c99 User's Manual

Edition 1 : 85/09/21

Software and Documentation written and adapted by
Clint Pulley

****

User-supported software

The c99 compiler, libraries, associated software and documentation are provided for your use and that of your friends and colleagues.

You are encouraged to distribute it freely provided you charge no more than media and reasonable distribution costs. All copies of the c99 release diskette must include this disclaimer.

If you are using this software and find it of value, your donation ($20.00 suggested) to the author will be appreciated and will help support further development of this product. Contributing users will be placed on a mailing list and advised of new releases.

If you develop useful applications using c99 you may market them provided that the software and documentation acknowledge the use of c99. The author would appreciate receiving a complimentary copy of any such programs.

Please mail all correspondence to:

Clint Pulley
38 Townsend Avenue
Burlington, Ontario
Canada L7T 1Y6

416/639-0583
STC TI7395

Any correspondence requiring a reply must include a stamped, self-addressed envelope. Requests for copies of the c99 release diskette must include a formatted diskette (SSSD please) in a mailer and return postage.

This software carries no warranty , either written or implied, regarding its performance or suitability. Neither the author nor any subsequent distributor accepts any responsibility for losses which might derive from its use.

****

I. INTRODUCTION

c99 is based on small-c version 1 which was published by Ron Cain in Dr. Dobb's Journal No. 45, May 1980. Small-c is a useful subset of the C programming language. It produces assembler source code as its output. This code is assembled to produce an object file which is loaded together with all required libraries and run.

c99 was written and adapted by Clint Pulley as a personal project. It has the potential for considerable further development if sufficient interest is shown by the TI-99/4A user community.

c99 has these features:

- -It supports a subset of the C language.
- -Most of the compiler is coded in c99.
- -It is syntactically identical to standard C.
- -It produces assembler source code rather than an object file.
- -It is a stand-alone single pass compiler. It does its own syntax checking and parsing.
- -It can compile itself.

Although c99 lacks many of the features of standard C systems and produces code which is less than optimal, it is, in the opinion of the author, a worthwhile addition to the software repertoire of the TI-99/4A computer. For the first time a structured language with a true compiler is available to TI users. c99, even at its present level of development, is sufficiently powerful for the development of utilities, text processors, database systems and games. With future compiler enhancements and new libraries its capabilities will increase.

Initial testing of this version (including compilation of the compiler itself) has not revealed any glaring errors. If you find bugs or wish to suggest improvements, please drop the author a line or leave a message on The Source (TI7395). Collect phone calls will not be accepted.

c99 was developed on a 1983 (black) TI-99/4A with 32k memory expansion, TI disk controller, two SSSD drives, RS232 interface and a Centronics printer. The Editor/Assembler software environment was used for all software development. All code was written or adapted by Clint Pulley.

This manual assumes a knowledge of standard C or the availability of a suitable reference. The file C99SPECS, found on the release diskette, identifies the features of C which are available in c99.

II. USING c99

A. Entering the source program

Input to the compiler must be a display/ variable 80 disk file or (for test purposes) the console keyboard. The standard TI editor is normally used for this purpose. If TI-Writer is preferred, be sure to save the source program with the Print File option to avoid getting a line of tab data at the end.

B. Compiling the source program

From the Editor/Assembler menu, select option 5 (Load Program File). Enter DSKn.C99C, where n is the diskette drive containing the compiler disk. When the compiler has been loaded it will identify itself and ask about a number of options. The questions asked are:

Include c-text?

This provides the option of including the c99 source code as comments in the output file. If your response is y or Y each line of c source will appear in the output file preceded by an asterisk. This option should not be selected for large programs as it will result in a very large output file.

Main program?

This controls the insertion of startup code at the beginning of the output file. You must reply with y or Y if the function main() occurs in the input file. If you are building a function library, the startup code is not wanted so reply n or N.

The compiler will then prompt for the input and output filenames. In each case, respond with the full filename (in upper or lower case). If c99 is unable to open a file it will display "Bad filename" and prompt for another name. If the response is a null name (pressing enter only) that file will map to the keyboard or screen. This may be useful for providing a quick check of a short program. Screen output will pause when a key is pressed.

The compiler will now proceed to process the source program. As each function header is encountered, the first six characters of the function name are displayed on the screen. If it is desired to abort execution at any time, press FCTN 4 (CLEAR). This will terminate the compiler and close all files.

When the compiler has finished, it will display !!Errors!! on the screen only if errors were found. It will then ask if more compilations are to be done. A response of y or Y will restart the compiler, a response of n or N will exit.

C. Assembling the compiler output

The output file should be assembled using the TI Assembler (option 2 from the Ed/Asm menu). The R option is not required since c99 generates numeric register references. This reduces the size of the output file and speeds the assembly slightly.

D. Loading the program and libraries

The object file from the assembly may be loaded using option 3 from the Ed/Asm menu. After loading the object file the CSUP library must always be loaded. If the program uses file I/O and contains the statement:

```
#include dsk1.stdio
```

then the CFIO library must also be loaded.

E. Running the loaded program

When the last file has been loaded, press enter to display the program name prompt. All c99 programs begin execution at the entry point START. When this has been entered, your program will (hopefully) execute correctly. At program exit the message "c99exit-press any key" will be displayed. This ensures that the screen will not be cleared at the instant the program stops. Press any key to exit to Ed/Asm.

III. ERROR MESSAGES

When c99 detects an error in the source program it inserts three comment lines in the output file. The first line displays the source line containing the error. The second line is a pointer to the approximate location of the error. The third line is a message of the form:

```
"*!! Error nn"   where nn is the error number
```

If any errors occur during a compilation, "!!Errors!!" is displayed on the screen.

The documentation file C99ERRORS contains a description for each error. The program ERRFIND, included on the release diskette, will scan an output file and pause at each error.

Notes:

a) The source line displayed has been pre-processed, so all multiple spaces have been removed, all names have been truncated to six characters, and all macro substitutions have been performed.

b) The error handling in this version of c99 leave something to be desired in that a single error will often produce a veritable multitude of error messages. This problem has been noted in other (expensive) C compilers.

IV. LIBRARIES AND INCLUDE FILES

The object libraries provided with this release are:

a) CSUP - The compiler support library. This contains the initialization, exit, and direct support (C$) functions required by all c99 programs as well as console I/O functions.

b) CFIO - The file input/output library. This contains the file tables and all file I/O functions.

The include files provided with this release are:

a) CONIO - I/O definitions for console functions only.

b) STDIO - I/O definitions for console and file functions as well as REF directives for all functions in CFIO. If STDIO is included in a program, CONIO must not be included or duplicate definitions will result.

V. LIBRARY FUNCTIONS

Each function is introduced by a sample call. If a function returns a value, an assignment is shown. You may, of course, discard the function result. Arguments must be of the same type as the sample.

The following declarations specify the type of all variables and arguments used in the sample calls.

```
int c,row,col,key,unit;
char buff[81];
char *mode,*name,*string;
```

Functions in CSUP:

-Read one character from the keyboard.

```
c=getchar();
```

Waits for a key to be pressed and returns the character value. The character is echoed to the screen. If the character is "ENTER", the screen spaces to the start of a new line and a value of 10 (EOL) is returned. If the character is CTRL-Z, -1 (EOF) is returned.

