Programación en Pascal. Expresiones

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Diciembre de 2019

Contenidos

Directivas para el compilador

En este curso, siempre le pediremos al compilador que sea especialmente cuidadoso con los errores. Nos advertirá o prohibirá ciertas construcciones que en principio son legales, aunque peligrosas. Para ello añadimos la siguiente línea antes de la cabecera del programa

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

- No te preocupes por su significado concreto, cópiala en todos tus programas

Ejemplo completo

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

program holamundo;
begin
    writeln('Hola, mundo');
end.
```

Palabras reservadas

En prácticamente cualquier lenguaje de programación hay una serie de palabras que forman parte del propio lenguaje, no se pueden usar como identificadores
Como referencia, incluimos aquí las de nuestro dialecto de Pascal (Object Pascal):

```
and array asm begin break case const constructor continue
destructor div do downto else end false file for function
goto if implementation in inline interface label mod nil
not object of on operator or packed procedure program
record repeat set shl shr string then to true type unit until
uses var while with xor
```

Identificadores

Identificador: nombre para un elemento del programa (programa, función, procedimiento, constante, variable, etc)

- Normalmente definido por el programador (o por el programador de una librería)
- En Pascal solo podemos usar letras inglesas para los identificadores
  - Esto no suele ser un problema, cualquier programa medianamente serio estará escrito en inglés (identificadores y comentarios). Otros idiomas como el español se usan solo en el interface de usuario, si procede
- El lenguaje Pascal no distingue mayúsculas de minúsculas Apellidos, APELLIDOS y APElliDOS resulta equivalente
  - La mayoría de los lenguajes de programación sí distinguen mayúsculas de minúsculas)

Tipos de datos

En Pascal manipulamos datos. Son de 5 tipos:

- Integer. Números enteros
- Real. Números reales
- Char. Carácteres
  'a' es un tipo char. También ' ' y '0', que no debemos confundir con 0
- String. Cadenas de texto
- Boolean. Valores booleanos
  Solo puede tomar dos valores: TRUE o FALSE

Declaración de constantes

Una constante es una entidad (una *caja*) que contiene un dato, que no cambiará durante la ejecución del programa

- Nos referiremos a ella con un identificador. Por convenio, en este curso las constantes las escribiremos empezando por letra mayúscula

Para usar una constante:

1. La declaramos. Indicamos su tipo
   Nombre de la constante, dos puntos, tipo de dato, punto y coma
2. La definimos, indicamos su valor
   Nombre de la constante, igual, valor, punto y coma

Ejemplo:

```
const
    Pi: real;
    Pi = 3.14159265358979;
```

Si declaramos y definimos, es recomendable declarar y definir al mismo tiempo:

```
const
    Pi: real = 3.14159265358979;
    E: real = 2.71828182845904;
```

Se declaran y definen

- Dentro de un bloque, en la parte declarativa , después de la palabra reservada `const` y antes de la lista de sentencias (begin end)

Observaciones:

- Después de `const` no va un punto y coma

Las constantes también se pueden definir y no declarar. Esto es, indicar el valor pero no el tipo[1]

```
const
    N = 100;
```

[1] En el tema 8 veremos que el tamaño de los arrays es necesario definirlos así, sin declarar el tipo

Las constantes pueden declarse

- Al principio del programa
  Serán constantes globales, visibles en todo el programa. Deben usarse lo mínimo posible, solo para valores *universales*, que no cambien fácilmente: p.e. el número Pi, el radio de la tierra, el tamaño de un campo en un fichero estandarizado, etc [2]
- Al principio de un subprograma
  Típicamente, al principio del cuerpo del programa principal. Solo serán visibles en este subprograma. Aquí podemos definir p.e. datos concretos de nuestro programa

Si tenías nociones de programación, echarás de menos las variables. Por motivos didácticos, en este curso las veremos en el tema 5. No las uses hasta entonces

[2] Suponiendo que todo esto sea constante en el ámbito de nuestro problema, en ciertos escenarios todos estos valores podrían ser cambiantes

Escritura en pantalla

- El procedimiento `write` escribe en la consola (la pantalla) los argumentos que recibe
- El procedimiento `writeln` escribe en la consola los argumentos que recibe, y a continuación, un salto de línea
- Los valores reales se escriben en notación científica. Para usar notación decimal, añade a continuación `:0:n`, donde
  - El 0 significa que el número puede ocupar todo el espacio que necesite
  - `n` representa el número de decimales.

Ejemplo: `write(tiempo_segundos:0:1)`

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

const
    Precio: real = 12.50;
    Porcentaje_descuento: real = 10;
begin
    writeln('El precio es');
    writeln( Precio);

    write('El precio es ');
    writeln(Precio:0:2);

    write('Precio: ', Precio:0:2);
    write(' Descuento: ', Precio * 0.01 * Porcentaje_descuento :0:2 );

    write(' Precio final:');
    writeln( Precio * (1 - 0.01 * Porcentaje_descuento):0:2 );
end.
```

