Programación en Pascal. Funciones

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Septiembre de 2019

Definición de problemas

La definición de un problema consiste en tres cosas:

- Saber qué resolvemos
- Saber qué nos hace falta para resolverlo
- Saber qué tendremos cuando lo hayamos resuelto

Esto coincide con la declaración de una función

- Cómo se llama la función
- Qué necesita
- Qué devuelve

Funciones en pascal

Una función básica en Pascal tendrá la suguiente estructura

```
function NOMBRE_FUNCION( LISTA_DE_PARAMETROS): TIPO_DEVUELTO;
begin
    result := EXPRESION_CON_LA_SOLUCION;
end;
```

Ejemplo:

```
function long_circunf(r: real): real;
begin
    result := 2.0 * Pi * r;
end;
```

`result` es una palabra reservada que en nuestro dialecto de Pascal (Object Pascal) emplemos para darle valor a la función

- En Pascal estándar usaríamos el mismo nombre de la función, pero esto tiene algunos inconvenientes

```
function long_circunf(r: real): real;
begin
    long_circunf := 2.0 * Pi * r;
end;
```

Luego llamaremos a la función con un valor concreto p.e.

```
long_circunf(4.2);
```

En este caso

- 4.2 es el argumento, el valor concreto que suministramos a la función
- r es el parámetro, el identificador que usamos para nombrar al argumento

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

program longitud_circunferencia;

const
    Pi : real = 3.14159265;

function long_circunf(r: real): real;
begin
    result := 2.0 * Pi * r;
end;

begin
    writeln( long_circunf(4.2):0:3 );  // Escribe 26.389
end.
```

Una función en Pascal bien escrita es muy similar a una función matemática

- Acepta uno o más parámetros
- Devuelve un valor, calculado a partir de los parámetros (y de ninguna otra cosa)

Y nada más

- NO lee ningún otro valor del exterior, solo sus parámetros
- NO depende del estado del programa
- NO depende del orden de evaluación de los parámetros
- NO provoca ningún cambio en ninguna parte, solo devuelve su valor
    - Esto incluye NO escribir nada en pantalla

A esto se llama *transparencia referencial*.

Así, no hay ninguna diferencia entre

- Llamar a una función, con sus argumentos
- Escribir el valor devuelto por la función

Ejemplo. Tenemos un programa con la función

```
function incrementa(x: integer): integer;
begin
    result := x + 1;
end;
```

Luego llamamos a esta función

```
incrementa(3);
```

Si cumplimos la propiedad de transparencia referencial (que en este curso es obligatorio), nuestro programa se comportará exactamente igual en todo si, en vez de escribir esta llamada, escribimos

```
4;
```

Se denomina *efecto lateral* a cualquier acción producida por una función. Por tanto,
Estas dos cosas son equivalentes

- Una función tiene transparencia referencial
- Una función no tiene efectos laterales

Podemos decir que una función con efectos laterales *hace lo que no debe*

- El compilador no lo va a impedir, no se va a generar ningún error. Pero eso es código de muy mala calidad, propenso a errores (aunque lo podemos ver incluso en algunos libros de programación)
- En este curso no permitiremos funciones con efectos laterales (excepto tal vez generar trazas)

Ejemplo de función con varios parámetros:

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

program media_v1;

function media(x: real; y:real): real;
begin
    result :=  (x + y) / 2;
end;

begin  // Aquí empieza el cuerpo principal
    writeln( media(4, 5) );
end.
```

Observa que writeln está en el cuerpo principal, nunca debe estar dentro de la función (porque eso sería un efecto lateral)

Si una función tiene varios parámetros del mismo tipo, también podemos declararlos así

```
function media(x, y: real): real;
begin
    result :=  (x + y) / 2;
end;
```

Una función devuelve una expresión. En cualquier lugar donde se espera una expresión, se puede escribir una función

```
writeln( media(4, 5) + 1 );    // Escribe 5.5
```

En la llamada a una función, el argumento puede ser cualquier expresión: un literal, una constante, otra función...

```
writeln( media(2 + 2, 3 + 2) );
```

