

BDS C User's Guide Addenda
v1.43 Edition — March, 1981

Leor Zolman
BD Software (New Address!)
33 Lothrop st.
Brighton, Ma. 02135
(617) 782-0836

Before getting on with the business at hand (where I shamelessly display all the horrible bugs that have plagued previous versions of the compiler), I'd like to take a moment to answer one of the more common questions that have been asked of me by users and potential users of BDS C. Hopefully, this will save some of you the expense of a phone call (which can run pretty high when I get to rambling...)

Q. What is the royalty arrangement for software developed using BDS C?

A. There is NO royalty arrangement AT ALL. Both the BDS C runtime package and function libraries, in either source or object form (or both), may be freely distributed with commercial (or non-commercial) application programs. The reason for this policy is to promote the use of C for anything and everything, without wrapping up potential applications in miles of red tape and ineffective security measures. Software authors: PLEASE include the source listings to your software with your packages! I understand that there are some markets where such generosity is considered suicidal, and I sympathize in many cases, but I also want to see BDS C selling more copies, and providing the source to applications programs will encourage users to obtain the compiler. Hopefully, some of them may even BUY it.

OK, now it's time for the bug reports. Following, in decreasing order of severity, are the bugs found and fixed for v1.43, and some additional notes:

0. Another logical-expression-related bug caused incorrect code to be generated when a subexpression of a binary operation used the && or || operators. For example,

```
if (x > (i==5 && j<7)) printf("Foobar\n");
```

might have caused a crash when executed.

0.5 A bitwise or arithmetic binary operation in which the left argument was a logical expression of any kind and the right argument was a binary expression of higher precedence failed to evaluate correctly. For example,

```
if (!kbit() & a<5) printf("foo\n");
```

didn't work.

1. A missing comma, such as in the statement:

```
sprintf(dest "x = %d\n", x);
```

went undiagnosed and caused wierd code to be generated. (The bug fixed in the last release had only corrected the case of a missing comma AFTER a format string specification, not BEFORE it...)

2. If a comment was begun on a line which contained an "#include" preprocessor directive, and not terminated until a later line, then CCl became confused. 2a. Several users have complained about not being able to put the character sequence '/' into a quoted string. This is a justifiable gripe, but I'm afraid you'll have to say things like "\/" to get the same effect. The reason comment delimiters are not tolerated within quotes
3. Mismatched curly-braces in a source file now draw a more meaningful diagnostic than the previous "Unexpected EOF encountered" message: a pointer is now provided to the line at which the badly-balanced function begins.
4. When an illegal constant was encountered by CCl at any place where a constant is required, an incorrect "Unmatched left parenthesis" diagnostic was displayed with an impossibly large line number. (Actually, the correct line number was obtainable by subtracting the exact size of the text file from the given line number. Guess what I forgot to initialize between passes...)
5. When using the "-w" option with CLINK, a terminating control-Z was NOT put out to the SYM file when the length of the SYM file worked out to be an exact multiple of 128 bytes. This gave CLINK a headache when "-y" was used to read the SYM file back in.
6. There was another bug in the "getc" library function that caused some trouble when the "fgets" function was used to read in lines from a text file that wasn't terminated with control-Z (CPMEOF). This was fixed by changing the line:

```
return ERROR;
```

to:

```
return iobuf->_nleft++;
```

7. Mismatched square brackets in an expression had drawn an "Unexpected EOF encountered" error instead of something more meaningful.
8. The word "main" is NO LONGER A KEYWORD. In previous versions, the fact that "main" was treated as a keyword made its use in any situation other than as the first line of a "main" function impossible. I.e, attempts to call "main" recursively were not accepted by the compiler. There is now no longer anything special about the word "main". In addition, previous versions had substituted an undocumented one byte code (9D hex) for the name "main" in CRL file directories, thereby probably causing a lot of confusion. This bizarre scheme is no longer used, although the linker will still recognize the special 9D code as meaning "main" when encountered in a CRL file (of course, "MAIN" will now also be recognized...)

9. A bug in the "-y" option handler in CLINK caused CLINK to crash when there wasn't enough room in the reference table to hold all the symbols being read in from a SYM file. Sorry about that, chief. Note, by the way, that the POSITION of "-y" on the command line IS VERY SIGNIFICANT. If the "-y" option appears to the right of names of CRL files to search, then the SYM file specified will not be used until AFTER the previous CRL files have already been scanned and loaded from. I.e., the "-y" option should appear BEFORE the names of any CRL files that contain functions that might not need to be loaded (due to their definition in the SYM file). A new feature of CLINK is that whenever a previously defined symbol is encountered in the process of loading the symbols from a SYM file, a message to that effect will be printed, allowing the user an opportunity to rearrange the command line so that the SYM file is read in earlier and some redundancy possibly eliminated.

10. An obscure feature of the "printf", "sprintf" and "fprintf" library functions, as described in the Kernighan & Ritchie book, is that a field-width specification value preceded by a '0' caused 0-fill instead of space-fill. I'd never NOTICED that before, until a user brought it to my attention (and conveniently provided a fix.) Note that this solves a problem often encountered when printing hex values. Now, the following "printf" call:

```
printf("%4x; %04x\n",8,8);
```

will produce the output:

```
8; 0008
```

11. The body of a function definition now MUST be enclosed in curly-braces. Formerly, the following sort of thing was tolerated as a function definition, but no more:

```
putchar(c) bdos(4,c);
```

12. A bug in the CMAC.LIB macro package had NOT allowed lines such as:

```
exrel <1xi h,>,putchar
```

while the following kind of lines were properly handled:

```
exrel call,putchar
```

13. A new low-level character I/O function package, named CIO.C, has been added for greater flexibility in console interaction, especially for game-type applications. Note, however, that code generated using this facility is NON-PORTABLE from one system to another unless the "other" system is also equipped with a C compiler. If you HAVE to, go ahead and use it, but please resist the temptation to give out a copy of the compiler to your friends along with your source code.

14. Quoted strings containing an open-comment delimiter sequence ('/*') had caused CCL to think an actual comment was intended. I.e., the statement

```
printf("this is an open-comment sequence: /* \n");
```

would have drawn a "string too long..." error. Not any more.

15. The handling of string constants by the code generator has been improved. Now, instead of putting the text right where it is used and generating a jump around it, the compiler accumulates up to 50 text strings in a function and places them all at the end of the function. If more than 50 strings appear, then after the 50th it goes back to doing it the old way for the remainder of the function (there's only so much table space worth allocating to hacks like this.)

16. Speaking of hacks, here's one that'll get you either excited or sick: You say you need some "static" variables? Consider the following method of simulating a "static array of characters":

```
char *static;  
...  
static = "0123456789";  
...
```

The result is that the variable "static" may be used just like a static array of ten characters. If declared as an "int" instead of a "char", it could be used as an array of five integer variables (or ten, if you make the quoted string twice as long...). Steve Ward makes use of this technique in his CIO.C library. Kludgy, yes, but it gets the job done and it's even portable...

17. The default CCl symbol table size for modified versions of the compiler (v1.43T) has been upped from 6K to 7K. The "-r" option still lets you explicitly set the table allocation, if you want to.