# Definición de tipos en Pascal

- Hay **tres formas** de asignar un tipo en Pascal:
  - Declaración **`const`**
    - Ejemplo: **`const`**

```
pi=3.1415926;
```

  - Declaración **`var`**:
    - Ejemplo: **`var`**

```
a,b,c:integer;
```

  - Declaración **`type:`**
    - Permite definir un nuevo tipo de datos y darle nombre con un identificador
    - Ejemplo: **`type`**

```
dia,mes=integer;
var
hoy,manyana:dia;
inicio_cuatrimestre:mes;
```

# Tipos de datos en Pascal

- En Pascal los **tipos** pueden ser:
  - Tipos simples:
    - Tipos que proporciona el Pascal:
      - entero (**`integer`**), real (**`real`**), carácter (**`char`**), lógico (**`boolean`**)
    - Definidos por el usuario:
      - enumerativo y subrango
  - Tipos estructurados:
    - Arreglos (**`array`**), registros (**`record`**), fichero (**`file`**), conjunto (**`set`**)
  - Tipo cadena
  - Tipo puntero
- Ya se han visto los tipos simples predefinidos, presentaremos a continuación algunos de los tipos restantes.

# Tipos simples definidos por el usuario: enumerados (1)

- Tipos enumerados
  - Sirven para crear **nuevos tipos ordinales**.
  - Para definir un tipo enumerado, se listan los identificadores que pueden tomar los datos de este tipo.
- Declaración:
  - **type**

    *nombre_tipo* = (*constante1*,*constante2*,...,*constanteN*);
  - Ejemplo :

```
type
     dias_semana = (lunes,martes,miercoles,jueves,
                    viernes,sabado,domingo);
var
     dias : dias_semana;
```

  - El **propósito** principal de los tipos enumerados es permitir al programador el uso de **nombres con significado** en los datos.
  - No pueden utilizarse **directamente** por los procedimientos de entrada/salida.

# Tipos simples definidos por el usuario: enumerados (2)

- Ejemplo de tipo enumerado:

```
program dias_semana; (* muestra los dias de la semana con tipos enumerados *)
type
   dia_semana = (lunes,martes,miercoles,jueves, viernes,sabado,domingo);
var
   dias :dia_semana;
begin
   for dias:=lunes to domingo do
     case dias of
               lunes :writeln('lunes ');
               martes :writeln('martes ');
               miercoles :writeln('miercoles');
               jueves :writeln('jueves ');
               viernes :writeln('viernes ');
               sabado :writeln('sabado ');
               domingo :writeln('domingo ')
     end;
end.
```

# Tipos simples definidos por el usuario: subrango (1)

- Tipo subrango
  - Un tipo subrango se define a partir de un tipo ordinal, especificando dos constantes de ese tipo, que actúan como **límite inferior y superior** del conjunto de datos de ese tipo
  - El tipo subrango es un tipo ordinal y sus valores se ordenan de igual modo que en el tipo del que se deducen.
- Declaración:

```
type nombre = límite inferior .. límite superior;
```

- Ejemplos:
  - `type digito = 0 .. 9;`
    - consta de los elementos 0,1,2,3,4,5,6,7,8,9
  - `type digito_caracter = '0' .. '9';`
    - este subrango consta de los caracteres '0' a '9'

# Tipos simples definidos por el usuario: subrango (2)

- Ejemplos de tipo subrango

```
program dias_semana; (* muestra los dias de la semana con tipos enumerados *)
type
    dia_semana = (lunes,martes,miercoles,jueves, viernes,sabado,domingo);
    dia_trabajo = lunes..viernes;
    dia_descanso= sabado..domingo;
var
    dias :dia_semana;
    diat: dia_trabajo;
    diad: dia_descanso;
begin
    writeln('Dias de trabajo:');
    for diat:=lunes to viernes do
      case diat of
                  lunes :writeln('lunes ');
                  martes :writeln('martes ');
                  miercoles :writeln('miercoles');
                  jueves :writeln('jueves ');
                  viernes :writeln('viernes ');
      end;
    writeln('Dias de descanso:');
    for diad:=sabado to domingo do
      case diad of
                  sabado :writeln('sabado ');
                  domingo:writeln('domingo');

      end;
end.
```