-Write one character to the screen.

```
c=putchar(c);
```

Writes the character whose ascii value is c to the screen. If c== 10 (EOL), the screen spaces to the start of a new line. If c==8 (BS), the cursor is backspaced. If c==12 (FF),the screen is cleared and the cursor is homed. If c represents a non-printing character, a \ is echoed. The value of c is returned.

-Read a line from the keyboard.

```
c=gets(buff);
```

Reads one line from the keyboard into a character array. The line is terminated with ENTER, CTRL-Z or the 80th character. The array is assumed to be 81 characters characters long and a null (0) character is appended to the end of the string. A value of buff is returned unless CTRL-Z is pressed. In that case, 0 (NULL) is returned. Use of the backspace key (FCTN-S) for inline editing is supported.

-Write a string to the screen.

```
puts(string);
```

Writes a string to the screen, stopping when it finds a null character. The null character is not written. The cursor is not spaced to the start of a new line unless newline (\n) is encountered.

-Exit the program.

```
exit(c);    or    abort(c);
```

Branches to the c99 exit function which closes any open files. The value of c may be between 0 and 7. If c==0, the normal exit message is displayed. Otherwise the value of c is displayed in an error message. The c99 exit function is also executed when function main terminates normally.

-Locate the cursor on the screen.

```
locate(row,col);
```

Places the cursor at the screen location specified by row and col. Subsequent console I/O will start at the new cursor location. Row and column numbering start at 1 as in TI Basic. The

validity of row and col is not checked.

-Check keyboard status.

```
key=poll(c);
```

Scans the keyboard and returns the key value (if one is pressed) or 0. If c != 0, the program will pause while a key is down. If c != 0 and FCTN-4 (CLEAR) is pressed, the program will branch to the c99 exit function.

Functions in CFIO:

Note: In all of the file I/O functions which reference the argument "unit", the operation will default to the corresponding console I/O function if the value of unit is <= 0. For this reason, the values for stdin, stdout, and stderr as defined in STDIO are -1, -2 and -3.

-Open a file.

```
unit=fopen(name,mode);
```

Opens the named file in the specified mode. Both name and mode must be strings or pointers to strings. Currently supported modes are:

```
"r" - read
"w" - write
"u" - update
"a" - append
```

A unit number is returned for use with the file I/O functions. This unit number must not be altered. If the open fails, NULL (0) is returned. No more than four files may be opened and no more than three may be disk files. At this time, only display/variable 80 files are supported. The filename may be upper or lower case.

-Close a file.;

```
c=fclose(unit);
```

Performs the appropriate file closing action and makes the unit available for another file. In the case of output files being written with putc, an incomplete line is lost. This function returns NULL if the close fails and non-null if it succeeds. All open files are closed automatically if a program terminates

normally.

-Read one character from a file.

```
c=getc(unit);
```

Reads and returns the next character from the file corresponding to unit. If the end-of-line is reached a value of 10 (EOL) is returned. If the end-of-file is reached, a value of -1 (EOF) is returned. If an error occurs, -2 (ERR) is returned.

-Write one character to a file.

```
c=putc(c,unit);
```

Writes the character whose ascii value is c to the file. If c==10 (EOL), the actual write operation occurs. The value of c is returned. If an error occurs, -2 (err) is returned.

-Read a line from a file.

```
c=fgets(buff,col,unit);
```

Reads one line from the file into a character array. At most, col-1 characters will be transferred. A null character is appended to the end of the line. If a partial line is transferred, the remainder of the line is discarded. If unit<=0, gets is called and the value of col is ignored. This could result in buffer overflow. A value of buff is returned. If an end-of-file or error condition occurs, NULL is returned.

-Write a string to a file.;

```
c=fputs(string,unit);
```

Writes a string to a file, stopping when it finds a null character or eighty characters have been transferred. A value of buff is returned. On end-of-file or error, NULL is returned.

-Rewind a file.

```
rewind(unit);
```

If a disk file is open for read or append it is rewound. All other cases are ignored.

VI. ASSEMBLY LANGUAGE INTERFACE

Interfacing to assembly language is relatively straightforward.

The "#asm ... #endasm" form allows the placing of assembly source code directly into the program. Since the compiler considers it to be a single statement, it may be used as:

```
while(1) #asm ... #endasm
or
if(expression) #asm ... #endasm else ...
```

In actual program coding, the #asm directive must be the last item on a line and the #endasm directive must appear on a line by itself. Since the compiler is free-format otherwise, the expected format is:

```
if(expression) #asm
---
---
#endasm
else statement;
```

A semicolon is not required after #endasm.

Assembly code within the "#asm ... #endasm" form has access to all global symbols and functions by name. It is the programmer's responsibility to know the data type of the symbol (whether "int" or "char" implies word or byte access). Stack locals and arguments may be retrieved by offset.

The push-down stack used by c99 is located in the upper part of the low (8k) bank of the TI-99/4A memory expansion. Register 14 in the c99 workspace is reserved as the stack pointer. The stack begins at >3FFE and grows towards >2000.

External assembly language routines accessed by function calls from c code may use registers R0 thru R7. They may push items on the stack, but must pop them off before exit. It is the responsibility of the calling program to remove arguments from the stack after a function call. Since arguments are passed by value, the arguments on the stack may be modified by the called program.

VII. MEMORY UTILIZATION

-PAD Usage.

c99 reserves PAD locations >8300 thru >832F for its workspace and support code. Other PAD locations not used by console routines are available to the programmer. In particular, locations >8330 thru >8348 can be used to store frequently accessed global variables by use of AORG and DORG directives. The file SIEVE;C, found on the release diskette, uses this technique.

The c99 workspace is at >8300. Register utilization is:

R0-R7	temporary storage
R8	primary computation register
R9	secondary computation register
R10	subroutine address for indirect calls
R11	return address for BL instruction
R12	address of recursive subroutine call routine
R13	address of recursive subroutine return routine
R14	the stack pointer
R15	first word of the PUSH routine (hence BL 15)

-Memory Expansion Usage.

The entire 24k bank of memory is available for program usage. The 8k bank contains the standard Editor/Assembler utilities (>2000 thru >2676). As previously mentioned, the c99 stack grows down from >3FFE. Since typical stack usage is a few hundred bytes, the intervening space is available.

Since no indication of stack/program overlap is provided in this version of c99, very large programs could crash. However, the c99 compiler which uses all of the 24k bank and much of the 8k bank is able to compile itself successfully, so this problem may never arise for most users.

-VDP Ram Usage.

Aside from the areas normally used in the Ed/Asm environment, c99 reserves VDP memory locations >1B80 thru 1FFE for file I/O requirements. This area was chosen to permit future implementation of bit-map graphics.

VIII. STACK USAGE

c99 makes extensive use of the previously-mentioned push-down stack for temporary storage. Function arguments are pushed onto the stack as they are encountered between parentheses, so the last argument is at the "top" of the stack. This inverse order is somewhat unconventional. After all arguments have been pushed on the stack, the return address is pushed on by the recursive call code accessed via R12. Since the stack grows downwards in memory, the last argument value is located 2 bytes above the stack pointer's current contents at function entry.

