Anteproyecto de calculadora

Jorge Alonso*

Vigo, 07/2005 — v2.0.0

Índice

1. Introducción

Hace unos... ¿5? años, Marcos Castro quería programar una calculadora sencillita en la que se pudiese definir funciones. Realicé entonces un escrito donde definía su sintaxis, que resultó un poco farragosa. Fue uno de tantos proyectos que no llegó a ningún lado.

Repensando en él, lo he simplificado bastante, por si algún día alguien se anima. Más que otra cosa, se trata de apuntes, no de una definición exhaustiva.

*Mi correo es `soidsenatas@yahoo.es`, y mi página web es `http://es.geocities.com/soidsenatas/`.

2. Sintaxis

2.1. General

La calculadora ejecutará las típicas expresiones matemáticas, como por ejemplo:

```
328*COS(34) + (1768^2 - 483)/691
```

El resultado es impreso en la pantalla, y a la vez almacenado en la variable interna `RESP`, para poder utilizarlo. Con cada nuevo cálculo, esta variable es actualizada.

Se trabaja con números reales, que pueden expresarse en notación científica:

```
548319.13269e-48
```

No distingue entre mayúsculas y minúsculas, ni letras con o sin acentos, pero respeta la notación primera de las expresiones. Los espacios son ignorados, pero son necesarios para definir alguna sintaxis especial, como se verá.

Si, durante la ejecución de un cálculo, se pulsa la tecla de escape, el cálculo es abortado.

2.2. Valores iniciales

El programa tiene una serie de valores y comportamientos iniciales por defecto, algunas de las cuales pueden alterarse pulsando las teclas de función del teclado.

Al iniciarse (y al reiniciarse), el programa lee un archivo de configuración, que puede editarse para fijar sus características iniciales. También, puede incluir la definición de funciones especiales. Además, existe una orden que permite seleccionar y ejecutar otros archivos de configuración para, por ejemplo, ampliar su repertorio de constantes y funciones:

```
CARGAR("archivo.calc")
```

Dentro de un archivo, los saltos de línea se consideran equivalentes a un espacio.

2.3. Separadores

Hay tres tipos de separadores.

El separador decimal, que se utiliza para separar la parte entera de la decimal de los números. Por defecto, se utiliza un punto, pero puede cambiarse a la coma, y también a que ambos signos sean válidos.

El separador de parámetros, utilizado en las llamadas a las funciones. Por defecto es la coma, pero puede cambiarse al punto y coma.

El separador de instrucciones, que es por defecto el punto y coma, pudiendo cambiarse a dos puntos.

Un mismo signo sólo servirá como un tipo de separador.

2.4. Operadores

El programa admite múltiples operadores, de los que se da aquí una visión general.

Además de los típicos `+ - * /`, ya se ha visto que admite el de la potenciación `^` (que también puede escribirse como `**`). Para el resto de la división (entre enteros) se utiliza `\` o `MOD`. El signo `%` simplemente divide el número anterior entre 100, y `!` calcula su factorial.

Los condicionales son `== > < >= <= <>`, que admiten otras formas de escritura.

Para asignar valores a una variable, además del signo igual, pueden simplificarse expresiones del tipo `x=x^y` a `x^=y`

Para sumar 1 a una variable basta con utilizar `++` (`x++`), y para restar 1 con `--`.

También se dispone de operadores lógicos `&` para Y, `|` para O y `NOT` para la negación.

2.5. Correcto e incorrecto

Las constantes internas `CORRECTO` e `INCORRECTO`, se emplean en los condicionales para indicar si son ciertos o no, y en las respuestas de algunas órdenes y funciones, para indicar si han hecho lo que se esperaba de ellas o no.

El valor de `INCORRECTO` es cero.

2.6. Múltiples instrucciones

Pueden indicarse simultáneamente varias instrucciones:

```
156*459; RESP/315
```

Por defecto, sólo se muestra el resultado final, y no los intermedios.

Para agrupar varias instrucciones como si fuese una sola, se usan las llaves:

```
MEDIA(12, {156*459; RESP/315}, 644)
```

El resultado del grupo es el del último cálculo efectuado en él.