# Tipos de datos estructurados

- Datos estructurados:
  - Mediante **un identificador común** se pueden representar **múltiples datos individuales**, pudiendo cada uno de ellos ser referenciados separadamente.
- Tipos de datos estructurados:
  - El tipo arreglo (array)
    - representa un conjunto de datos del mismo tipo
  - El tipo registro (record)
    - representa un conjunto de datos no necesariamente del mismo tipo
  - El tipo conjunto (set)
    - representa un conjunto ordenado de datos del mismo tipo
  - El tipo fichero (file)
    - representa una colección de datos del mismo tipo que pueden almacenarse en un dispositivo auxiliar de almacenamiento (disco)

# Tipos estructurados: arrays (1)

- Arrays
  - representa un **conjunto de datos del mismo tipo**.
- Arrays unidimensionales:
  - representan el concepto matemático de vector

$$\text{lista} = \begin{pmatrix} \text{lista}[1] \\ \text{lista}[2] \\ \dots \\ \text{lista}[n] \end{pmatrix}$$

- Declaración de un array unidimensional:
  - **var** *nombre_del_array*: **array** [*tipo_de_índice*] **of** *tipo*
  - El tipo de índice puede ser cualquiera de los siguientes datos simples: entero, char, booleano, enumerativo o subrango.
  - El tipo del array puede ser cualquiera, incluyendo cualquier tipo estructurado (por ejemplo otros arrays).

# Tipos estructurados: arrays (2)

- Ejemplos de declaración de arrays

```
var lista : array [1..100] of real;

type indice = 1..100;
var lista : array [indice] of real;

type color = (rojo,blanco,azul);
var indice:color
valores: array [indice] of integer;
```

- Referenciación de elementos en arrays unidimensionales:
  - Cada elemento del array (cada uno de sus datos) se **referencia** mediante el nombre del array seguido de su índice encerrado entre corchetes.
  - Ejemplos:
    - lista [33]; (es un real), valores [blanco]; (es un entero)

# Tipos estructurados: arrays (3)

- Arrays multidimensionales
  - representan el concepto matemático de **matriz**

$$\text{tabla} = \begin{pmatrix} \text{tabla}[1,1] & \text{tabla}[1,2] & \dots & \text{tabla}[1,n] \\ \text{tabla}[2,1] & \text{tabla}[2,2] & \dots & \text{tabla}[2,n] \\ \dots & \dots & \dots & \dots \\ \text{tabla}[m,1] & \text{tabla}[m,2] & \dots & \text{tabla}[m,n] \end{pmatrix}$$

- Declaración de un array multidimensional:
  - **var** *nombre_del_array*: **array** [*tipo_de_índice1, tipo_de índice2 ,... , tipo_de_índiceN*] **of** *tipo;*
  - El tipo de índice puede ser cualquiera de los siguientes datos simples: entero, char,booleano, enumerativo o subrango.
  - El tipo del array puede ser cualquiera, incluyendo cualquier tipo estructurado (por ejemplo otros arrays).

# Tipos estructurados: arrays (4)

- Ejemplos

```
var tabla : array [1..60,1..150] of real;

type indice1 = 1..60;
     indice2=1..150;
var tabla : array [indice1,indice2] of real;

type color = (rojo,blanco,azul);
var indice:color;
valores: array [1..100,indice] of boolean;
```

- Referenciación de elementos en arrays multidimensionales
  - Cada elemento del array (cada uno de sus datos) se **referencia** mediante el nombre del array seguido de sus índices separados por comas y encerrados entre corchetes.
  - Ejemplos:
    - tabla [33,7]; (es un real), valores[blanco,3]; (es un boolean)

# Tipos estructurados: arrays (5)

- Ejemplo:

```
program producto_escalar; (*Calcula el producto escalar de dos vectores)
const
  tamanyo=5
var
  vector1,vector2 :array [1..tamanyo] of real;
  i: integer;
  resultado: real;
begin
  writeln('Introduzca el primer vector');
  for i:=1 to tamanyo do
    readln(vector1[i]);
  writeln('Introduzca el segundo vector');
  for i:=1 to tamanyo do
    readln(vector2[i]);

  resultado:=0

  for i:=1 to tamanyo do
    resultado:=resultado+vector1[i]*vector2[i];

  writeln('El resultado es ',resultado);
  end.
```

