GST Computer Systems Limited

CONTENIDO

1 INTRODUCCION

QC es una versiòn de C especialmente producida para el Sinclair QL. Es el subconjunto del lenguaje standard C descrito por Kernighan y Ritchie con los siguientes extras:

sentencias switch, for, do y goto
Operadores lògicos && ññ
Operadores unitarios ñ *.
Expresiones con coma.
Operadores de asignaciòn,
Enteros cortos y largos,
Variables locales inicializables
Variables y funciones estàticas.

Todo el lenguaje se describe con màs detalle en la secciòn 3, y sus diferencias principales con el standard en el apèndice D.

Se recomienda leer completamente èste manual antes de usar el compilador.

1.1 Objeto de este manual.

El "QC USER MANUAL" facilita una introducciòn al compilador de C del Sinclair QL, incluyendo toda la informaciòn precisa para utilizar el compilador sin pretender enseñar el lenguaje.

1.2 Convenciones usadas en este manual.

Cuando se mencione una palabra clave de QC, operadores o variables apareceran en negrilla. Las MAYUSCULAS se usaràn para palabras-clave de superBASIC o ficheros QDOS.

Cuando se describa el lenguaje en la secciòn 3, los ejemplos de sintaxis llevaràn las palabras-clave en negrilla y los elementos a proporcionar en parèntesis angulados Ej:

do (<sentencia>) **while** (<sentencia>);

Las rutinas de libreria descritas en la secciòn 4 , todos tienen la declaraciòn de comienzo de funciòn sus paràmetros y sus tipos de paràmetros. Todo esto aparece en negrilla.

El uso de la palabra ¡módulo¡ en este manual, se refiere a un grupo de subrutinas compiladas juntas de una vez, sin importar cuantos ficheros vayan a usarlas. Puede referirse tanto al fichero fuente en ensamblador, al fichero binario relocalizable o al fichero(s) fuente en C.

1.3 Lista de componentes del QC.

Se facilita el compilador QC bon los siguientes elementos:
Dos cartuchos de microdrive con el compilador, ensamblador, linker, librerias y programas ejemplo.
Dos cartuchos blancos para copia de seguridad.
Un ejemplar de "A book on C" de Berry & Meekings.
Un cuaderno A5 con èste manual.

La cinta de microdrive etiquetada como QC1 contiene:

QC	El compilador QC
QCASM	El ensamblador QC (paso 2 de compilaciòn)
LINK	El linker de QC
QC_LINK	El fichero de control del linker
BACKUP	Un programa para copiar este cartucho.

La cinta de microdrive etiquetada como QC2 contiene:

QC_LIB	La libreria QC standard con las rutinas descritas en la secciòn 4 de este manual
QDOS_LIB	Las libreria de rutinas extra del QDOS

definidas en la sección 5 de este manual
STDIO.H El fichero de cabeceras standard de entrada-salida.(ver 4).
COMPILE Un programa en C para gestionar el compilador, ensamblador y el linker (ver 2.4).
WINDOW_MGR Un programa en C para ajustar las ventanas de omision del resto de programas.
COMPILE_C Listado fuente del programa COMPILE.
BACKUP Un programa para copiar este cartucho.

Para tener suficiente espacio para programas, se deben copiar los ficheros de libreria y el de cabeceras en un microdrive el cual tendrà espacio para sus programas. Copie el resto de ficheros a otro microdrive ya que se usaràn menos frecuentemente.

1.4 Que va a necesitar para usar el compilador y escribir programas C.

Este compilador funciona en cualquier QL sin expansiòn de memoria, pero para crear los ficheros fuente precisarà un editor de texto como el programa ED del Sinclair Assembler.

Si desea escribir grandes programas, los floppy disk seràn de utiliddad dado su espacio extra de almacenamiento, pero los microdrives cumplen bien con programas de unos pocos centenares de lineas.

Si no tiene un editor de texto puede usar QUILL para crear los ficheros fuentes. Los ficheros producidos directamente por QUILL no los puede leer el compilador, primero ha de imprimirlo a un fichero de impresiòn usando un controlador de impresora adecuado (intente con longitud de pàgina 0, longitud de linea 80, separador de linea LF sin ningùn preàmbulo o postàmbulo).

1.5 Haciendo copias de seguridad.

Es muy importante que haga copias de seguridad de las cintas originales antes de utilizarlas. El programa BACKUP suministrado en ambas puede usarse. Le sugerimos que utilice la nueva copia para ejecutar el compilador y guarde las originales como copias.

Para ejecutar BACKUP ponga la cinta a copiar en MDV1 y una en blanco en MDV2 y entonces introduzca el comando

```
EXEC_W MDV1_BACKUP
```

El programa preguntarà por el nombre de directorio de fuente y de destino. Si presiona ENTER asumirà que la fuente es MDV1 y el destino MDV2, mientras preguntarà si desea foormatear el destino, de no hacerlo preguntarà si quiere sobregrabar los ficheros que ya existan. Esta opciòn es útil para hacer copias de seguridad de las cintas de trabajo. Entonces copiarà la cinta.

1.6 Otros libros y manuales de utilidad.

The C programing language
B.W.Kernighan y D.M.Ritchie (Prentice-Hall).
A C Reference Manual
S.P.Harbison y C.L.Steele (Prentic-Hall).
The C Programing Tutor
L.A.Wortman y T.O.Sidebottom (Prentice-Hall).

Nòtese que algunos de los programas de ejemplo de estos libros no correràn con QC dado que alguna caracteristica no la soporta el compilador.

1.7 Responsabilidades.

Bajo ninguna circustancia GST Computer Systems Limited se hace responsabble de cualquier daño directo o indirecto, incluyendo pèrdida de datos o contratos por cualquier error o fallo del software del compilador QC.

GST Computer Systems Limited

GST Computer Systems Limited tiene una politica de desarrollo y mejoras constante de sus productos reservandose el derecho dee modificar software o manuales en cualquier momento sin notificaciòn.

1.8 **Derechos y Marcas Registradas.**

El compilador QC en cartucho microdrive junto con el QC User Manual poseen el Copyright (C) 1984, GST Computer Systems Limited.

QC es una marca registrada de GST Computer Systems Limited.
QLKit y QL Toolkit son marcas registradas de QJUMP Limited.
QL, QDOS y Microdrive son marcas registradas de Sinclair Rese#rch Limited.
UNIX es una marca registrada de los laboratorios Bell.

2 COMO USAR EL COMPILADOR.

El compilador QC toma programas fuente de C (creados con un editor de texto) y los convierte en un fichero de texto en ensamblador. Este debe ser ensamblado usando el ensamblador QC (O el Sinclair Macro Assembler). Esto produce un fichero binario relocalizable que ha de combinarse con la libreria de tiempo de ejecución mediante el linker para crear un programa ejecutable.

2.1 Compilador.

El programa compilador se llama QC. Ponga la cinta marcada QC1 (o una copia de ella) en MDV1 y su cinta de datos (que es creada con una copia de QC2) en MDV2, entonces introduzca el comando

EXEC_W MDV1_QC

El compilador se inicializará y pedirá una linea de comando. Ahora puede teclear el nombre del fichero del programa C que quiera compilar (esta es la linea de comando más simple).

Si el nombre de fichero que usted da es MDV2_MIPROGRAMA el compilador intentará primero buscar un fichero MDV2_MIPROGRAMA_C y de no encontrarlo buscará el fichero MDV2_MIPROGRAMA.

El compilador producirá un fichero de salida nombrado como el de fuente pero con la extensión MDV2_MIPROGRAMA_ASM.

Usted puede poner más nombres de fichero en la linea de comando en cuyo caso todos son leidos como fichero de entrada y el nombre del primero se usará para dar nombre al fichero de salida.

También puede incluir algunas opciones en la linea de comando. Todas ellas comienzan con un guión, seguido de una letra (mayúscula o minúscula):

-M monitor: Escribe la primera linea de cada función en la pantalla como es compilada.

-A alarm: El compilador producirá un sonido cada vez que imprima un mensaje de error en la pantalla

-P pause: Después de imprimir un mensaje de error en la pantalla el compilador esperará a que presione ENTER antes de continuar

-C comments: El programa fuente de C es incluido en el fichero de salida en ensamblador como comentarios.

-D <dir> directory: El directorio especificado es localizado para incluir sus ficheros. Cualquier unidad o nombre de directorio hay que especificarla aquí. Nótese que el parámetro es obligatorio en éste comando.

-L <name> listing: El listado de salida del compilador se envia al fichero o dispositivo especificado en el nombre <name>.

Si se proporciona una linea en blanco , el compilador solicitará una nueva linea de comando. Si no se le proporciona ningun fichero, el compilador tomará los datos introducidos por teclado como fuente de compilación y enviará su salida a pantalla.

Si se detecta algún error de compilación, QC enviará un mensaje de error completo al QDOS.

2.2 Ensamblador.

La salida del compilador es en forma de fuente de ensamblador. Se incluye un ensamblador con el compilador. Pongale en marcha tecleando:

EXEC_W MDV1_QCASM

Cuando el ensamblador solicite una linea de comando, introduzca el nombre de su programa (la extensión _ASM puede omitirse) ej:

MDV2_MIPROGRAMA

Esto generará un fichero en binario llamado de igual modo que el fuente pero con la extensión _REL (MDV2_MIPROGRAMA_REL).

La línea de comando del ensamblador consta de dos partes. Primero vienen una serie de paràmetros posicionales (los paràmetros se interpretan por su posiciòn en la lista). Pueden haber desde uno a tres paràmetros posicionales todos nombre de fichero. Estos van seguidos de las opciones. Una opciòn es una palabra que comienza con un guiòn y puede que haya de estar seguida de un nombre de fichero como paràmetro.

Los paràmetros posicionales son :

Primero	**Fichero fuente.**
Segundo (opcional)	**Fichero de listado.**
Tercero (opcional)	**Fichero binario.**

Las opciones son:

-NOLIST	No produce fichero de listado.
-ERRORS <Nombre de Fichero>.	Envia los mensajes de error y la línea errònea al fichero nombrado. Esta opciòn conecta la opciòn **-NOSYM.**
-LIST <N.de fichero>	Envia el listado al fichero especificado.
-NOBIN	No produce ninguna salida binaria relocalizable.
-BIN <N.de fichero>	Envia la salida binaria relocalizable al fichero especificado.
-NOSYM	No produce la tabla de simbolos ni la de referencia cruzada.
-SYM	Produce la tabla de simbolos y la de referencia cruzada
-NOLINK	Produce còdigo binario absoluto en vez de relocalizable.

Los nombres de fichero de estas opciones son todos opcionales, pero de especificarse se superponen a los de los ficheros de la lista de paràmetros posicional. Si en ninguno de los dos lugares aparece especificado un nombre de fichero, se construye un nombre por defecto con el nombre del fichero fuente y las extensiones _LIS para los ficheros de listado y _REL para los ficheros binarios relocalizables.

Por defecto, el ensamblador QC no produce un fichero de listados. Si se selecciona uno se incluirà por defecto una tabla de simbolos.

Si el ensamblador detecta algùn error, èste enviarà al QDOS un mensaje de "no completado". Funciones y variables son chequeadas con el mismo nombre que los registros del procesador 68000. Usted puede ignorar todos los avisos provinientes del ensamblador.

2.3 Linkador o linker.

El linker se utiliza para combinar sus programas con la librería de tiempo de ejecuciòn. Existe un fichero de control del linker en QD1 llamado QC_LINK. Para correr el linker escriba:

```
EXEC_W MDV1_LINK
```

entonces el linker solicitarà una línea de comando, teclee el nombre del fichero binario (la extensiòn _REL puede omitirse) seguido del nombre del fichero de control y cualquier otra opciòn del linker, ej:

```
MDV2_MIPROGRAMA  MDV1_QC_LINK -NOLIST
```

Esto producirà un programa MDV2_MIPROGRAMA_BIN que puede ser ejecutado usando EXEC o EXEC_W.

Nòtese que el fichero de control del linker asume que la libreria standard QC_LIB se halla en MDV2. Para modificarlo, tendrà que editar el fichero de control del linker.

La línea de comando del linker consta de dos partes, como la del ensamblador. Primero vienen una

serie de paràmetros posicionales (cada paràmetro depende de su posiciòn en la lista). Puede haber hasta 4 paràmetros posicionales. Estos van seguidos de las opciones que comienzan por un guiòn y pueden seguirse de un nombre de fichero como paràmetro.

Los parametros posicionales son:

Primero	**Nombre de fichero binario relocalizable**
Segundo	**Nombre del fichero de control del linker.**
Tercero	**Nombre del fichero de listado.**
Cuarto	**Nombre del fichero de programa.**

Las opciones son:

-WITH <N. de fichero>	Esto identifica el fichero de control del linker si no se especifica fichero binario relocalizable en los paràmetros posicionales.
-NOPROG	No se produce fichero de programa.
-PROG <N. de fichero>	Genera un fichero de programa.
-NOLIST	No produce un fichero de listado.
-LIST <N. de fichero>	Envia un listado al fichero nombrado.
-NODEBUG	No genera un fichero de depurado.
-DEBUG <N.de fichero>	Genera un fichero de depurado.
-NOSYM	No produce ninguna tabla de simbolos ni de referencia cruzada.
-SYM	Produce una tabla de simbolos.
-CRF	Produce una tabla de referencias cruzadas.
-PAGELEN <número>	Especifica el número de lineas por pàgina en el listado.

Los nombres de fichero especificados en las opciones, son todos opcionales, pero de especificarlos todos se superponen a los de los paràmetros posicionales. Si un nombre de fichero no es especificado en ninguno de los dos lugares, se crearà un ficnero por defecto construido desde el fichero fuente con las extensiones: _MAP para el listado, _DEBUG para el fichero de depurado y _BIN para el fichero de programa.

Si existe algùn error o aviso provinientes del linker sobre la existencia de algùn simbolo indefinido o oefinido varias veces, no intente corregirlo modificando el listado de ensamblador, modifique mejor el del compilador.

