Programación en Pascal. Definición de tipos y registros

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Noviembre 2020

Contenidos

Nuevos tipos datos

Ya conocemos los tipos de datos primitivos básicos en Pascal: integer, real, boolean, char, string

- Una variable integer (entera) contendrá un dato de tipo integer, serán *literales* como p.e. 2, 17 o -1250
- Una variable real contendrá un dato de tipo real, con literales como 4.3909999999999997E+001 o -5.7294910000000003E+004
- Una variable boolean contendrá un dato boolean, con los literales TRUE o FALSE
- Una variable de tipo char (carácter) caracter contendrá datos de tipo char, con literales como 'V', '@' o '3'
- Una variable de tipo string (cadena) contendrá un dato de tipo string, con literales como 'hola, mundo' o 'x'

Pero podemos definir nuevos tipos de datos para manejar en el programa entidades abstractas propias de nuestro problema

- Tipos enumerados
  Contendrán literales con sentido en el universo de nuestro problema, como *sota*, *caballo*, *rey*
  o
  despegue, *ascenso*, *descenso*

Tipos Enumerados

Definimos un nuevo tipo a base de enumerar sus elementos, que serán literales especificados por nosotros, literales propios del dominio de nuestro problema

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program enumerados_01;
type
    TipoDiaSem = ( lun, mar, mie, jue, vie, sab, dom );

    TipoCarta = ( uno, dos, tres, cuatro, cinco, seis, siete,
        sota, caballo, rey );

    TipoPalo = ( oros, bastos, copas, espadas );
var
    carta: TipoCarta;
const
    No_laborable : TipoDiaSem = dom;
begin
    carta := uno;
    writeln(carta);   // Escribe  uno
    writeln(No_laborable);   // Escribe dom
end.
```

Como sabes, la declaración de una variable es p.e.

```
var
    x : real;
```

La definición de un nuevo tipo es p.e.

```
type
    TipoFigura = (sota, caballo, rey);
```

- La definición empieza tras la palabra reservada *type*. Sin ';'
- Es conveniente poder distinguir fácilmente los identificadores que correspondan a nuevos tipos de datos, en este curso estableceremos el convenio de que el nombre del tipo empiecen por la palabra *Tipo* y con *NotacionCamello*
  - Juntar palabras sin guiones ni barras bajas, poniendo en mayúscula la primera letra de cada palabra
- Después del nombre de tipo se escribe '=', ni ':' ni ':='
- Definimos el tipo enumerando todos sus elementos, entre paréntesis y finalizando cada uno en ';'

- Hemos visto que el número 3 y el caracter '3' son distintos, aunque se vean iguales en pantalla
- Por el mismo motivo, la cadena *oros* y el TipoPalo *oros* son distintos, aunque se vean iguales en pantalla

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program enumerados_02;
type
    TipoPalo = ( oros, bastos, copas, espadas );
var
    palo: TipoPalo;
    s: string;
begin
    palo := oros;
    s := 'oros';
    writeln(palo)  ;  // oros
    writeln(s);     // oros

  { palo := 'oros'; ¡ERROR!  palo es de TipoPalo, no string   }
  { s := oros;      ¡ERROR!  s es de tipo string, no TipoPalo }
end.
```

Hay lenguajes que no tienen tipos enumerados, se puede programar sin ellos, empleando por ejemplo cadenas
Pero los enumerados tienen muchas ventajas, nombraremos 4 (hay más)

- Ventaja 1
  No hace falta filtrar valores erróneos, con código como

```
if (valor <> 'oros') and (valor <> 'bastos')
    and (valor<>'copas') and (valor <> 'espadas') then error();
```

  (Porque `valor` siempre estará dentro de los enumerados, el compilador impediría lo contrario)

- Ventaja 2
  Los elementos tienen orden, podemos escribir expresiones como `(carta > sota) and (carta < rey)`
- Ventaja 3
  Disponemos de las funciones `succ()` y `pred()`, que devuelven el valor posterior y el valor anterior, respectivamente
- Ventaja 4
  Disponemos de las funciones `high(TIPO_ENUMERADO)` y `low(TIPO_ENUMERADO)` que nos devuelven el mayor y el menor valor posible dentro del tipo enumerado