2.7. Multiplicación abreviada

En algunas ocasiones, puede obviarse el signo de multiplicación. La expresión:

```
541+415(23+86)(458-47COS(19))/23PI
```

Es equivalente a:

```
541+415*(23+86)*(458-47*COS(19))/(23*PI)
```

Obsérvese que la multiplicación abreviada tiene precedencia sobre la división.

2.8. Paréntesis omitidos

La expresión:

```
COS 13 + COS^2 13 + COS(13)^3
```

Es equivalente a:

```
COS(13) + (COS(13))^2 + (COS(13))^3
```

Obsérvese que en el primer coseno se ha requerido del uso de un espacio entre el nombre de la función y su parámetro, y en el segundo, entre el exponente y el ángulo.

También pueden colapsarse paréntesis mediante corchetes. La expresión:

```
1+(52+78(16-48(25+76)))/43
```

Puede simplificarse a:

```
1+(52+78(16-48(25+76]/43
```

2.9. Unidades angulares

Por defecto, el programa trabaja con radianes. Para indicar grados sexagesimales, basta con utilizar la notación:

```
48º125’7.86"
```

Para indicar los segundos, pueden utilizarse también dos comillas simples.

Es posible alterar este comportamiento, y cambiar la unidad angular con que se introducen y muestran los resultados.

Existen constantes internas (bueno, propiamente no se las puede llamar constantes) que funcionan como multiplicadores de conversión. Por ejemplo, para conocer el coseno de la suma de 3π radianes y 2 revoluciones y media, basta con indicar:

```
COS(3 PI RAD + 2.5 REV)
```

Y esta expresión es independiente de qué unidad angular esté seleccionada.

2.10. Variables y constantes

Por ejemplo, para definir la variable `x`, con valor inicial 7813, basta con escribir `x=7813`. Una vez que se define una variable, ésta permanece en la memoria de la sesión actual del programa, conservando su valor actual. También es posible borrarla en cualquier momento de la memoria.

Otra forma es que simplemente aparezca en un cálculo. Si está previamente definida, el programa utiliza su valor actual. En caso contrario, suspende el cálculo, y pide al usuario que se la defina. Puede cambiarse este comportamiento, de forma que tome cero como el valor por defecto.

Para forzar a que el programa pida que se introduzca el valor de una variable, se utiliza `VAR`. Admite un segundo parámetro, que será el valor que se sugiere para esa variable:

```
VAR(x, 7813)
```

Para definir una constante, por ejemplo `f` con el valor `145.541`, hay que utilizar la orden `CTE`:

```
CTE(f, 145.541)
```

El nombre de una variable o de una constante (o de una función) ha de empezar obligatoriamente por una letra (incluyendo el signo de subrayado), pero después puede incluir números.

2.11. Condicionales

La función condicional `SI` tiene múltiples sintaxis.

Si la condición `x>56` es cierta, ejecuta `x-56`:

```
SI(x>56, x-56)
```

Ídem, pero si es falsa, ejecuta `x+56`:

```
SI(x>56, x-56, x+56)
```

Si la condición `y<23` es cierta, ejecuta `y+47`; si es falsa, entonces si `x=94` ejecuta `y=x-16`:

```
SI(y<23, y+47, x=94, y=x-16)
```

Ídem, pero si ninguna expresión es correcta, ejecuta `x=y/2`:

```
SI(y<23, y+47, x=94, y=x-16, x=y/2)
```

Y así sucesivamente.

2.12. Bucles

Los bucles son muy primitivos, utilizándose una etiqueta y un salto condicional.

Las etiquetas se indican con `ETIQUETA`, cuyo parámetro es un número identificativo. La orden `SALTO` tiene dos parámetros, el primero una condición, y el segundo un número identificativo de una etiqueta; si la condición es cierta, el flujo de ejecución de las instrucciones salta a esa etiqueta.

