Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TEMA 4:
TIPOS ABSTRACTOS DE DATOS (TADs)

TIPOS ABSTRACTOS DE DATOS (TADs)

- Introducción
- Un contraejemplo completo
- Metodología de la programación de TADs
- Elaboración e implementación de un TAD para el ejemplo anterior

INTRODUCCIÓN

- La definición de nuevos tipos de datos a partir de nuevas estructuras de datos representa:
  - Una herramienta útil que permite el uso de variables de dicho tipo siempre que sean necesarias (reutilización) .
  - Un proceso de abstracción de la misma forma que se siguió con las funciones y procedimientos.
- La abstracción de datos (TADs) permite al programador despreocuparse de los detalles menores concentrandose sólo en el uso del tipo y de sus operaciones asociadas.
- Los TADs ponen a disposición del programador un conjunto de objetos junto con sus operaciones básicas que son independientes de la implementación elegida. (todas sus características vienen definidas por su definición y no por su implementación.)

UN CONTRAEJEMPLO COMPLETO

- La representación en Pascal de un conjunto tiene una fuerte limitación en cuanto al valor máximo del cardinal (en TP7.0 es de 256).

- El ejemplo propone representar el tipo abstracto de dato "conjunto" sin esta restricción y poder emplear dicho tipo en la resolución de un algoritmo, la Criba de Eratóstenes, que permite obtener todos los números primos menores a uno dado.

DEFINICIÓN DEL PROBLEMA

- Idea: Disponer de un conjunto inicial de números enteros positivos menores que uno dado del cual se puedan eliminar aquellos que no son primos.

- Una vez detallada la idea se debe decidir que implementación se elige para la representación del conjunto. Para este ejemplo se elige una representación mediante una lista enlazada ordenada ascendentemente, sin repeticiones.

ALGORITMO

- Primera etapa de diseño:

  Leer cota $\in$ N

  Generar el conjunto inicial {2,...,cota}

  Eliminar los números no primos del conjunto

  Escribir los números del conjunto.

- Generar el conjunto inicial...

  Crear un conjunto vacío

  Añadir números a la lista de 2 a cota

- Eliminar los números no primos del conjunto

  Para todo e $\in$ conjunto, entre e y sqrt(cota)

  eliminar del conjunto todos los múltiplos de e.

- Eliminar del conjunto todos los múltiplos de “e”

  coeficiente:=2

  repetir

  eliminar del conjunto mientras (e*coeficiente) menor o igual que cota

  coeficiente:=coeficiente+1

  hasta que e sea mayor o igual que sqrt(cota)

```
PROGRAM CribadeEratostenes;
{Prec. Input=un número entero >=2}
TYPE
    tConjunto=^tNodoEnt;
    tNodoEnt=record
            elem:integer;
            sig:tconjunto
    end; {tNodoEnt}
VAR
    cota,e,coef:integer;
    conjunto,aux,puntprimo,auxelim:tconjunto;
```

```
BEGIN
    writeln('cota: '); {leer cota}
    readln(cota);
    new(conjunto);
    aux:=conjunto;
    aux^.elem:=2;
    for e:=3 to cota do begin
            new(aux^.sig);
            aux:=aux^.sig; {enlazado de los nodos}
            aux^.elem:=e
    end; {for}
    aux^.sig:=nil { valor nil al último nodo de la lista}
    {Eliminación de los números no primos del conjunto}
    {puntero que señala al primer nodo }
    puntprimo:=conjunto
```

```
repeat
        e:=puntprimo^.elem;
        coef:=2;
        aux:=puntprimo;
        while (e*coef≤cota) and (aux^.sig<>nil) do begin
                if (aux^.sig^.elem)<(coef*e) then
                        aux:=aux^.sig
                else if aux^.sig^.elem=coef*e then begin
                        auxelim:=aux^.sig;
                        aux^.sig:=aux^.sig^.sig;
                        Dispose(auxelim);
                        coef:=coef+1
                end; {else if}
                  else if aux^.sig^.elem>coef*e then
                        coef:=coef+1
        end {while}
        puntprimo:=puntprimo^.sig
until (e≥sqtr(cota)) or (puntprimo=nil);
```

```
    {Escribir los números del conjunto}

    aux:=conjunto;
    while (aux<>nil) do begin
            write(aux^.elem:4);
            aux:=aux^.sig
    end; {while}
end. {CribadeEratostenes}
```

APRECIACIONES AL EJEMPLO ANTERIOR

- Los detalles del algoritmo quedan mezclados con los relativos a la representación de la estructura de datos. Esto implica lo siguiente:
  - El código obtenido es complejo (difícil corrección y verificación) por lo que su mantenimiento será innecesariamente costoso
  - la estructura de datos empleada no puede ser reutilizada.
- Todos estos inconvenientes son los que se quieren superar mediante la utilización de los TADs.

