  - Como puede observarse, desde el punto de vista lógico, la lista es una estructura de datos dinámica, ya que su tamaño vendrá determinado por el numero de elementos a contener. Una lista puede ser implementada de diversas formas. Con los TDA que se han visto hasta ahora se haría mediante arreglos. Así, la lista, que es dinámica, independientemente de su implementación, se realiza mediante una implementación estática. Sin embargo también es posible implementarla dinámicamente mediante ***punteros.***

1.2.3 Tipos de datos abstractos: Punteros y Listas enlazadas

Un puntero es un *tipo de datos simple* cuyo valor es la dirección de una variable de otro tipo, denominada *variable referenciada*.

La utilidad de los punteros radica en que permiten realizar una reserva dinámica de memoria. Las *variables referenciadas* son variables dinámicas que son creadas y ***destruidas*** en *"tiempo de ejecución"* (recordad los tipos de datos que se pueden declarar en un programa: Const, Type, Var, que ocupan espacio durante toda la existencia del programa).

En la mayoría de los computadores el valor de un puntero es un número entero, sin embargo, este valor no es de tipo entero, sino una dirección, en algunas máquinas puede corresponder a una estructura compleja donde se especifique una dirección base y un desplazamiento.

En Modula2 la declaración de los tipos puntero se realiza en la sección `TYPE` con la siguiente sintaxis:

```
TYPE
      Tipo_puntero = POINTER TO Tipo_variable_referenciada;
```

donde `Tipo_variable_referenciada` es el tipo de datos al que pertenecerán las variables referenciadas. La declaración de variables puntero se realiza en la sección VAR con la siguiente sintaxis:

```
VAR
      variable_puntero : Tipo_puntero;
```

La variable referenciada es accedida con el puntero mediante el *operador apuntador*, cuyo símbolo en Modula2 es el acento circunflejo (^).

La memoria necesaria para albergar las variables puntero es reservada y ocupada de la misma forma que las variables estáticas. Sin embargo, la variable referenciada no existe hasta que no ha sido creada. Para crear una variable referenciada en Modula2 se dispone del procedimiento `Allocate`,cuya sintaxis es:

```
      Allocate(variable_puntero, SIZE(Tipo_variable_referenciada));
```

Este procedimiento realiza dos acciones. En primer lugar toma la memoria libre necesaria para albergar la variable referenciada (determinada por la función `SIZE` que determina el tamaño de la variable referenciada). En segundo lugar, asigna la dirección de memoria que ha reservado al puntero, `variable_puntero`. Tras la ejecución de este procedimiento, la variable referenciada existe en el sentido de que hay espacio de memoria reservado para ella y además, puede ser accedida a través del puntero.

Una vez que una variable referenciada no va a volver a ser utilizada, se destruirá mediante el procedimiento `Deallocate` cuya sintaxis es:

```
      Deallocate(variable_puntero, SIZE(Tipo_variable_referenciada));
```

El ejemplo de la figura 1.3 de la página 14 ilustra estas acciones:

```
MODULE ejemplo;

TYPE

   puntero = POINTER TO integer;

VAR

   p,q: puntero;

BEGIN

   ALLOCATE(p, SIZE(integer));

   p^:=8;

   q=p;

   q^:=5;

   DEALLOCATE(p, SIZE(integer));

END
```

Debe prestarse especial atención a la manipulación de punteros, ya que una acción incorrecta puede llevar a perder la variable referenciada. Por ejemplo si en el programa anterior hubiéramos querido que otra variable referenciada por ***q*** almacenara el valor ***5***, deberíamos haber reservado espacio para ésta. Obsérvese que la acción de igualar q a p (q=p), implica que a partir de ese momento, ambos punteros (p y q) apuntan a la misma variable referenciada y por consiguiente, una acción como p^:=5, tendrá el mismo resultado que q^:=8. Es decir, la variable referenciada por p recibirá el valor 5, pero posteriormente, esa misma variable recibirá el valor 8.

Cuando un puntero no apunta a nada, o deseamos que no apunte a nada, mostrará o deberemos asignar la constante `NIL`.

Como definición final, podemos decir que el TDA puntero es un tipo de datos simple cuyos valores son direcciones de memoria, y sus operadores asociados son la asignación de punteros y la comparación de punteros.

Por último, debe tenerse en cuenta el ahorro de memoria (almacenamiento principal) que significa el utilizar punteros. Los punteros son la base de construcción de las ***estructuras dinámicas,***como los árboles o las listas enlazadas.