2.4 El programa de control COMPILE.

Un programa llamado COMPILE se incluye para controlar el compilador, el ensamblador y el linker de modo conjunto. Preguntarà por el nombre del fichero a compilar y con ello formarà las lìneas de comando para cada programa. No hace falta que ponga la extensiòn _C. Para ponerlo en marcha teclee:

```
EXEC_W MDV2_COMPILE
```

Presione F1 p#ra seleccionar la opciòn de compilaciòn:

```
Name of file to compile?      MDV2_MIPROGRAMA
```

COMPILE tambièn permite ejecutar otros programas incluidos como el gestor de ventanas y el de copias de seguridad (BACKUP).

El fichero fuente de èste programa se incluye con el compilador como un programa ejemplo y para permitirle que lo modifique si lo desea. Si lo recompila, habrà de reajustar los requerimientos de memoria de datos para dejar suficiente memoria para el compilador (para ello puede usar el programa gestor de ventanas (ver 2.6).

2.5 Linkando programas màs complicados.

Para linkar programas màs complicados que soporten màs de un màdulo, tendrà que crearse su propio fichero de control del linkador. Si ustèd quiere especificar todos los nombres de màdulos en el fichero de control, tome el fichero de control suministrado (QC_LINK) y sustituya la linea "INPUT *" con una orden de input para cada uno de los màdulos de C que quiera linkar juntos. Entonces ejecute el linker con una linea dde comando como esta:

-WITH MDV2_MIFICHERO_CONTROL.

Alternativamente si todavia quiere nombrar algùn màdulo en la linea de comando del linker, deje la linea INPUT * (que le dice al linkador que lea el nombre de màdulo en la linea de comando) y prosiga con una orden INPUT para cada uno de los otros màdulos a linkar.

El fichero de control del linkador podria verse asi:

```
SECTION   S.HEADER
SECTION   S.CBASE
SECTION   S.GLOB
SECTION   S.CCODE
SECTION   S.BBASE
SECTION   S.RELOC
SECTION   S.TRAILER
INPUT     MDV2_MIPROGRAMA1_REL
INPUT     MDV2_MISRUTINAS_REL
INPUT     MDV2_OTROCODIGO_REL
LIBRARY   MDV2_QDOS_LIB
LIBRARY   MDV2_QC_LIB
DEFINE    LIB$ = EXIT
DEFINE    G$ = C_GLOBBA
DEFINE    M$ = C_ENDGLO + $8000
DATA              8K
```

Si tiene un programa complicado que use mucha memoria de pila, tedrà que modificar la linea DATA de èste fichero de control para que limpie el espacio de trabajo que requiera su programa. Si no usa ninguna de las rutinas descritas en la secciòn 5 de èste manual puede suprimir la linea que solicita la libreria de QDOS (LIBRARY QDOS_LIB) para que el linkado sea màs ràpido. Otras lineas del fichero de control no deben modificarse.

Se puede crear una libreria de subrutinas uniendo ficheros relocalizables juntos. Si este fichero mixto se incluye en el fichero de control con una instrucciòn INPUT, todos los màdulos se incluiràn en el programa final. De hacerlo con una instrucciòn LIBRARY sòlo se incluiran aquellos màdulos que sean solicitados por referencia externa de otros màdulos del fichero de control.

2.6 El programa gestor de ventanas.

Este programa se facilita para ajustar la ventana de omisiòn de cualquier programa escrito en C o cualquiera de las utilidades de la cinta.

```
EXEC_W MDV2_WINDOW_MGR
```

El programa solicitarà el nombre del programa a ajustar. La posiciòn y tamaño de la ventana se pueden ajustar de modo interactivo, asi como el colos del papel y la tinta. Se pueden definir las ventanas con una laguna superior reservada para escribir el nombre del programa.

El programa tambièn permite modificar el espacio de memoria de pila reservado. No deberà modificar èste dato en los programas suministrados.

Cuando usted ejecuta un programa con una ventana enmarcada, esta serà limpiada al inicializarse el

programa. Si la ventana no está delimitada se deja un espacio en el programa C hasta que sea escrito, lo que permite crear programas que no tengan definida una ventana de pantalla.

2.7 Uso del compilador con unidades de disco.

Si dispone de unidades de disco en su QL, puede pasar los programas a ellos para obtener un mejor rendimiento en velocidad y mayor espacio de almacenamiento.

Ninguno de los programas requiere modificación para ejecutarlos desde disco (el programa COMPILE comprueba que exista una unidad de discos FLP1 y de no haberla ejecuta los programas desde MDV1 por defecto.

Necesitará eso sí, modificar el fichero de control del linkador para que tome los datos de los ficheros de disco en vez de los de microdrive. Este es una fichero de texto ordinario que se puede modificar con cualquier editor.

3.0 El lenguaje QC.

Esta sección define las facilidades de lenguaje del compilador QC, no intenta enseñar el lenguaje C para ello se le recomienda lea el libro suministrado con el compilador o cualquiera de los recomendados en la introducción.

Los comentarios en C comienzan con /* y finalizan con */ no pudiendo estar contiguos.

3.1 Variables y tipos.

Las variables son los datos fundamentales manejados en un programa. Todas las variables en QC han de estar declaradas antes de usarse.

3.1.1 Nombres de variable.

Los nombres de variable se pueden formar con letras, números y el signo de subrayar "_", debiendo comenzar por una letra. Sólo son significativos los 8 primeros caracteres del nombre de variable.

Los nombres en mayúsculas y los en minúsculas son tratados por el compilador como diferentes. Por convención se usan las minúsculas para nombres de variable y las mayúsculas para nombres de constantes (ver 3.6.2).

Algunos nombres están reservados como las palabras-clave, los nombres que comiencen por "c_" (reservados para uso de la librería de tiempo de ejecución) y los nombres de los registros del 68000 (A0,D1, USP).

3.1.2 Declaraciones.

Las variables se pueden declarar globalmente (fuera de cualquier función) en cuyo caso son accesibles desde que se declaran hasta el final del módulo, o localmente en cuyo caso el uso de la variable está restringido para ese bloque o rutina.

Los tipos soportados por QC son:

int	32 bits de anchura
long	32 bits de anchura (la misma que int)
short	16 bits de anchura
char	8 bits de anchura
pointer	32 bits de anchura

Una declaración ha de especificar: un identificador de tipo de almacenamiento opcional, seguido de un especificador de tipo opcional seguido de una lista de nombres de variable.

Las variables globales pueden tener las siguientes tipos de almacenamiento.

(ninguno) La variable es accesible fuera del módulo mediante directivos extern.

extern La variable debe estar declarada (sin identificador de clase de almacenamiento) en algún otro módulo (ver 3.1.4)

static La variable es particular del módulo.

Los identificadores de tipo de almacenamiento son ignorados en las variables locales. Se permiten señalar los tipos **auto** y **register** que no son usados normalmente. Los nombres son aceptados por compatibilidad con otros compiladores. Los tipos de almacenamiento **static** y **extern** no son aceptados por QC en las variables locales.

Los tipos de dato **short** y **long** se pueden usar solos o seguidos de **int**.

Un asterisco antes de el nombre crea un apuntador a los objetos de ese tipo mientras que los paréntesis cuadrados se usan para definir matrices y encerrar el tamaño de la matriz.

Ejemplos:

```
int lower, upper, step;
long value;
short int word;
char c, buffer(512), *str;
```

Nòtese que cuando se define una matriz con N elementos éstos son accedidos usando los subìndices 0 a N-1.

3.1.3 Inicializando variables.

Las variables locales se pueden inicializar con cualquier expresiòn o valor salvo las matrices locales. Las variables globales char e int pueden inicializarse a valores constantes, ej:

```
int i = 0,
    j = (435*678)+123;
char c = '?';
```

las variables inicializadas o no, pueden mezclarse en la misma expresiòn. Las matrices globales pueden inicializarse dando los valores iniciales entre llaves:

```
short int array(10) = { 0, 1, 2, 3 }
```

El nùmero de los elementos de matriz inicializados puede ser menor que el tamaño de la matriz en cuyo caso el resto de la matriz no se inicializa. El tamaño de la matriz puede no ponerse, en cuyo caso el nùmero de elementos inicializados es tomao como tamaño de la matriz.

Una matriz de caracteres o un apuntador de caracter puede inicializarse con una constante de cadena, por ejemplo:

```
char
        *pointer = "Una cadena",
        string() = "Otra cadena";
```

En C las cadenas son matrices de caracteres nerminadas en un byte a cero.

3.1.4 Variables externas.

Las variables globales usadas en otros mòdulos, pueden referenciarse usando el calificador extern . Este no debe usarse dentro de funciones. La variable es tratada como si fuera una global y se mantiene accesible hasta el final del mòdulo.

Si una matriz es declarada externa, no debe especificar el tamaño de la matriz pero use un par de corchetes vacios.

Los indicadores de variable externa no deben inicializar valores, la variable se puede inicializar en el mòdulo que la declare.

Ejemplo:

```
extern int
        qwerty, *pointer,
        datarray();
```

3.1.5 Valores sin signo.

La palabra clave unsigned se puede poner al principio de cualquier declaraciòn para indicar que la variable debe usarse con nùmeros sin signo. Entonces se usarà artmètica sin signo en expresiones que contengan estas variables. Ejemplos:

```
short i;
unsigned short u;
i = 40000;          /* el desbordamiento es tratado como -25536 */
u = 40000;
i = i/2   /* el resultado es -12768 */
u = u/2   /* el resultado es 20000 */
```

char	valores en el rango desde -128	hasta	127
short	valores en el rango desde -32768 hasta		32767
short	valores en el rango desde -2147483648	hasta	32767
unsigned char	valores en el rango desde 0	hasta	255
unsigned short	valores en el rango desde 0	hasta	65536
unsigned int	valores en el rango desde 0	hasta	4294967295

3.2 Operadores y expresiones.

3.2.1 Constantes.

Las constantes aceptadas por QC son:
números decimales ej: -1, 76890, 1000000000. -999
números octales ej: 012, 0377
(comienzan con un dígito cero).
números hexadecimales ej: 0x0a, 0xFF
(comienzan con 0x).
Constantes de caracter ej: 'a', 'DE'
Constantes de cadena ej: "Yo soy una cadena"

Nótese que las constantes de caracter actualmente generan valores int, y pueden tener uno o dos caracteres en ellos, mientras que una constante de cadena es actualmente una matriz de caracteres terminado por un byte nulo. El valor de una constante de cadena es la dirección de la matriz.

En las constantes y de caden y de caracter se interpreta el backslash "Ñ" de un modo especial para indicar ciertos caracteres.

ÑnCbdigo de nueva línea.
ÑfCbdigo de paso de página ASCII (form feed)
ÑtCbdigo ASCII de tabulación (TAB)
ÑbCbdigo ASCII de espacio atràs (BS)
Ñ'Comilla simple (para uso en costante de caracter).
Ñ"Comillas dobles (Para uso en una cadena).
ÑÑCaracter backslash.
Ñ123 Hasta tres dígitos en octal.
Ñx12 Hasta dos dígitos en hexadecimal
ÑOCaracter nulo (un caso especial de octal)

3.2.2 Expresiones.

Algunos operadores requieren un operando tipo "lvalor". lvalor es una expresión que se refiere a un objeto: Un nombre de variable es el caso màs simple; una matriz de texto indexado, y un puntero indirecto también son lvalor.

Cuando se usan **char** o **short** con signo, el signo se extiende al tamaño de **int**. Si se usan **char** o **short** sin signo en alguna expresión, son rellenados con ceros hasta completar el tamaño de **unsigned long**.

Un resumen de las prioridades en las operaciones aparece al final de esta sección (3.2.8).

El desbordamiento se ignora siempre al evaluar una expresión.

3.2.3 Expresiones primarias.

Constantes, variables y cadenas son todas expresiones primarias.

Todas las expresiones entre paréntesis son unitarias sirviendo éste para controlar el orden de evaluación de la expresión.

Una expresión primaria seguida de una expresión entre corchetes es una expresión primaria. Esto se usa en matrices indexadas.

Una expresión primaria seguida de una secuencia de cero o màs expresiones entre paréntesis es una llamada de función, siendo cada una una expresión primaria.

3.2.4 Operadores unarios.

Se evaluan de derecha a izquierda.

* Indirecto: La expresiòn es normalmente un puntero y el resultado de la expresiòn es el valor apuntado

& direcciòn de: Solo se puede usar para expresiones <lvalor>, y retorna la direcciòn de <lvalor>.

- menos unario: Proporciona el complemento a dos de la expresiòn.

! negaciòn logica: Retorna 0(falso) si la expresiòn es distinta de cero (cierta), y 1 (cierto) si la expresiòn es cero (falso).

~ complemento a uno: Retorna el complementario de la expresiòn bit a bit.

++ Incremento y decremento: Pueden ser operadores prefijo y sufhjo.

-- El operando ha de ser <lvalor>. Si son usados como prefijo el valor es incrementado o decrementado y el resultado de la expresiòn es el nuevo valor. Si son usados de sufijos el valor es incrementado o decrementado pero la expresiòn toma el valor original.

3.2.5 Operadores binarios.

Todos los operadores binarios se leen de izquierda a derecha. Si en una expresiòn hay operadores de igual prioridad, la expresiòn se analiza desde la izquierda. Ejemplo:

a - b + c es analizado como ((a - b) + c)

Los operadores aparecen en esta secciòn de mayor a menor prioridad, y los que son de igual prioridad, en grupo.

Si algun operando o el operador son unsigned , los dos operandos son tratados como unsigned y el resultado serà unsigned.

* Multiplicaciòn
/ Divisiòn.
% Resto.

Los desbordamientos de multiplicaciòn son ignorados. La divisiòn es truncada a unidades. El signo del resto es el mismo que el del divisor. Siempre es cierto que:

((a / b) * b + a % b) = = a /* suponiendo que b es distinto de cero */

+ Suma.
- Resta.

Si un entero es sumado o restado a un puntero serà transformado por el tipo base del apuntador. De tal modo que si p es el apuntador a una tabla, p+1 es el apuntador al siguiente elemento de la tabla.

Si dos punteros a objetos de el mismo tipo son restados, el resultado es transformado para dar el nùmero de objetos entre los dos punteros. Sòlo es util si los dos punteros apuntan a elementos de la misma tabla.

<< Desplazamiento a la izquierda.
>> Desplazamiento a la derecha.

