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

Octubre de 2018

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*
- Subrangos
  Como 1..6 (de 1 a 6) o -100..100 (de -100 a 100)

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 varible 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.
```

Rango de un enumerado

Al usar enumerados, hay que tener cuidado de no desbordar el rango

- Si intentamos invocar a la función `pred()` pasando como argumento el primer elemento del tipo enumerado, obtendremos un error de ejecución porque no hay ningún valor anterior al primero
- Si intentamos invocar a la función `succ()` pasando como argumento el último elemento del tipo enumerado, obtendremos un error de ejecución porque no hay ningún valor posterior al último

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program enumerados_04;
type
    TipoTemperatura = ( muy_frio, frio, caliente, muy_caliente);
var
    t: TipoTemperatura;
begin
    t := mUY_FRio;
        // Pascal es 'case insensitive' (insensible a mayúsculas)
        // Si cambiamos las mayúsculas el resultado es el mismo,
        // aunque es una mala costumbre.

    writeln(t);          // muy_frio
    writeln(pred(t))     // Error de ejecución, t no tiene predecesor
end.
```

Resultado de la ejecución:

```
muy_frio
Runtime error 201 at $000000000040026E
  $000000000040026E line 14 of enumerados_04.pas
  $000000000040018C
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program siguiente_01;
type
    TipoDiaSem = (lun, mar, mie, jue, vie, sab, dom);
var
    dia: TipoDiaSem;
begin
    dia := sab;
    dia := succ(dia);
    writeln(dia);  // Escribe dom
    dia := succ(dia)   // Genera error de ejecución
end.
```

Resultado:

```
Dom
Runtime error 201 at $0000000000400271
  $0000000000400271 line 12 of dia_siguiente.p
  $000000000040018C
```

Comprobamos que estamos en el rango

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program siguiente_02;
type
    TipoDiaSem = (lun, mar, mie, jue, vie, sab, dom);

function dia_siguiente(dia: TipoDiaSem): TipoDiaSem;
begin
    if dia = dom  then
        result := lun  // Recuerda que aquí no hay ';'
    else
        result := succ(dia)
end ;

var
    dia: TipoDiaSem;
begin
    dia := sab;
    dia := dia_siguiente(dia);
    writeln(dia);  //  dom
    dia := dia_siguiente(dia);
    writeln(dia)  //  lun
end.
```

Mejor aún: lo comprobamos con high() / low()

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program siguiente_03;
type
    TipoDiaSem = (lun, mar, mie, jue, vie, sab, dom);

function dia_siguiente(dia: TipoDiaSem): TipoDiaSem;
begin
    if dia = high(TipoDiaSem)  then
        result := low(TipoDiaSem)
    else
        result := Succ(dia)
end ;

var
    dia: TipoDiaSem;
begin
    dia := sab;
    writeln(dia);  // sab
    writeln(dia_siguiente(dia_siguiente(dia)))   // lun
end.
```

Subrangos

Otra tipo de datos definido por el programador son los subrangos

- Definimos un subrango de forma muy similar a un enumerado, pero
  - En vez de enumerar todos los valores posibles entre paréntesis
  - Indicamos el primer valor posible y el último, separados por '..'

Un tipo subrango es compatible con el tipo base (el tipo a partir del que se definió el subrango)

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program subrangos;
type
    TipoDiaSem = (lun, mar, mie, jue, vie, sab, dom);
    TipoDiaLab = lun .. vie ;
    TipoDiaMes = 1 .. 31 ;
    TipoNatural = 0 .. high(integer) ;
var
    hoy : TipoDiaSem ;
    dia_reunion : TipoDiaLab ;
begin
    hoy := mie ;
    dia_reunion := hoy ;
        // Mezclamos tipo base y tipo subrango, sin problema
        // Son tipos distintos, pero compatibles
    hoy := dom
 {  dia_reunion := hoy ;
        // Error de ejecución, el dia de hoy (domingo) está fuera
        // del rango TipoDiaLab
 }
end.
```

Registros

Ya conocemos

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

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)
end.
```

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
end.
```

Ejemplo: siete y media

En el juego *las siete y media*, el jugador debe conseguir siete puntos y medio

- Se utiliza baraja española, con cartas del 1 al 7. Sin 8 ni 9. Los números 10, 11 y 12 corresponden a las figuras, sota, caballo y rey
- La puntuación de las cartas del 1 al 7 coincide con su valor facial (el número impreso)
- La puntuación de las figura es medio punto

El siguiente programa de ejemplo

- Emplea registros para las cartas. Con un campo para el valor facial de la carta y otro para el palo. Ambos campos son enumerados
- Tiene una función que acepta una carta y devuelve una cadena con su nombre
- Otra función acepta una carta y devuelve un entero con la puntuación de esa carta

Observa que

- Si quisiéramos añadir información adicional a `TipoCarta`, por ejemplo una imagen, bastaría con añadir una línea a la definición de `TipoCarta`, el resto del programa funcionaría igual
- Si quisiéramos usar baraja francesa, bastaría cambiar `TipoPalo` y la función `nombre_carta()`
- Si quisiéramos cambiar la puntuación bastaría cambiar la función `puntuacion()`

Esto es algo muy importante: que un cambio de una característica del programa afecte lo menos posible al resto. Se dice entonces que los elementos del programa están *débilmente acoplados*
Lo contrario, elementos *fuertemente acoplados* hace programas de mala calidad

Código fuente del programa:
https://gsyc.urjc.es/~mortuno/fpi/ejemplos/siete_y_media.pas