Published: 06/01/67
(Supersedes BN.7.04 03/11/67)

Identification

EPL String Operations
D. B. Wagner and M. D. McIlroy

Purpose

EPL operations on strings are performed either directly by the compiled code or through calls to the procedures described here, according to the whim of the compiler. [At present, EPL compiles most string operations in-line when all strings involved are of known length and 36 bits long or less.]

Each of the procedures described here can take either varying or non-varying strings as argument. 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 to stgop_ are listed below. Following each call to stgop_ is the call to a string routine which stgop_ invokes. Following this is the approximately equivalent PL/I statement indicating its effect. B1, b2, b3, are bit strings, varying or non-varying; c1, c2, c3 are character strings, varying or non-varying. The other variables mentioned are declared

```
dcl answer bit (1), n fixed bin (24), fx fixed bin (63);
```

It will be noted that all of these procedures are entries into the one segment stgop_.

```
call stgop_$bsbs_(b1,b2);
call movstr_$movb_(b1,b2);

     b2 = b1;

call stgop_$cscs(c1,c2);
call movstr_$movc_(c1,c2);

     c2 = c1;
```

```
call stgop_$ ctbs_(b1,b2,b3);
call catstr_$catstrb_(b1,b2,b3);

     b3 = b1||b2;

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

     c3 = c1||c2;

call stgop_$ixbs_(b1,b2,n);
call index_$indexb_(b1,b2,n);

     n = index(b1,b2);

call stgop_$ixcs_(c1,c2,n);
call index_$infrxc_(c1,c2,n);

     n = index(c1,c2);

call stgop_$ntbs(b1,b2);
call movstr_$not_(b1,b2);

     b2 = b1;

call stgop_$ndbs_(b1,b2,b3);
call andstr_$andstr_(b1,b2,b3);

     b3 = b1 & b2

call stgop_$orbs_(b1,b2,b3);
call andstr_$orstr_(b1,b2,b3);

     b3 = b1|b2;

call stgop_$eqbs_(b1,b2, answer);
call strcmp_$eqb_(b1,b2, answer);

     answer = (b1=b2);

call stgop_$eqcs_(c1,c2,answer);
call strcmp_$eqc_(c1,c2,answer);

     answer = (c1=c2);

call stgop_$nebs_(b1,b2,answer);
call strcmp_$neb_(b1,b2,answer);

     answer = (b1=b2)
```

```
call stgop_$necs_(c1,c2,answer);
call strcmp_$nec_(c1,c2,answer);

     answer = (c1=c2);

call stgop_$lebs_(b1,b2,answer);
call strcmp_$leb_(b1,b2,answer);

     answer = (b1<=b2);

call stgop_$lecs_(c1,c2,answer);
call strcmp_$lec_(c1,c2,answer);

     answer = (c1<=c2);

call stgop_$gebs_(b1,b2,answer);
call strcmp_$geb_(b1,b2,answer);

     answer = (b1>=b2);

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

     answer = (c1>=c2);

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

     answer = (b1<b2);

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

     answer = (c1<c2);

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

     answer = (b1>b2);

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

     answer = (c1>c2);

call stgop_$bsfx_(b1,fx);

     fx = fixed (b1,63);
```

There are a number of places where it may be useful to call these procedures directly from an EPL program: for example in the File System modules which may not use varying strings (because of the danger of embarassing 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 and BN.5.00 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.