As specified in the C language definition, parameter passing is "call by value". If X and Y are global variables, the compiler generates the following code for this C statement:

```
X=function1(X,Y,zf());
```

```
MOV @X,8      value of X to primary register
BL 15         push onto stack
MOV @Y,8      ditto for Y
BL 15
BL *12        recursive call to zf
DATA ZF       function value is returned in primary reg
BL 15         push value onto stack
BL *12        call function1
DATA FUNCTI   note 6 characters used
AI 14,6       restore stack pointer
MOV 8,@X      primary reg to X
```

As function1 is entered, the stack contents are:

```
      .
      .
      .
-------------
value of X
-------------
value of Y
-------------
zf fcn value
-------------
return addr     <=== stack pointer (R14)
-------------
```

In this case, the value of Y could be accessed by "MOV @4(14),8".

Local variables allocate as much stack space as needed and are assigned the current value of the stack pointer (after allocation) as their address. The compiler ensures that each variable is located on a word boundary. The declaration:

```
int z;
```

generates:

```
DECT 14
```

which allocates space on the stack for 2 bytes (not initialized). References to z will now be made to *14 (stack pointer+0). If the next declaration is:

```
char array(5);
```

the code generated is:

```
AI 14,-6
```

Note that the stack pointer changes by 6 bytes, ensuring that any following declarations fall on a word boundary.

Now, array[0] is at *14, array[1] is at @1(14), array[2] is at @2(14), etc. z is now accessed as @6(14). For this reason, imbedded assembly language code using "#asm ... #endasm" cannot access local variables by name, but must know their location relative to the stack pointer's current contents.

IX. PROGRAMMING INFORMATION.

-When a c99 program begins execution, the screen is in text mode (40 characters/line) displaying black characters on a cyan background. Future releases will include functions to change modes and colors. It is possible to access VDP registers and memory from c99 by using the appropriate values with pointers.

-The logical operators "&&" and "||" are not available in c99. However, the bitwise operators "&" and "|" will yield the correct results in logical expressions. c99 follows the usual convention in using a non-zero value to represent a true condition and a zero value to represent a false condition.

-Global variables result in more efficient code than local variables but they cannot be used in recursive situations. In addition, global variables (especially arrays) increase the size of a program module.

-Functions may be passed the names of other functions as arguments for indirect calling. The dummy argument for a function name must appear in the argument list as a simple name and must be declared as an integer pointer.

-Since c99 programs are self-contained, they may be saved as program files. Two object files, C99PFI and C99PFF, have been placed on the release diskette to facilitate program file creation. C99PFI must be loaded before user programs and libraries since it defines the entry points SFIRST and SLOAD and contains code to load the standard utilities from the Ed/Asm GROM. C99PFF must be loaded after all programs and libraries as it defines the entry point SLAST. If the Ed/Asm program SAVE is then loaded and run, a useable program file should result. The c99 compiler and utility program files were created in this manner.

-The release diskette contains a file, CONV;C, which consists of functions for string to int (atoi) and int to string (itod) conversions. Use of these functions is documented in the file. CONV;C may be edited or included in any program requiring the functions. Future releases of c99 will contain a library of conversion functions as well as printf and scanf.

X. REFERENCES.

-The C Programming Language by Brian Kernighan and Dennis Ritchie
Prentice-Hall 1978

-The C Primer by Les Hancock and Morris Kreiger
McGraw-Hill 1982

c99 Compiler Specifications (Revised 85/12/04)

A : Features of c99

c99 currently supports:

1. Data type declarations of:

   "char" (8 bits)
   "int" (16 bits)
   pointers to either of the above by using an "*"
   before the variable name.

2. Arrays:

   One-dimensional arrays may be of type char or int.

3. Expressions:

   unary operators:
      "-" (minus)
      "*" (indirection)
      "&" (address of)
      "~" (ones complement)
      "!" (logical negation)
      "++" (increment, either prefix or postfix)
      "--" (decrement, either prefix or postfix)
   binary operators:
      "+" (addition)
      "-" (subtraction)
      "*" (multiplication)
      "/" (division)
      "%" (modulo, ie. remainder from division)
      "|" (bitwise inclusive or)
      "^" (bitwise exclusive or)
      "&" (bitwise and)
      "==" (test if equal)
      "!=" (test if not equal)
      "<" (test if less than)
      "<=" (test if less or equal)
      ">" (test if greater than)
      ">=" (test if greater or equal)
      "<<" (arithmetic left shift)
      ">>" (arithmetic right shift)
      "=" (assignment)
   primaries:
      array[expression]
      function(arg1,arg2,...,argn)
      constant
        decimal number
        quoted string ("sample string")
        primed string ('a' or 'ab')
      local variable or pointer
      global (static) variable or pointer

4. Program control:

```
if(expression) statement;
if(expression) statement; else statement;
while(expression) statement;
break;
continue;
return;
return expression;
; (null statement)
{ statement; statement; ... } (compound statement)
```

5. Pointers:

Local and static pointers can contain the address of char or int data elements.

6. Compiler commands:

```
#define name string
   (pre-processor will replace name with string)
#include "filename"
   (take input from filename until end-of-file.
   cannot be nested.)
#asm        (not in standard C)
   (allows all code between "#asm" and "#endasm" to
   be passed unchanged to the assembler. this
   command is actually a statement and may be used
   as: "if (expression) #asm ... #endasm else ...")
#outseg     (not in standard C)
   (closes and reopens the output file so that very
   large programs may be compiled with segmented
   output files which may be read with the editor)
```

7. Miscellaneous:

Expression evaluation maintains the same heirarchy as in standard C.

Pointer arithmetic recognizes the data type of the destination. ptr++ will increment by 2 if ptr was declared "int *ptr".

Pointer compares are unsigned since addresses are not signed numbers.

Operations whch require more than two words of instructions generate calls to routines in the CSUPP libary to minimize the size of the program. all such support routines have names beginning with C$.

The generated code is re-entrant as required by C. each time a function is referenced, the local variables refer to a fresh area of the stack.

c99 Compiler Error Codes (Rev. 85/09/11)

1 Missing }
2 Bad filename for include
3 Array size must be a constant
4 Illegal array size specification
5 Illegal name for function or declaration
6 Missing (in function declaration
7 Illegal argument name
8 Comma was expected
9 Incorrect number of arguments
10 argument name was expected
11 missing semicolon
12 Illegal symbol name
13 Symbol already defined
14 Missing bracket or token
15 Must be lvalue
16 Can't subscript
17 Global symbol table overflow
18 Local symbol table overflow
19 Too many active loops
20 No active whiles
21 Missing quote
22 Missing apostrophe
23 Line too long
24 Macro table full
25 Output file error
26 Bad sddress
27 Invalid expression
28 String space full

```
flg=spcall();
```
Returns true if any sprites are in coincidence.

Notes :

For sprite automotion to occur, interrupts must be enabled at least 60 times per second. The functions poll, key, and joyst enable interrupts when called. If they are not called often enough, a function such as the following may be used :

```
intон();
{
#asm
 LIMI 2
 LIMI 0
#endasm
}
```

inton should be invoked at a suitable place in the program to ensure smooth sprite motion.