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

program media_v3;

function media(x, y: real): real;
begin
    result :=  (x + y) / 2.0;
end;

begin
    writeln( media(4, 5)+1) ;       // Escribe  5.5000000000000000E+000

    writeln( (media(4, 5)+1):0:3 );   // Escribe  5.500

    writeln( media(2+2, 3+2) );  // Escribe  5.5000000000000000E+000
end.
```

De nuevo, todos los writeln están fuera de la función. Lo contrario sería un defecto muy severo

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

program llamadas_funcion;

function suma(x, y: integer): integer;
begin
    result :=  x + y
end;

function incrementa(x: integer): integer;
begin
    result := x + 1
end;

const
    a : integer = 4;
    b : integer = 1;

begin
    writeln( suma(9, 2) );              // Escribe 11

    writeln( suma(a, b) );              // Escribe 5

    writeln( suma(a, incrementa(b))) ; // Escribe 6
end.
```

Los problemas ser resuelven dividiéndolos en subproblemas, que serán subprogramas
Lo habitual es:

1. Diseñar la solución *top-down* (arriba-abajo)
   Empezamos por el problema global, suponiendo que tenemos resueltos los subproblemas. Aplicamos esto las veces necesarias, hasta que tengamos un problema con solución directa
2. Programar *bottom-up* (abajo-arriba)
   Programamos los subproblemas, empezando por los más sencillos. Probamos cada subprograma, con valores de ejemplo. Cuando estemos razonablemente seguros de que funciona, programamos un subprograma más complejo

Volumen de un cilindro hueco

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program cilindro_hueco;
const       // Constantes globales
    Pi: real = 3.1415926;

function area_circulo(r: real): real;
begin
    result := Pi*sqr(r);
end;

function area_corona(r_interior: real; r_exterior: real): real;
begin
    result := area_circulo(r_exterior) - area_circulo(r_interior);
end;

function vol_cil_hueco(r_interior, r_exterior, alt: real): real;
begin
    result := area_corona(r_interior, r_exterior) * alt;
end;
```

```
const    // Constantes del cuerpo del programa
    Rmin: real = 3.2;
    Rmax: real = 4.3;
    Alt: real = 2.8;

begin
    writeln('v = ', vol_cil_hueco(Rmin, Rmax, Alt) :0:3);
end.
```

El nombre de la constante que usamos como argumento no tiene relación que el nombre del parámetro. Pueden coincidir, pero en general no tienen por qué

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

program nombres_parametros;

function media(x, y: real): real;
begin
    result := (x+y) / 2
end;

const
    a: real = 2;
    b: real = 1;

begin
    writeln( media(a,b) );
end.
```

División de un número en sus dígitos (por medios aritméticos)
A partir de p.e. 4297 queremos el 4, 2, el 9 y el 7

```
4297 = 4 * 1000 + 2 * 100 + 9 * 10 + 7
     = 4 * 10**3 + 2 * 10 ** 2 + 9 * 10**1 + 7 * 10**0
```

- Haremos la división entera entre 1000, 100, 10 y 1 para obtener 4, 42, 429 y 4297
- A partir de este resultado, tomaremos el resto de la división entera entre 10 para obtener 4, 2, 9 y 7

- Cuarto dígito (por la derecha)
  4297 div 1000 = 4

  pos = 4
  1000 = 10 ** 3 = 10 ** (pos - 1)

- Tercer dígito
  4297 div 100 = 42
  42 mod 10 = 2

  pos = 3
  100 = 10 ** 2 = 10 ** (pos-1)

- Segundo dígito
  4297 div 10 = 429
  429 mod 10 = 9

  pos = 2
  10 = 10 ** 1 = 10 **(pos-1)

- Primer dígito
  4297 div 1 = 4297
  4297 mod 10 = 7

  1 = 10 ** 0 = 10 **(pos-1)

Generalizamos la fórmula para que el cuarto dígito se calcule como los demás

```
Cuarto dígito
4297 div 1000 = 4
4 mod 10 = 4
```

Por tanto, la fórmula para obtener el dígito `pos` del número `n` es

```
(n div (10 ** (pos-1))) mod 10;
```

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

program separar_digitos;

uses math;

function aisla_digito(n, pos: integer): integer;
begin
    result := (n div 10**(pos-1)) mod 10;
end;

const
    Num: integer = 4297;
begin
    writeln(aisla_digito(Num, 4));   // Escribe 4
    writeln(aisla_digito(Num, 3));   // Escribe 2
    writeln(aisla_digito(Num, 2));   // Escribe 9
    writeln(aisla_digito(Num, 1));   // Escribe 7
end.
```