El desplazamiento a la izquierda rellena con bits a cero. El desplazamiento a la derecha rellena con ceros si el valor es sin signo y si lo tiene, lo preserva.

< Menor que
<= Menor o igual que
> Mayor que
>= Mayor o igual que
== Igual que
!= No igual que.

Estos operadores retornan 1 (cierto) si la relación se cumple, en caso contrario retornan 0 (falso). Las comparaciones **unsigned** son ciertas si ambos parámetros son **unsigned** (sin signo).

& AND bit a bit.
^ XOR bit a bit.
! OR bit a bit.

Estos operadores ejecutan una operación bit a bit en el valor del segundo operando dando como resultado un valor entero.

&& AND lógico.
!! OR lógico.

Los operadores lógicos comprueban si los valores son cero o no y retornan un valor que es cero o uno, garantizando una evaluación de izquierda a derecha, salvo en el caso de que el primer operando determine el resultado, en que no se evalua el operando de la derecha.

3.2.6 Operadores de asignación.

Los operadores de asignación, agrupan de derecha a izquierda y su resultado es el valor asignado siendo posible realizar asignaciones múltiples. Ej:

```
*fred = bert(1) = thing = 0;
```

Esto iguala thing a cero y entonces pone una locación de la matriz bert a cero y entonces pone el objeto apuntado por fred a cero.

<lvalor> = <expresión>

Esta es la signación simple: La expresión es evaluada y el resultado se salva en la variable o lugar especificado por lvalor.

Cuando asignamos a un **char** el valor es truncado a 8 bit perdiéndose los bit superiores. Si asignamos a **short** se asignan los 16 bit bajos.

Otros operadores de asignación efectuan alguna operación aritmética en un objeto:

<lvalor> <op>= <expresion>.

<op> puede ser uno de entre: + - * / % >> << & ^ |

Nótese que no debe haber espacio entre el <op> y el simbolo igual.

El significado de a <op>= b es el mismo que a = a <op> b pero sblo se evalua a una vez, siendo más rápido.

3.2.7 Expresiones con coma.

<expresión> , <expresión>

Un par de expresiones separadas por una coma se evaluan de izquierda a derecha. El valor de la primera expresión es descartado, y la expresión de la derecha da el resultado de toda la expresión.

Las expresiones con coma se usan tipicamente cuando la sintaxis requiere una expresión, pero con una serie de efectos laterales, por ejemplo en el control de un bucle **while**.

```
while ( ++ptr1 , fred ++ incremento , c = *ptr1, c != EOF )
        print ( c );
```

Observe que cuando el contexto exige otro análisis del caracter coma, (ej en la lista ddd parámetros de una función) las expresiones con coma deben encerrarse entre parántesis para evitarle confusiones al compilador, por ej:

```
func ( a, ( b=2, b+3 ), c );
```

3.2.8 Sumario de prioridades de los operadores.

parentesis
<primario> (<expresion>/es>..,) Llamada a funciòn
<primario> (<expresion>) Indexaciòn de matriz
*<expresion> valor en la direcciòn
&<lvalor> direcciòn de
-<expresiòn> menos unario (complemento a dos)
!<expresiòn> negaciòn lògica unaria
~<expresiòn> complementario bit a bit unario.
++<lvalor> pre incremento
--<lvalor> pre decremento
<lvalor>++ post incremento
<lvalor>-- post decremento
* / % operadores multiplicativos.
+ - operadores aditivos.
<< >> desplazamientos aritmèticos.
< <= > >= Inigualdades
== != operadores de igualdad
& ^ | operadores lògicos de bit.
<expresion> && <expresiòn> AND lògico
<expresiòn> || <expresiòn> OR lògico
= += -= *= /= %= >>= <<= &= ^= |= operadores de asignaciòn
<expresiòn>,<expresiòn> expresiones con coma.

3.3 Control de flujo y sentencias.

Una nota sobre el punto y coma. En C, el punto y coma no se usa para separar o finalizar las sentencias (como en PASCAL) sino que forma parte de la sintaxis de algunas sentencias.

3.3.1 Sentencia de expresión.

Es una expresión seguida de punto y coma. Normalmente estas sentencias de expresión son asignaciones o llamadas de función, ej:

```
a = 1;
ptr++;
func ( a, b, c );
```

3.3.2 Sentencia compuesta (bloque).

(<declaraciones> <sentencias>)

Una sentencia compuesta permite usar varias sentencias donde la sintaxis requiere sólo una sentencia. También permite declarar y definir variables locales.

Una sentencia compuesta tiene un par de abrazaderas (llaves) que albergan cero o más declaraciones seguidas de cero o mas sentencias, por ejemplo:

```
(               int a, b;
                a = getval();
                b = getval();
                putval ( a + b );
)
```

3.3.3 Sentencia condicional.

Las dos formas de sentencia condicional son:

if (<expresión>) <sentencia1>
if (<expresion>) <sentencia1> else <sentencia2>

En ambas la expresión es evaluada, y si es cierta (distinta de cero), la sentencia1 es ejecutada. En el segundo caso si la expresión es falsa (cero) la sentencia2 es ejecutada.

En C no existe la construcción if.. endif y el **else** siempre es adjudicado al **if** incompleto más cercano.

3.3.4 Sentencia while.

while (<expresión>) <sentencia>

La expresión se evalua y si es cierta (no cero) la sentencia se ejecuta y se vuelve a repetir hasta que la expresión sea falsa (cero).

3.3.5 Sentencia do.

do <sentencia> while (<expresión>);

La sentencia es ejecutada y después se evalua la expresión y de ser cierta (no cero) el ciclo se repite.

3.3.6 Sentencia for.

```
for ( <expresiòn1> ; <expresiòn2> ; <expresiòn3> ) <sentencia>
```

Es funcionalmente equivalente a:

```
<expresiòn1>;
while ( <expresiòn2> ) { <sentencia> <expresiòn3>; }
```

La primera expresiòn inicializa el bucle, la segunda es el test de finalizaciòn que se evalua antes de cada vuelta. La tercera es el reinicializador de bucle, frecuentemente incrementa un contador.

Cualquiera de las tres expresiones pueden suprimirse, pero si se suprime la segunda, el bucle se repite continuamente (ver 3.3.8) por ejemplo:

```
for (count=0 ; ; ++count )
{ int c;
  c = getchar();
  if ( c == EOF ) break;
  putchar(c);
}
```

Las variables usadas en una sentencia **for** deben ser declaradas previamente, no como en otros lenguajes (en cambio pueden ser modificadas dentro del bucle).

3.3.7 Sentencia switch.

```
switch ( <expresiòn> ) <sentencia compuesta>
```

La expresiòn es evaluada, y entonces se usa el resultado para determinar què parte de la sentencia compuesta se ejecuta. Cada sentencia de la sentencia compuesta puede etiquetarse con uno o màs prefijos **case :**

```
case <exprresiòn constante> :
```

Puede haber un prefijo por defecto de la forma:

```
default:
```

El valor de la expresiòn del **switch** es comparado con cada constante **case.** Si alguna es igual, el control pasa a la sentencia que sigue al **case.** Si ninguna constante es igual y no hay ninguna sentencia etiquetada **default:** no se ejecuta ninguna sentencia del bloque. Si existe **default:** el control pasa a la sentencia detràs de èste prefijo.

Vea la sentencia **break** (3.3.8) para salir de una sentencia **switch** . En caso de no usarse la sentencia break se ejecutaràn todas las sentencias siguientes del bloque compuesto atravesando los **case** y **default:**.

No se permiten declaraciones en la sentencia compuesta de una sentencia **switch.**

Ejemplo:

```
switch ( ch )
{
case 'a': case 'e' : case 'i' : case 'o' : case 'u':
                print ( "Es una vocal" );
                break;
case 'y':
                print ( "Calidad de vocal" );
                break;
case 'f' : case 'h' : case 'l' : case 'r' : case 's' :
                print ( "Consonantes suaves" );
                break;
default:
                if ( isalpha( ch ) )
                    print ( "Consonante" );
                else print ( "No es una letra" );
                break; /* No es necesario */
}
```

3.3.8 Sentencias break, continue y return.

La sentencia

```
break;
```

produce la salida del bucle o sentencia switch actual.
La sentencia

```
continue;
```

hace que se ignore el resto de sentencias del bucle en ejecución y se comience una nueva vuelta al bucle.
Las sentencias

```
return;
return <expresión>;
```

acaban la función en ejecución. En el primer caso el resultado de la función queda indefinido. En el segundo la expresión es evaluada y su resultado es devuelto como resultado de la función.

3.3.9 Sentencia goto.

La sentencia **goto** tiene la forma:

```
goto <etiqueta>;
```

donde <etiqueta> está definida en la función en ejecución de la forma

```
<etiqueta> :
```

Un nombre de etiqueta se construye del mismo modo que un nombre de variable. QC no permite gotos hacia dentro o fuera de bloques que contengan variables declaradas.

3.3.10 Sentencia nula.

La sentencia nula consiste en un punto y coma. Se usa en ciclos en que todo el trabajo es realizado por la expresión de control, como en el ejemplo:

```
while (*dest++ = *source++ )
   ;
```

Aquí los octetos apuntados por source son copiados al puntero dest e incrementado. El bucle termina cuando se transfiere un 0 .

3.4 Funciones y estructura del programa.

3.4.1 Estructura del programa.

Un programa en C se forma de un grupo de funciones (no se distinguen procedimientos y funciones en C, todas las funciones retornan un valor, pero el valor puede estar indefinido o ser ignorado). Noexiste un programa principal en C, pero cada programa tiene una función llamada main que es llamada cuando el programa es ejecutado.

Un programa fuente de C puede contener un número de declaraciones de funciones cada una con sus opcionales variables externas o globales (ver 3.1.2) y/o ordenes al preprocesador opcionales (ver 3.6).

3.4.2 Declaración de una función.

```
<nombre> ( <lista de argumentos>...) <declaración de argumentos>...
( <declaraciones>... <sentencias>...)
```

Comienza con el nombre de función (se construye del mismo modo que los nombres de variable) seguido de un paréntesis que encierra la lista de argumentos: la forman cero o mas nombres de argumento separados por comas. Estos son el nombre con que son conocidos los argumentos dentro de la función y son accesibles desde cualquier parte de la función.

La declaración de argumentos es una serie de declaración que especifica el tipo de los argumentos. Cada argumento ha de declararse como char o como int y en todo caso como puntero a uno de estos dos. Si el nombre se precede de un asterisco nombra a un puntero hacia el tipo especificado. ej:

```
afunction ( i, j, ch, str ) int i, *j; char ch, *str; (......)
```

En este ejemplo i se declara como entero y j como puntero a un entero. Igualmente ch es un caracter y str es un auntador a un caracter.

Las matrices son pasadas a las funciones como la dirección del primer elemento en la matriz. Como alternativa a declarar un rgumento de tipo matriz como un puntero (usando un asterisco) se puede declarar usando un par de coorchetes, ej:

```
func ( array ) int array();
```

En este caso el parámetro array es todavia un puntero a la base de la matriz pero en este caso la sintaxis deja ver mejor al programador que una matriz va a ser procesada.

Después viend dl cuerpo de la función que ha d$ ser una sentencia compuesta.

Una función que no tenga parámetros se declara como sigue:

```
función () (.....)
```

3.4.3 Llamada a funciones.

Para llamar una función, el nombre de función debe seguirse de los parámetros entre paréntesis.

```
afunction ( value()-123, &intarray((17), '0'+digit, string )
```

Cada expresión de parámetro es evaluada y se hace una copia del valor antes de llamar la función. Es ésta copia la que toma la función como su parámetro. Si quiere que una función sea capaz de modificar una variable, pase la dirección de la variable a la función , ej:

```
main()
{
   int a, b;
   ;
   swap ( &a, &b ) /* llamada a la función intercambio de variables */
   ;               /* observe que se pasan las direcciones */
}

swap ( addra, addrb )   /* esta es la función swap */
   int *addra, *addrb;
```

```
                    /* los parámetros son definidos como punteros */
                {
                    int temp;
                    temp = *addra;    /* pone en temp el valor apuntado por addra */
                    *addra = *addrb;  /* pone el valor apuntado por addrb en el valor
                                         apuntado por addra */
                    *addrb = temp;    /* pone en el valor apuntado por addrb el antiguo
            valor apuntado por addra */
                }
```

Si una función no tiene parámetro se llama con el nombre de función seguido de dos paréntesis vacíos ej: función();

3.4.4 Resultados de las funciones.

Todas las funciones retornan resultados, pero el resultado retornado no se define hasta que la función ejecuta una sentencia con un valor return.

En QC todas las funciones retornan un valor int. Pueden retornarse punteros dado que la variable puntero tiene el mismo número de bits que un int y QC no cambia el valor cuando un puntero es asignado a un entero o viceversa.

3.4.5 Funciones static.

Si una función se declara static su nombre no es accesible desde otros módulos, de igual modo que las variables estáticas. El nombre de la función se precede por la palabra static en la declaración:

static <nombre> (<lista-argumentos>...) <declaración-argumentos>...{...}

3.4.6 Funciones extern.

Si una función está en otro módulo puede ser declarada como externa del siguiente modo:

extern int <nombre>();

extern int (*<nombre>) ();

Observe que la lista de la función no se declara aquí. El tipo por omisión es en una declaración extern es int y puede suprimirse. QC considera el segundo modo como equivalente pero los compiladores completos de CC lo tratarían como un apuntador a una función.

Al final de un módulo, cualquier símbolo indefinido se le trata como si fuera una función extern.

3.4.7 Paso de (la dirección de)funciones como parámetros.

Se puede llamar a una función pasando como un parámetro (la dirección de) otra función que es llamada. Esto se puede hacer en QC usando la siguiente sintaxis en la declaración de una llamada:

(* <nombre>)

Por ejemplo, func1 es una función que tiene una función como parámetro y es declarada como sigue:

```
func1 ( arg ) int (* arg) ();  /* arg es un puntero a una función */
{
                (* arg )(); /* la función es llamada de esta manera */
}
```

La llamada a esta función se hace pasando el nombre de la segunda función como parámetro pero sin ningún paréntesis.

```
func1 ( func2 );
```

3.4.8 Funciones con un número variable de parámetros.

Normalmente, el número de parámetros que se pasa a una función al llamarla es igual al total de parámetros declarados en la función.

