

# Array con tope y Registros variantes

Programación 1

InCo - FING

## Array con tope

# ¿Qué es y para qué se usa?

- Un **array con tope** es una estructura *conceptual* que se define en términos de los tipos estructurados registro y arreglo.

## Type

```
ArrTope = record
    elems : array [1..N] of T;
    tope   : 0..N
end;
```

- El campo *tope* lleva el control de la cantidad de elementos **válidos** almacenados en el arreglo y toma valores entre 0 y N.
- Los elementos válidos se almacenan al principio del arreglo desde la posición 1 hasta la posición del tope.
- Cuando el tope vale 0 significa que la estructura no contiene elementos.

## ¿Qué es y para qué se usa? (2)

- Lo que está después del tope **no tiene significado** para la lógica del programa.
- Array con tope **no** es un tipo de dato de Pascal, ni de Free Pascal, como lo son los arreglos o los registros. No figura como tema en el libro del curso.

## ¿Qué es y para qué se usa? (2)

- Lo que está después del tope **no tiene significado** para la lógica del programa.
- Array con tope **no** es un tipo de dato de Pascal, ni de Free Pascal, como lo son los arreglos o los registros. No figura como tema en el libro del curso.
- Puede llegar a verse como una estructura **dinámica** ya que se le puede agregar o quitar elementos a lo largo de la ejecución del programa.

## ¿Qué es y para qué se usa? (2)

- Lo que está después del tope **no tiene signifcado** para la lógica del programa.
- Array con tope **no** es un tipo de dato de Pascal, ni de Free Pascal, como lo son los arreglos o los registros. No figura como tema en el libro del curso.
- Puede llegar a verse como una estructura **dinámica** ya que se le puede agregar o quitar elementos a lo largo de la ejecución del programa.
- Pero no es *dinámica* porque esto se da dentro de una estructura **estática** (un record que contiene un arreglo de un cierto tamaño) que establece en su declaración el espacio máximo que va a utilizar.
- Estructuras dinámicas serán vistas cuando veamos punteros.

## Ejemplo: conjuntos

Un posible uso de array con tope es para la representación de conjuntos:

Type

```
Conj = record
    elems : array [1..N] of T;
    tope   : 0..N
end;
```

Se requiere que los valores de tipo T sean comparables por igualdad.

# Cómo funciona

- El array `elems` almacena los elementos del conjunto.
- Los elementos se almacenan en las celdas 1 a `tope`.
- El campo `tope` apunta a la última posición “ocupada” del array.
- El conjunto vacío se representa mediante el campo `tope` en 0.
- Pueden representarse conjuntos que contengan de **0** a **N** elementos.



# Operaciones sobre conjuntos

A continuación veremos la implementación de varias operaciones sobre la representación de conjuntos.

```
type
  Conj = record
    elems : array [1..N] of T;
    tope  : 0..N
  end;
```

# Operaciones sobre conjuntos

A continuación veremos la implementación de varias operaciones sobre la representación de conjuntos.

```
type
    Conj = record
        elems : array [1..N] of T;
        tope   : 0..N
    end;
```

Crear conjunto vacío.

```
procedure CrearConjuntoVacio(var S : Conj);
begin
    S.tope := 0
end;
```

# Insertar un elemento

# Insertar un elemento

```
procedure Insertar(e : T; var S : Conj);  
{ pre-condicion: (e no pertenece a S) y (S.tope < N)}  
begin  
  with S do  
    begin  
      tope := tope + 1;  
      elems[tope] := e  
    end  
  end;  
end;
```

# Eliminar un elemento

Si el elemento a eliminar no está el procedimiento no hace nada.

# Eliminar un elemento

Si el elemento a eliminar no está el procedimiento no hace nada.

```
procedure Eliminar(e : T; var S : Conj);  
var i : integer;  
begin  
    i := 1;  
    { evaluacion por circuito corto }  
    while (i <= S.tope) and (S.elems[i] <> e) do  
        i := i + 1;  
  
    if i <= S.tope then  
        begin  
            S.elems[i] := S.elems[S.tope];  
            S.tope := S.tope - 1  
        end  
    end;  
end;
```

# Búsqueda de un elemento

```
function pertenece(e : T; S : Conj) : boolean;
var
    i: integer;
begin
    i:= 1;
    { evaluacion por circuito corto }
    while (i <= S.tope) and (S.elems[i] <> e) do
        i:= i+1;

    pertenece:= i <= S.tope

end;
```

Notar que nunca se itera más allá del tope.

## Registros variantes



- Se quiere representar una entidad que puede pertenecer a diferentes categorías.
- Según la categoría hay diferentes datos.
- Existe un conjunto de datos que son comunes a todas las categorías (eventualmente vacío).

# Ejemplo

Categorías:

- Estudiante:
  - Año de Ingreso
  - Cantidad de materias
- Docente
  - Carga horaria
  - Grado
- Egresado
  - Año de egreso
  - Título

Datos comunes a todas las categorías:

- Cédula.
- Credencial cívica.

## Ejemplo (cont)

type

```
TOrden = (docente,estudiante,egresado);
```

```
TUniversitario = record
```

```
    cedula      : TCedula;
```

```
    credencial  : TCredencial;
```

```
    case orden : TOrden of
```

```
        docente : (grado: 1..5;  
                   carga: 0..40);
```

```
        estudiante : (semestre: 1..15;  
                     materias: integer);
```

```
        egresado : (egreso: 1900..3000;  
                   titulo: TTitulo);
```

```
end;
```

## Ejemplo: Figuras

```
type RGBColor = record
    red,green,blue : 0..255;
end;
punto = record
    x,y: real;
end;
TipoFigura = (circulo,cuadrado,rectangulo);
figura = record
    color: RGBColor;
    case clase: TipoFigura of
        circulo      : (radio: real;
                        centro: punto);
        cuadrado     : (lado: real;
                        verticeSupIzq: punto);
        rectangulo   : (base,altura: real;
                        verticeInfIzq: punto);
    end;
```

## Creando una figura

Para crear una figura se deben asignar los campos correspondientes de acuerdo a la categoría.

(\* creación de un rectángulo \*)

```
r.color.red := 15;
```

```
r.color.green:= 121;
```

```
r.color.blue:= 203;
```

```
r.clase:= rectangulo;
```

```
r.base:= 12.4;
```

```
r.altura:= 345.90;
```

```
r.verticeInfIzq.x:= 0.9;
```

```
r.verticeInfIzq.y:= 19.78;
```

# Área de una figura

El campo discriminante es el que permite identificar cada categoría.

```
function areaFigura(fig : Figura): real;
begin
  with fig do begin
    case clase of
      circulo      : areaFigura := PI * sqr(radio);
      rectangulo   : areaFigura:= base * altura;
      cuadrado     : areaFigura:= sqr(lado);
    end; { case }
  end; { with }
end; {areaFigura}
```

# Comparación de posición

La función `masArriba` determina si una figura dada está completamente por encima de un cierto valor del eje de las ordenadas.

```
function masArriba(fig: figura; alt: real): boolean;
begin
  with fig do begin
    case clase of
      circulo      : masArriba:= (centro.y - radio) > alt;
      rectangulo   : masArriba:= verticeInfIzq.y > alt;
      cuadrado     : masArriba:= (verticeSupIzq.y - lado) > alt;
    end { case }
  end { with }
end; {masArriba}
```