It is neccessary to include REFs for each function referenced. An include file (GRF1RF) containing REFs for all functions in this library is on this diskette. It is accessed with:

```
#include "dsk1.grf1rf"
```

The following functions may also be used in text mode :

chrdef hchar vchar gchar joyst key

```
/* grf1 references
*/
#asm
 REF GRF1,TEXT,SCREEN,COLOR,CHRDEF,CLEAR,HCHAR,VCHAR
 REF GCHAR,JOYST,KEY,SPRITE,SPDEL,SPDALL,SPCOLR,SPPAT
 REF SPLOCT,SPMAG,SPMOTN,SPMCT,SPPOSN,SPDIST,SPDRC
 REF SPCNC,SPCRC,SPCALL
#endasm
```

B : Limitations of c99

c99 does not support:

1. Structures and unions.
2. Multi-dimensional arrays.
3. Data types other than "char" and "int".
4. Function calls returning other than "int" values.
5. The unary "sizeof" and casts.
6. The operators: &&, !!, ?: and , .
7. The assignment operators: +=, -=,*=, /=, %=, >>=, <<=, &=, ^=, !=.
8. Storage classes: auto, static, extern, register and typedef.
9. The statements: for, do while, switch, case, default and goto.
10. The use of arguments within a "#define" command.
11. Conditional compilation (#ifdef, etc).
12. Initializers on declarations.
13. Pointers to anything but char or int.
14. The use of _ (underscore) in names.

Other limitations of c99:

Since c99 is a single-pass compiler, undefined names are not detected and are assumed to be function or variable names not yet defined. If this assumption is incorrect, an undefined reference error will occur when the compiled program is assembled.

Because a single-pass compiler scans the source code only once, very little object code optimization is possible. For example, the statement x=1+2; results in code to add 1 and 2 at runtime.

Names may be of any length but the compiler only recognizes the first six characters.

REF directives are generated for functions in the CSUPP library only. REFs to functions in other libraries must be provided explicitly using #asm and #endasm. The include file STDIO contains REFs for all functions in the CFIO library.

The file I/O library (CFIO) is currently limited to processing Display/Variable 80 type files. Up to four files may be open at one time but only three may be disk files.

```
/*
** CONIO : C99 console I/O definitions
*/
#define stdin  (-1)
#define stdout (-2)
#define stderr (-3)
#define EOF    (-1)
#define YES     1
#define NO      0
#define NULL    0
#define EOL    10
#define FF     12
#define BS      8


/* simple conversion functions */
/*
** n=atoi(s) - convert string to integer
*/
atoi(s)  char *s;
{ int sign,n;
  while(*s==' ')++s;
  sign=1;
  if(*s=='-') { sign=-1; ++s; }
  if(*s=='+') ++s;
  n=0;
  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
  return(sign*n);
}
/*
** itod(nbr,str,sz) -
**         convert nbr to signed decimal string of width sz
**         right justified, blank filled, result in str[].
**         sz includes 0-byte string terminator
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
}
```

```
/* c99 error finder
**
** this program scans a c99 compiler output file,
** pausing on errors. during execution, pressing a
** key will cause the display to halt. pressing CLEAR
** (FCTN-4) will terminate execution.
*/
#include "DSK1.STDIO"
#define LEN 81
char fn[40],line[LEN];
int c,inp;
main()
{ while(1)
  { putchar(FF);
    puts("c99 error finder\n\n");
    inp=getfn("Input","r");
    putchar('\n');
    while(fgets(line,LEN,inp))
    { puts(line);
      putchar(EOL);
      if(line[1]=='!')errpause();
      poll(YES);
    }
    fclose(inp);
    puts("\nmore files?");
    c=getchar();
    if((c=='N')|(c=='n'))break;
  }
}
/* getfn returns unit # or 0 if null name entered
*/
getfn(text,m) char *text,*m;
{ int unit;
  unit=0;
  while(1)
  { puts(text);
    puts(" filename?");
    gets(fn);
    if(!*fn)break;
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
}
/* errpause waits for a key
*/
errpause()
{ puts("\nPress a key to continue:");
  getchar();
  putchar(EOL);
}
```

```
/* c99 file copy utility
**
** this utility program copies a display/variable 80
** file. Any legal filename may be used.
** responding to a filename prompt by pressing ENTER
** will result in that file being mapped to the
** console. keyboard input is terminated with CTRL-Z.
** during execution, pressing a key will pause the
** program, CLEAR (FCTN-4) will abort.
*/
#include "DSK1.STDIO"
#define LEN 81
char fn[40],line[LEN];
int c,inp,out;
main()
{ while(1)
  { putchar(FF);
    puts("c99 file copy utility\n\n");
    inp=getfn("Input","r");
    out=getfn("Output","w");
    putchar('\n');
    while(fgets(line,LEN,inp))
    { fputs(line,out);
      if(!out)putchar(EOL);
      poll(YES);
    }
    fclose(inp);
    fclose(out);
    puts("\nmore copies?");
    c=getchar();
    if((c=='N')!(c=='n'))break;
  }
}
/* getfn returns unit # or 0 if null name entered
*/
getfn(text,m) char *text,*m;
{ int unit;
  unit=0;
  while(1)
  { puts(text);
    puts(" filename?");
    gets(fn);
    if(!*fn)break;
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
}
```

```
#asm
 REF FOPEN,FCLOSE,GETC,PUTC,FGETS,FPUTS,REWIND
#endasm
/*
** STDIO : standard c99 I/O definitions
*/
#define stdin  (-1)
#define stdout (-2)
#define stderr (-3)
#define EOF    (-1)
#define ERR    (-2)
#define YES      1
#define NO       0
#define NULL     0
#define EOL     10
#define FF      12
#define BS       8



/* Malcolm's test program 1 */
#include "dsk1.conio"
int row,col;
main()
{ while(1)
  { row=0;
    putchar(FF);
    while(++row<25)
    { col=6;
      while(++col<35)
      { locate(row,col);
        putchar(42);
      }
    }
    if(getchar()<1) break;
  }
}
/*
** c99 random number functions
**
** by Clint Pulley, based on TI-FORTH
** last edit 85/11/16 0810
*/
/*
** initialize the random seed
*/
randomize()
{
#asm
 MOVB @>8802,0  RESET VDPSTA
 CLR  1
R#1
 INC  1         COUNT
 MOVB @>8802,0  READ VDPSTA
 ANDI 0,>8000
 JEQ  R#1       IF NOT VDP INT
 MOV  1,@>83C0  STORE SEED
#endasm
}
/* generate a 16-bit random number
*/
rndnum()
{
#asm
 MOV  @>83C0,0  GET SEED
```

```
 MPY  @R#C1,0
 A    @R#C2,1
 SRC  1,5
 MOV  1,8        RETURN NUMBER
 MOV  1,@>83C0   NEXT SEED
#endasm
}
#asm
R#C1  DATA >6FE5
R#C2  DATA >7AB9
#endasm
/* generate a random number between
** 0 and n-1
*/
rnd(n) int n;
{
#asm
 BL   *12       CALL rndnum
 DATA RNDNUM
 ABS  8
 CLR  7
 MOV  @2(14),9 GET n
 DIV  9,7       REMAINDER IN R8
#endasm
}



/* random function test
*/
int i,avg,rn;
char cb[6];
main()
{ randomize();
  i=avg=0;
  while(++i<160)
  { rn=rnd(100);
    avg=avg+rn;
    puts(itod(rn,cb,6));
  }
  puts("\nAverage =");
  puts(itod(avg/160,cb,6));
}
#include dsk1.random;c
/*
** itod(nbr,str,sz) -
**        convert nbr to signed decimal string of width sz
**        right justified, blank filled, result in str[].
**        sz includes 0-byte string terminator
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
  return str;
}
```

```
/* c99 code optimizer v1.2 by Clint Pulley
**
** Last Edit 85/10/21 2030
**
*/
£include dsk1.stdio
£define MAXLIN 81
£define LINES  10
char lbuf[810];
char *line1,*line2,*line3,*line4,*line5;
char *pat1,cn[6];
int inp,otf,ni,nb;
int out,in,rptr,eoflg;
/*
*/
main()
{ puts("c99 optimizer v1.2\n\n");
  pat1=" MOV *8,8";
  ni=nb=in=eoflg=0;
  out=-LINES;
  while(1)
  { inp=getfn("Input","r");
    if(inp)break;
    puts("Illegal filename\n");
  }
  roll(LINES);
  otf=getfn("Output","w");
  fputs("*c99opt v1.2",otf);
/* main loop */
  while(read(&line1))
  { poll(1);
    if(match(line1," MOV 14,8"))
    { if(type1()) continue;
    }
    if(match(line1," BL 15"))
    { if(type2()) continue;
    }
    if(match(line1," MOV *14+,9"))
    { if(type3()) continue;
    }
    fputs(line1,otf);
    roll(1);
  }
  fclose(inp);
  fclose(otf);
  puts("\nInstructions optimized =");
  puts(itod(ni,cn,6));
  puts("\nBytes of memory saved  =");
  puts(itod(nb,cn,6));
}
/* get filename and open file
*/
getfn(text,m) char *text,*m;
{ int unit;
  char fn[20];
  unit=0;
  while(1)
  { puts(text);
    puts(" filename? ");
    gets(fn);
    if(!*fn)break; /* return 0 if null name */
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
```

```
}
type1()
{ read(&line2);
  if(match(line2,pat1))
  { fputs(" MOV *14,8",otf);
    ++ni;
    nb=nb+2;
    roll(2);
    return 1;
  }
  if(match(line2," INCT 8"))
  { return(type1a());
  }
  if(nmatch(line2," AI 8,",6))
  { return(type1b());
  }
  reset();
  return 0;
}
type1a()
{ read(&line3);
  if(match(line3,pat1))
  { fputs(" MOV @2(14),8",otf);
    ++ni;
    nb=nb+2;
    roll(3);
    return 1;
  }
  reset();
  return 0;
}
type1b()
{ read(&line3);
  if(match(line3,pat1))
  { strcpy(line1," MOV @");
    strcat(line1,line2+6);
    strcat(line1,"(14),8");
    fputs(line1,otf);
    ++ni;
    nb=nb+4;
    roll(3);
    return 1;
  }
  reset();
  return 0;
}
type2()
{ read(&line2);
  if(nmatch(line2," LI 8,",6) | match(line2," CLR 8"))
  { read(&line3);
    if(match(line3," MOV *14+,9"))
    { fputs(" MOV 8,9",otf);
      fputs(line2,otf);
      ++ni;
      nb=nb+2;
      roll(3);
      return 1;
    }
  }
  reset();
  return 0;
```

```
}
type3()
{ read(&line2);
  if(match(line2," A 9,8"))
  { fputs(" A *14+,8",otf);
    ++ni;
    nb=nb+2;
    roll(2);
    return 1;
  }
  reset();
  return 0;
}
/* match entire string
*/
match(s,t) char *s,*t;
{ while(*s==*t)
  { if(!*s)return(1);
    ++s; ++t;
  }
  return(0);
}
/* match partial strings
*/
nmatch(s,t,n) char *s,*t; int n;
{ while(n--)
  { if(*s++==*t++)continue;
    return(0);
  }
  return(1);
}
/* concatenate t to end of s
*/
strcat(s,t) char *s,*t;
{ --s;
  while(*++s);
  while(*s++ = *t++);
}
/* copy t to s
*/
strcpy(s,t) char *s,*t;
{ while(*s++=*t++);
}
/*
** itod(nbr,str,sz) -
**   convert nbr to signed decimal string of width sz
**   right justified, blank filled, result in str[].
**   sz includes 0-byte string terminator.
**   returns str.
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
  return str;
```

```
}
/*
** circular lookahead buffer handling
*/
/* roll n lines in circular buffer
*/
roll(n) int n;
{ char *cptr;
  rptr=out=(out+n)%LINES;
  if(eoflg)return;
  while(n--)
  { cptr=bufloc(in);
    if(!fgets(cptr,MAXLIN,inp))
    { *cptr=127;
      return;
    }
    in=++in%LINES;
  }
}
/* read a line by setting ptr to the buffer loc
** return 0 if end-of-file
*/
read(ptr) int *ptr;
{ char *cptr;
  *ptr=cptr=bufloc(rptr);
  rptr=++rptr%LINES;
  if(*cptr==127)return 0;
  return 1;
}
/* reset rptr to starting line
*/
reset()
{ rptr=out;
}
/* return buffer location for line n
*/
bufloc(n) int n;
{  return(lbuf+n*MAXLIN);
}
```