Algunas funciones como **printf** en la libreria (ver 4.5.1) pueden usar un nùmero variable de paràmetros. Si quiere una funciòn con un nùmero variable de paràmetros como èsta, puede encontrar el nùmero de paràmetros pasados llamando a la rutina de libreria **ccargc()**. (esta rutina se debe llamar al principio de una funciòn antes de llamar a cualquier otra), con ella y sabiendo que los argumentos se almacenan como **int**s en orden inverso dentro de la pila, es posible acceder correctamente a los argumentos. Observe que esto no es transportable. Ejemplo:

```
func ( arg )        /* declara un argumento de funciòn */
                    int arg; /* como un entero */
(
                    int numargs, /* tendrà el nùmero de argumentos */
                        *argp; /* tendrà un apuntador a ellos */

                    numargs = ccargc(); /* obtiene el nùmero de argumentos */
                    argp = &arg + numargs-1; /* apuntador al primer argumento */
                    while ( numargs--) /* para cada argumento en orden */
                         process ( *argp-- );  /* lo procesa y pasa al siguiente */
)
```

Quedaria ahora definir que se quiere hacer con cada paràmetro en la funciòn process. (N del T)

Vea la secciòn 6 para màs informaciòn sobbre la disposiciòn de los paràmetros de funciòn en el stack o pila.

3.5 Punteros y matrices.

3.5.1 Punteros.

Son variables que soportan el apuntador de direcciòn a otra variable. Facilitan un mecanismo flexible y potente de procesar datos. El tipo del puntero se especifica por el del objeto apuntado para asi poder distinguir ints de chars. Se manipulan de acuerdo al tamaño del dato: si incrementa 1 a un puntero a un char apuntarà al siguiente que estarà 1 octeto màs allà, si incrementa en uno un puntero a un int apuntarà al siguiente int que estarà 4 octetos màs allà.

Cuando tomemos la diferencia entre dos punteros es el nùmero de objetos entre ambos punteros no el nùmero de octetos. (si los punteros no son del mismo tipoel resultado queda indefinido).

La sintaxis para declarar un puntero es la misma que para declarar variables salvo que el nombre de variable tiene un asterisco como prefijo.

```
<tipo> *<nombre>;
```

Si un puntero es usado en una expresiòn sobre si mismo, la direcciòn se usa o se altera de acuerdo a la expresiòn. Si el puntero se precede de un asterisco, el valor apuntado serà modificado por la expresiòn. Ej:

```
int fred, *prt1, *prt2;
prt2 = 0;       /* pode el puntero a cero */
prt1 = &fred;   /* pone el puntero a apuntar a fred */
*ptr1 = 0;      /* pone el objeto apuntado por prt1 (fred) a cero */
fred = *prt1;   /* usa el entero apuntado por prt1 */
prt2 = prt1     /* copia el valor del puntero */
```

3.5.2 Matrices.

Las matrices en QC estàn restringidas a una sola direcciòn (pero vea la secciòn siguiente). Se declaran siguiendo al nombre de la matriz unos corchetes con el tamaño de la matriz dentro (que ha de ser una constante).

```
<tipo> <nombre> [ <expresiòn constante> ]
```

Por ejemplo:

```
int array [ 10 * 100 ];
char buffer [ 512 ];
```

Si una matriz es declarada como **extern** suprima eel tamaño de la matriz que irà dfinido en el otro mòdulo.

Para apuntar a un elemento de la matriz se usa la siguiente sintaxis:

```
<nombre> [ <expresiòn> ]
```

Si una matriz tiene M elementos el primer elemento se obtiene con un incremento de cero y el ultimo con el incremento M-1.

Un nombre de matriz apunta a la direcciòn (del primer elemento) de la matriz. Se puede usar en expresiones como la direcciòn de una matriz (puede asignarse a un puntero por ej) pero no se puede alterar a si misma.

Observe que la sintaxis de punteroo tambièn puede usarse deireccionando matrices. Las siguientes expresiones son equivalentes:

```
array [ offset ];
* ( array + offset );
```

3.5.6 Simulando matrices multidimensionales.

Aunque QC no crearà matrices multidimensionales es posible hacer una matriz de punteros (a veces conocidos como vectores sin vida) lo que permite acceder a una matriz unidimensional como si fuera bidimensional.

Una matriz de punteros se declara poniendo un asterisco y corchetes en el nombre de la matriz, por ejemplo:

```
int * index [16],  /* Matriz de apuntadores a enetros */
```

```
                        table (256);  /* Matriz de enteros */
```

Si los elementos de cada puntero de index se inicializan apuntando hacia cada dieciseisavo elemento de table simularemos una matriz de 16 por 16. Los punteros podrian inicializarse asi:

```
int i;
for ( i=0 ; i< 16 ; ++i )
                        index ( i ) = &table ( 16*i );
```

Y la matriz podria accederse como sigue:

```
index (a)(b);
```

index(a) toma uno de los punteros en index y proporciona un puntero a la matriz table. El segundo acceso de matriz proporciona un elemento en esa parte de la matriz table.

3.6 Comandos del preprocesador.

Estos comandos del preprocesador, manipulan el fichero fuente de C antes de ser compilados. Se le puede conriderar como un paso extra del compilador pero los cambios se realizan a medida que el compilador lee el texto. Todas estas instrucciones de preproceso deben usar una línea para sí.

3.6.1 La instrucción include.

La instrucción **include** tiene dos formas (los parèntesis en ángulo son parte de la sintaxis):

#include "nombre_de_fichero"

#include <nombre_de_fichero>

No se admiten espacios entre las comillas o parèntesis.

Esta instrucción permite que un fichero se introduzca en la compilación en este punto. Puede usarse para hacer el mismo conjunto de definiciones y direcciones en cada uno de un conjunto de módulos, por ejemplo.

El compilador acepta las dos formas y son tratadas igual (Las dos formas indican a algunos compiladores de C que busquen el fichero en diferentes directorios).

El nombre de fichero de esta instrucción ha de ser un nombre completo de QDOS salvo si se está usando la opción -D del compilador (ver sección 2) en cuyo caso la unidad o directorio especificado puede omitirse. El nombre de fichero puede ir en mayúsculas o en minúsculas.

Los ficheros **include** no deben estar anidados: si el ficheroA incluye al ficheroB y este incluye al ficheroC, entonces al final del ficheroC se salta el resto del ficheroB y se sigue con el ficheroA.

Esta instrucción sólo puede ir a nivel más alto del programa, por ejemplo entre las declaraciones de función y las de variables globales. No puede usarse dentro de una función.

3.6.2 Sustitución de macros.

#define <nombre_de_macro> <Texto del macro>

Esta instrucción define un macro que entonces tendrá eltexto arbitrario del resto de la línea. Cuando más tarde se encuentre un noombre de macro más tarde en el módulo, el compilador lo cambiará por el texto del macro.

Los nombres de macro deben seguir las misaas normmas que los de variables pero por convención suelen usarse en mayúsculas.

Suelen usarse para reemplazar a constantes pero se pueden usar para reemplazar cualquier cosa.

Los macros sólo se pueden definir fuera de funciones aunque la sustitución de macros puede darse en cualuier parte del programa (salvo en cadenas y constantes **char**).

Observe que la sustitución de macros también se puuede hacer en las líneas de definicción de macros por lo que los macros pueden definirse en tèrminos de macros anteriores.

No estan implementados en QC macros con paràmetros.

3.6.3 Compilación condicional.

#ifdef <nombre_de_macro>

#ifndef <nombre_de_macro>

#else

#endif

Estas instrucciones permiten al compilador, compilar distintas secuencias de código según esté o no definido un macro, pudiéndose hacer distintas versiones de un mismo programa solo cambiando las instrucciones **#define** del principio. Las instrucciones de compilación condicional puden ponerse en cualquier lugar de un programa.

3.6.4 Código en ensamblador.

#asm

#endasm

Estas instrucciones permiten incluir còdigo en ensamblador en el listadoo de un programa. Pueden usarse en cualquier nivel del programa, tanto dentro como fuera de las funciones. Todas las instrucciones del 68000 se admiten. Para detalles de la sintaxis de ensamblador vea la guia del manual del Macro assembler Sinclair. Vea la secciòn 6 para mayor informaciòn sobre el acceso y direccioones del còdigo en ensamblador.

3.6.5 Instrucciones de control de listado.

#nolist
#list
#page

Estas instrucciones pueden usarse para controlar la disposiciòn y contenido del listado del compilador. Tal vez usted quiera quitar el listado de un fichero include conteniendo gran nùmero de macros o tal vez quiera que cada funciòn comience en una pàgina nueva.

para entrada, salida y canal de error standard.

Los datos escritos en **stderr** siempre se envian a la pantalla sin poderlos redirigir como **stdout**. También se permite leer desde **stderr** en cuyo caso el teclado será leido independientemente de **stdin**.

Estos ficheros standard pueden usarse sin llamar **fopen**.

4.3.1 **fopen (nombre, modo)**
char *nombre, *modo;

Abre un canal al fichero nombrado. Retorna un descriptor de fichero o si falla el valor 0. **modo** puede ser alguna de las siguientes cadenas:

"r" Abre para leer (el fichero debe existir).
"w" Abre para escribir (en cuyo caso de existir es borrado y recreado).
"a" Abre para añadir (El fichero si no existe se crea y si existe se posiciona al final del fichero. No puede usarse para periféricos como la pantalla).
"d" Abre el directorio (ver 5.4.11).

4.3.2 **freopen (nombre, modo, fd)**
char *nombre, *modo;
int *fd;

Se usa para cerrar un canal y reabrirlo en otro fichero. Retorna **fd** y si falla retorna cero.

4.3.3 **fclose (fd) int *fd;**

Cierra el canal y afluye en el cualquier dato que pudiera quedar en memoria
Retorna cero o en caso de error un valor distinto de cero.

4.3.4 **getc (fd) int *fd;**
fgetc (fd) int *fd;

Estas rutinas son equivalentes. Retornan un caracter leido del canal de entrada especificado o **EOF** si llegan al final del fichero.

4.3.5 **ungetc (c, fd) char c; int *fd;**

Esta función se usa para devolver un caracter a un fichero sin reposicionarlo pudiendose usar en fuentes secuenciales. La próxima vez que se lea el fichero el valor será retornado. La función devuelve el valor c o devuelve **ERR** si estamos en la posición EOF.

4.3.6 **fgets (str, tamaño, fd)**
char *str;
int tamaño, *fd;

Esta rutina se usa para leer una linea de un fichero. **str** es un puntero a un buffer y **tamaño** es el número máximo de octetos que se pueden leer. La entrada termina con el caracter de newline (LF). Tras el newline en el buffer se pone un byte a 0.

Si la linea es muy grande para el buffer, apareceran en el buffer tamaño-1 octetos terminando con un octeto a 0.

La rutina retorna str o en caso de error **NULL**.

4.3.7 **putc (c, fd) char c; int *fd;**
fputc (c, fd) char c; int *fd;

Estas rutinas son equivalentes. Escriben el caracter c en el fichero indicado por **fd**. Retornan el valor de c o en caso de error **EOF**.

4.3.8 **fputs (str, fd) char *str; int *fd;**

Escribe la cadena str al fichero fd eliminando el caracter nulo del final y sin añadir al final un caracter de newline.

4.3.9 **fflush (fd) int *fd;**

Esta rutina se llama para escribir todos los datos pertenecientes a un fichero que estèn todavia en el buffer de la libreria de tiempo de ejecuciòn de QC. Se puede usar para forzar a una linea parcial a que sea escrita a un fichero. Los datos destinados a pantalla se afluyen automàticamente al final de cada linea. Esta rutina es llamada por **fclose.**

4.3.10 **Isatty (fd) int *fd;**

Esta rutina retorna YES si el canal està abierto a un dispositivo serie (la pantalla o la consola...) o NO si el periferoco es de acceso aleatorio (MDV o floppy).

4.3.11 **Iscons (fd) int *fd;**

Esta rutina retorna YES si el canal està conectado a la pantalla o al teclado, culquier otro retorna NO.

4.3.12 **delete (nombre) char *nombre**
unlink (nombre) char *nombre

Estas rutinas son equivalentes. Borran el fichero especificado con el nombre de fichero retornando NULL o en ccaso de error EOF.

4.3.13 **feof (fd) int *fd;**

Retorna YES si se ha llegado al final de fichero o NO si no se ha llegado.

4.3.14 **ferror (fd) int *fd;**

Esta rutina devuelve el còdigo de estado de sistema asociado a la ùltima llamada del sistema a **fd. ferror (0)** es un caso especial que devuelve el còdigo de estado de sistema asociado a la ùltima llamada de apertura de fichero.

4.3.15 **clearerr (fd) int *fd;**

Limpia cualquier estado de error asociado con el fichero **fd.**

4.4 <u>Entrada y salida de acceso aleatorio.</u>

Las siguientes rutinas pueden usarse para movernos a lo largo de un fichero. La posiciòn es un valor entero positivo que representa el nùmero de bytes desde el comienzo del fichero.

4.4.1 **rewind (fd) int *fd;**

Esta rutina reposiciona un fichero al principio. Devuelve **NULL** o en caso de error **EOF.** Es equivalente a **lseek(fd,0,0)** (ver màs abajo).

4.4.2 **getpos (fd) int *fd;**

Esta rutina devuelve la posiciòn actual del fichero o EOF si el canal no soporta acceso aleatorio.

4.4.3 **lseek (fd, desplazamiento, desde) int *fd, desplazamiento, desde;**

Esta rutina posiciona el fichero a otra indicada por **desplazamiento** en bytes desde la posiciòn:

desde == 0 posiciòn relativa al comienzo del fichero
desde == 1 posiciòn relativa a la posiciòn actual
desde == 2 posiciòn relativa al final del fichero

La rutina devuelve **NULL** o en caso de error **EOF.**

4.5 <u>Entrada y salida con formato.</u>

4.5.1. **printf (str, arg1, arg2, ...) char *str;**

Esta funciòn imprime una cadena de caracteres con un formato en el canal standarad de salida. **str** es

una cadena de control que contiene caracteres ordinarios y especificaciones de conversiòn. Los caracteres ordinarios son escritos tal cual. Cada especificaciòn de conversiòn indica como han de convertirse los arg correspondientes antes de darlos salida. La funciòn devuelve como resultado el nùmero de caracteres escrito.

