Published: 9/25/68

Identification

The BCPL command
D. M. Ritchie

Purpose

This section describes the use of the BCPL command. For details on the language itself, see MSPM BZ.6.

Usage

The user types the command

```
bcpl segname -(opt1 ... optn)-
```

which invokes the BCPL compiler to compile a segment named "segname.bcpl". Segname may be an entry name, in which case it is assumed to be in the user's working directory, or an absolute or relative path name. Opt1, ... , optn are options for the compiler; none of them need be given. These will change their format when the regular option mechanism is available.

Except as modified by options, the compiler will leave in the user's working directory the following segments:

segname.source
-- which contains a printable listing of segname.bcpl, complete with line numbers and interspersed error comments.

segname.error
-- which contains a printable listing of the error comments from the compiler.

segname
-- which is the text segment from the compilation.

segname.link
-- which is the linkage segment.

segname.list
-- which is the assembly listing of the compiled code.

The last three of these are actually produced by EPLBSA, which is called automatically by the compiler unless suppressed.

The Options

The following options are available. From time to time various other options may appear and go away, but these will be useful only for debugging the compiler and not will not be listed here.

old
-- the old option causes the compiler to accept the old BCPL syntax, which is almost exactly a superset of CTSS BCPL syntax. This option should be used for all programs brought over from CTSS. See below for details.

listty
-- causes the compiler to produce its ".source" segment online instead of in a special listing segment.

errtty
-- causes the compiler to produce all its comments on source code errors online instead of in the ".error" segment.

pname
-- causes the compiler to produce, as part of the ".source" segment, a cross-reference listing of the occurrences of each identifier in the program.

nobsa
-- prevents the compiler from calling EPLBSA when compilation is done. The text and link segments will not be produced; the primary output will be a segment called segname.eplbsa in the working directory.

savebsa
-- causes the compiler to leave "segname.eplbsa" in the working directory even if EPLBSA is called. This segment may then be assembled at a later time.

Error handling

BCPL can generate two kinds of errors: those arising from syntactic or semantic mistakes in the source program, and those arising from various file system problems. The first kind are recorded in the ".source" listing and in the ".error" file (or online, if the proper options are set), but the second kind of error is communicated directly to the console. This type of error includes: inability to find the input segment or one of the "get" segments (the "get" facility is like "include" in EPL); inability to create one of the several intermediate segments in the process directory; and inability to move one of the output segments into the working directory.

The bcpl command will be changed to use the regular MULTICS error handling mechanism when this becomes available.

Note that EPLBSA can encounter various difficulties which may also be reported on the console. See MSPM BX.7.03 for details.

Method

The BCPL compiler proper is written entirely in BCPL. The bcpl command, since it must call many EPL procedures, is written in EPL. Its task is essentially trivial: the options are interpreted, intermediate segments are created, and the compiler is called. On return, the output segments are moved into the working directory under the proper name and finally EPLBSA is called. The command also sets up a stack segment for the compiler in such a way that no BCPL procedure that the user has executed or will execute later can interact undesirably with the compiler.

The compiler operates in three phases: syntax analysis, translation, and code generation. The first phase produces a tree representation of the input program and stores it in the compiler's stack; the second phase produces an intermediate-language representation of the program; this is stored in an intermediate segment. The third phase produces EPLBSA code.

Space-Time

The BCPL compiler occupies about 35K(10) words of storage, counting all text and link segments. In its present (unbound) form it consists of 22 segments of which 4 are common to all phases. Each phase uses about 10K(10) words, and the phases are called successively, so that about 15K(10) are in core at one time. Using the MULTICS system as of 8/12/68, a 393-line program took 1:06 minutes to compile, not counting EPLBSA time. This yields a speed of about 360 lines per minute of CPU time. Experiments with zero-length programs reveal that about 20 seconds of the time used by BCPL is spent in the bcpl command, the remainder in the compiler proper. It is hoped that the former time can be reduced by a more sophisticated use of EPL.

As a comparison, to compile the same program into EPLBSA using the GECOS version of the BCPL compiler took 40 seconds of processor time. On the other hand, the real time consumed by the GECOS compilation was about 2 minutes. This large difference is traceable to the rather slow disc I/O of GECOS.

EPLBSA required about 1:40 minutes to assemble the program resulting from the 393-line BCPL program mentioned above.

Notes on the "old" option

Only three BCPL constructions are treated differently depending on the "old" option. Strings and character constants in CTSS BCPL are both indicated using single quotes ('). In MULTICS BCPL, character constants use single quotes, string constants use double quotes ("); a character constant is exactly one character long. With the "old" option on, the CTSS interpretation of the single quote is used; with the option off, a string of more than one character enclosed by single quotes calls forth a diagnostic. However, the long character constant is compiled in the same way

as a string constant, just as in CTSS BCPL. Thus the only effect of the "old" option in this case is on the presence of a warning diagnostic.

The symbols "%(" and "%)" (RBRA and RKET) are recognized only with the "old" option on. These were used for grouping of arithmetic expressions, but in MULTICS BCPL ordinary parentheses (,) are used. In MULTICS BCPL with the "old" option off, "%" is used as the remainder operator.

The last difference occurs in the treatment of subscripts. In CTSS BCPL, subscripting is indicated by the character sequence "*(", so "V*(i)" is an instance of a subscripted variable. MULTICS BCPL with the "old" option on treats "*(...)" as the subscript operator; with the "old" option not on, "*" is always the multiplication operator. The standard way of indicating a subscript in MULTICS BCPL is by square brackets " ... ", and this notation is accepted regardless of the setting of the "old" option.

The only other way in which MULTICS BCPL fails to be a superset of CTSS BCPL is that the arithmetic and logical operators have slightly different precedence relations. There is no way of making MULTICS BCPL behave like the CTSS version in this respect, so parenthesization should be used in cases of doubt. (In fact, this difference is seldom important; in the entire BCPL compiler, much of which is directly copied from CTSS, there was only one instance of different interpretation of an expression.)