GRF1 : c99 graphics mode 1 function library

Interim Documentation (85/12/05)

This library provides functions analogous to those in Extended Basic for character graphics and sprite control. Except as noted, all functions perform the same operation as the equivalent Ex Basic call although names may differ.

All arguments are of type int except for the character definition string in chrdef. Arguments preceded by & are used to return values. Omission of & (address of) will result in highly unpredictable results!

Numbering conventions:

Character positions : row 1-24, column 1-32

Colors : 1-16

Characters : 0-255 (full ascii set available)

Character sets : 0-31 (add 4 to ExBasic set number)

Sprites : 0-31 (32 sprites available)

Sprite characters : 0-255 (co-incide with normal characters)

Sprite positions : row 0-255 (invisible above 191)
col 0-255

Functions available in graphics 1 mode :

grf1();
Set to graphics 1 mode (24x32 characters). Char char set colors set to black/transparent, backdrop set to cyan.

text();
Set to text mode (24x40 characters, no sprites). Screen set to black/cyan.

screen(c);
Set screen (backdrop) color to c.

color(cs,f,b);
Change colors for char set cs to f/b.

chrdef(ch,str);
Define character pattern(s) starting at character ch. Up to 4 characters may be defined with one call. str is either a quoted string or a pointer to a string consisting of 1-64 hex (0-9, A-F) digits. Incomplete patterns are zero-filled.

clear();
Clear the screen.

hchar(r,c,ch,n);
Place character ch at row r, col c and repeat n times horizontally.

```
vchar(r,c,ch,n);
```
Place character ch at row r, col c and repeat n times vertically.

```
c=gchar(r,c);
```
Return the value (int) of the character at row r, col c.

```
s=joyst(u,&x,&y);
```
Read joystick u (1 or 2) and return the x and y values (0 , +4, or -4). s is true if x or y != 0.

```
c=key(u,&s);
```
Read keyboard u (0-5) and return char value c and status s (-1, 0, or 1).

```
sprite(spn,ch,col,dr,dc);
```
Define sprite number spn with char ch, color col, located at dr,dc.

```
spdel(spn);
```
Delete sprite spn.

```
spdall();
```
Delete all sprites and clear automotion table.

```
spcolr(spn,col);
```
Set sprite spn to color col.

```
sppat(spn,ch);
```
Set pattern for sprite spn to char ch.

```
sploct(spn,dr,dc);
```
Socate sprite spn at row dr, col dc.

```
spmag(f);
```
Set sprite magnification to f (1-4).

```
spmotn(spn,rv,cv);
```
Set row velocity rv and col velocity cv for sprite spn.

```
spmct(n);
```
Enable automotion for the first n sprites (numbers 0 to n-1).

```
spposn(spn,&rp,&cp);
```
Return row position rp and col position cp for sprite spn.

```
dsq=spdist(spn1,spn2);
```
Return the square of the distance between sprites spn1 and spn2.

```
dsq=spdrc(spn,dr,dc);
```
Return the square of the distance between sprite spn and location dr,dc.

```
flg=spcnc(spn1,spn2,tol);
```
Returns true if the distance between sprites spn1 and spn2 is <= tol.

```
flg=spcrc(spn,dr,dc);
```
Returns true if the distance between sprite spn and location dr,dc is <= tol.

```
.co RUNOFF text formatting program - documentation
.he          RUNOFF documentation  85/10/07
.fo          Page #
.in 10
.rm 70
```

