BDS C User's Guide Addenda
v1.44 Edition -- April, 1981

Leor Zolman
BD Software
33 Lothop St.
Brighton, Massachussetts 02135
(617) 782-0836

Please note my NEW new address and phone number...some earlier versions of the new documentation have said that my new city and zip code were Allston, 02134, which is where I THOUGHT I was. Actually, I'm in Brighton, 02135, and any mail sent me addressed to Allston may have been returned to the sender stamped with something like "No such address known." Sorry about that.

Here are the bug fixes/extensions for version 1.44:

1. (Applies to v1.43a only): the character sequence \\ appearing at the END of a quoted string caused the preprocessor in CC1 to screw up and stop stripping comments for the rest of the source file. For example, the statement:

```
printf("This backslash would cause big trouble: \\");
```

would have done it.

2. The "qsort" library function didn't work when the total size of the data array being sorted exceeded 32K bytes. This has been fixed by changed the declarations of certain variables in qsort from "int" to "unsigned".

3. CC1, CC2, and CLINK may now be aborted in the middle of execution by typing a control-C.

4. A new CLINK option has been added (as if there weren't enough of them already...) The "-f" option, when specified immediately before the name of an extra CRL file to be searched, FORCES all functions in that CRL to be loaded into the current linkage—even if they haven't been previously referenced. This provides a simple solution to the backwards-reference problem; a typical case when this would be used comes up when you want to use a special version of a low-level function such as "putchar." If you have a complete program such as:

```
main()
{
        printf("this is a test\n");
}
```

and would like your OWN version of putchar to be loaded from a library called, say, SPECIAL.CRL (which you have previously compiled), then simply saying:

```
clink test special <cr>
```

would NOT work, because the "putchar" function doesn't become "needed" until AFTER the library file DEFF.CRL, which contains "printf", is searched...which doesn't happen until AFTER special is searched! So the "putchar" finally loaded would come from DEFF2.CRL, which is the library file automatically searched after DEFF.CRL. To make this do what you want, all you'd have to do now is:

```
clink test -f special <cr>
```

which would force everything in SPECIAL.CRL to be loaded right away, before the DEFF files are scanned. Then, when "printf" gets loaded from DEFF.CRL, the correct "putchar" function will already have been loaded and the one in DEFF2.CRL will be ignored.

The "rename" library function had a rather serious problem: whenever executed, it would zero out the three bytes of code immediately after the end of the function (i.e., the first jump instruction of the next function in memory would get clobbered.) This problem was fixed by increasing the amount of storage declared in the "ds" at the end of "rename" from 49 bytes to 53 bytes.

The "setfcb" function requires that the buffer allocated to hold the resulting fcb is AT LEAST 36 BYTES LONG! "Setfcb" zeroes out the random-record field bytes of the fcb just in case the CP/M 2.x random-record file I/O mechanism is later used. But whether you use the random stuff or not, the fcb you allocate still has to be 36 bytes long.

This bug applies to v1.43 only: A character constant consisting of the double-quote character enclosed in single quotes ('"'), when encountered by ccl, caused ccl to stop stripping comments while reading in the rest of the source file from disk. This was a bug in the v1.43 code added to allow comment delimiters within quoted strings.

Whenever the type information for a function definition was placed on a line separate from the actual name of the function, then the compiler would "lose" a line of code and all errors found past that point in the source file would be reported with an incorrect line number. For example, the following kind of function definition would've caused this problem:

```
char *
foo()
{
        ...
}
```

A new library function, "execv", has been added to the package (source is in DEFF2.ASM). This function allows chaining to another COM file with a variable number of command line parameters (note that "execl" requires all of the arguments to be explicitly passed as string pointer parameters to the function, so that one particular call can only have the number of arguments that it was written with.) The format of the "execv" function is:

```
execl(prog,argvp)
char *prog, **argvp;
```

where `prog' points to the name of the COM file to be chained to, and `argvp' is an `argv'-like pointer to an array of pointers to text parameters. The final pointer in the list must be followed by a null pointer. As an example, note that the "execl" call

```
execl("stat","badspots","$r/o",0);
```

can be written in terms of "execv" as follows:

```
char *args[3];
...
args[0] = "badspots";
args[1] = "$r/o";
args[2] = NULL;
execv("stat",args);
```

10. Directed I/O and pipes, of sorts, are now available to BDS C programmers. The files DIO.C and DIO.H make up a cute little directed I/O package, allowing for directed input, directed output and pipes (a la Unix) on the command lines to programs compiled with this special I/O package. See the comments in DIO.C for complete details. Note that the presence of this package does NOT contradict certain comments made in the User's Guide about kludging advanced Unix features under CP/M; those comments were directed toward systems in which the I/O redirection/generalization is forced upon the user, along with all the entailing overhead, when the redirection isn't needed or wanted for many applications. The DIO package, being written in C and separately compiled, lets YOU the USER decide when you want it and when you do not. If you don't want it, it takes up zero space; if you do, it takes up a bit of room and yanks in all the buffered I/O, but it DOES give you redirection and pipes!

11. A "standard error" buffered I/O stream number has been added to the list of special devices recognized by the "putc" buffered output function. An iobuf value of 4 causes the character given to be written to the CP/M console output, always, while an iobuf value of 1 causes the character to be written to the standard output (which might be a file if the DIO package is being used.) Note that 4 was used instead of the Unix Standard-error value of 2 because 2 had already been taken (by the CP/M LST: device.)

12. String constants may now contain zero bytes within them. Previous versions have flagged lines such as

```
foo = "Jan\0Feb\0Mar\0Apr\0May\0Jun\0Jul\0Aug\0Sep\0Oct\0Nov\0Dec\0";
```

with the error message:

```
Zero bytes are not allowed within strings; to print nulls, use \200
```

Note that allowing the above kind of string constant makes it easier to initialize a table of homogenously-sized strings; the example with the months could be part of a function that returns a pointer to the name of some month n, where n is a passed

value ranging from 0 to 11 (or from 1 to 12, or whatever...)