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program enumerados_03;
type
    TipoTemperatura = ( muy_frio, frio, caliente, muy_caliente);
var
    t: TipoTemperatura;
begin
    t := frio;

    if t < caliente then
        writeln('Encender calefaccion');  // Escribe el mensaje

    writeln(high(TipoTemperatura)) ;    // muy_caliente
    writeln(low(TipoTemperatura)) ;     // muy_frio
    writeln(succ(t)) ;                  // caliente
    writeln(pred(t));                   // muy_frio
end.
```

Registros

Ya conocemos

- Tipos primitivos (boolean, integer, real, char, string)
- Tipos enumerados

Todos ellos son *tipos elementales*, definen un único elemento, un único valor. Además de estos, en Pascal como en casi cualquier lenguaje de programación, tenemos tipos compuestos, formados por la agregación de varios tipos simples
Veremos ahora los *registros*, un tipo de datos compuesto

Es muy común que tengamos datos variados pero con mucha relación entre ellos, que queramos tratar como una única cosa

- Porque sean propiedades del mismo ente
- Porque se usan juntos habitualmente
- Porque son la salida de una función
- ...

Podemos definir un nuevo tipo de datos *record* (registro), como una serie de datos elementales agregados
A cada elemento de un registro se le llama *campo*

Definición de un registro

Para definir un tipo de datos registro

- Usamos la palabra reservada *type*, como para los demás tipos
- El nombre del tipo lo escribimos en NotacionCamello, anteponiendo la palabra *Tipo*
- Después del nombre, el carácter '=', ni ':' ni ':='
- Palabra reservada *record*
- Declaramos el nombre y tipo de cada elemento, finalizado en ';'
- Concluimos con *end*
  - Observa que es un *end* similar al de *case*: es imprescindible y no llega ningún *begin* asociado

```
type
    TipoNotas = record
        entrega : real;
        teorico : real;
        practico : real;
    end;
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program registros_01;
type
    TipoNotas = record
        entrega : real;
        teorico : real;
        practico : real;
    end;
var
    nota_jperez : TipoNotas;
    nota : TipoNotas;
begin
    nota_jperez.entrega := 7.5;
    nota_jperez.teorico := 4.3;
    nota_jperez.practico := 4.2;

    // Ahora nota_jperez es una única variable que contiene
    // los tres valores

    nota := nota_jperez;
    // Podemos copiarlos todos a la vez en otra variable

    // writeln(calcula_media(nota_jperez));
    // Podemos pasarlos a una función como un único parámetro
end.
```

Para acceder a cada campo, escribimos el nombre del registro, un punto y el nombre del campo

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program registros_02;
type
    TipoNotas = record
        entrega : real;
        teorico : real;
        practico : real;
    end;

var
    nota_jperez : TipoNotas;
    nota_media : real;
begin
    nota_jperez.entrega := 7.5;
    nota_jperez.teorico := 4.3;
    nota_jperez.practico := 4.2;

    nota_media :=  ( nota_jperez.entrega + nota_jperez.teorico +
                     nota_jperez.practico) / 3 ;
end.
```

No podemos escribir todo el registro con una única llamada a `write()` o `writeln()`, estos procedimientos no saben manejar los registros. Es necesario escribir campo a campo

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program registros_03;
type
    TipoNotas = record
        entrega : real;
        teorico : real;
        practico : real;
    end;
var
    nota_jperez : TipoNotas;
begin
    nota_jperez.entrega := 7.5;
    nota_jperez.teorico := 4.3;
    nota_jperez.practico := 4.2;

   {writeln( nota_jperez ); ERROR, no se pueden escribir directamente}
    writeln(nota_jperez.entrega:0:2);
    writeln(nota_jperez.teorico:0:2);
    writeln(nota_jperez.practico:0:2);
end.
```

Lo más razonable es usar un procedimiento, escrito por nosotros, que sí sepa manejar nuestro registro

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program registros_04;
type
    TipoNotas = record
        entrega : real;
        teorico : real;
        practico : real;
    end;
procedure escribe_nota(nota:TipoNotas);
begin
    writeln('Entrega de prácticas:',nota.entrega:0:2);
    writeln('Examen teórico:',nota.teorico:0:2);
    writeln('Examen práctico: ',nota.practico:0:2);
end;

var
    nota_jperez : TipoNotas;
begin
    nota_jperez.entrega := 7.5;
    nota_jperez.teorico := 4.3;
    nota_jperez.practico := 4.2;
    escribe_nota(nota_jperez);
```

Pascal tampoco sabe comparar registros, tendremos que programar nuestra función

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program registros_05;
{ declaramos TipoNotas como en los ejemplos anteriores}

function notas_iguales(n1,n2:TipoNotas): Boolean;
begin
    result := (n1.entrega = n2.entrega) and
        (n1.teorico = n2.teorico) and
        (n1.practico = n2.practico);
end;
var
    nota_jperez, nota_mgarcia : TipoNotas;
begin
    nota_jperez.entrega := 7.5;
    nota_jperez.teorico := 4.3;
    nota_jperez.practico := 4.2;
    nota_mgarcia.entrega := 7.5;
    nota_mgarcia.teorico := 4.3;
    nota_mgarcia.practico := 4.2;

