Depuración de programas

Miguel Ortuño
Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

Septiembre de 2022

Contenidos

Formato de cadenas

Ya sabemos usar *write* y *writeln* para componer un mensaje en pantalla. Supongamos que necesitamos algo como esto

```
x: 3.47 y: 12.15 z: 0.29
```

Podemos generarlo así:

```
write('x:',x:0:2);
write(' y:',y:0:2);
writeln(' z:',z:0:2);
```

Esto funciona, pero tiene tres problemas

1. Es farragoso y propenso a errores
2. No permite un ajuste fino de la presentación de los números
3. No es aplicable en algunos casos. Por ejemplo en la librería de depuración que usaremos aquí

Casi todos los lenguajes de programación nos ofrecen una solución más conveniente: dar formato a una cadena

Esta forma de componer cadenas se hace popular por las funciones *printf* y *sprintf* del lenguaje C. Muchos otros lenguajes lo han incluido, con las mismas opciones y sintaxis

- Usan un *microlenguaje* muy rico, especialmente para indicar cómo queremos que se muestren los números
- Aquí veremos solo las opciones más elementales, pero hay muchas otras: alineamiento a derecha, izquierda, centrado, número de decimales, ceros a la izquierda, diversos tipos de notación científica, base decimal, hexadecimal, etc

Función format

En Pascal disponemos de la función *format*.

- Para usarla es necesario añadir al comienzo del programa

```
uses SysUtils;
```

Recibe dos argumentos

- Una *format string* que especifica el texto invariable y el formato del contenido variable
  Ejemplo:

```
'Nombre: %s   Nota: %f   Convocatoria:%d'
```

  Un signo de porcentaje seguido de una letra es un *especificador de formato*

- Un array de argumentos, que indica cual será el contenido variable. Casi siempre serán variables
  Ejemplo:

```
[nombre,nota,convocatoria]
```

Devuelve una cadena, donde cada especificador de formato habrá sido reemplazado por un argumento del array

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program ejemplo_format_01;

uses  SysUtils;

var
    nombre : string;
    nota : real;
    convocatoria : integer;
    mensaje: string;

begin
    nombre := 'Juan García';
    nota := 6.25;
    convocatoria := 1;

    mensaje := format('Nombre: %s   Nota: %f   Convocatoria:%d',
        [nombre,nota,convocatoria]);

    writeln(mensaje);
end.
```

Resultado:

```
Nombre: Juan García   Nota: 6.25   Convocatoria:1
```

Observa que el primer especificador de formato (`%s`) se reemplaza por el primer argumento del array, el segundo especificador (`%f`) por el segundo argumento y así sucesivamente

- `%s` indica una cadena
- `%f` indica un número real (en lenguaje C el tipo es *float*)
- `%d` indica un número entero, decimal

Hay muchos otros especificadores, estos son los principales[1]

[1] puedes encontrar el resto buscando en el web 'printf format'

Para escribir el simbolo de porcentaje como literal y no como especificador, se usa %%

```
format( 'Aprobados: %f %%',[aprobados]);
```

Resultado:

```
Aprobados: 68.31 %
```

Obviamente, tendremos que prestar atención para que

- El número de especificadores coincida con el número de argumentos del array
- Los tipos coincidan. Por ejemplo no tiene sentido indicar %s para un valor numérico

Si nos equivocamos, no obtendremos un error de compilación sino uno de ejecución

```
An unhandled exception occurred at $00000000004348B2:
EConvertError: Invalid argument index in format
"Nombre: %s   Nota: %s   Convocatoria:%d"
  $00000000004348B2
  $00000000004359EA
  $0000000000435F26
  $0000000000401182 line 17 of ejemplo_format_01.pas
```

Depuración de un programa

Depurar es el proceso de localizar y corregir los fallos de un programa

- Es una habilidad especialmente importante: es normal que un programador emplee la mayor parte de su tiempo en la depuración, no en el análisis ni en la escritura de código
- Los programas raramente funcionan a la primera. El programador tiene que detectar los problemas y corregirlos. Lo fundamental es *ver* qué está pasando
- La depuración empieza en el mismo momento de la programación. Cuando empezamos a escribir cada función deberíamos ir pensando cómo depurarla, sin esperar a que se manifiesten los problemas

Como estudiante siempre podrás pedir ayuda al profesor. Pero antes, debes intentar depurar el programa por tí mismo

- Un buen profesor con frecuencia no arreglará tu programa, sino que te dará las pautas para depurarlo
- Te estás formando. Que tu programa funcione es un objetivo subordinado al objetivo principal: que aprendas a programar y por tanto a depurar
- Es muy frecuente que un profesor de programación tenga que decirle a un estudiante algo como: *Tu programa no tiene buena solución. El código es complicado, no está bien dividido en subprogramas, tienes muchos niveles de if/while/for... Tú mismo no lo entiendes, es seguro que no hay un fallo sino muchos, porque te has puesto a escribir líneas de forma desorganizada y sin comprobar cada paso. Borra todo y vuelve a empezar. Pero ahora haz esto, esto y ...*

Supongamos que necesitamos una función que calcule una potencia (sin usar **) y escribimos algo así