Las especificaciones de conversiòn comienzan con el caracter % y acaban con una letra. Entre estos dos caracteres puede haber otros datos opcionales con informaciòn extra de formato:

Un signo menos Indica justificaciòn a la izquierda dentro del campo.

Un nùmero decimal Da el ancho del campo que si va precedido de un cero el campo serà rellenado en sus huecos con ceros.

Una fracciòn decimal nùmero de caracteres a tomar de una cadena.

La letra del final indica el tipo de conversiòn:

b	entero sin signo lo convierte a binario
c	caracter
d	entero con signo lo convierte a decimal (con shgno)
o	entero sin signo lo convierte a octal
s	direcciòn de una cadena
u	entero sin signo lo convierte a decimal (sin signo)
x	entero sin signo lo convierte a hexadecimal.

Si se encuentra un caràcter invàlido èste se escribe de salida como un caracter de texto, y de encontrarse %% en la cadena de formato se escribirà en la salida un "%".

Vea màs adelante algunos ejemplos de uso de **printf**.

4.5.2 **fprintf (fd, str, arg1, arg2,...) int *fd; char *str;**

Esta rutina es jùstamente igual que **printf** sòlo que el primer paràmetro indica el canal en el que hay que escribir.

4.5.3 **scanf (str, arg1, arg2,...) char *str;**

Esta funciòn efectua la operaciòn inversa a **printf**. Lee texto desde el canal de entrada standard y lo interpreta de acuerdo a la especificaciòn de conversiòn **str**.

La cadena de control **str** sòlo contiene especificaciones de conversiòn y espacios en blanco (que son ignorados).

Todos los argumentos pasados deben de ser las direcciones de las variables donde los resultados seràn puestos. (o para cadenas el apuntador al buffer)

La funciòn devuelve el nùmero de campos procesados. Terminarà prontamente si si algùn dato de entrada no se ajusta a la especificaciòn de conversiòn. Si no se ha leido ningùn campo y se ha llegado al final del fichero devolverà **EOF**.

Las especificaciones de conversiòn son similares a las de **printf** pero entre % y la letra sòlo se puede poner un asterisco (que indicarà que se salte dicho campo) y/o un nùmero decimal que indique la profundidad del campo.

Un campo es normalmente una secuencia de caracteres imprimibles terminados por un caracter de espacio en blanco. El campo tambièn termina cuando la profundidad del campo (si es especificada) ha sido completada. Si la especificaciòn de conversiòn es %c se lee un ùnico caracter sin saltar el espacio en blanco.

Las especificaciones de formato son:

b entero binario
c caracter
d nùmero decimal con signo
o entero octal
s cadena de caracteres
u nùmero decimal sin signo
x nùmero hexadecimal

Vea màs adelante ejemplos de la funciòn **scanf**.

4.5.4 **fscanf (fd, str, arg1, arg2,...) int *fd; char *str;**

Esta rutina trabaja como **scanf** pero el primer paràmetro indica el canal que ha de ser leido.

EJEMPLO de PRINTF

Las barras verticales se han incluido en la cadena de formato de los ejemplos para mostrar el efecto de espaciado y de anchura de campo en la salida usando printf. No son preceptivas.

Si nombre es la cadena "fred" y puntuación es un entero de valor 67, entonces: printf("%s's score is %d%%%n", nombre, puntuación); escribirá fred's score is 67%

cadena de formato	parametro(s)	impresión de salida
%%d%	1234	%1234%
%%6d%	1234	% 1234%
%%-6d%	1234	%1234 %
%%06d%	1234	%001234%
%%d%	-1	%-1%
%%o%	-1	%4294967295%
%%x%	-1	%ffffffff%
%%b%	1234	%100110100010%
%%06o%	1234	%002322%
%%04x%	1234	%04d2%
%%c%	'A'	%A%
%%9c%	'A'	% A%
%%2x%	'A'	%41%
%%a%	"computer"	%computer%
%%5a%	"computer	%computer%
%%12a%	"computer"	% computer%
%%-12a%	"computer"	%computer %
%%12.4a%	"computer"	% comp%
%%-9s %02d/%02d%	"Thursday",14,2,85	%Thursday 14/02/85%

EJEMPLO DE SCANF

Considerando la siguiente sentencia:

scanf (" %s %c %c %%s %d %3d %d ", str, &c1, &c2, &i1, &i2, &i3);

si la entrada contiene el siguiente texto

abc defg -12 345678 9

entonces las variables recibirán los siguientes valores:

str: "abc" lee una cadena terminada en un espacio
c1: ' ' lee el siguiente caracter
c2: 'd' lee el siguiente caracter
la siguiente cadena es saltada "efg"
i1: -12 lee un número terminado en un no-digito
i2: 345 lee un número de tres digitos
i3: 678 lee un número terminado por un no-digito

La siguiente llamada de entrada leerá comenzando en el espacio después de "345678".

Otro ejemplo podria ser la siguiente sentencia:

num = scanf ("%d %%c %d %%c %d %%c ", &i1, &i2, &i3);

que puede usarse para leer números terminados con no-digitos

123, 456, 789,

donde %d lee hasta el no-digito y %*c salta un caracter.

4.6 Funciones de conversión de formato.

4.6.1 **stoi (str) char *str;**
Convierte un número decimal (con signo) en una cadena en un entero. El espacio en blanco del final se salta, y la conversión termina en el primer no-digito que se encuentre. Un caracter de signo + o - puede preceder al primer digito.

4.6.2 **stoib (str, base) char *str; int base;**
Convierte un número (sin signo) en la base especificada en un entero. El espacio en blanco del final se salta. Se permiten bases de la 2 a la 16. Esta rutina destruye el calor de la cadena.

4.6.3 **itos (num, str) int num; char *str;**
Convierte el número en una cadena decimal justificada por la izquierda. Si el número es negativo, se genera un signo menos. La cadena termina con un byte null. Los enteros de 32bit requieren hasta 10 octetos (o 12 incluyendo el signo y el null).

4.6.4 **itosb (num, str, base) int num; char *str; int base;**
Convierte el número (sin signo) en una cadena usando la base especificada. Se permiten bases de la 2 a la 16. La cadena se termina con un caracter null. Se pueden generar hasta 33 bytes en el caso de la base 2.

4.6.5 **dtoi (str, num) char *str; int *num;**
Convierte la cadena decimal con signo poniendo el resultado en num. Devuelve el número de digitos en el número o ERR si se produce un error (desbordamiento o digitos no válidos). La rutina acepta un signo al principio opcional pero no admite espacios en blanco al principio.

4.6.6 **otoi (str, num) char *str; int *num;**
Convierte una cadena octal sin signo poniendo el resultado en num. Devuelve el número de digitos en el número o en ccaso de desbordamiento ERR. No admite espacios al principio.

4.6.7 **utoi (str, num) char *str; int *num;**
Convierte una cadena decimal sin signo poniendo el resultado en num. Retorna el número de digitos o en caso de desbordamiento ERR. No admite espacios en blanco delante.

4.6.8 **xtoi (str, num) char *str; int *num;**
Convierte la cadena hexadecimal sin signo poniendo el resultado en num. Retorna el número de digitos o en caso de desbordamiento ERR. Acepta letras en mayúsculas y en minúsculas y no salta espacios en blanco al principio.

4.6.9 **itod (num, str, tamaño) int num, tamaño; char *str;**
Convierte un número entero en una cadena decimal con signo justificada por la derecha. Si tamaño es >0 se colocan tamaño-1 octetos en la cadena seguidos de un caracter null. Si tamaño es = 0 se busca colocar el caracter null cuando acabe la conversión en la cadena. Si tamaño es negativo, se convierte en una cadena de tamaño bytes sin poner un caracter null al final. Devuelve str.

4.6.10 **itoo (num, str, tamaño) int num, tamaño; char *str;**
Convierte a octal un número entero, el resto como itod.

4.6.11 **itou (num, str, tamaño) int num, tamaño; char *str;**
Convieerte a decimal sin signo un número entero. El resto como itod.

4.6.12 **itox (num, str, tamaño) int num tamaño; char *str;**
Convierte a hexadecimal un nùmmero entero. El resto como itod. Los digitos A-F se escriben en

Primero vienen los còdigos de control, luego otros caracteres no alfabèticos, Letras mayùsculas inmediatemente seguidas de su minùscula correspondiente. al final fiene el caracter DEL (Copyright). Los caracteres del segundo juego (128-255) quedan ideflnidos.

4.8 Funciones de clasificaciòn de caracteres.

Estas rutinas devuelven YES o NO de acuerdo con la pertenencia del caracter a un determinado grupo de caracteres. Actualmente no reconoce ningun caracter del juego extendido de caracteres Sinclair para lenguas extranjeras.

4.8.1 **isalnum (c) char c;** caracteres alfanumèricos a-z, A-Z, 0-9.
4.8.2 **isalpha (c) char c;** caracteres alfabèticos a-z, A-Z.
4.8.3 **isascii (c) char c;** caracteres ASCII 0-127.
4.8.4 **iscntrl (c) char c;** caracteres ASCII de control 0-31.
4.8.5 **isdigit (c) char c;** caracteres numèricos 0-9
4.8.6 **isgraph (c) char c;** caracteres graficos ASCII 33-127.
4.8.7 **islower (c) char c;** caracteres minùscula a-z.
4.8.8 **isprint (c) char c;** caracteres imprimibles 32-127(espacio incluido)
4.8.9 **ispunct (c) char c;** caracteres de puntuaciòn ASCII: todos menos los alfabèticos los numèricos y los de control
4.8.10 **isspace (c) char c;** caracter espacio en blanco.
4.8.11 **isupper (c) char c;** letras mayùsculas A-Z.
4.8.12 **isxdigit (c) char c;** digitos hexadecimales A_F, 0-9.

4.9 Funciones de conversiòn de caracteres.

4.9.1 **toascii (c) char c;** convierte c a ascii (lo retorna sin cambio)

4.9.2 **tolower (c) char c;** convierte c a una minùscula si es una mayùscula, en cualquier otro caso devuelve c.

4.9.3 **toupper (c) char c;** convierte c a mayùscuila si es una minùscula, en cualquier otro caso devuelve c.

4.10 Otras facilidades del sistema.

Estas son facilidades variadas del sistema que fecuentemente son facilitadas en otras implementaciones de C en una forma similar. Las facilidades del sistema especificas de QDOS aparecen en la secciòn 5.

4.10.1 **abs (n) int n;** Retorna el valor absoluto del entero n.
4.10.2 **sign (n) int n;** devuelve -1, 0 b +1 segùn el signo de n.

4.10.3 **fread (buff, size, count, fd) char *buff; int size, count, *fd;**
Lee desde el fichero fd dentro de buff count datos de size tamaño de bytes. Ejecuta una transferencia binaria y retorna el nùmero actual de items leidos. Use **feof()** o **ferror()** para determinar la llegada a **EOF** o si existe algùn error.

4.10.4 **fwrite (buff, size, count, fd) char *buff; int size, count, *fd;**
Escribe count items de size bytes de largo desde buff al fichero fd. Ejecuta una transferencia binaria y retorna el nùmero actual de items escritos. Use **feof()** o **ferror()** para determinar la llegada a **EOF** o si existe algùn error.

4.10.5 **read (fd, buff, count) char *buff; int count, *fd;**
Lee desde el fichero fd dentro de buff count bytes. Ejecuta una transferencia binaria y retorna el

número actual de items leidos. Use **feof()** o **ferror()** para determinar la llegada a EOF o si existe algún error.

4.10.6 **write (fd, buff, count) char *buff; int count, *fd;**

Escribe count bytes desde buff al fichero fd. Ejecuta una transferencia binaria y retorna el número actual de items escritos. Use **feof()** o **ferror()** para determinar la llegada a EOF o si existe algún error.

4.10.7 **calloc (count, size) int count, size;**

Localiza count * size bytes de memoria y los inicializa a cero. Devuelve un apuntador a la memoria o cero si no hay memoria libre.

4.10.8 **malloc (count) int count;**

Reserva count bytes de memoria no inicializada. Devuelve un punttero a la memoria o cero si no hay memoria libre.

4.10.9 **avail (abort) int abort;**

Esta rutina devuelve la cantidad de espacio libre entre la pila y el programa. Si abort es distinto de cero y la pila està sobreescribiendo el programa, el programa será abortado.

4.10.10 **free (pointer) char *pointer;**

cfree (pointer) char *pointer;

Estas rutinas son equivalentes. Devuelven la memoria a la pila que había sido grabada por **calloc** o **malloc**. Cualquier reserva puede ser devuelta a la pila en cualquier momento y no hace falta que se liberen en el orden inverso a que fueron reservadas. El puntero ha de ser el que retornò la llamada a **calloc** o **malloc** de otro modo la pila del sistema será destrozada: un puntero a la pila no podrá trabajar.

4.10.11 **getarg (n, str, size, argc, argv)**

char *str; int n, size, argc, *argv;

Extrae el enèsimo paràmetro de la cadena de paràmetros del programa y lo copia en str (con size de tamaño màximo). Vea la sección 7 para màs información sobre la cadena de paràmetros de programa.

4.10.12 **poll (pause) int pause;**

Esta rutina busca si hay alguna pulsación de teclado pendiente para el programa. Si la pause es cero cualquier caracter es devuelto a la llamada (o cero si no había ninguno esperando). Si pause es distinta de cero y el caracter es CTRL-S entonces el programa se suspende hasta que se introduzca un nuevo caracter por el teclado, caracter que no será leido por el programa.

4.10.13 **abort (errcode) int errcode;**

exit (errcode) int errcode;

Estas rutinas son equivalentes. Cierran todos los canales abiertos y devuelven al sistema el errorcode (código de error) que será cero (para indicar èxito) o un mensaje de error de QDOS.

4.10.14 **ccargc()**

Esta función se la llama para determinar el número de paràmetros que se pasan a una función (vea 3.4.8).