This is a translation of the text formatting program from Kernighan and Plauger's book "Software Tools", modified to take advantage of the features of C. If the following command summary is not sufficient, please refer to the book.

```
.sp 2
.nf
.in +2
```

command	break?	default	result
.bp n	yes	+1	begin page numbered n
.br	yes		cause a paragraph break
.ce n	yes	1	center the next n lines
.co	no		includes a comment in input
.fi	yes		begins fill mode
.fo s	no	empty	sets footer text
.he s	no	empty	sets header (page top) text
.in n	no	0	indent lines by n spaces
.ls n	no	1	sets line spacing
.nf	yes		stops filling
.pl n	no	66	sets lines per page
.rm n	no	60	sets right margin
.sp n	yes	1	spaces down n lines
.ti n	yes	0	temporary indent of n spaces

```
.in -2
.sp 2
```

Run Instructions:

After RUNOFF has been loaded in the usual manner, it will identify itself and prompt for an output filename. This will normally be the same printer filename (PIO, RS232.BA=4800, etc.) that is used for listing files with the Editor. Next, the program will request keyboard input. This allows titles, margins, etc. to be input. Press CTRL-Z to end keyboard input. The program will then prompt for an input filename and proceed to print it. When the input file has been printed, the program will prompt for another filename. When all files have been printed, it is essential to respond to the prompt by pressing enter. This causes the fill buffer to be flushed and the last footer to be printed before program exit.

```
.sp 2
```

Notes:

```
.in +3
.fi
```

1) In the above table, n designates a numeric constant. If it unsigned, it is an absolute value. If the number is preceded by + or - it is a relative value and is added to or subtracted from the present value of that parameter.

2) In the table, s represents a string of characters used for a title. If the string includes a # , the current page number is substituted for the #.

3) Filling consists placing as many words as possible on each line and right-justifying them. Paragraph breaks are generated by .br, an empty line, or a line beginning with space(s). Paragraph indentation is preserved.