Resultado de la ejecución:

```
El precio es
 1.2500000000000000E+001
El precio es 12.50
Precio: 12.50 Descuento: 1.25 Precio final:11.25
```

El delimitador de cadena es la comilla recta (la tecla a la derecha de la tecla 0) y no la comilla invertida (la tecla a la derecha de la tecla p)

Operadores

Un operador es un símbolo o una palabra reservada que indica que se debe realizar una operación matemática o lógica sobre unos *operandos* para devolver un resultado

- Los operandos son expresiones (valores) de entrada, en Pascal la mayoría de los operadores tienen 2 operandos, algunos tienen 1.

Ejemplo:

- $2 + 2$

El operador $+$ tiene dos operandos y devuelve como resultado su suma

Los operadores están muy vinculados a los tipos de datos, cada operando solo puede recibir ciertos tipos concretos de datos, para devolver cierto tipo de datos

Ejemplo

El operador `div` es la división entera. Sus operandos han de ser números enteros. En otro caso, el compilador produce un error

Uso correcto:

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program ej_div;
begin
    writeln( 5 div 2);   // Escribe 2
end.
```

Uso incorrecto:

```
writeln( 5 div 2.0);
```

```
koji@mazinger:~/pascal$ fpc ej_div.p
Free Pascal Compiler version 3.0.0+dfsg-2 [2016/01/28] for x86_64
Copyright (c) 1993-2015 by Florian Klaempfl and others
Target OS: Linux for x86-64
Compiling ej06.p
ej06.p(4,16) Error: Operator is not overloaded: "ShortInt" div "Single"
ej06.p(8) Fatal: There were 1 errors compiling module, stopping
Fatal: Compilation aborted
Error: /usr/bin/ppcx64 returned an error exitcode
```

Otro ejemplo incorrecto

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

const
    X: integer = 3;
    Y: char = '4';
begin
    write( 'X vale ');
    writeln( X );

    write( 'Y vale ');
    writeln( Y );

    write( 'La suma vale ');
    write( X + Y );  // ¡¡MAL!!  La constante Y no es numérica
end.
```

```
Compiling tipos_mal.pas
tipos_mal.pas(15,14) Error: Operator is not overloaded: "LongInt" + "Char"
tipos_mal.pas(17) Fatal: There were 1 errors compiling module, stopping
Fatal: Compilation aborted
Error: /usr/bin/ppcx64 returned an error exitcode
```

- Pascal es *fuertemente tipado*, esto significa que en general no se pueden mezclar tipos de datos, hay que convertirlos antes. Convertir un dato de un tipo a otro se llama *ahormado*. En español solemos emplear la palabra inglesa: *casting*
- Algunas conversiones de tipos las hace el compilador automáticamente

Operadores numéricos disponibles para enteros y reales

```
**   Exponenciación
+ - * /
-   (operador unario de cambio de signo)
```

Si un operando es entero, el compilador lo convierte en real, automáticamente

Operandos para enteros

```
div    División entera
mod    Resto de la división entera
```

Solo para enteros, si un operando es p.e. real el compilador da error

Para poder usar el operador de exponenciación, es necesario añadir la cláusula `uses math;` en la cabecera, esto añade la librería matemática