5 Rutinas de la librería extra de QDOS.

Las rutinas definidas en esta sección se incluyen para permitir el acceso a las facilidades del sistema operativo QDOS. Són únicas de QC y no son portables a otros compiladores de C.

5.1 Acceso al QDOS.

trap1(punteroreg)
trap2(punteroreg)
trap3(punteroreg)

Estas tres rutinas permiten el acceso directo a muchas de las facilidades del QDOS inplementadas como traps. Vea la "QDOS Software Development Guide" facilitada por Sinclair Research Ltd, para más información sobre el uso de los traps (También en QL programación avanzada de Adrian Dickens ed RAMA en español (N del T)).

Punteroreg es un puntero a una matriz de 8 enteros ordenados como: D0, D1, D2, D3, A0, A1, A2, A3, todos los cuales son puestos al día por los resultados del trap.

Para acceder al canal QDOS asociado con un fichero QC fd (definido int *fd) use *fd. Como ejemplo aquí tiene algunas rutinas de entrada/salida sin buffer:

```
/* INKEY
** Esta rutina proporciona una facilidad semejante a la función INKEY$ del
** SuperBASIC: lee el teclado directamente o devuelve cero si no se
** presiona ninguna tecla durante el periodo_de_espera.
*/
inkey ( periodo_de_espera ) int periodo_de_espera;
{
int regs(8);
regs(0) = 1;   /* IO.FBYTE */
regs(3) = periodo_de_espera;
regs(4) = *stderr; /* A0 = canal a CON_ */
trap3( regs );
if (regs(0) == 0 )    /* si no hay error */
   return ( regs(1) );  /*devuelve el código de la tecla */
else
   return ( 0 );
}
/* OUTSCR
** Es una rutina que acompaña a INKEY y se usa para escribir caracteres a
** la pantalla evitando los buffer de QC.
*/
outscr ( ch ) int ch;
{
int regs(8);
regs(0) = 5;    /* IO.SBYTE */
regs(1) = ch;   /* D1 = caracter a escribir */
regs(3) = -1;
regs(4) = *stderr; /* A0 = canal a CON_*/
trap3( regs );
return ( regs(0) ); /*devuelve el código de error de haberlo */
}
```

5.2 Funciones de pantalla y ventana.

Todas las funciones de ésta sección llaman al trap 3 para modificar la pantalla, retornando códigos

de estado de QDOS. Para cambiae el estilo o color de texto enmedio de una linea, deberà llaaar **fflush()** para escribir el texto anterior a la llamada del controlador de pantalla.

5.2.1 **selwindow(fd) int *fd;**
Esta rutina se usa para elegir la ventana que queremos que se use con las rutinas del controlador de pantalla y gràficos. Por omisiòn se usa la ventana conectada a **stderr.**

5.2.2 **getwindow(flag,puntero) int flag, puntero;**
Esta rutina se utiliza para averiguar el tamaño de la ventana y la posiciòn del cursor. Flag es 0 para tomar la posiciòn en coordenadas de pixel o distinto de cero para obtenerlo en coordenadas de caracter. Puntero especifica el lugar donde se deben colocar los resultados. Debe ser una matriz de cuatro enteros (o un puntero a una matriz mayor).
puntero(0) anchura de la ventana
puntero(1) altura de la ventana
puntero(2) coordenada x del cursor desde laizquierda
puntero(3) coordenada y del cursor desde lo alto de la ventana

5.2.3 **border(size,colour) int size, colour;**
Se utiliza para modificar el borde de la ventana. Size es la anchura que queremos dar al borde y colour el color deseado. (nùmero de color de QDOS)

5.2.4 **window(ancho, alto, x, y) int ancho, alto, x, y;**
Modifica la definiciòn de ventana. Ancho y alto determinan el nuevo tamaño de la ventana y x e y especifican la posiciòn de la misma. El cursor se repone a la esquina superior izquierda de la ventana (posiciòn 0,0).

5.2.5 **cursen(switch) int switch;**
Esta rutina activa o desactiva el cursor de la ventana segùn el valor de switch sea distinto de cero o cero respectivamente.

5.2.6 **at(fila,columna) int fila, columna;**
Esta rutina posiciona el cursor en las coordenadas de caracter especificadas dentro de la ventana.

5.2.7 **tab(col) int col;** Posiciona el cursor en la columna especificada.
5.2.8 **nextline()** Posiciona el cursor al comienzo de la linea siguiente.
5.2.9 **curleft()** Mueve el cursor a la izquierda un espacio.
5.2.10 **curright()** Mueve el cursor un espacio a la derecha.
5.2.11 **curup()** Mueve el cursor a la linea superior màs pròxima.
5.2.12 **curdown()** Mueve el cursor a la siguiente fila.
5.2.13 **cursor (xpos,ypos) int xpos, ypos;**
Esta rutina mueve el cursor a la posiciòn en pixels especificada.

5.2.14 **scroll(distancia,zona) int distancia, zona;**
La ventana o una parte de ella es desplazada verticalmente la distancia especificada. Una distancia positiva mueve el texto hacia abajo en la ventana. La zona se interpreta como sigue:
zona = 0 Toda la pantalla
zona = 1 La zona superior a la linea del cursor de la ventana
zona = 2 La zona inferior de la ventana desde la linea del cursor incluso.

5.2.15 **pan (distancia, zona) int distancia, zona;**
La ventana o una zona de ella es desplazada horizontalmente la distancia especificada. Una distancia positiva desplaza la ventana a la derecha. La zona se interpreta como sigue:
zona = 0 Toda la ventana
zona = 3 Toda la linea del cursor

zona = 4 La linea del cursor desde el cursor hasta el final de la linea.

5.2.16 **cls (zona) int zona;**

La ventana o parte de ella es limpiada con el color de papel actual. La zona se interpreta como sigue:

zona = 0 Limpia toda la ventana.
zona = 1 limpia la ventana por encima del cursor.
zona = 2 Limpia la ventana por debajo de la linea del cursor.
zona = 3 limpia la linea del cursor.
zona = 4 Limpia la linea del cursor desde el cursor nasta el final.

5.2.17 **fount (juego1, juego2) char *juego1, *juego2;**

Esta rutina se utiliza para cambiar el juego de caracteres de la ventana. Una direcciòn de juego 0 elige el juego por defecto. (Vea documentaciòn de QDOS para detalles sobre la estructura de datos de los juegos de caracteres). Si el caracter a escribir no està definido en juego1 entonces se usa juego2.

5.2.18 **recol (tabla) char *tabla;**

Esta rutina recolorea una ventana. Tabla es una matriz de ocho bytes (char) que proporciona los nuevos colores (0-7) para cada uno de los colores que aparecen en la ventana en ese momento. (el orden es 0=negro, 1=azul, 2=rojo, 3=magenta, 4=verde, 5=cyan, 6=amarillo, 7=blanco).

5.2.19 **paper (color) int color;**

Pone el color del papel. Este serà el que se use cuando se limpie la ventana o cuando se realice un scroll o un pan.

5.2.20 **strip (color) int color;**

Pone el color de tira. Este se usa como color de fondo cuando el texto se escribe en la pantalla salvo que se estè escribiendo como texto transparente. Frecuentemente se pone al mismo color que el del papel.

5.2.21 **ink (color) int color;**

Pone el color de la tinta. Este se usa para texto y gràficos salvo que se estè en modo XOR.

5.2.22 **flash (switch) int switch;**

Pone o quita el modo de parpadeo segùn switch sea no-cero o cero. El parpadeo sòlo funciona si la màquina està en el modo de 8 colores.

5.2.23 **under (switch) int switch;**

Pone o quita el modo subrayado segùn switch sea no-cero o cero.

5.2.24 **over (switch) int switch;**

Define el modo de producciòn de caracteres y gràficos:
switch = -1 XOR de la tinta con el fondo.
switch = 0 El fondo del caracter es el color de tira.
switch = 1 El fondo del caracter es transparente.

5.2.25 **csize (ancho, alto) int ancho, alto;**

Pone el tamaño del caracter y su espaciado:
ancho = 0 Espacio de 6 pixels.
ancho = 1 Caracter de 6 pixels en un espacio de 8 pixels.
ancho = 2 ancho de 12 pixels.
ancho = 3 Caracter de 12 pixels en un ancho de 16 pixels.
alto = 0 Altura simple de 10 pixels.
alto = 1 altura doble de 20 pixels.

5.2.26 **block (ancho, alto, xpos, ypos, color)**
int ancho, alto, xpos, ypos, color;
Rellena un rectàngulo (especificado en coordenadas de pixel) del ancho y alto especificados con el color indicado.

5.3 Rutinas gràficas.

En estas rutinas todas las coordenadas estàn en unidades gràficas que dependen de una escala. Los àngulos se especifican en centèsimas de radian dado que QC no soporta la coma flotante.
La excentricidad de la elipse debe pasarse tambièn como centèsimas del número requerido.
Los gràficos se escribiràn en la ventana elejida con **selwindow** (5.2.1) usando el color elegido con **ink** (5.2.21).
Observe que las coordenadas gràficas tienen su origen en la esquina inferior izquierda, no como las coordenadas de pixel que parten de la esquina superior izquierda de la ventana.

5.3.1 **point (x, y) int x, y;** Dibuja un punto.
5.3.2 **line (x1, y1, x2, y2) int x1, y1, x2, y2;** Dibuja una linea.

5.3.3 **arc (x1, x2, y1, y2, àngulo) int x1, y1, x2, y2, àngulo ;**
Se dibuja un arco entre los dos puntos de final. El sentidoo del arco depende del signo del àngulo dibujandose la curva en sentido contrario al de las agujas del reloj.

5.3.4 **circle (xpos, ypos, radio) int xpos, ypos, radio;**
Esta rutina dibuja una circunferencia. Es un caso especial de la rutina ellipse (elipse de excentricidad 1):

```
circle (xpos, ypos, radius) int xpos, ypos, radius;
(
ellipse (xpos, ypos, radius, 100, 0 );
)
```

5.3.5 **ellipse (xpos, ypos, radio, excentricidad, àngulo)**
int xpos, ypos, radio, excentricidad, àngulo;
Radio indica uno de los radios de la elipse. El otro se determina como radio veces la excentricidad. Observe que el paràmetro excentricidad viene en centèsimas. El àngulo es el de la elipse respecto a su eje mayor en centiradianes.
5.3.6 **scale (factor, xorg, yorg) int factor, xorg, yorg;**
La escala se ajusta a la altura de la ventana hacièndola equivalente al factor de escala. xorg e yorg dan las coordenadas internas de gràficos respecto a la esquina inferior izquierda de la ventana. El origen de omisiòn es 0,0 y la escala por defecto 100.

5.3.7 **gcursor (xorg, yorg, derecha, abajo) int xorg, yorg, derecha, abajo;**
Se pone el cursor en la posiciòn xorg y org en coordenadas gràficas con un desplazamiento en pixels derecha abajo.

5.3.8 **fill (switch) int switch;**
Pone el relleno de àreas o lo quita segùn switch valga no-cero o cero.

5.4 Otras facilidades del QDOS.

5.4.1 **delay (ticks) int ticks;**
Espera un corto periodo de tiempo. Un tick es un cincuentavo de segundo.

5.4.2 **adate (segundos) int segundos;**
Ajusta el reloj adelante o atràs el número de segundos indicado. Devuelve el valor del reloj en

segundos desde el 1 Enero 1961.

5.4.3 **qdosdate()** devuelve el valor del reloj.

5.4.4 **date (clock, datos) int clock; int datos();**
convierte clock en un formato de fecha; datos ha de ser un puntero a una matriz de siete enteros distribuidos como sigue:
datos(0) = año
datos(1) = mes
datos(2) = dia del mes
datos(3) = dia de la semana (0 = domingo... 6= sàbado)
datos(4) = hora
datos(5) = minuto
datos(6) = segundo

5.4.5 **sdate (valor) int valor;** pone el reloj a valor.

5.4.6 **beep (duraciòn, tono) int duraciòn, tono;**
Esta rutina proporciona un acceso sencillo al generador del sonido del QL. Duraciòn es un periodo en unidades de 72 microsegundos (14000 = 1 seg). El tono es màs agudo cuanto màs bajo sea su valor.
Otra forma sencilla de generar un sonido se proporciona en la libreria principal: escribir un CTRL-G en la pantalla genera un pequeño pitido. Observe que ambas rutinas regresan inmediatamente. Para esperar a que termine beep debe llamarse delay. El tiempo a esperar serà aproximadamente duraciòn/28 ticks.

5.4.7 **warbie (duraciòn, tono1, tono2, intervalo, paso, envol, ruido, rand)**
int duraciòn, tono1, tono2, intervalo, paso, envol, ruido, rand;
Esta rutina proporciona control total al usuario sobre el generador de sonido del QL. Vea el manual de SuperBASIC para mayor informaciòn de los paràmetros anteriores.

5.4.8 **keyrow (fila) int fila;**
Lee el teclado del QL directamente. Puede usarse para comprobar si se pulsa una o màs teclas. Vea la documentaciòn de SuperBASIC y de QDOS para conocer màs detalles. Los bit a uno en el resultado de èsta funciòn indican que teclas de la fila estàn pulsadas.

5.4.9 **random()** Devuelve un entero aleatorio.
5.4.10 **rnd (min, max) int min, max;** Da un entero aleatorio en el rango.

5.4.11 **readdir (fd, fname, dirinfo) int *fd, *dirinfo; char *fname;**
Esta rutina se utiliza para obtener el contenido de un directorio. fd serà el resultado de una llamada **fopen** con la opciòn = "d". fname es un puntero a una matriz de caracteres lo suficientemente grande para un nombre de fichero (37 caracteres es bastante). y dirinfo es un puntero a una matriz de 8 enteros.
La rutina devuelve normalmente 0 o EOF al final del directorio. Un ejemplo de uso de esta rutina:

```
main()
(
int dirinfo(8); char filename(40); int *fd;
fd = fopen ( "mdv1_", "d" );
while (readdir ( fd, filename, dirinfo ) == 0 )
   ( printf (...
   )
)
```

La informaciòn de la entrada de directorio se pone en la matriz como sigue:
dirinfo(0) tamaño del fichero
dirinfo(1) clave de acceso
dirinfo(2) tipo de fichero

dirinfo(3) información dependiente del tipo
dirinfo(4) información dependiente del tipo
dirinfo(5) fecha de la última modificación
dirinfo(6) fecha del último acceso
dirinfo(7) fecha de la última copia

Lea la documentación del QDOS para mayores detalles sobre la información del directorio. Observe que no todos estos campos de información los soportan todos los controladores de dispositivo.