4) The book "Software Tools" by B.W. Kernighan and P.J. Plauger (Addison-Wesley 1976) is a standard reference which is well worth reading.

```
/*
** text formatter transcribed from the book Software Tools
**
** Converted to c99 by Clint Pulley
**
** Last edit 89/10/07 2130
**
*/
£include dsk1.stdio
£define PAGEWIDTH 60 /* default page width */
£define PAGELEN   66 /* default page length */
£define MAXLINE   82
£define PAGECHAR '£' /* page number escape char */
£define HUGE   32700
/*
** global storage
*/
int fill,  /* in fill mode if YES */
    lsval, /* current line spacing */
    inval, /* current indent; >= 0 */
    rmval, /* current right margin */
    tival, /* current temporary indent */
    ceval, /* number of lines to center */
    curpag,/* current output page number */
    newpag,/* next output page number */
    lineno,/* next line to be printed */
    plval, /* page length in lines */
    m1val, /* top margin, including header */
    m2val, /* margin after header */
    m3val, /* margin after last text line */
    m4val, /* bottom margin, including footer */
    bottom,/* last live line on page: */
           /*  = plval - m3val - m4val */
    outp,  /* index into outbuf */
    outw,  /* width of text in outbuf */
    outwds,/* number of words in outbuf */
    dir,   /* directions flag */
    inp,   /* input unit */
    out;   /* output unit */
char header[MAXLINE], /* top of page title */
     footer[MAXLINE], /* bottom of page title */
     outbuf[MAXLINE], /* lines to be filled go here */
     inbuf[MAXLINE];  /* input line buffer */
/*
*/
main()
{ inval=tival=ceval=curpag=lineno=0;
  outp=outw=outwds=dir=0;
  lsval=newpag=1;
  m1val=m3val=2;
  m2val=m4val=3;
  fill=YES;
  rmval=PAGEWIDTH;
  plval=PAGELEN;
  init();
  puts("RUNOFF text formatter v1.1\n");
  out=getfn("Output","w");
  puts("\n Keyboard input? (y/n) ");
  gets(inbuf);
  putchar('\n');
  if((*inbuf&95)=='Y')
  { puts("Enter data, terminate with CTRL-Z\n\n");
```

```
 roff(stdin);
  }
  while(inp=getfn("Input","r"))
  {
    roff(inp);
    fclose(inp);
  }
  if (lineno > 0)
    space(HUGE);
  fclose(out);
}
/* initialize header, footer, and line-count invariant
*/
init()
{
  bottom = plval - m3val - m4val; /* invariant */
  *header=NULL;
  *footer=NULL;
}
/* get filename and open file
*/
getfn(text,m) char *text,*m;
{ int unit;
  char fn[20];
  unit=0;
  while(1)
  { puts(text);
    puts(" filename? ");
    gets(fn);
    if(!*fn)break; /* return 0 if null name */
    if(unit=fopen(fn,m))break;
    puts("bad filename-try again\n");
  }
  return(unit);
}
/* format current file
*/
roff(f) int f;
{ while (fgets(inbuf,MAXLINE-1,f) != NULL)
  { strip(inbuf);
    if (*inbuf == '.')
      command();
    else
      text();
  }
}
/* strip out TI-Writer control characters
*/
strip(b) char *b;
{ if(*b==FF)
  { strcpy(b,".bp");
    return;
  }
  if(*b>127) /* TIW tab data */
  { strcpy(b,".co");
    return;
  }
```

```
  --b;
  while(*++b);
  if(*--b==13)*b=NULL;
}
/* perform formatting command
*/
command()
{ int val, spval,argtyp;
  val = getval(&argtyp);
  if (lookup("bp"))
  {
    if (lineno > 0)
      space(HUGE);
    set(&curpag,val,argtyp,curpag+1,-HUGE,HUGE);
    newpag = curpag;
  }
  else if (lookup("br"))
    brk();
  else if (lookup("ce"))
  {
    brk();
    set(&ceval,val,argtyp,1,0,HUGE);
  }
  else if (lookup("co"));
  else if (lookup("fi"))
  {
    brk();
    fill = YES;
  }
  else if (lookup("fo"))
    strcpy(footer,inbuf+3);
  else if (lookup("he"))
    strcpy(header,inbuf+3);
  else if (lookup("in"))
  {
    set(&inval,val,argtyp,0,0,rmval-1);
    tival = inval;
  }
  else if (lookup("ls"))
    set(&lsval,val,argtyp,1,1,HUGE);
  else if (lookup("nf"))
  {
    brk();
    fill = NO;
  }
  else if (lookup("pl"))
  {
    set(&plval,val,argtyp,PAGELEN,m1val+m2val+m3val+m4val+1,HUGE);
    bottom=plval-m3val-m4val;
  }
  else if (lookup("rm"))
    set(&rmval,val,argtyp,PAGEWIDTH,tival+1,HUGE);
  else if (lookup("sp"))
  {
```

```
    set(&spval,val,argtyp,1,0,HUGE);
    space(spval);
  }
  else if (lookup("ti"))
  {
    brk();
    set(&tival,val,argtyp,0,0,rmval);
  }
  else
    return; /* ignore unknown commands */
}
/* lookup routine for commands
*/
lookup(string) char *string;
{ return (inbuf[1] == string[0]) & (inbuf[2] == string[1]);
}
/* evaluate optional numeric argument
*/
getval(argtyp) int *argtyp;
{ int i;
  i = 3;
  /* ..find argument.. */
  while (inbuf[i] == ' ')
    ++i;
  *argtyp = inbuf[i];
  if (*argtyp == '+' | *argtyp=='-')
    i++;
  return(atoi(inbuf+i));
}
/* set parameter and check range
*/
set(param,val,argtyp,defval,minval,maxval)
int *param,val,argtyp,defval,minval,maxval;
{ int t;
  t=*param;
  if(argtyp == NULL)
    *param = defval;
  else if (argtyp == '+')
    *param = t+val;
  else if (argtyp == '-')
    *param = t-val;
  else
    *param = val;
  *param = min(*param,maxval);
  *param = max(*param,minval);
}
/* process text lines
*/
text()
{ char wrdbuf[MAXLINE];
  int i;
  if (*inbuf==' ')
    leadbl(inbuf);  /* go left. set tival */
  if (ceval > 0)
  { center(inbuf);
    put(inbuf);
    ceval--;
  }
```

```
    else if (!*inbuf)
    { brk();
      put(" ");
    }
    else if (fill == NO)
      put(inbuf);
    else
    { i = 0;
      while (getwrd(inbuf,&i,wrdbuf) > 0)
        putwrd(wrdbuf);
    }
}
/* delete leading blanks.  Set tival
*/
leadbl(buf) char *buf;
{ int i,j;
  brk();
  /* find first non blank */
  i = 0;
  while (buf[i] == ' ')
    i++;
  if (buf[i] != NULL)
    tival = tival+i;
  else ++tival;
  /* move line to left */
  j = 0;
  while ((buf[j++] = buf[i++]) != NULL) ;
}
/* put out line with proper spacing and indenting
*/
put(buf) char *buf;
{ int i;
  if ((lineno == 0) | (lineno > bottom))
    phead();
  i = 1;
  while (i++ <= tival)
    putc(' ',out);
  tival = inval;  /* tival good for one line only */
  while(*buf)
  { putc(*buf++,out);
  }
  putc('\n',out);
  skip(min(lsval-1,bottom-lineno));
  lineno = lineno+lsval;
  if (lineno > bottom)
    pfoot();
}
/* put out page header
*/
phead()
{ curpag = newpag++;
  if (m1val > 0)
  {
    skip(m1val-1);
    puttl(header,curpag);
  }
```

```
  skip(m2val);
  lineno = m1val+m2val+1;
}
/* put out page footer
*/
pfoot()
{ skip(m3val);
  if (m4val > 0)
  {
    puttl(footer,curpag);
    skip(m4val-1);
  }
}
/* put out title line with optional page number
*/
puttl(buf,pageno) char *buf; int pageno;
{ while(*buf)
  { if (*buf == PAGECHAR)
      numout(pageno);
    else
      putc(*buf,out);
    ++buf;
  }
  putc('\n',out);
}
/* print page number without ldg spaces
*/
numout(num) int num;
{ char nbuf[6],*ptr;
  itod(num,nbuf,6);
  ptr=nbuf-1;
  while(*++ptr==' ');
  while(*ptr)putc(*ptr++,out);
}
/* space n lines or to bottom of page
*/
space(n) int n;
{ brk();
  if (lineno > bottom)
    return;
  if (lineno == 0)
    phead();
  skip(min(n,bottom+1-lineno));
  lineno = lineno+n;
  if (lineno > bottom)
    pfoot();
}
/* output n blank lines
*/
skip(n) int n;
{ while ((n--) > 0)
  { putc(' ',out);
    putc('\n',out);
  }
}
```

```
/* get non-blank word from in[i] into out.
** increment *i.
*/
getwrd(in,ii,out) char *in,*out; int *ii;
{ int i, j;
  i = *ii;
  while (in[i]  == ' ') i++;
  j = 0;
  while((in[i]) & (in[i] != ' '))
    out[j++] = in[i++];
  out[j] = NULL;
  *ii = i;     /* return index in ii */
  return(j);  /* return length of word */
}
/* put a word in outbuf; includes margin justification
*/
putwrd(wrdbuf) char *wrdbuf;
{ int last, llval, w, nextra;
  w = width(wrdbuf);
  /* new end of wrdbuf */
  last = strlen(wrdbuf)+outp+1;
  llval = rmval-tival;
  if ( (outp > 0) & ((outw+w>llval) | (last>=MAXLINE)) )
  {
    /* too big */
    last = last-outp; /* remember end of wrdbuf */
    nextra = llval-outw+1;
    spread(outbuf,outp-1,nextra,outwds);
    if (nextra > 0 & outwds > 1)
      outp = outp+nextra;
    brk();  /* flush previous line */
  }
  strcpy(outbuf+outp,wrdbuf);
  outp = last;
  outbuf[outp-1] = ' '; /* blank between words */
  outw = outw+w+1;      /* 1 extra for blank */
  outwds++;
}
/* spread words to justify right margin
*/
spread(buf,outp,ne,outwds) char *buf; int outp,ne,outwds;
{ int nholes, i, j, nb;
  if (ne <= 0 | outwds <= 1)
    return;
  dir = 1-dir;  /* reverse direction */
  nholes = outwds-1;
  i = outp-1;
  /* leave room for EOS */
  j = min(MAXLINE-2,i+ne);
  while (i < j)
  {
    buf[j] = buf[i];
    if (buf[i] == ' ')
    {
      /* compute # of blanks */
      if (dir == 0)
        nb = (ne-1)/nholes+1;
      else
        nb = ne/nholes;
      ne = ne-nb;
      nholes--;
      /* put blanks in buffer */
      while (nb-- > 0)
```

```
            buf[--j] = ' ';
        }
        i--;
        j--;
    }
}
/* compute width of character string
*/
width(buf) char *buf;
{ int val;
  val = 0;
  while(*buf++)
    val++;
  return(val);
}
/* end current filled line
*/
brk()
{ if (outp > 0)
  {
    outbuf[outp-1] = NULL;
    put(outbuf);
  }
  outp = outw = outwds = 0;
}
/* center a line by setting tival
*/
center(buf) char *buf;
{ tival = max((rmval+tival-width(buf))/2,0);
}
max(a,b) int a,b;
{ if(b<a)return(a); else return(b);
}
min(a,b) int a,b;
{ if(b>a)return(a); else return(b);
}
/* concatenate t to end of s
*/
strcat(s,t) char *s,*t;
{ --s;
  while(*++s);
  while(*s++ = *t++);
}
/* copy t to s
*/
strcpy(s,t) char *s,*t;
{ while(*s++=*t++);
}
/* return length of s
*/
strlen(s) int *s;
{ char *t;
  t=s-1;
  while(*++t);
  return(t-s);
}
```

```
/*
** itod(nbr,str,sz) -
**        convert nbr to signed decimal string of width sz
**        right justified, blank filled, result in str[].
**        sz includes 0-byte string terminator
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
}
/*
** n=atoi(s) - convert string to integer
*/
atoi(s)  char *s;
{ int sign,n;
  while(*s==' ')++s;
  sign=1;
  if(*s=='-') { sign=-1; ++s; }
  if(*s=='+') ++s;
  n=0;
  while((*s>='0')&(*s<='9')) n=10 * n + *(s++) - '0';
  return(sign*n);
}
```

```
/*
** itod(nbr,str,sz) -
**      convert nbr to signed decimal string of width sz
**      right justified, blank filled, result in str[].
**      sz includes 0-byte string terminator
*/
itod(nbr,str,sz) int nbr,sz; char str[];
{ char sgn;
  sgn=' ';
  if(nbr<0) { nbr=-nbr; sgn='-'; }
  str[--sz]=0;
  while(sz)
  { str[--sz]=nbr%10+'0';
    if(!(nbr=nbr/10))break;
  }
  if(sz) str[--sz]=sgn;
  while(sz) str[--sz]=' ';
}
/*
** nvatoi(s) - convert string to integer
*/
atoi(s) char *s;
{ int sign,n;
  while(*s==' ')++s;
  sign=1;
  if(*s=='-') { sign=-1; ++s; }
  if(*s=='+') ++s;
  n=0;
  while(*s>='0'&&*s<='9') n=10 * n + *(s++) - '0';
  return(sign*n);
}
```