Ejemplo: $2{,}1^{3{,}1}$

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

program ej_potencias;
uses math;
begin
    writeln( 2.1 ** 3.1);  // Escribe 9.9742399926587076014 5E+0000
end.
```

El programador puede hacer ciertas conversiones de tipos explícitamente

- Un entero se puede convertir en real

  ```
  real(3)
  ```

- Un carácter se puede convertir en entero (obteniendo el código ASCII correspondiente)

  ```
  integer('a')
  ```

- Un entero se puede convertir en carácter (obtenemos el carácter correspondiente a ese código ASCII)

  ```
  char(97)
  ```

Código ASCII:
https://es.wikipedia.org/wiki/ASCII

Un número real no se puede ahormar directamente a entero,
Pero disponemos de las funciones predefinidas
`trunc()` y `round()`
que reciben un número real y devuelven un entero, truncado sus decimales o redondeando

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

program casting;

const
    X: char = 'a';
    Y: integer = 98;
    Z: real = 65.7;
begin
    writeln( integer(X));  // Escribe 97

    writeln( char(Y));   // Escribe 'b'
    writeln( real(Y));   // Escribe 9.8000000000000000E+001

    { writeln( integer(Z));   // ¡Esto es ilegal!  }

    writeln( round(Z));   // Escribe  66
    writeln( trunc(Z));   // Escribe  65

    writeln( char( trunc(Z) ) ) ;   // Escribe  'A'
end.
```

Operadores de comparación

Sus argumentos pueden ser enteros o reales. Devuelven un booleano

```
=    Igual
<>   Distinto
<    Menor
<=   Menor o igual
>    Mayor
>=   Mayor
```

Un error frecuente es confundir el operador de comparación de igualdad = con el operador de asignación := [3]

[3] O con el operador de comparación de igualdad en C y derivados ==, o con el operador de asignación en C y derivados, =

Es materia de los temas 3 y 4, pero adelantamos aquí este ejemplo

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

program comparacion;

function posible_matricula(nota: real):boolean;
begin
    if nota = 10
    then
        result := TRUE
    else
        result := FALSE;
end;

const
    Nota_ejemplo: real = 9.5;
begin
    writeln( posible_matricula(Nota_ejemplo));
end.
```

Operadores booleanos

- `op1 and op2`
  Devuelve TRUE cuando ambos operandos son ciertos.
  Devuelve FALSE en otro caso
- `op1 or op2`
  Devuelve TRUE cuando un operando es cierto o cuando dos operandos son ciertos.
  Devuelve FALSE en otro caso
- `not op`
  Devuelve TRUE cuando el operando es FALSE.
  Devuelve FALSE cuando el operando es TRUE
- `op1 xor op2`
  Devuelve TRUE cuand un operando es TRUE y otro es FALSE.
  Devuelve FALSE en otro caso.
  equivale a
  `(op1 and (not op2)) or ((not op1) and op2)`

Expresiones booleanas

- Ya estás familiarizado con las expresiones numéricas, que combinan operandos numéricos con operadores numéricos. P.e

```
1.23 + (2.4 * 12)
```

- En programación se usan mucho, además, las expresiones booleanas. Los operandos son booleanos y, por supuesto, también los operadores

```
FALSE or not (TRUE and FALSE)
Diabetico and not Menor_edad
```

- Los operadores booleanos solo admiten operandos booleanos

```
Diabetico and 20   //  ¡¡Mal!!
```

- Los operadores de comparación aceptan enteros o reales como operandos, y devuelven un booleano, con el que ya si podemos construir expresiones booleana

```
Diabetico and (Edad >= 18)
```

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

const
    Diabetico : boolean = TRUE;
    Menor_edad : boolean = FALSE;
    Edad: integer = 20;
begin
    writeln(FALSE or not (TRUE and FALSE));  // Escribe  TRUE
    writeln(Diabetico and not Menor_edad);  // Escribe TRUE

    { writeln(Diabetico and not Edad);     ¡¡ Esto es un error !!  }
    { writeln(Diabetico and not Edad < 18); ¡¡ Esto es un error !! }

    writeln(Diabetico and not (Edad < 18)); // Escribe TRUE
    writeln(Diabetico and (Edad >= 18)); // Escribe TRUE
end.
```

