Table of Contents

Syntax A self-defining structure is one that contains information describing the lengths and bounds of some of its subordinate elements (and therefore their locations).

PL/I implements self-defining based structures using the REFER option in the declaration. REFER specifies a field in the structure to hold the length and bound information, called a *REFER object.* At run-time, the generated code uses the value of the refer object to access the data. These values are initially set from specifications in the declaration when the structure is allocated, and may be modified by the user's program thereafter, as long as this does not cause the structure to be larger than initially allocated.

Self-defining structures are an IBM extension to PL/I, and are not specified in either the full PL/I or the "subset" standards.

Syntax

The general syntax for a REFER option declaration is:

```
expression REFER ( variable )
```

expression is the declared maximum length or bound, used when the structure is allocated, and *variable* is name of the REFER object.

An example of a complete declaration is:

```
DECLARE init_array_size FIXED BINARY(7) INITIAL(5);
DECLARE 1  structure BASED,
                      2  size_of_array     FIXED BINARY(31),
                      2  variable_elem  (init_array_size
REFER(size_of_array)) CHARACTER(8);
```

(A constant could be used in place of *init_array_size* in this declaration.)

When this structure is allocated, storage for five (*init_array_size*) occurrences of *variable_elem* will be reserved, and *size_of_array* will be initialized to five. During execution the program may set *size_of_array* to any value not exceeding five, and store that many values in *variable_elem.* If structure is written to a file, only

the number of elements specified are written. When the structure is subsequently read, *size_of_array* will be set to indicate the number originally written.

The syntax is similar for string lengths, for example:

```
… 2  variable_elem CHARACTER(init_array_size REFER(size_of_array) );
```

Multiple REFER options may be specified for an element, for example specifying lower and upper bounds and string length, and multiple elements can be declared with the REFER option. The same REFER object may be specified for multiple elements (assuming the values are the same, naturally).

Changing the value of a REFER object will “remap” the structure, and possibly change the locations of some elements. This will not cause any current data to be moved.

Restrictions

- All REFER objects must be at logical level two in the structure, that is not part of a minor structure.
- REFER objects cannot be array elements.
- All REFER objects must precede the first element declared with a REFER option.
- All must be FIXED BINARY(31,0).
  [IBM PL/I requires FIXED BINARY but allows any precision]

Tips

- Self-defining structures involve substantial overhead. Avoid them if other solutions are available.
- When declaring bound information, no check is made that the upper bound is greater than the lower bound.
- The refer objects should never be set such that the current size of the structure would exceed its initially declared size.
- Files read or written should be declared `ENV(VARLS RECSIZE(n))`, where n is at least the length of the longest possible record plus 2 bytes.
- Only direct assignments to REFER objects are recognized. For example, the following might not function correctly.:

```
DECLARE p POINTER;
DECLARE fb FIXED BINARY(31) BASED;
```

```
p = ADDR(some_refer_object);
…
p->fb = foo;
```

Do not use builtin functions such as PLIFILL or PLIMOVE to make assignments to the structure.

Declaring the structure with the ABNORMAL attribute will allow this to work correctly, at the cost of significantly more overhead.

