RECENT BCPL SYSTEM MODIFICATIONS
19 December, 1974

BCPL

<SWINEHART>BCPL.DM contains a version of BCPL (named BCPL) providing all the goodies described below. There may be some problems with new things, but old things should be totally unaffected.

BUG FIXES

The "-5" bug (error -5 issued from the compiler) was fixed, along with some other petty annoyances, in September.

"frame space overflow" will occur much less frequently now with programs which declare a lot of unused symbols, due to directory condensation during the last pass.

Warning: A + or - may not directly follow a <cr>, except in a table. This prevents 5<cr>-2<cr> from evaluating to a single entry, 3, within a table declaration. It also prevents i = i<cr>-2<cr>. Complaints to Curry.

SPEEDUP -- since 9-19-74

BCPL, running under DOS Rev 4, is about 20% faster, due to buffering of I/O characters. The results are more dramatic if the disk is configured just right. It is not known exactly how to do that.

PARTIAL COMPILATION -- since 11-6-74

This feature is provided to allow declarations to be "precompiled", then applied to several compilations which use them. Its performance is not terribly dramatic (about 20%, in general), because of file opening and copying overhead.

Any source file (which may contain get statements), may be precompiled, using the global /G switch, vis:

```
BCPL/G DECLDRIVER
```

DECLDRIVER is typically just a list of get statements, consolidating declaration files.

Subsequently, one can obtain the effect of compiling the concatenation of DECLDRIVER and, say, TESTG, using the local /G switch:

```
BCPL DECLDRIVER/G TESTG
```

This eliminates the time required to scan the DECLDRIVER files, enter names in the directory, and write lexemes to a temporary file. It does, however, involve reading two files and writing one (with no processing), which currently is somewhat expensive.

Redundant "gets": the compiler remembers all file names used in "get" statements, and ignores any "get" statement for a file which has already been "got". This allows one to leave the "get" statements in a source (like TESTG, above), even when it is compiled with a precompiled file (e.g., DECLDRIVER). There is therefore no conversion requirement when using the /G feature.

Notification: the compiler now adds to the listing file (TTY if none specified), the names of each "get" file as it is opened. The name is surrounded by parentheses if it is redundant (second appearance.) If this is too wordy for anyone, we can put it under a switch, or consolidate it one one line, or something.

CONDITIONAL COMPILATION -- since 12-19-74

This facility allows alternative code sequences to be chosen at compile time, based on the value of "constant expressions" (must be comprised of operations on manifests and numeric constants, evaluatable at compile time). A conditional construct may appear wherever "let" is legal (e.g., not within a statement or declaration, or directly following a "then", "ifso", "ifnot", or "case" (must be bracketed).

Conditional selections are done at a time after "get" files have been "got", so "get" operations will not be affected by conditionals (they will be "got" anyway). You can only turn off "gets" by commenting them out.

The syntax of conditional compilation parallels that for conditional statements, except that "[" and "]" brackets are always required. The German equivalents of the BCPL conditional reserved words are used.

"Wenn" is "if"
"dann" is "then" (ifso)
"sonst" is "otherwise" (ifnot). And
"probieren" is "test"
(subject to consultation with a German dictionary.)

A <sequence> is a legally separated sequence of commands and declarations.

```
wenn <expression> dann [ <sequence> ]
```

Will compile <sequence> only if <expression> is true. The <sequence> may contain declarations which will apply to commands which follow the <wenn> construct, as long as the uses of the declared variables are also conditionally compiled.

```
probieren <expression>
   dann [ <sequence1> ]
   sonst [ <sequence2> ]
```

compiles <sequence1> if <expression> is true, <sequence2> otherwise. The "dann" and "sonst" constructs may appear in either order.

The "newname" statement. This construct allows one to determine if a name has been previously declared. It, like "get", is evaluated before conditional compilation operations are performed.

```
newname <bcpl name>
```

evaluates at compile time to "true", if the <new name> is appearing for the first time. It evaluates to "false" if that name has appeared before (including preceding newname constructs.) A sample use is:

```
wenn newname switch dann [ manifest swtch = 0 ]
```

COMMAND LINE MANIFESTS -- since 12-19-74

This feature allows one to declare manifest symbols in the command line, and optionally to set their initial values.

n/V, n a decimal number

sets to n the value to be used in subsequent /M entries. The initial setting for this value is "true" (-1).

name/M, name a legal BCPL identifier

causes "manifest name = value" to be issued. This value will apply throughout the compilation, excluding any part of the program introduced using the /G (precompilation) option.

If used in conjunction with the "newname" facility, this feature can be used to override standard settings for parameters.

Caution: if it appears in a command line issued directly from the terminal, a command line manifest will be declared in upper case (DOS uppifies everything). You can get a lower case declaration by including the /M switch in a command file -- but it's probably better just to use upper case switches. In no case (sic) will this trigger the "automatic upper case for everything if the first symbol in the program is upper case" feature.

BLDR -- since 8-22-74

<mcdaniel>xldr.dm is a nova style dump file which contains a new version of bldr. This program is named xldr. The files it uses are xldr.y*. It provides an intelligibile message when you attempt to load a file with more than 32k words. (it then kills itself). When you are satisfied it works, delete your old bldr.* files and rename xldr files (all of them).