METODOLOGÍA DE LA PROGRAMACIÓN DE UN TAD

- Un tipo de dato abstracto se puede definir como un sistema con tres componentes:

  1. Un conjunto de elementos (estructura)

  2. Un conjunto de descripciones sintácticas de operaciones

  3. Un descripción semántica, esto es, un conjunto suficientemente completo de relaciones que especifiquen el funcionamiento de las operaciones.

ESPECIFICACIÓN DE UN TAD

- El lenguaje idóneo para poder definir adecuadamente las especificaciones de un TAD es el matemático.
- La definición sintáctica vendrá dada por la declaración de los encabezamientos de las operaciones.
- Mientras que la semántica se recoge en los comentarios que describen el comportamiento de las operaciones sin ningún tipo de ambigüedad.

IMPLEMENTACIÓN DE UN TAD

- Para la implementación del TAD es fundamental disponer de mecanismos que permitan ENCAPSULAR, el tipo de dato y sus operaciones, y OCULTAR la información al programador usuario.

- Estas dos características son fundamentales para poder llevar a cabo esta abstracción.

  - Encapsulamiento implica reutilización,
  - Ocultación implica no manipulación y por tanto un mejor mantenimiento del sistema.

- Turbo Pascal 7.0 provee de unidades que permiten, en cierta medida, cumplir con ambas premisas.

DESARROLLO DE PROGRAMAS CON TADs

- El primer paso a seguir es definir un TAD (colección de objetos junto con el conjunto de operaciones definidas para dichos objetos).

- Una vez definido el TAD y sin necesidad de conocer como ha sido implementado ya se puede utilizar de una forma abstracta, siendo suficiente la información suministrada por su definición.

ETAPAS EN LA ELABORACIÓN DE UN TAD

- Reconocimiento de los objetos candidatos a elementos del nuevo tipo de datos
- Identificación de las operaciones del tipo de datos.
- Especificación de las operaciones de forma precisa, sin ambigüedad.
- Selección de una buena implementación

DESARROLLO DEL TAD PARA EL EJEMPLO ANTERIOR

TYPE

tConj=Abstracto

{los objetos del TAD son los conjuntos con un número arbitrario de números enteros}

- Estos objetos se podrán manipular con las siguientes operaciones:
  - **Crearconj(conj):**crea un conjunto vacío
  - **Annadirelemconj(elem,conj):** añade un nuevo elemento al conjunto.
  - **Quitarelemconj(elem,conj):** quita un elemento del conjunto.

DESARROLLO DEL TAD PARA EL EJEMPLO ANTERIOR

- **Pertenece(elem,conj):** determina si el elemento se encuentra o no en el conjunto.
- **Escribirconj(conj):** muestra todos los elementos de un conjunto.
- **Estarvacioconj(conj):** determina si el conjunto está vacío o no.

ESPECIFICACIÓN PRECISA DE LA OPERACIONES

PROCEDURE Crearconj(var conj: tconj);

{Efecto. Conj:=Ø}

PROCEDURE Annadirelemconj(elem:integer; var conj:tconj);

{Efecto. Conj:=conj ∪ [elem]}

PROCEDURE Quitarelemconj(elem:integer; var conj:tconj);

{Efecto. Conj:=conj / [elem]}

FUNCTION Pertenece(elem:integer; conj:tconj):boolean;

{Dev. True si (elem∈Conj) y False en otro caso}

PROCEDURE Escribirconj (conj:tconj);

{Efecto. Escribe en el output los elementos del conjunto}

FUNCTION Estarvacioconj(conj:tconj):boolean;

{Dev. True si Conj:=∅ o False en caso contrario}

CATEGORIAS DE LAS OPERACIONES APLICABLES A UN TAD

- **Operaciones de creación**: (constructoras o primitivas). Obtienen nuevos objetos del tipo (crearconj).
- **Operaciones de consulta:** Realizan funciones tal que empleando un argumento del tipo devuelven un valor de otro tipo (pertenece, estarvacioconj) o tareas frecuentes (Escribirconj).
- **Operaciones de modificación:** Permiten modificar un objeto del TAD (annadir y eliminar)
- **Operaciones propias del tipo:** Operaciones propias del tipo que se emplean en las anteriores operaciones descritas.

