Chipmunk BASIC 1.0
David Gillespie
Versión XControl por Oliver Teuber

---

## COMANDOS

**LIST** línea(s)

Lista la líneas especificadas del programa. Por ejemplo,

```
LIST 10, 100-200
```

lista la línea 10, y de la 100 a la 200, inclusive.

**RUN** [línea]

Empieza la ejecución del programa desde la primera línea, o desde la línea especificada. Todas las variables se borran.

**RUN** fichero[,línea]

Carga y ejecuta un programa. Por ejemplo,

```
RUN "FOO", 30
```

carga el programa del fichero FOO.TEXT y comienza su ejecución en la línea 30.

**NEW**

Borra un programa de la memoria.

**LOAD** fichero

Carga un programa en la memoria. El programa que anteriormente estuviese en la memoria será borrado. El nombre del fichero debe de estar entre comillas, la extensión .TEXT se añade de manera automática. El fichero es un fichero ASCII que contiene el listado del programa. Todas las líneas del fichero deben de empezar con un número de línea, pero no hace falta que estos se incrementen en orden.

**MERGE** fichero

Se carga un programa en memoria. El programa anterior permanece en memoria; si existe una línea en ambos programas la más nueva es la que permanece.

**SAVE** fichero

Almacena el programa en memoria en un fichero.

**BYE**

Vuelve al sistema operativo.

**DEL** línea(s)

Borra las líneas especificadas del programa. Los números de línea deben de estar separados por comas o guiones, como en una lista. Si se usa dentro de un programa, DEL terminará la ejecución del programa solo si borra la línea en la que aparece.

**RENUM** [inicio[,incremento]]

Renumera las líneas del programa. Por defecto, la nueva secuencia sería 10, 20, 30,... El primera argumento es el número inicial de línea; el segundo argumento es el incremento que habrá entre las líneas.

## DECLARACIONES

**REM** comentario

Un marcador; es ignorado. Los comentarios pueden contener cualquier carácter con la excepción de que REM no puede ir inmediatamente seguido por un carácter alfanumérico.

**[LET]** variable = expresión

Asigna un valor a una variable. Los nombres de variables pueden tener una longitud de 20 caracteres, consistente de letras en mayúsculas o minúsculas, números, caracteres de subrayado y el símbolo dolar. Los nombres de variables son sensibles a las mayúsculas/minúsculas. Las variables pueden normalmente contener números reales o cadenas hasta de 255 caracteres si su nombre termina en $.

Ejemplos:

```
LET X=20
X$="FOO"
X$=X$+"BAR"
```

**DIM** variable(dimensiones), ...

Localiza memoria para matrices. Las matrices pueden tener hasta 4 dimensiones, con un rango que va desde el 0 hasta el especificado en la declaración DIM. El mismo nombre no debe de ser usado tanto para una variable simple como para una matriz.

Si se usa una matriz antes de su dimensionamiento, este es declarado a 10.

Ejemplo:

```
INPUT "Cuantos elementos? "; x
DIM matriz(x,1)
FOR i=1 TO x : INPUT matriz(i,0), matriz(i,1) : NEXT
```

**PRINT** items

Imprime los items en pantalla. Los items pueden ser tanto numéricos como expresiones de cadena y pueden ir separados por comas, puntos y coma o por nada.

Los números normalmente terminan en espacios. Para evitar este espacio convierte el número a cadena con STR$.

La línea es terminada con un CR/LF, a menos que la lista de items termine en coma o punto y coma.

La palabra PRINT puede ser abreviada con un cierra interrogante.

Ejemplos:

```
PRINT "1+2=", 1+2
PRINT X$ "=" Z$;
? x; y+z
```

**INPUT** [indicador;] variables

Si se pone un indicador, este es imprimido. De otra forma se imprime un interrogante. Entonces el ordenador espera que se introduzca el valor de cada variable. Si hay más de una variable, sus nombres deben separarse por comas.

Si las variables son numéricas, sus valores deben ser introducidos en distintas líneas o combinados con comas. Cualquier expresión numérica es una respuesta válida.

Si las variables son cadenas, cada cadena es escrita en una línea separada. El carácter imprimido es copiado literalmente en la cadena.

No se pueden mezclar variables de cadena o numéricas en un declaración INPUT.

Ejemplos:

```
INPUT X$
INPUT "Escribe 3 n£meros: "; X, Y, Z
```

**GOTO** línea

Comienza la ejecución de la declaración de la línea especificada. El número de línea puede ser una expresión numérica.

Si lo prefieres puedes usar palabra GO TO en vez de GOTO.

**IF** condición **THEN** línea/declaraciones **ELSE** línea/declaraciones

Si la condición se cumple (por ejemplo, la expresión numérica tiene un valor distinto de cero), las declaraciones que siguen a la palabra THEN son ejecutadas. De otra forma, las declaraciones que siguen la palabra ELSE son ejecutadas. Si no hay una cláusula ELSE se ejecuta la siguiente línea del programa.

Se puede usar un número de línea después tanto del THEN como del ELSE, de esta forma se implementa una declaración GOTO.

**END**

Termina el programa. La declaración END no es necesaria.

**STOP**

Termina el programa con un mensaje identificativo: "Break".

**FOR** variable = primero **TO** último [**STEP** incremento]
{declaraciones}
**NEXT** [variable]

Ejecuta {declaraciones} repetidamente mientras que la variable cuenta del "primero" al "último", incrementandose de 1 en 1, o por el valor determinado en STEP. Si el valor en STEP es negativo, la variable se decrementa.

