Identification

PL/I String Operations
D. B. Wagner and M. D. McIlroy

Purpose

PL/I operations on strings will be performed either directly
by the compiled code or through calls to the procedures
described here, according to the whim of the compiler.
[EPL compiles most string operations in-line when all
strings involved are of known length and 36 bits long
or less. One can imagine PL/I being considerably cleverer,
in particular performing all important operations on non-
varying strings directly in the compiled code. - D.B.W.]

Each of the procedures described here can take either varying
or non-varying strings as arguments. The details of how
the procedures distinguish varying strings from non-varying
strings, and what they do with varying - string answers,
are given in Implementation, below.

Usage

The possible calls are listed below with approximately
equivalent PL/I statements indicating their effect. B1,
b2, b3 are bit strings, varying or non-varying. The other
variables mentioned are declared:

        dcl answer bit(1), n fixed bin (24);

It will be noted that all of these procedures are entries
into the one segment stgop_. Stgop_ does not itself perform
the operation but "dispatches" any call to the appropriate
procedure to do the actual work.

        call stgop_$bsbs_(b1,b2);

            b2=b1;

        call stgop_$cscs_(c1,c2);

            c2=c1;

        call stgop_$ctbs_(b1,b2,b3);

            b3=b1 | |b2;

        call stgop_$ctcs_(c1,c2,c3);

            c3=c1 ||c2;

```
        call stgop_$ixbs_(b1,b2,n);
            n=index(b1,b2);
        call stgop_$ixcs_(c1,c2,n);
            n=index(c1,c2);

        call stgop_$ntbs_(b1,b2);
            b2=¬ b1;

        call stgop_$ndbs_(b1,b2,b3);
            b3=b1&b2;

        call stgop_$orbs_(b1,b2,b3);
            b3=b1 |b2;

        call stgop_$eqbs_(b1,b2,answer);
            answer=(b1=b2);
        call stgop_$eqcs_(c1,c2,answer);
            answer=(c1=c2);

        call stgop_$nebs_(b1,b2,answer);
            answer=(b1¬ =b2);
        call stgop_$necs_(c1,c2,answer);
            answer=(c1¬ =c2);

        call stgop_$lebs_(b1,b2,answer);
            answer=(b1< =b2);
        call stgop_$lecs_(c1,c2,answer);
            answer=(c1< =c2);
```

```
call stgop_$gebs_(b1,b2,answer);

    answer=(b1> =b2);

call stgop_$gecs_(c1,c2,answer);

    answer=(c1> =c2);


call stgop_$ltbs_(b1,b2,answer);

    answer=(b1< b2);

call stgop_$ltcs_(c1,c2,answer);

    answer=(c1< c2);


call stgop_$gtbs_(b1,b2,answer);

    answer=(b1> b2);

call stgop_$gtcs_(c1,c2,answer);

    answer=(c1>  c2);
```

There are a number of places where it may be useful to
call these procedures directly from a PL/I program: for
example in the File System modules which may not use varying
strings (because of the danger of embarassirg segment
faults), using direct calls rather than assignment statements
to perform string operations will prevent the compiler
from creating varying-string temporaries.   The compiler
may be clever enough to avoid these unnecessary temporaries,
but it is probably not advisable to count on this.

Implementation

See BP.2.01 for the representation of strings.   There
are three possible identity codes in the dope for a string:

    200(8) non-varying, aligned

    240(8) non-varying, packed

    202(8) varying

Thus the procedures can easily work with any kind of string
passed to them.

If the result of an operation is varying, so that its
current value is kept in a free storage area, the procedure
must allocate sufficient storage for the new value, perform
the operation, and then free the storage associated with
the old value. (Allocating and freeing is done using
the procedures described in BP.4.02.) The reason for
the insistence upon not freeing the old value until the
new value has been calculated is that otherwise the compiler
would have to make a special case out of such a statement
as

        a=a;

where a is a varying string.

There is still a problem with a statement like

        a=b||a;

where a is non-varying. This statement cannot be implemented
through the call

        call stgop_$ctcs_(b,a,a);

as one might expect, since ctcs_ will "clobber" a while
it still needs its value. The only way the concatenation
procedure could be sure of avoiding this problem would
be to place its result into a temporary in the stack,
then move this to the result string. It was apparently
felt that this would be too inefficient for the general
case. Therefore whenever a non-varying string variable
is being assigned the value of an expression involving
itself, a temporary must be used. [EPL, of course, always
produces a temporary. One hopes the PL/I compiler can
be cleverer.]