Ejemplo: años bisiestos

- Descripción en lenguaje natural:

```
Los años múltiplos de 4 son bisiestos.
Excepción: los múltiplos de 100, que no lo son.
Excepción a la excepción: los múltiplos de 400 sí lo son.
```

- Descripción algorítmica

```
Un año es bisiesto si es múltiplo de 4 y
(no es múltiplo de 100 o es múltiplo de 400)
```

- Implementación en Pascal

```
(A mod 4 = 0 ) and   (not (A mod 100 = 0)   or  (A mod 400 = 0) )
```

- Implementación en Pascal, un poco más legible

```
(A mod 4 = 0 ) and   ( (A mod 100 <> 0)   or  (A mod 400 = 0) )
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program bisiesto ;
const
    Anyo : integer = 1940;

begin
    write(Anyo, ' es bisiesto:');
    writeln(
        (Anyo mod 4 = 0 )
        and
        ( (Anyo mod 100 <> 0)  or  (Anyo mod 400 = 0 ))
    );
end.
```

Lógica proposicional

Doble negación

- $\neg\neg p \Leftrightarrow p$
- No es cierto que no llueva $\Leftrightarrow$ Llueve

Ambas expresiones son lógicamente equivalentes, aunque la primera es más clara

En lenguaje natural, es habitual usar expresiones del tipo *si... entonces*

- *Si me avisas, entonces llevo más dinero*
- *Si suspendes las prácticas, entonces suspendes la asignatura*

En la especificación de un algoritmo hay que tener mucho cuidado, porque en rigor, con esta estructura estamos diciendo qué sucede si se cumple la condición, pero no estamos diciendo qué pasa si no se cumple

- En algunos casos, posiblemente los humanos supongan que si no se cumple la condición, no se cumple la consecuencia
  Si no me avisas, entonces no llevo más dinero
- En otros, posiblemente no haremos esa suposición
  Si apruebas las prácticas, ya veremos, dependes de los exámenes

Estas imprecisiones propias del lenguaje natural no son admisibles en la especificación de un algoritmo, es importante dejar claro qué pasa si la condición es falsa: si estoy diciendo algo para ese caso o si no estoy diciendo nada

Caso 1. Equivalencia

- *Si me avisas, entonces llevo más dinero. Y si no, no*
  Esto se convierte en una equivalencia lógica. El aviso equivale a llevar más dinero.
  $aviso \implies mas\_dinero \wedge \neg aviso \implies \neg mas\_dinero$
  $aviso \Leftrightarrow mas\_dinero$
- $p \implies q \wedge \neg p \implies \neg q$
  $p \Leftrightarrow q$

Caso 2. Implicación

- *Si suspendes las prácticas, entonces suspendes la asignatura. Y si apruebas las prácticas, ya veremos*
  $suspender\_practicas \implies suspender\_asignatura$
  (Y ya está, en un entorno formal está claro que no estoy haciendo ninguna afirmación si no se da la condición)
- $p \implies q$

Ojo con sacar conclusiones erróneas

- $p \implies q$
  ¿Equivale a ?
  $\neg p \implies \neg q$
  ¡No!

Con otras palabras, es lo mismo que acabamos de ver. De la afirmación *si me avisas, entonces llevo más dinero*, no se puede deducir que si no me avisas, no lo llevo. Es necesario indicarlo explícitamente (si procede)

Transposición de la implicación

La conclusión que sí podemos extraer es

- $p \implies q \Leftrightarrow \neg q \implies \neg p$
- *ser asturiano implica ser español*
  equivale a
  no ser español implica no ser asturiano

Otro ejemplo

- Si he venido es porque no lo sabía
- Si lo se no vengo

Un poco más claro, en presente

- Si voy es porque no lo se
- Si lo se, no voy

Leyes de De Morgan

Las leyes de De Morgan[4] también permiten generar expresiones booleanas equivalentes desde el punto de vista lógico

- `not (a and b)`
  equivale a
  `(not a) or (not b)`
- `not (a or b)`
  equivale a
  `(not a) and (not b)`

[4] No es una errata, el nombre de su descubridor es Augustus De Morgan

Aplicando la doble negación y las leyes de De Morgan, podemos escribir las expresiones booleanas de formas distintas, que