# Tipos estructurados: registros (1)

- Registros
  - Representa un conjunto de datos **no necesariamente del mismo tipo**. Los elementos individuales se llaman **campos** del registro
- Declaración de un registro:

```
type
  nombre_reg=   record
                        campo1;
                        campo2;
                        ...;
                        campoN
                end;
var
  nombre_var:nombre_ reg;
```

o de forma equivalente:

```
var nombre_var: record campo1;campo2;...;campoN end;
```

cada declaración de campo es similar al de una variable individual:

```
variable:tipo
```

# Tipos estructurados: registros (2)

- Ejemplo:

```
type cuenta=record
                    numcliente:integer;
                    saldocliente:real
          end;
var cliente:cuenta;
```

- Cada campo de un registro puede ser una variable de cualquier tipo incluyendo otros tipos estructurados. Por ejemplo:

```
    type tarjeta= (visa,4b,clavecard);
           texto=array [1..80 ] of char;
           fecha=record
                    mes:1..12;
                    dia:1..31;
                    anyo:1900..2100
                 end;
          cuenta=record
                    nombrecliente:texto;
                    tarjetacliente:tarjeta;
                    saldocliente:real;
                    ultimopago:fecha
                end;
  var clientes=array[1...9999 ] of cuenta; clientepreferido:cuenta;
```

# Tipos estructurados: registros (3)

- Referenciación de elementos en registros
  - La forma de **acceder** a un elemento de un registro es:

    *nombre_de_variable.campo*
  - Ejemplo:

    `clientepreferido.saldocliente` (es un real)

    `clientepreferido.nombrecliente[0]` (es un char)

    `clientes[33].tarjetacliente` (de tipo tarjeta)

    `clientes[50].ultimopago.dia` (de tipo entero)
- Asignación de registros:
  - Dos registros del mismo tipo pueden **copiarse** íntegramente mediante una sentencia de asignación.
  - Ejemplo:

```
clientes[51]:=clientes[89];
```

# Tipos estructurados: registros (4)

- La estructura with:
  - Se utiliza para **facilitar** el trabajo con los registros al evitar la repetición de su nombre.
  - Forma de la sentencia:

```
with nombre_de_variable do
  sentencia
```

  - Ejemplo:

```
with clientepreferido do
begin
  nombrecliente:='José Sanchez';
  tarjetacliente:=visa;
  saldocliente:=100.000;
  with ultimopago do
  begin
    dia:=12;
    mes:=2;
    anyo:=2000
  end;
end;
```

# Tipos estructurados: registros (5)

- Ejemplo:

```
program ejemplo_record;
type
   datos_alumno=
         record
               inicial_nombre:char;
               inicial_primer_apellido:char;
               inicial_segundo_apellido:char;
               edad:integer;
          end;
var
   estudiante : datos_alumno;
begin
   estudiante.inicial_nombre:='G';
   estudiante.inicial_primer_apellido:='B';
   estudiante.inicial_segundo_apellido:'P';
   estudiante.edad:=18
   writeln('El estudiante ', estudiante.inicial_nombre,
   estudiante.inicial_segundo_apellido, estudiante.inicial_primer_apellido,
   'tiene ', estudiante.edad, 'años')
end.
```

# Tipos estructurados:conjuntos (1)

- Conjuntos
  - Representa un conjunto **ordenado** de datos todos del **mismo tipo**
- Declaración de conjunto:

```
type
  base=( dato1,dato2,...,datoN);  o también
  base=dato1..datoN;
  tipo_conjunto= set of base;
var
  nombre_de_variable:tipo_conjunto;
```

  - Ejemplo:

```
type
  ingredientes=(queso,jamon,tomate,atun,gambas,champinyones);
  pizza=set of ingredientes;
var
  margarita,marinera:pizza;

type minusculas = set of 'a'..'z'
type digitos = set of '0'..'9'
```

# Tipos estructurados:conjuntos (2)

- Asignación en conjuntos:
  - La forma de **asignar** un conjunto es

    *nombre_de_variable*:=[*elemento1*,..., *elementoN*];
  - Ejemplo:

    ```
    margarita:=[queso,jamon,tomate];
    marinera:=[queso,jamon,tomate,atun,gambas];
    ```
- Operaciones con conjuntos:
  - Operaciones algebraicas: Requiere conjuntos del mismo tipo y el resultado es también del mismo tipo
    - Unión: Se utiliza el operador +.
      - Ejemplo: `marinera:=margarita+[atun,gambas];`
    - Intersección: se utiliza el operador *
      - Ejemplo: `basepizza:=margarita*marinera;`
    - Diferencia: se utiliza el operador -
      - Ejemplo: `ingredientesadicionales=marinera-margarita;`