Si "primero" es mayor que "último" (o menor si el STEP es negativo), la ejecución procede directamente a la declaración NEXT, sin llegar a ejecutar el cuerpo del bucle.

El nombre de la variable es opcional en la declaración NEXT.

**WHILE** [condición]
{declaraciones}
**WEND** [condición]

Ejecuta {declaraciones} repetidamente hasta que la condición WHILE (si es dada) se convierte en falsa, o hasta que la condición WEND se convierte en cierta. Esta estructura puede emular el WHILE-DO y REPEAT-UNTIL del Pascal o ambas a la vez. Si no se da ninguna condición el bucle no se termina a no ser que se use un endiablado GOTO.

**GOSUB** línea
**RETURN**

Ejecuta las declaraciones empezando en la línea especificada, entonces cuando se encuentra un RETURN hace que se vuelva a la declaración que había detrás del GOSUB.

**READ** variables
**DATA** valores
**RESTORE** línea

Lee valores numéricos o cadenas de la declaración DATA. La lectura empieza en la primer declaración DATA y continua hasta la última. Leer más allá del último DATA genera un error.

Los valores DATA pueden ser tanto numéricos como cadenas, de acuerdo con el tipo de variable a ser leída. Leer el tipo erroneo de expresión genera un Syntax Error.

La declaración RESTORE hace que el siguiente READ reuse la primera declaración DATA en el programa, o el primer DATA después de una línea en particular.

**GOTOXY** columna, fila

Mueve el cursor a la posición de pantalla indicada, los valores son entre 0,0 y 79,23.

**ON** expresión **GOTO** línea, línea, ...
**ON** expresión **GOSUB** línea, línea, ...

Si el valor de la expresión, redondeado como un entero, es N, va al N-avo número de línea de la lista. Si N es menor que uno o demasiado grande, la ejecución continua en la siguiente declaración después del ON-GOTO o ON-GOSUB.

**POKE** dirección, dato

Almacena un byte en la dirección especificada.

## EXPRESIONES NUMERICAS

x **AND** y

AND lógico de dos enteros.

x **OR** y

OR lógico de dos enteros.

x **XOR** y

XOR lógico de dos enteros.

**NOT** x

Complemento lógico de un entero.

x+y, x-y, x*y, x/y, x^y, -x

Operaciones aritméticas típicas de coma flotante.

x=y, x<y, x>y, x<=y, x>=y, x<>y

Comparaciones; resultado 1 si es verdad, 0 si es falso.

x **MOD** y

Módulo de dos enteros.

**SQR** x

Cuadrado de X. Los paréntesis no son requeridos si el argumento de la función es una entidad simple; por ejemplo SQR SIN X no necesitaría paréntesis, pero SQR(1+X) si.

**SQRT** x

Raiz cuadrada de X.

**SIN** x, **COS** x, **TAN** x, **ARCTAN** x

Funciones trigonométricas típicas, en radianes.

**LOG** x, **EXP** x

Logaritmo natural y e a la potencia de X.

**ABS** x

Valor absoluto de X.

**SGN** x

Signo de X: 1 si X es positiva, 0 si es cero y -1 si es negativa.

**VAL** x$

Valor de la expresión contenida en la cadena X$. Por ejemplo, VAL "1+2" da 3. X$ puede ser un literal simple, variable o función, o una expresión de cadena en paréntesis.

**ASC** x$

Código ASCII del primer carácter en X$, o 0 si X$ es null.

**LEN** x$

Número de caracteres en X$.

Prioridad:

```
Paréntesis
Funciones  (incl. NOT y menos simples)
     ^
 *, /, MOD
   +, -
=, <, >, <=, >=, <>
    AND
  OR, XOR
```

## EXPRESIONES DE CADENA

"string" o 'string'

Cadena literal. Las comillas simples son convertidas a comillas dobles internamente.

x$+y$

Concatenación. El resultado debe ser de 255 caracteres o menos.

x$=y$, x$<y$, x$>y$, x$<=y$, x$>=y$, x$<>y$

Comparación de cadenas; resultado 1 si es cierto y 0 si es falso.

**STR$**(x)

El número X expresado como una cadena de dígitos. No se ponen máscaras ni espacios, la notación científica se usa si el valor absoluto es mayor a 1E12 o menor que 1E-2.

**CHR$**(x)

El carácter cuyo código ASCII es X.

**MID$**(x$, y)
**MID$**(x$, y, z)

(Paréntesis obligatorios) La subcadena consistente de los Z caracteres empezando por el Y de la cadena X$. La posición 1 es el primer carácter de la cadena. Si se omite Z se usa por defecto 255, por ejemplo la parte derecha entera de una cadena.

## CONVENCIONES

Se pueden escribir distintas declaraciones en una línea, separadas por dos puntos:

```
10 INPUT X : PRINT X : STOP
```

Actualmente no hay diferencia entre comandos y declaraciones; ambas pueden ser usadas tanto dentro como fuera de programas. Algunos comandos, como NEW, por supuesto, detendrán la ejecución del programa.

Los números de línea pueden ser un entero de 1 a MAXINT.

Para borrar una línea usa DEL, o escribe el número de línea solo:

```
10
```

Pulsar CLR I/O para detener la ejecución del programa. [Esto no se soporta en la traslación de p2c] Para salir del BASIC usar el comando BYE.

Las palabras reservadas deberían escribirse todas el mayúsculas o minúsculas; siempre son convertidas en mayúsculas de forma interna. Los espacios son ignorados salvo cuando están entre comillas. Los corchetes son convertidos a paréntesis. Las comillas que faltan al final de una línea son añadidas de forma automática, como en el comando:

```
SAVE "PROGRAM
```