5.4.12 exec (nombreprograma, optstr, flag)

```
char *nombreprograma, *optstr; int flag;
```

Esta rutina le permite ejecutar otro programa de nombre nombreprograma pasando optstr al programa como línea de parámetros. Si flag vale cero la rutina vuelve inmediatamente después de arrancar el subprograma. Si es no-cero la rutina espera a que el subprograma acabe. Un código de estado de QDOS es devuelto por esta rutina que si el flag es no-cero será el que devuelva el subprograma al acabar. (vea sección 7 (N del T))

6 Acceso con código en ensamblador.

6.1 Uso de los registros.

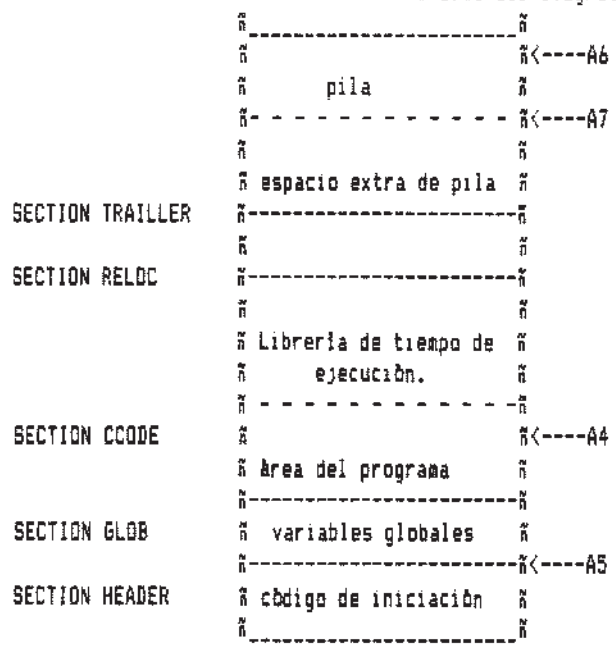
QC utiliza los registros del 68000 del siguiente modo:

D0 Tiene valores temporales en cálculos matemáticos
D1 Tiene valores temporales en cálculos matemáticos
D2 Tiene el número de parámetros pasados a una rutina.
D3 No se usa
D4 No se usa
D5 No se usa
D6 Siempre tiene el valor 1.
D7 Siempre tiene el valor 0.
A0 Es el registro primario. Tiene el valor de la expresión que esta siendo evaluada y el resultado de la función cuando se vuelve de la llamada.
A1 Es el registro secundario. Se usa al evaluar la expresión y suele tener la dirección de la variable que se está actualizando.
A3 No se usa.
A4 No se usa.
A5 Apunta a la mitad del programa. Todas las funciones se acceden mediante un desplazamiento sobre el puntero A4.
A6 Es el puntero de orden de la pila. Apunta a la variable local actualmente en uso de la función dentro de la pila.
A7 Es el puntero a lo alto de la pila del sistema. En el 68000 la pila se desplaza hacia abajo desde la zona alta de la memoria.

Si usted escribe código en ensamblador para usarlo junto a QC, puede usar cualquier registro pero los siguientes han de restaurarse a sus valores originales antes de volver al código QC: D6, D7, A4, A5, A6 y A7.

6.2 El mapa de memoria de un programa en QC.

Las direcciones más altas están en lo alto del diagrama



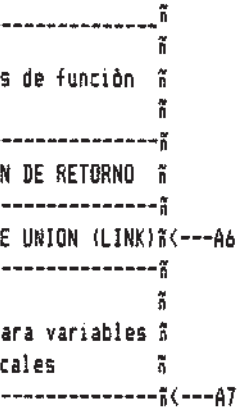
Las secciones HEADER, TRAILLER y RELOC no se deben usar por el programador en ensamblador: todo el código se deberá colocoar en la sección CODDE en compañía del código generado por el compilador. El lugar apuntado por A4 está referenciado por el símbolo M$ pudiéndose llamar a las funciones del siguiente modo: BSR func-M$(A4). Usando este mecanismo se puede acceder a cualquier función de un modo independiente de su posición, permitiendo a los programas en QC ser de unos 64K de tamaño. (M$ está definido actualmente a 32k desde la base del programa.

Se pueden declarar variables globales en ensamblador accediendo a la sección GLOB y usando DC.L o DC.B para declararlas e inicializarlas. El lugar apuntado por A5 lo referencia el símbolo G$ pudiéndose usar otras variables globales del siguiente modo:

```
MOVE.L   globname-G$(A5),D0
MOVE.B   D1,globname-G$(A5)
```

6.3 Estructura de la pila de QC.

QC utiliza las instrucciones LINK y UNLK del 68000 para mantener la estructura principal de la pila. Al llamar una función la llamada pone en la pila sus parámetros. La función usa la instrucción LINK para reservar espacio para las variables globales declarados en el otro nivel de la función. La pila estonces aparece como sigue: (Direcciones altas en lo alto)



Los parámetros de la función se acceden con desplazamientos positivos sobre A6 y las variables locales con desplazamientos negativos. Si un bloque dentro de la función declara también variables locales no se genera otra instrucción LINK, tan sólo se modifica el apuntador de pila A7.

Al final de la función, se utiliza UNLK para restaurar el estado de la pila antes de llamar RTS. Los parámetros los retira de la pila la función que hizo la llamada.

Todos los parámetros se pasan a las funciones como int ej palabras largas de 32bit. Se ponen en la pila en el orden en que se declaran estando el primer parámetro lejos de A6 y el último en la dirección 8(A6) (antes de la dirección de retorno).

Respecto de las variables locales, las variables int y pointers (punteros) ocupan 4octetos y están en frontera de palabra. Short ocupan 2bytes y char 1byte- habrá huecos entre las variables si tiene una mezcla de variables char e int. Si tiene alguna duda de como acceder a variables locales en ensamblador, mire al listado de salida del compilador.

El modo más simple de utilizar código en ensamblador es utilizar el hecho de que el compilador da el resultado de una expresión en el registro A0, el cual lo puede usar en su código en ensamblador. Si al final de una función aparece código en ensamblador, el valor devuelto por la función estará en el registro A0.

6.4 Ejemplo de inserción de código.

Este ejemplo está tomado del módulo de gráficos en la librería. Utiliza las rutinas de coma flotante de QDOS para manipular números reales en la pila de coma flotante. fp_tos es un puntero dentro de la matriz fp_stack.

```
char fp_stack(300), *fp_tos;
/* FP_DIVIDE
** Divide el siguiente de la pila por el primero de la pila (nos/tos)
*/
fp_divide()
{
fp_tos; /* toma el puntero de la pila de coma flotante en A0 */
#asm
    MOVE.L   A0,A1     toma el puntero de la pila en A1
    MOVE.L   A6,-(SP)  salva el valor de A6 de QC
    SUBA.L   A6,A6     pone A6 a cero para el QDOS
    MOVEQ    #$10,D0   código de QDOS de dividir
    MOVE.L   $11C,A2   toma la dirección de la rutina de coma flotante
    JSR      (A2)      llamada a la rutina de coma flotante de QDOS
    MOVE.L   (SP)+,A6  restaura el valor de A6 de QC.
#endasm
fp_tos += 6; /* actualiza nuestra copia del puntero a la pila */
```

7 Línea de comando y redirección de entrada/salida.

Si usted posee el QL Toolkit o si usted inicia un programa con la rutina de librería exec, tendrá la posibilidad de pasar información al programa cuando usted lo ejecute. También podrá modificar los canales standard de entrada y de salida.

7.1 Como pasar una línea de comando al programa.

Para pasar una cadena de texto a un programa en QC puede utilizar los comandos ET, EX ó EW. Vea la documentación del Toolkit para mayor información. El comando podría verse así:

EX <nombre de programa> ; <cadena>

donde <cadena> debe ser una expresión de cadena de SuperBASIC.

La cadena de la línea de comando la analizará el código de iniciación de QC y las palabras dentro de ella serán procesadas individualmente. (el analizador busca secuencias de caracteres separadas por espacios).

7.2 Como redirigir los canales standard de entrada salida.

Si los nombres de fichero de datos están incluidos en la línea de comando de EX, el primer fichero de datos será tomado como entrada standard del programa y el último como salida standard (pero si se han depfinido pipes, estos serán usados en su lugar). Ejemplos:

EX QC,MIFICHERO_C,MIFICHERO_ASM

Esto ejecutará el compilador, que leerá de MIFICHERO_C como unidad por defecto y escribirá en MIFICHERO_ASM. Observe que se han omitido los nombres de dispositivo de directorio tal y como se indica en el Toolkit pero la extensión hay que definirla explícitamente.

EX MIPROGRAMA_BIN,FICHERODEDATOS

Esto ejecutará el programa MIPROGRAMA_BIN que leerá de FICHERODEDATOS pero la salida standard permanecerá en la pantalla.

EX MIPROGRAMA_BIN,FICHERODEDATOS,UNFICHERO TO OTROPROGRAMA

En este caso MIPROGRAMA leerá de FICHERODEDATOS y mandará su salida por el pippe a OTROPROGRAMA ignorando a UNFICHERO. (Vea la documentación del QL Toolkit).

Por compatibilidad con operativos como el UNIX también se pueden usar parèntesis angulados para la redirección de E/S.

Si en la cadena de opciones de un comando EX aparece lo siguiente sobreescribirá cualquier otra redirección de E/S. No se admiten espacios entre los parèntesis angullados y los nombres de fichero.

<nombredefichero Abre el fichero como entrada standard.

>nombredefichero Abre el fichero como salida standard.

>>nombredefichero Abre el fichero como salida standard pero añadiendo el texto al final del fichero.

Especificaciones de redirección como estas no pueden pasarse a los programas de usuario como paràmetros (ver más abajo).

7.3 Interpretación de una línea de comando desde un programa.

Para utilizar la línea de comando pasada desde SuperBASIC, la función **main** del programa deberá declararse como sigue:

main (argc, argv) int argc, *argv();

argc es el número de argumentos pasados al programa (mas uno por el nombre de programa), y **argv** es un puntero a una matriz de punteros a las cadenas de los argumentos. Si por ejemplo, un programa se arranca con el siguiente comando: EX PROG;" <MDV2_DATAFILE ABC >>MDV1_FILENAME def 99 @ "

argc tendrá el valor 5 y argv apuntará a una variable como esta:

Observe que argv(0) apunta a una cadena que contiene un asterisco. Este es un puntero al nombre del programa y sistema operativo que lo soporta.

La rutina de librería **getarg** puede usarse para acceder a los parámetros en esta estructura de datos. Ella se define así:

getarg (n, str, tamaño, argc, argv) char *str; int n, tamaño, argc, *argv;

La rutina copia el enèsimo parámetro en la cadena str y puede truncarse a tamaño octetos si es muy larga. argc y argv deben pasarse como se reciben del sistema operativo. Ejemplo:

```
main ( argc, argv ) int argc, *argv;
{
char buff(20);
getarg ( 2, buff, 20, argc, argv );
}
```

getarg devuelve EOF y pone una bytes null en str si llega al final de la lista de argumentos.

Apèndice A Mensajes de error del compilador.

Cuando el compilador detecta un error en el còdigo, imprime un mensaje de error a la pantalla y si se està produciendo un listado, también en el listado. El mensaje de la pantalla consiste en la linea errònea seguida de un puntero señalando el lugar de la linea donde està el error y seguido de un mensaje.

Si se ha elejido la opciòn -A el compiladar emitirà un pitido para llamar la atenciòn y si se ha elejido la opciòn -P esperarà a que usted presione enter para continuar.

Al final de la compilaciòn, el compilador imprimirà una lista con cualquier simbolo indefinido en la pantalla y en el fichero de listado. Primero vienen los nombres de funciones no declarados ue se asume seràn funciones externas; seguidos de los nombres no declarados que no estan como funciones que se asumiràn como **global** no definidos y se trataràn como error.

A continuaciòn se describen en orden alfabètico los mensajes de error.

already defined El simbolo ya ha sido usado. Puede crear una variable local con el mismo nombre de una global o de una local de otro bloque pero no puede haber dos variables globales con el mismo nombre ni dos locales iguales en el mismo bloque.

bad label La etiqueta es invàlida o se ha perdido.

can't subscript Sòlo tienen subìndices punteros y matrices.

cannot assign to pointer Los punteros no se pueden inicializar salvo los de char que pueden inicializarse con una constante de cadena.

cannot assign Se ha intentado asignar a alo que no es un lvalor (también se produce con los operadores de incremento con algo que no sea lvalor). Debe poner algo vàlido al lado izquierdo de la asignaciòn como una variable, un puntero indirecto o una matriz con subìndice.

cannot initialise local arrays Sòlo se pueden inicializar matrices globales las locales no.

error opening file Se ha producido un error al abrir un fichero en la compilaciòn: este es un error fatal.

expresion too complicated El compilador admite solo hasta 12 niveles de anidamiento de parèntesis. Si se obtiene este error, simplifique la expresiòn o ròmpala en varias expresiones màs pequeñas y use variables temporales.

failed to open include file Se ha producido un error al acceder a un fichero include. El compilador proseguirà despuès de este error.

function body must be a compound statement QC requiere un cuerpo de funciòn hecho de una sentencia compuesta, otros tipos de sentencia no se permiten.

global symbol table overflow Hay demasiados simbolos globales para la tabla del compilador. Hay un espacio para unos 500 simbolos globales. Si obtiene este error inntente romper el programa en mòdulos màs simples y pequeños.

illegal address El operador de direcciòn & ha sido usado sobre algo que no tiene direcciòn.

illegal argument name Hay algùn error de sintaxis en el nombre de argumento de una funciòn (ej es un nombre reservado).

illegal array size Los tamaños de matriz negativos son ilegales.

illegal function or declaration Este es un error de sintaxis en el nivel de definiciòn de funciones o de variables globales: el compilador no encuentra sentido a la declaraciòn.

illegal symbol El nombre de simbolo tiene caracteres invàlidos o es un nombre reservado.

invalid expression Un tèrmino de la expresiòn es invàlido. Son tèrminos vàlidos constantes, variacles o constantes de cadena. Los nombres de etiquetta y palabras reservadas son invàlidos.

line too long Despuès de que el preprocesador ha hecho la sustituciòn de macros la linea excede los 125 caracteres. Deberà simplificar la linea o romperla en varias lineas.

literal queue overflow Hay demasiadas constantes de cadena en una funciòn o estas son muy largas. El compilador salva las constantes de cadena hasta el final de la funciòn. Hay espacio para unos 800 caracteres. Si obtiene este error, rompa la funciòn en funciones màs pequeñas.

local symbol table overflow Hay demasiadas variables locales en la funciòn para la tabla de simbolos locales del compilador. Rompa la funciòn en funciones màs pequeñas.

locals not allowed in switch No se permiten variables locales en el bloque de una sentencia switch.