- Desde el punto de vista lógico y mátemático, serán equivalentes
- Considerando la claridad para el humano, no serán equivalentes. Las personas entedemos mejor la *lógica positiva* (afirmaciones sin negaciones) que la *lógica negativa* (afirmaciones con negaciones)
  En programación, normalmente lo más importante es el programador, presente o futuro. En general deberemos usar la expresión equivalente con menos negaciones

Aplicando la transposición de la implicación, podemos mejorar la claridad de las implicaciones en lógica booleana

Ejemplos

- `p = not q and (not r or s )`

  `not p = not ( not q and (not r or s ) )`

  `not p = q or not (not r or s)`

  `not p = q or (r and not s)`

- `p = (a >= 65) or (a <= 16) or q`

  `not p = (a < 65) and (a > 16) and not q`

Ejercicio

1. Escribe en una hoja de papel un par de expresiones booleanas parecidas a las de la transparencia anterior
   - Una equivalencia, con la constante *p* a la izquierda de la igualdad y las constantes q, r y s a la derecha. Con los operadores and, or, algún not y algún paréntesis
   - Otra equivalencia con la constante *p* a la izquierda, valores numéricos y constantes a la derecha. Con operadores de comparación, and y or
   - Usa la notación de Pascal (and, or, not), no uses notación lógica ($\wedge, \vee, \neg$)
2. Escribe en otra hoja las expresiones booleanas equivalentes, negando ambos lados de la igualdad
3. Entrega la primera hoja a un compañero para haga lo mismo. Resuelve tú su ejercicio. Comparad las soluciones para comprobar que sean iguales, corregid el problema si hay errores

Precedencia de operadores

En matemáticas normalmente podemos distribuir los operandos entre varias líneas, haciendo cosas como $\frac{4+2}{1+1} = 3$

- En casi todos los lenguajes de programación, nos vemos obligados a usar una sola línea
- Si intentamos escribir la expresión anterior como $4 + 2/1 + 1$ estaremos cometiendo un error, porque el compilador lo interpreta como
  $4 + \frac{2}{1} + 1 = 7$

Los operadores tienen una *reglas de precedencia*

- Los operadores se evalúan de mayor precedecia a menor precedencia
- A igualdad de precedencia, se evalúa de izquierda a derecha

Precedencia en Pascal, de mayor a menor:

```
**  not  - (cambio signo)
* / div mod and
or xor + - (resta)
```

En un programa la claridad es fundamental, así que debemos usar paréntesis. Incluso es recomendable hacerlo en los casos en los que, por la precedencia de los operadores, no sería necesario
Ejemplo

- $\frac{4+2}{1+1}$
  lo escribimos
  $(4 + 2)/(1 + 1)$
- Si quisiéramos escribir
  $4 + \frac{2}{1} + 1$
  bastaría $4 + 2/1 + 1$
  Pero es preferible ser muy claro:
  $4 + (2/1) + 1$

Elementos predefinidos

Como en prácticamente cualquier lenguaje, en Pascal hay *elementos predefinidos*: funciones, operaciones y constantes definidos inicialmente en el lenguaje. No podemos declarar nuevos identificadores que usen estos nombres.

```
abs(n)      valor absoluto
trunc(n)    truncar a entero
round(n)    redondear a entero
sqr(n)      elevar al cuadrado
sqrt(n)     raíz cuadrada
chr(i)      carácter en posición i
ord(c)      posición del carácter o valor c
pred(c)     carácter o valor predecesor de c
succ(c)     carácter o valor sucesor de c
arctan(n)   arcotangente
cos(n)      coseno
exp(n)      exponencial
ln(n)       logaritmo neperiano
sin(n)      seno
low(x)      menor valor o índice en x
high(x)     mayor valor o índice en x
```

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program predefinidos;
begin
    writeln( abs(-3) );       // Escribe 3
    writeln( trunc(3.64) );   // Escribe 3
    writeln( round(3.64) );   // Escribe 4
    writeln( sqr(3) );        // Escribe 9
    writeln( sqrt(9) );       // Escribe 3e0
    writeln( succ('a') );     // Escribe b
    writeln( pred('B') );      // Escribe A
end.
```