C99REL1 - Release Notes 85/12/04

by Clint Pulley

This is the first release of c99 but I sincerely hope it's not the last. C99REL1 includes version 1.3 of the c99 compiler, console and file I/O libraries, sample and utility programs, and documentation.

The compiler is based on Ron Cain's original small-c as revised by Jeff Lomicka. Considerable insight and some improvements resulted from looking at Jim Hendrix's small-c version 2.

The initial development of the c99 compiler was done on another computer using standard C. I started with Ron's Z-80 version of small-c and, three months and 32 revisions later, had a c99 output file to download to my TI. The release version of the compiler has been revised and rebuilt several times on the TI. The I/O libraries, written in assembly code, were developed entirely on the TI.

I do not expect that the compiler is bug-free (what 2,000+ line program ever is!!). However, it works well enough to recompile itself reliably. This, together with much testing and poring over generated code, leads me to believe that c99REL1 is a useable piece of software.

If you are an experienced assembly language programmer you will find that the generated code is very strange looking. c99 is a rather primitive, single pass compiler, and it produces somewhat inefficient code. However, it runs the Byte Magazine prime number sieve benchmark in 68 seconds, compared to 175 seconds in TI-Forth and almost 4,000 seconds in console Basic. I have replaced the call to the stack PUSH subroutine with inline instructions and reduced this to 62 seconds (a new compile option in rel2?). With all its warts, c99 has enough of the power of C to enable significant programs (such as compilers) to be written and can run 2.5 times faster than Forth.

Enough rambling! Here is a summary of the contents of the C99REL1 diskette:

C99C/D/E	The c99 compiler pgm files
C99ERRORS	Text of error messages
C99MAN1/2/3	The c99 user's manual
C99PFF/PFI	Obj files for pgm file generation (cf manual)
C99SPECS	c99 specifications
CFIO	Obj file I/O library
CONIO	Include file (cf manual)
CONV;C	c source for string </> int conversions
CSUP	Obj support and console I/O library
ERRFIND	Pgm file of error finder
ERRFIND;C	c source for above
FCOPY;C	c source for file copy pgm
SD	Pgm file of my disk directory pgm (not in c)
SIEVE;C	c source of benchmark
STDIO	Include file (cf manual)
TEST1;B	Basic/ExB test pgm
TEST1;C	c source of above
TEST1;O	Obj file for above

The TEST1 program is a simple demonstration of c99 vs Basic. Run TEST1;B in either console or extended Basic, noting the time taken to fill the screen. Then load TEST1;O and CSUP with Ed/Asm and run program START. The program logic is the same in both versions, but not so the execution time. CTRL-Z exits the c version.

The source files for ERRFIND and FCOPY include brief instructions for use.

During final testing, a problem was encountered with the rewind function. TI's designers regarded attempting to read past an end-of-file as a fatal error, so rewinding a file which is at EOF doesn't work. It is neccessary to close and re-open the file instead. The next release will include a soft EOF test (as in Basic) to handle this case. The rewind function does work when the file is not at EOF.

All documentation files were written and saved with TI-Writer. They do not require the formatter and contain no form feeds so they can be listed with the Editor on any printer.

Much more could be done to and with c99. My plans include:

-Compiler changes to provide "for" and "switch/case", function calls with variable argument count (needed by printf), and better code;

-Library additions including printf, scanf and graphics functions;

-Some classic C applications such as AR and ROFF.

If new libraries and applications are not too large I will upload them to the Source/Texnet. Major changes such as a new compiler will require a new release diskette.

Please let me know what you think of c99.

Clint

PS : This diskette now contains v1.32 of the compiler, which corrects a bug and generates better code for the ++ and -- operators. The restriction on an octal constant following the \ in a string has been removed. The filename in the #include statement may now be enclosed in quotes to prevent the pre-processor from truncating the name to six characters. Also, an object file called printf contains the printf function. It performs as in standard C and supports c, d, o, s, u and x conversions.

If you know of anyone who has received an earlier version of c99, please feel free to provide them with this version.

PPS : The CSUP library now contains a function to change the text mode screen colors. Usage is:

tscrn(f,b); where f and b are fgd/bgd colors using Basic color numbers.

It is neccessary to provide a REF directive if this function is used.

c99 extra diskette - 85/11/16

A number of people have sent me 2 diskettes for C99REL1 so I have put together this extra diskette containing some things which may be of interest. I wish I had enough applications in c99 to fill a disk but developing the compiler has taken most of my time.

GRF1 is a function library for graphics 1 mode. All of the graphics facilities in Extended Basic have been implemented except CHRPAT and CHRSET.

OPT is a c99 program which optimizes a number of the less efficient instruction sequences produced by the compiler. Its input file is the compiler output and its output file is the new assembler input. Typically, an improvement of about 10% is realized.

RANDOM provides a random number generator similar to TI-FORTH's.

RUNOFF is a text formatter in c99. It does a lot of the things that TI-Writer does but it does not require imbedded control characters for paragraphing and will offset text with tables, etc. in no fill mode which the TIW formatter cannot do. I think RUNOFF is a good example of a c99 application. To run it, load the program file RUNOFF1. This version has been optimized.

The files are:

GRF1	The graphics function library
GRF1DOCS	Documentation for GRF1
OPT;C	The source file for the c99 optimizer.
RANDOM;C	The source file for the random number generator
RNDTST;C	A test driver for random
RUNOFF;C	The c99 source program
RUNOFF1/2	The program files
RUNOFFDOC	A short writeup in RUNOFF format

TIW is a mini-memory resident loader for TI-WRITER whch supports all functions, including show directory and recover edit. It does not allow alternate filenames for the utility program load. TIW is the object file and TIWL;S is the assembler source.

SDIR;S is the source file for the SD program which is on C99REL1.

The files BTHU... are the object and documentation for a game which I wrote