APENDICE B Sumario de las rutinas de la librería.

Este apèndice contiene una lista ordenada de las funciones de la librería. Es un sumario de los parámetros de las funciones y se incluye para referncia ràpida. El número de sección donde se describe la función se proporciona a la derecha.

abort	(còdigodeerror) int còdigodeerror;	4.10.13
abs	(n) int n;	4.10.1
adate	(segundos) int segundos;	5.4.2
arc	(x1,y1,x2,y2,àngulo) int x1,y1,x2,y2,àngulo;	5.3.3
at	(fila,columna) int fila,columna;	5.2.6
atoi	(str) char *str;	4.6.1
atoib	(str,base) char *str; int base;	4.6.2
avail	(abort) int abort;	4.10.9
beep	(duraciòn,tono) int duraciòn,tono;	5.4.6
block	(ancho,alto,x,y,color) int ancho,alto,x,y,color;	5.2.26
border	(tamaño,color) int tamaño,color;	5.2.3
calloc	(cuenta,tamaño) int cuenta,tamaño;	4.10.7
ccarg	()	4.10.14
cfree	(puntero) char *puntero;	4.10.10
circle	(xpos,ypos,radio) int xpos,ypos,radio;	5.3.4
clearerr	(fd) int *fd;	4.3.15
cls	(zona) int zona;	5.2.16
csize	(ancho,alto) int ancho,alto;	5.2.25
curdown	()	5.2.12
curleft	()	5.2.9
curright	()	5.2.10
cursen	(switch) int switch	5.2.5
cursor	(xpos,ypos) int xpos,ypos;	5.2.13
curup	()	5.2.11
date	(clock,datevec) int clock,datevec();	5.4.4
delay	(ticks) int ticks;	5.4.1
delete	(nombre) char *nombre;	4.3.12
dtoi	(str,num) char *str; int num;	4.6.5
ellipse	(x,y,radio,exc,angulo) int x,y,rad,exc,angulo;	5.3.5
exec	(nprog,optstr,flag) char *nprog,*opstr; int flag;	5.4.12
exit	(codigodeerror) int codigodeerror;	4.10.13
fclose	(fd) int *fd;	4.3.3
feof	(fd) int *fd;	4.3.13
ferror	(fd) int *fd;	4.3.14
fflush	(fd) int *fd;	4.3.9
fgetc	(fd) int *fd;	4.3.4
fgets	(str,tamaño,fd) char *str; int tamaño, fd;	4.3.6
fill	(switch) int switch;	5.3.8
flash	(switch) int switch;	5.2.22
fopen	(nombre,modo) char *nombre,*modo;	4.3.1
fount	(fuente1,fuente2) char *fuente1,fuente2;	5.2.17
fprintf	(fd,str,arg1,arg2,...) int*fd,char *str;	4.5.2

fputs	(str,fd) char *str; int *fd;	4.3.8
fread	(buff,tamaño,cuenta,fd) char *buff;int tamaño,cuenta,*fd;	4.10.3
free	(puntero) char *puntero;	4.10.10
freopen	(nombre,modo,fd) char *name,*modo; int *fd;	4.3.2
fscanf	(fd,str,arg1,arg2,...) int *fd;char *str;	4.5.4
fwrite	(buff,tamaño,cuenta,fd) chr *buff;int tamaño,cuenta,*fd;	4.10.4
gcursor	(xorg,yorg,dcha,izda) int xorg,yorg,dcha,izda;	5.3.7
getarg	(n,str,tamaño,argc,argv) char *str;int n,tamaño,argc,*argv;	4.10.11
getc	(fd) int *fd;	4.3.4
getchar	()	4.2.1
getpos	(fd) int *fd;	4.4.2
getwindow	(flag,puntero) int flag,puntero();	5.2.2
ink	(color) int color;	5,2,21
isalnum	(c) char c;	4.8.1
isalpha	(c) char c;	4.8.2
isascii	(c) char c;	4.8.3
isatty	(fd) int *fd;	4.3.10
iscntrl	(c) char c;	4.8.4
iscons	(fd) int *fd;	4.3.11
isdigit	(c) char c;	4.8.5
isgraph	(c) char c;	4.8.6
islower	(c) char c;	4.8.7
isprint	(c) char c;	4.8.8
ispunct	(c) char c;	4.8.9
isspace	(c) char c;	4.8.10
isupper	(c) char c;	4.8.11
isxdigit	(c) char c;	4.8.12
itoa	(num,str) int num; cahr *str;	4.6.3
itoab	(num,str,base) int num; char *str; int base;	4.6.4
itod	(num,str,tamaño) int num,tamaño;char *str;	4.6.9
itoo	(num,str,tamaño) int num,tamaño;char *str;	4.6.10
itou	(num,str,tamaño) int num,tamaño;char *str;	4.6.11
itox	(num,str,tamaño) int num,tamaño;char *str;	4.6.12
keyrow	(fila) int fila;	5.4.8
left	(str) char *str;	4.7.1
lexcmp	(str1,str2) char *str1,*str2;	4.7.12
lexorder	(c1,c2) char c1,c2;	4.7.13
line	(x1,y1,x2,y2) int x1,y1,x2,y2;	5.3.2
lseek	(fd,desplaz,desde) int *fd,desplaz,desde;	4.4.3
malloc	(cuenta) int cuenta;	4.10.8
nextline	()	5.2.8
otoi	(str,num) char*str;int num;	4.6.6
over	(switch) int switch;	5.2.24
pan	(distancia,zona) int distancia,zona;	5.2.15
paper	(color) int color;	5.2.19
point	(x,y) int x,y;	5.3.1
poll	(pausa) int pausa;	4.10.10

printf	(str,arg1.arg2,...) char *str;	4.5.1
putc	(c,fd) char c;int *fd;	4.3.7
putchar	(c) int c;	4.2.2
puts	(str char *str;	4.2.3
qdosdate	()	5.4.3
random	()	5.4.9
read	(fd,buff,cuenta) char *buff;int cuenta,*fd;	4.10.15
readdir	(fd,nombref,dirinfo) char *nombref;int *fd,*dirinfo;	5.4.11
recol	(tabla) char *tabla;	5.2.18
reverse	(str) char *str;	4.7.11
rewind	(fd) int *fd;	4.4.1
rnd	(min,max) int min,max;	5.4.10
scale	(escala,xorg,yorg) int escala,xorg,yorg;	5.3.6
scanf	(str,arg1,arg2,...) char *str;	4.5.3
scroll	(distancia,zona) int distancia,zona;	5.2.14
sdate	(valor) int valor;	5.4.5
selwindow	(fd) int *fd;	5.2.1
sign	(n) int n;	4.10.2
strcat	(destino,fuente) char *destino,*fuente;	4.7.2
strchr	(str,c) char *str,c;	4.7.9
strcmp	(str1,str2) char *str1.*str2;	4.7.4
strcpy	(destino,fuente) char *destino,*fuente;	4.7.6
strip	(color) int color;	5.2.20
strlen	(str) char *str;	4.7.8
strncat	(destino, fuente,n) int n;char *destino,*fuente;	4.7.3
strncmp	(str1,str2,n) int n;char *str1,*str2;	4.7.5
strncpy	(destino,fuente,n) int n;char *destino,*fuente;	4.7.7
strrchr	(str,c) char*str,c;	4.7.10
tab	(col) int col;	5.2.7
toascii	(c) char c;	4.9.1
tolower	(c) char c;	4.9.2
toupper	(c) char;	4.9.3
trap1	(punteroreg) int *punteroreg;	5.1
trap2	(punteroreg) int *punteroreg;	5.1
trap3	(punteroreg) int *punteroreg;	5.1
under	(switch) int switch;	5.2.23
ungetc	(c,fd) char c;int fd;	4.3.5
unlink	(nombre) char *nombre;	4.3.12
utoi	(str,num) char *str; int *num;	4.6.7
warble	(duración,tono1,tono2,interv,paso,envolv,ruido,aleatorio)	5.4.7
window	(ancho,alto,x,y) int ancho,alto,x,y;	5.2.4
write	(fd,buffer,cuenta) char *buffer; int cuenta *fd;	4.10.6
xtoi	(str,num) char *str;int *num;	4.6.8

APENDICE C Sumario de las opciones del ensamblador y del linker.

(N del T) El contenido de èste apèndice ha sido trasladado a su lugar correspondiente en la sección 2 y es allí donde se deben referenciar las opciones de èstos programas.

APENDICE D Diferencias entre QC y el C standard.

Este apèndice proporciona un breve sumario de las diferencias entre el lenguaje QC y el C standard definido en el apèndice A del libro "The C Programing Lenguage" de Kernighan y Ritchie.

Diferencias principales
Coma flotante y estructuras y todas las caracteristicas del lenguaje relacionadas con ellos (uniones, bitfields, typedef) no son soportadas.
El tipo retornado por una función es siempre int y no puede especificarse otro.
QC no soporta matrices multidimensionales.

Otras diferencias (Numeradas de acuerdo al libro de K y R)

2.2 **Identificadores:** Los identificadores externos noo pueden empezar con un subrayado y los 8 primeros caracteres se convierten a mayúsculas. Los nombres de los registros del 68000 estàn reservados
2.4 **Constantes:** Las constantes octales no aceptan los digitos 8 y 9. La constante long explicita (letra L al final) no es aceptada.
2.5 **Cadenas:** Las cadenas no pueden dividirse usando \ seguido de newline.
7.2 **Operadores unarios:** Los operadores Casts y sizeof no son soportados.
7.13 **Operador condiconal:** No es soportado en QC.
8.1 : Declaraciones estàticas no pueden usarse en medio de un bloque.
8.6 **Inicialización:** Nose pueden inicializar los punteros globales (salvo *char que puede inicializar una constante de cadena)
9.7 **Sentencia switch:** No se admiten variables locales en una sentencia Switch.
9.11 **Sentencia goto:** QC No admite goto en funciones con variables locales en el bloque
12.1 **Reemplazo de token:** QC no soporta macros con paràmetros.
12.3 **Compilación condicional:** La instrucción #if no se soporta.
12.4 **Control de linea:** La instrucción #line no se soporta.

ANEXO AL COMPILADOR QC

LIBRERIA QDOSBIS.LIB ISCSoft 1987

Como complemento al compilador [illegible] de GST Computer Systems Limited, se presenta [illegible] de librerias que [illegible] todas las facilidades del QDOS que no aparecian disponibles con [illegible] original [illegible] mas sencillo de las llamadas de entrada [illegible] de un modo mas parecido al comando [illegible] de SuperBasic.

Anexo.1 freemem()
Esta funcion [illegible] la cantidad de memoria disponible en el area de usuario.

Anexo.2 mode (res) int res;
[illegible]

Anexo.3 baud (frec) int frec;
Permite asignar la frecuencia de transmision en los puertos serie. [illegible] 2400, 4800, 9600 o 19200 [illegible]

Anexo.4 alchp (mem) int mem;
[illegible] en el area de usuario [illegible] su direccion.

Anexo.5 rechp (puntero) char *puntero;
[illegible] una zona de memoria asignada [illegible]

Anexo.6 format (medio) char *medio;
Esta rutina formatea un medio [illegible] sectores

Anexo.7 inkey (espera, fd) int espera, *fd;
[illegible]

Anexo.8 sbyte (ch, fd) int ch, *fd;
[illegible]

Anexo.9 edlin (longbuff, fd, strbuff) int longbuff, *fd; char *strbuff;
[illegible]

Anexo.10 mdinf (fd, nombre) int *fd; char *nombre;
[illegible]

Anexo.11 lbytes (largo, base, fd) int largo, *base, *fd;
Esta rutina carga un fichero abierto al canal [illegible] memoria asignada [illegible] todo el fichero.

Anexo.12 sbytes (largo, base, fd) int largo, *base, *fd;
[illegible]

Anexo.13 heads (fd, buffer) int *fd; char *buffer;
[illegible]

Las cabeceras standard de los ficheros tienen 64 octetos:

$00 longitud del fichero

$04 acceso al fichero (octeto)

$05 tipo de fichero (0 para los ficheros ordinarios, 1 para los ejecutables)

$06 8 octetos de información dependiente del tipo; los 4 primeros octetos contienen el espacio asignado correspondiente al espacio de datos del programa.

$0E longitud del nombre del fichero

$10 Nombre del fichero (hasta 36 caracteres)

$34 fecha de actualización

$38 fecha de referencia

$3C fecha de copia

Anexo.14 headr (fd, buffer) int *fd;char *buffer;

Esta rutina lee la cabecera del fichero abierto al canal fd y lo coloca en el buffer especificado en memoria. El buffer ha de tener por lo menos 64 octetos de longitud.

Anexo.15 input (str, size, fd) int size *fd;char *str;

Esta rutina acepta una cadena terminada por un LF a través de un canal fd de una longitud máxima especificada por size. Esta rutina elude los buffers del C y coloca la cadena en memoria sin procesar llamada str.

Anexo.16 report (status) int status;

Esta rutina manda un mensaje de error estándar de QDOS negativo especificado en status al canal #0.