Sample Program

```
 /* refsamp                                                  */
 /***********************************************************/
 /*                                                         */
 /* Module:   refsamp                                       */
 /*           Peter Flass -- Jan 2023                       */
 /*                                                         */
 /* Function: Sample program for self-defining structures */
 /*           with file I/O.                                */
 /*                                                         */
 /*           Program allocates a self-defining structure */
 /*           and creates and writes records of different */
 /*           lengths. The data is then read back and     */
 /*           printed.                                      */
 /*                                                         */
 /* Usage:    refsamp                                       */
 /*                                                         */
 /* Dependencies:                                           */
 /*                                                         */
 /*                                                         */
 /***********************************************************/
refsamp: proc options(main);
  dcl    (p,q)               ptr;
  dcl     n                  fixed bin(31);
  dcl     i                  fixed bin(31);
  dcl   1 struc              based(p),
          5 char_occ         fixed bin(31),
          5 char             char(64 refer(char_occ));
  dcl     vfile              file env(varls recsize(80));
  dcl     eof                bit(1)              init('0'b);

  /*-------------------------*/
  /* Test Data               */
  /*-------------------------*/
  dcl     test_string     ( 8)char(16)  varying static   init(
    'String One', 'Str 2', 'Three',        'String 4',
    'St 5',       'Str 6', 'String SEVEN', 'Last' );

  on endfile(vfile) eof='1'b;
```

```
put skip list('refsamp: Test self-defining structures');
alloc struc set(p);                    /* Allocate the structure    */
q = p;                                 /* Save addr for later       */
put skip edit('allocated size',stg(p→struc))(a,p'zz9');
open file(vfile) output title('ReferTest.dat');

put skip(2) edit('Writing',hbound(test_string),' records')
                (a,p'zz9',a);
/*-------------------------*/
/* Write records           */
/*-------------------------*/
do n=1 to hbound(test_string);
  p->char_occ = length(test_string(n));
  p->char     = test_string(n);
  i = cstg(p→struc);
  put skip edit('writing',i,heximage(p,i))(a,p'zz9',x(1),a);
  write file(vfile) from(p→struc);
  end; /* do n */
put skip list('all records written');
close file(vfile);

/*-------------------------*/
/* Reread and print records*/
/* using READ SET()        */
/*-------------------------*/
open file(vfile) input title('ReferTest.dat');
put skip(2) list('trying READ SET()');
read file(vfile) set(p);
do while(^eof);
  put skip edit(length(p->char),': "',p→char,'"')(p'zzz9',a,a,a);
  read file(vfile) set(p);
  end; /* do while */
close file(vfile);

/*-------------------------*/
/* Reread and print records*/
/* using READ INTO()       */
/*-------------------------*/
p = q;                                 /* Res addr after read(set)  */
eof='0'b;                              /* Reset EOF                 */
```

```
open file(vfile) input title('ReferTest.dat');
put skip(2) list('trying READ INTO()');
read file(vfile) into(struc);
do while(^eof);
  put skip edit(length(p->char),': "',p→char,'"')(p'zzz9',a,a,a);
  read file(vfile) into(struc);
  end; /* do while */
close file(vfile);

put skip(2) list('refsamp test complete');
end refsamp;
```

Sample Output

Here is the printed output of the sample program, followed by a hexadecimal dump of the file created.

```
./refsamp

refsamp: Test self-defining structures
allocated size 68

Writing  8 records
writing 14 0A0000005374726696E67204F6E65
writing  9 0500000053747220 32
writing  9 0500000054687265 65
writing 12 080000005374726696E672034
writing  8 0400000053742035
writing  9 0500000053747220 36
writing 16 0C0000005374726696E6720534556454E
writing  8 040000004C617374
all records written

trying READ SET()
  10: "String One"
   5: "Str 2"
   5: "Three"
   8: "String 4"
   4: "St 5"
   5: "Str 6"
  12: "String SEVEN"
   4: "Last"

trying READ INTO()
  10: "String One"
   5: "Str 2"
   5: "Three"
   8: "String 4"
   4: "St 5"
   5: "Str 6"
  12: "String SEVEN"
   4: "Last"

hexdump -C ReferTest.dat
00000000  0e 00 0a 00 00 00 53 74  72 69 6e 67 20 4f 6e 65  |......String One|
00000010  09 00 05 00 00 00 53 74  72 20 32 09 00 05 00 00  |......Str 2.....|
00000020  00 54 68 72 65 65 0c 00  08 00 00 00 53 74 72 69  |.Three......Stri|
```

```
00000030  6e 67 20 34 08 00 04 00  00 00 53 74 20 35 09 00  |ng 4......St 5..|
00000040  05 00 00 00 53 74 72 20  36 10 00 0c 00 00 00 53  |....Str 6......S|
00000050  74 72 69 6e 67 20 53 45  56 45 4e 08 00 04 00 00  |tring SEVEN.....|
00000060  00 4c 61 73 74                                    |.Last|
```