  { writeln (nota_jperez = nota_mgarcia);  ¡Mal!   }
    writeln( notas_iguales(nota_jperez,nota_mgarcia));   // TRUE
```

Distancia entre dos puntos

Otro caso donde claramente es conveniente usar registros: coordenadas de un punto

Por ejemplo, coordenadas cartesianas de un punto en el plano

- Un punto será un registro
- Los campos x e y contendrán sus coordenadas

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program coordenadas;
uses math;

type
    TipoPunto= record
        x: real;
        y: real;
    end;

function distancia(a, b: TipoPunto): real;
begin
    result := sqrt((b.x-a.x)**2 + (b.y-a.y)**2);
end;
```

Distancia entre dos puntos en un plano
https://tinyurl.com/yxn9b523

```
procedure escribe_punto(a: TipoPunto);
begin
    write('(', a.x:0:3);
    write(',');
    write(a.y:0:3,')');
end;
```

```
var
    p1,p2 : TipoPunto;

begin
    p1.x := 4;
    p1.y := 0;
    p2.x := -1;
    p2.y := 0;

    write('La distancia entre ');
    escribe_punto(p1);
    write(' y ');
    escribe_punto(p2);
    writeln(' es ', distancia(p1,p2):0:3);
end.
```

```
La distancia entre (1.000,3.000) y (0.000,-1.000) es 4.123
```

https://gsyc.urjc.es/~mortuno/fpi/distancia.pas

Devolver varios valores en una función

Sabemos que una función devuelve exactamente 1 valor. Si necesitamos más, tenemos dos soluciones

1. Usar un procedimiento con parámetros de salida, como vimos en el tema 5
2. Devolver un registro

En el siguiente ejemplo, la función *division_entera* devuelve un registro con el cociente y el resto de una división entera

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program devolver_registro;
type
    TipoResulDiv = record
        cociente : integer;
        resto : integer;
    end;

function division_entera(dividendo, divisor: integer):TipoResulDiv;
begin
    result.cociente := dividendo div divisor;
    result.resto := dividendo mod divisor;
end;

procedure escribe_resultado(dividendo, divisor: integer;
    resultado: TipoResulDiv);
begin
    write('La división entera entre ', dividendo);
    write(' y ', divisor);
    write(' es ', resultado.cociente);
    writeln(' con un resto de ', resultado.resto);
end;
```

```
var
    dividendo, divisor : integer;
    resultado : TipoResulDiv;
begin
    dividendo := 9;
    divisor := 2;

    resultado := division_entera(dividendo, divisor);
    escribe_resultado(dividendo, divisor, resultado);

end.
```

```
var
    dividendo, divisor : integer;
    resultado : TipoResulDiv;
begin
    dividendo := 9;
    divisor := 2;

    resultado := division_entera(dividendo, divisor);

    write('La división entera entre ', dividendo);
    write(' y ', divisor);
    write(' es ', resultado.cociente);
    writeln(' con un resto de ', resultado.resto);
end.
```

```
La división entera entre 9 y 2 es 4 con un resto de 1
```

Devolver varios valores en una función

Una situación habitual donde resulta muy conveniente que una función devuelva un registro es el siguiente:

- Si todo ha ido bien, queremos devolver el valor calculado
- Si algo ha fallado (precondición, postcondición o cualquier otro problema), queremos indicarlo

Para ello devolvemos un registro con dos campos

- Un código que indique si hubo problemas o no
- El valor, si todo fue bien. Si el código indica error, esta valor es irrelevante
  - Distintos lenguajes y liberías usan distintos convenios. P.e. el procedimiento `val`, y muchos otros, devuelve un entero con valor 0 en este caso

En el siguiente ejemplo, devolveremos una cadena

- *ok*, para indicar la ausencia de errores
- En otro caso, la descripción del problema

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program inverso_02;

type
    TipoResultado= record
        codigo:string;
        valor: real;
    end;
```

```
function inverso(x: real): TipoResultado;
var resultado : TipoResultado;
begin
    if x = 0 then
        resultado.codigo := 'Error, el divisor no puede ser nulo'
    else begin
        resultado.codigo := 'ok';
        resultado.valor := 1/x;
    end;

    result := resultado;

end;
```

```
var
    x : real;
    y : TipoResultado;

begin
    writeln('Escribe un número');
    readln(x);

    y := inverso(x);
    if y.codigo = 'ok' then begin
        write('El inverso de ', x:0:3 );
        writeln(' es ', y.valor:0:3);
    end
    else
        writeln( y.codigo );
end.
```

https://gsyc.urjc.es/~mortuno/fpi/inverso_02.pas