# Tipos estructurados:conjuntos (3)

- Operaciones con conjuntos:
  - Operaciones de **relación**:Se utilizan para formar expresiones de tipo booleano.
    - Hay cuatro tipos de relaciones: Igualdad (=), Desigualdad (<>), Inclusión (<= y >=)
    - Ejemplo:

      `margarita=marinera (false)  margarita<>marinera (true)`

      `margarita<=marinera (true)  margarita >=marinera (false)`

      `[]<=margarita (true)`

  - Operaciones de **pertenencia**: Sirven para comprobar la pertenencia a un conjunto:
    - Se utiliza la sentencia *`elemento`* **`in`** *`conjunto`*
    - Ejemplo:

      `queso` **`in`** `marinera (true)`

      `atun` **`in`** `margarita (false)`

# Tipos estructurados:conjuntos (4)

- Ejemplo:

```
program ejemplo_conjunto;
type
   conjunto_numeros = set of 1..10;
var
   misnumeros : conjunto_numeros;
   valor : integer;
   begin
     misnumeros := [2..6];
     valor := 1;
     while( valor <> 0 ) do
     begin
               writeln('Introduzca un número, (0 salir)');
               readln( valor );
               if valor <> 0 then
               begin
                        if valor in misnumeros then
                                 writeln('Está en el conjunto')
                        else
                                 writeln('No está en el conjunto')
               end
     end
end.
```

# Tipos cadena:strings (1)

- Strings
  - Es un tipo que proporcionan algunas versiones de Pascal para trabajar con **cadenas de caracteres.**
- Declaración de strings
  - `var nombre_variable:string[numero_caracteres];`
  - El máximo número de caracteres permitido en un string es 255.
  - Ejemplo:`var titulo:string [80];`
- Asignación de cadenas:
  - Se utiliza *`nombre_variable:=cadena_de_caracteres;`*
  - Ejemplo:

```
titulo:='Programación en Pascal'
```

- Operadores con cadenas:
  - Operaciones algebraicas
    - Concatenación: Se utiliza el operador +
      - Ejemplo: `titulo :='Programación '+ 'en '+'Pascal';`

# Tipos cadena:strings (2)

- Operaciones de relación:
  - Hay cuatro tipos de relaciones: Igualdad (=), Desigualdad (<>) y comparación (<= y >=). El resultado se basa en los códigos ASCII de los caracteres.
  - Ejemplo: `'A'<'B'` (`true`) `'A'<'a'` (`true`) `'A'>'b'` (`false`)
    `'ABC'<'ABCDEF'` (`true`) `'ABD'>'ABC'` (`true`)
- Procedimientos y funciones:
  - `delete(cad,ind,n)`
    - Suprime n caracteres de la cadena a partir de ind
    - Ejemplo: `str='cadena' delete(str,3,1)='caena'`
  - `insert(cadfuente,caddest,ind)`
    - Inserta caddest en cadfiente a partir de ind.
    - Ejemplo: `str1='micadena' str2='nueva '`
      `insert(str1,str2,4) = 'mi nueva cadena'`

# Tipos cadena:strings (3)

- `str(valor,cadena)`
  - Convierte un valor numérico en una cadena de caracteres.
  - Ejemplo: `str(22,cadena)` hace `cadena='22'`
- `val(cadena,variable,coderror)`
  - Convierte una cadena de caracteres en un número y devuelve en coderror (0=correcto 1=error).
  - Ejemplo: `val('22',variable,coderror)` hace `variable=22` y `coderror=0`
- `concat(cadena1,cadena2)`
  - Une las dos cadenas.
  - Ejemplo: `concat('dul','ce')='dulce'`
- `copy(cadena,pos,num)`
  - Genera la subcadena de cadena con num caracteres a partir de pos.
  - Ejemplo: `copy('bueno',3,2)='en'`

# Tipos cadena:strings(4)

- lenght(cadena)
  - Número de caracteres de la cadena.
  - Ejemplo `length('hola')` es igual a 4.
- pos(cadena1,cadena2)
  - Posición de la primera ocurrencia de cadena1 en cadena2. Ejemplo: `pos('ar','naranja')` es igual a 2.