IMPLEMENTACIÓN DEL TAD CONJUNTO

```
UNIT conjent;
{Implementación mediante listas enlazadas, sin cabecera,
 ordenados ascendentemente y sin repeticiones}
INTERFACE
TYPE
    tElem:integer;
    tconj=^tNodolista;
    tNodolista=record
            info:tElem;
            sig:tconj;
    end; {tNodolista}
```

IMPLEMENTACIÓN DEL TAD CONJUNTO

PROCEDURE Crearconj(var conj: tconj);

{Efecto. Conj:=Ø}

PROCEDURE Annadirelemconj(elem:integer; var conj:tconj);

{Efecto. Conj:=conj ∪ [elem]}

PROCEDURE Quitarelemconj(elem:integer; var conj:tconj);

{Efecto. Conj:=conj / [elem]}

FUNCTION Pertenece(elem:integer; conj:tconj):boolean;

{Dev. True si (elem∈Conj) y False en otro caso}

PROCEDURE Escribirconj (conj:tconj);

{Efecto. Escribe en el output los elementos del conjunto}

FUNCTION Estarvacioconj(conj:tconj):boolean;

{Dev. True si Conj:=∅ o False en caso contrario}

```
IMPLEMENTATION
PROCEDURE Crearconj(var conj: tconj);
Begin
    Listavacia(conj)
End; {Crearconj}

PROCEDURE Annadirelemconj(elem:integer; var conj:tconj);
VAR
    insertar:boolean;
Begin
            if  localiza(elem,conj)=nil then begin
                        insertar:=true
                else
                        insertar:= false;
            if insertar then
                    Inserorden(elem,conj)
End; {Annadirelemconj}
```

```
PROCEDURE Quitarelemconj(elem:integer; var conj:tconj);
VAR
    quitar:boolean;
Begin
        if  localiza(elem,conj)=nil then begin
                    quitar:=false
            else
                    quitar:= true;
        if quitar then
                Suprime(elem,conj);
End; {Quitarelemconj}

FUNCTION Pertenece(elem:integer; conj:tconj):boolean;
Begin
        Pertenece:=(Localiza(elem,conj)<>nil);
End; {Pertenece}
```

```
PROCEDURE Escribirconj (conj:tconj);
Begin
    if Estavacioconj(conj) then
        write('[ ]');
    else begin
        write('[',);
        Visualiza(conj);
        writeln(']')
    end; {else}
End; {Escribirconj}

FUNCTION Estarvacioconj(conj:tconj):boolean;
Begin
    Estarvacioconj:=Esvacia(conj)
End; {Estarvacioconj}
```

\+ todas las operaciones de la lista enlazada que se emplean para definir las operaciones públicas del nuevo tipo (operaciones propias del tipo).

```
End. {ConjEnt}
```

- Ahora sólo necesitamos cambiar en nuestro programa la declaración:

```
TYPE
    tconj: Abstracto

por:

USES
    ConjEnt
```

- Que hace que el tipo tconj declarado en la unidad esté disponible para su uso.

```
PROGRAM CribadeEratostenes;
{Prec. Input=un número entero >=2}
USES
    Conjent;
VAR
    cota,e,coef:integer;
    conjunto:tconj;
BEGIN
    writeln('cota: '); {leer cota}
    readln(cota);
    Crearconj(conjunto);
    {Generar el conjunto inicial}
```

```
    for e:=2 to cota do begin
        Annadirelemconj(e,conjunto);
    for e:=2 to Trunc (sqrt(cota)) do
        if  pertenece(e,conjunto) then begin
            coef:=2;
            repeat
                Quitarelemconj(e*coef,conjunto);
                coef:=coef+1;
            until (e*coef>cota)
        end; {if}
    Escribirconjunto(conjunto)
End. {CribadeEratostenes}
```