```
function potencia(base: real; exponente: integer): real;
var
    i : integer;
    mensaje : string;
begin
    result := 1;
    for i := 0 to abs(exponente) do begin
        result := result * base;
    end;

    if exponente < 0 then begin
        result := 1 / result;
    end;
end;
```

Los probamos por ejemplo con

```
base := 2;
exponente := 3;
```

Y obtenemos el resultado erróneo 16

- Tal vez nos demos cuenta de que el problema es que el bucle *for* debería empezar en 1 y no en 0, pero si no es el caso, iremos *generando trazas*
- Esto consiste en ver, paso a paso, los valores que se van generando, para analizar el comportamiento del programa

- Una solución sencilla y frecuente, aunque *chapucera* es ir escribiendo en pantalla los valores de la variable, usando *write* o la sentencia equivalente en nuestro lenguaje

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

uses slog, SysUtils;

function potencia(base: real; exponente: integer): real;
var
    i : integer;
    mensaje : string;
begin
    result := 1;
    for i := 0 to abs(exponente) do begin
        result := result * base;
        mensaje := format('i:%d  result:%f', [i,result]);
        writeln(mensaje);
    end;

    if exponente < 0 then begin
        result := 1 / result;
        writeln('Exponente negativo, multiplico por -1');
    end;
end;
```

```
var
    base : real;
    exponente : integer;
begin
    base := 2;
    exponente := 3;
    writeln(potencia(base, exponente):0:3);

    exponente := -3;
    writeln(potencia(base, exponente):0:3);
end.
```

Resultado:

```
i:0  result:2.00
i:1  result:4.00
i:2  result:8.00
i:3  result:16.00
16.000
i:0  result:2.00
i:1  result:4.00
i:2  result:8.00
i:3  result:16.00
Exponente negativo, multiplico por -1
0.063
```

- Esto tiene el problema de que *ensucia* la pantalla, mezclando información de depuración con la verdadera salida del programa. Rompe completamente el principio de la transparencia referencial: las funciones no deberían escribir nada en pantalla, pero escribir una traza es escribir
- La solución igualmente *chapucera* es: *cuando acabe de probarlo, quito los mensajes de traza*
  - Lo cual es pesado, lleva tiempo. Además puede ser complicado localizar dónde se generan la trazas
  - Es muy habitual que, en el futuro, se vuelvan a detectar problemas en el programa. Entonces habría que reescribir las trazas, un esfuerzo repetido, una pérdida de tiempo (al borrar y al reescribir)

La forma adecuada de generar trazas es mediante una librería de *logs* [2]

- Una librería es un fichero que contiene funciones, que son llamadas desde el código escrito en otro fichero
- En cualquier lenguaje tendremos librerías de log. O podremos programar la nuestra, es sencillo
- Para esta asignatura hemos preparado la librería *slog* (*simple log*. Puedes descargarla en https://gsyc.urjc.es/~mortuno/slog.pas

[2] La palabra española es bitácora, pero raramente se usa en este contexto

Uso de slog

Para usar slog en un programa

- Añade al principio del programa al menos la librería *slog*

```
uses slog;
```

  Probablemente querrás usar format, así que normalmente escribirás

```
uses slog, SysUtils;
```

- Incluye el fichero slog.pas en el mismo directorio que el programa

slog escribirá las trazas en el fichero de log, que por omisión se llamará *slog.log* y estará en el mismo directorio
Para escribir en el fichero de log