Por ejemplo, para calcular la suma de los cuadrados de los número enteros desde 75 hasta 123:

```
n=75; s=0; ETIQUETA(1); s += n^2; n++;
SALTO(n<=123, 1); s
```

También es posible marcas las etiquetas con cadenas de texto (sin olvidarse entonces de encerrarlas entre comillas dobles).

2.13. Definición de funciones

Por ejemplo, para definir la función `NORMA`, que admite dos parámetros, llamados `a` y `b`, y que devuelve el valor `(a*a+b*b)^2`, se escribe:

```
FUNCIÓN(NORMA, a, b, (a*a+b*b)^0.5)
```

Los parámetros que recibe la función, y los que se definen dentro de ella, son locales a la propia función, y no interfieren con otras que puedan tener el mismo nombre.

Otra forma de definirla es:

```
FUNCIÓN(NORMA, 2, (#1*#1+#2*#2)^0.5)
```

Donde el `2` indica el número de parámetros, que se representan respectivamente por `#1` y `#2`.

`FUNCIÓN` devuelve el valor `CORRECTO` si ha podido crearla.

Dos funciones pueden tener el mismo nombre, pero han de tener distinto número de parámetros. Así, se puede definir sin problemas:

```
FUNCIÓN(NORMA, a, b, c,
(a*a+b*b+c*c)^0.5)
```

Si una función no tiene ningún parámetro, hay que indicarlo explícitamente con el valor cero.

Si tiente un número indefinido de parámetros, no se indican nada. Dentro de la función hay que emplear las expresiones numéricas precedidas por # (#1, #2, #3...), siendo ## el número total de parámetros que ha recibido. Puede seleccionarse un elemento cualquiera mediante la función #(), por ejemplo #(i) (que puede simplificarse a #i).

La ampliación de la función NORMA sería:

```
FUNCIÓN(NORMA, {s=0; i=1; ETIQUETA 1; s
+= #i #i; i++; SALTO(i<=##, 1); s^0.5})
```

Se admiten funciones recursivas (funciones que se llaman a sí mismas).

Dentro de las funciones puede utilizarse la orden COM para indicar comentarios, y TEXTO para que muestre en pantalla textos o resultados:

```
...; COM("Inicio del bucle principal");
ETIQUETA 1; ...; TEXTO("El valor", n,
"no es válido"); ...
```

Por último, para terminar inmediatamente la ejecución de una función, devolviendo un determinado valor, se utiliza RETORNA:

```
...; RETORNA(s^0.5); ...
```

2.14. Bases de numeración

Para convertir el número hexadecimal 2e5a1 a base 10, se escribe:

```
BASE("2e5a1", 16)
```

El número ha convertir siempre ha de estar entre comillas dobles.

Y para pasar 2398 a base 2, se escribe:

```
CONV(2398, 2)
```

El resultado es un texto, no un número.

2.15. Opcional: Macros

De forma opcional, el programa podría incluir macros. Es decir, las instrucciones a ejecutar se preprocesan tratándolas como un texto, en el que se sustituyen subcadenas de texto por otras subcadenas de texto.

Por ejemplo, suponiendo que el programa sólo admita <>, y se quiera que acepte escribirlo también como ><, basta con:

```
MACRO("><", "<>")
```

3. Ejemplo

Para terminar, veamos una función que resuelve la ecuación de segundo grado, imprimiendo las soluciones en pantalla, y devolviendo INCORRECTO si se trata de soluciones imaginarias (escrito tal y como iría dentro de un archivo a cargar):

```
FUNCIÓN(ec2g, a, b, c, {

COM("a x^2 + b x + c = 0");
SALTO(d = b b - 4 a c < 0, "i");

d ^= 0.5;
TEXTO("Soluciones:")
TEXTO((-b+d)/2a);
TEXTO((-b-d)/2a);
RETORNA(CORRECTO);

ETIQUETA("i");
TEXTO("Soluciones imaginarias:");
d = (-d)^0.5 / 2 ABS(a);
e = -b/2a;
TEXTO(e, "+ i", d);
TEXTO(e, "- i", d);
INCORRECTO

})
```