- Llama a la función `slog.debug()`, pasando como argumento la cadena a escribir

Para que las trazas dejen de generarse

- Invoca al procedimiento *setquietmode* con el parámetro *True*

```
slog.setquietmode(True);
```

Ejemplo

```
program hola_log;

uses slog, SysUtils;

var
    cadena : string;
    mi_entero : integer;
    mi_real : real;
```

```
begin
    cadena := 'Hola,mundo';
    mi_entero := 3;
    mi_real := 2.5;

    slog.debug('Hola mundo, esto es un mensaje de depuración');

    slog.setquietmode(True);
    slog.debug('Esto no aparecerá en el log');

    slog.setquietmode(False);
    slog.debug('Esto vuelve a salir en el log');

    slog.debug(format('la cadena vale %s, el entero %d, y el real %f',
        [cadena, mi_entero, mi_real]));

    mi_entero := mi_entero + 1;
    mi_real := mi_real * 2;
    slog.debug(format('ahora el entero vale %d y el real %f',
        [mi_entero, mi_real]));
end.
```

Al ejecutar el programa anterior

- En pantalla no aparecerá nada
- Contenido del fichero *slog.log*:

```
2020-12-18 20:33:28 DEBUG Hola mundo, esto es un mensaje de depuración
2020-12-18 20:33:28 DEBUG Esto vuelve a salir en el log
2020-12-18 20:33:28 DEBUG la cadena vale Hola,mundo, el entero 3, y el real 2.50
2020-12-18 20:33:28 DEBUG ahora el entero vale 4 y el real 5.00
```

Cada vez que usemos slog, los mensajes nuevos se irán añadiendo al fichero *slog.log*, sin borrar los mensajes anteriores. Cuando ya no necesites la traza

- Escribe al principio de tu programa

```
slog.setquietmode(True);
```

- No es necesario que borres las llamadas a *slog.debug* (excepto tal vez las líneas muy obvias que probablemente no vuelvas a necesitar)

Otro ejemplo

```
{$mode objfpc}{$H-}{$R+}{$T+}{$Q+}{$V+}{$D+}{$X-}{$warnings on}
program potencia_03;
uses slog, SysUtils;
function potencia(base: real; exponente: integer): real;
var
    i : integer;
    traza : string;
begin
    result := 1;
    for i := 0 to abs(exponente) do begin
        result := result * base;
        traza := format('i:%d  result:%f', [i,result]);
        slog.debug(traza);
    end;

    if exponente < 0 then begin
        result := 1 / result;
        traza := format('Exp. negativo, multiplico por -1. result:%f',
            [result]);
        slog.debug(traza);
    end;
end;
```

```
var
    base : real;
    exponente : integer;

begin
    base := 2;
    exponente := 3;
    writeln(potencia(base, exponente):0:3);

    exponente := -3;
    writeln(potencia(base, exponente):0:3);
end.
```

En el fichero *slog.log* veremos la siguiente traza

```
2020-12-18 20:55:03 DEBUG i:0  result:2.00
2020-12-18 20:55:03 DEBUG i:1  result:4.00
2020-12-18 20:55:03 DEBUG i:2  result:8.00
2020-12-18 20:55:03 DEBUG i:3  result:16.00
2020-12-18 20:55:03 DEBUG i:0  result:2.00
2020-12-18 20:55:03 DEBUG i:1  result:4.00
2020-12-18 20:55:03 DEBUG i:2  result:8.00
2020-12-18 20:55:03 DEBUG i:3  result:16.00
2020-12-18 20:55:03 DEBUG Exp. negativo, multiplico por -1. result:0.06
```

Que debería servir para darnos cuenta de que el error ha sido empezar el bucle por 0 y ejecutarlo 4 veces

¿Qué trazar?

Cuando nuestro subprograma no hace lo que esperamos, el problema puede estar en

- Los parámetros que recibe, que no son los que deberían
- Las expresiones principales del cálculo
- La asignación del resultado al valor de retorno o a los parámetros de salida

En otras palabras: el error puede estar en cualquier parte, podemos necesitar trazar cualquier cosa

- Pero tampoco tiene sentido trazar todas y cada una de las variables y parámetros, esto añadiría demasiada confusión tanto al código fuente como a la traza
- En cada caso debemos decidir qué trazar