Legac-E Education

Passing a variable number of parameters in COBOL

On 20th March 2020 the question was asked about passing a variable number of parameters to a COBOL sub-routine. This had arisen because an installation had an existing Assembler sub-routine, called by many COBOL programs, and the desire was to rewrite the sub-routine in COBOL, without amending any of the multitude of calling programs.

There were two givens; the number of parameters passed was a truncated list rather than having intermediate parameters omitted, and that COBOL sets the high-order bit to one in the 4-byte address field of the last parameter. It was found that if the COBOL calling routine used the OMITTED keyword then the corresponding parameter address would be binary zeroes (NULL).

There did not appear to be an easy way of validating the high-order bit of the address field from within COBOL. Using the COBOL ENTRY statement does provide scope for alternative parameter lists, but that statement requires a different name to PROGRAM-ID, and hence requires ALIAS statements at Link-Edit time. This would then necessitate changing each caller to reflect the appropriate sub-routine name even if it were a single program with aliases.

The solution adopted was to code an Assembler interface module to inspect and modify the parameter list passed by the caller, prior to passing control to a COBOL sub-routine which performed the required logic. The end result is that the sub-routine receives a consistent set of address fields, albeit with the trailing, omitted parameter address set to binary zero. As the Assembler interface routine has the same name as the original routine, it means there are no changes to any of the calling program..

This is the second iteration of this document, as the original was produced and tested solely in a Batch environment and it was subsequently identified that there was a need for the routine to be incorporated into CICS programs. In consequence the z/OS LOAD and CALL macros, which enabled dynamic linkage to the next COBOL subroutine, were replaced by a sequence of instruction using a V-type constant to obtain the address of the subroutine to be invoked by this Assembler routine, with transfer of control being passed via a BASR. This routine and its COBOL subroutine must now be statically linked together.

The Language Environment compliant code which follows is the solution adopted, and was tested under z/OS 2.3 with Enterprise COBOL Version 6 Release 2 callers and sub-routines..

Passing a variable number of parameters in COBOL

```
         PRINT NOGEN
         TITLE 'BRIDGE - INTERFACE PROGRAM FOR TRUNCATED PARM LISTS'
*-----------------------------------------------------------------*
*        Author   : T R Sambrooks                                 *
*        Written  : 22nd-24th March 2020                          *
*        Modified : 2nd April 2020                                *
*-----------------------------------------------------------------*
*        When a COBOL program issues a CALL to a sub-routine with *
*        parameters USING the BY REFERENCE option, a consecutive  *
*        list of addresses is built representing the lpcation of  *
*        each parameter. If the COBOL program issues a subsequent *
*        CALL but using fewer parameters, the addresses of the    *
*        trailing omitted parameter will remain unchanged. COBOL  *
*        does set the high-order bit to one of the last actual    *
*        parameter but there does not appear to be a mechanism    *
*        within COBOL to test this bit. Even tests with the LE    *
*        routine, CEESITST proved fruitless.                      *
*                                                                 *
*        The purpose of this routine is to facilitate  passing of *
*        truncated parameter lists between COBOL programs. (Note that
*        OMITTED intermediate parameters are handled as they are  *
*        transparent to the routine.)                             *
*                                                                 *
*        The routine relies on the fact that the high-order bit of *
*        field containing the last parameter address will be on, i.e.
*        set to one. The routine sets a new parameter list to binary
*        zeroes prior to inspecting each passed address in turn, and
*        adds the passed address to the new list if the high-order *
*        bit is off. If the high-order bit is on, the routine stops
*        the inspection, adds the current parameter address to the *
*        new list, and calls the real COBOL subroutine. There is no
*        need to explicitly set the high-order bit of the last    *
*        parameter in the new list as it is implicitly set when   *
*        the original address is added.                           *
*                                                                 *
*        The effect of this process is that the COBOL subroutine will
*        receive an amended parameter list with binary zeroes (NULL)
*        in each parameter address field that is not required.    *
*-----------------------------------------------------------------*
*        The 2nd April 2020 modification was to remove the dynamic *
*        CALL to the next sub-routine. This makes the routine     *
*        usable in both Batch and CICS provided that a COBOL CALL, *
*        rather than CICS LINK is used and that the subsequent    *
*        routine does not want to exploit CICS commands.          *
*-----------------------------------------------------------------*
```

Legac-E Education

Passing a variable number of parameters in COBOL

```
*        To modify the routine to cater for a larger initial list of*
*        parameters do the following:                                *
*        - Add additional 1-byte fields between P7 and PARMSEND      *
*                                                                    *
*        The constraints of the MVC instruction used to zeroise the *
*        new parameter list restricts the maximum number of          *
*        parameters to 64 (256/4).                                   *
*--------------------------------------------------------------------*
BRIDGE   CEEENTRY PPA=MYPPA,AMODE=31,RMODE=31,MAIN=NO,PARMREG=1
         LR    R9,R1                  R9  = PARAMETER LIST ADDRESS
         LA    R10,P1                 R10 = NEW PARM LIST ADDRESS
         MVC   P1(PARMLEN),ZEROADDR   ZEROISE PARM ADDRESSES
CHECKVL  TM    0(R9),X'80'            IS VL BIT SET?
         BO    LASTPARM               YEP - DEAL WITH END OF LIST
         MVC   0(4,R10),0(R9)         PASS THE PARM ADDRESS
         LA    R9,4(R9)               R9  = NEXT ADDRESS SLOT
         LA    R10,4(R10)             R10 = NEXT ADDRESS SLOT
         B     CHECKVL                REPEAT TILL LIST EXHAUSTED
CALLSUB L     R1,=A(NEWPARMS)        R1  = PARAMETER LIST
         L     R15,=V(COBSUB)         R15 = SUB-ROUTINE ENTRY ADDR
         BASR  R14,R15                GO EXECUTE SUBROUTINE
         CEETERM RC=0                 LOGICAL END OF PROGRAM
LASTPARM MVC   0(4,R10),0(R9)         PASS LAST PARAMETER ADDR
         B     CALLSUB                GO EXECUTE SUBROUTINE
         LTORG
ZEROADDR DC    256AL1(0)              FOR ZEROISING PARMS
*--------------------------------------------------------------------*
*        Only insert code in this box if extending the dummy         *
*        parameter list.                                             *
*--------------------------------------------------------------------*
NEWPARMS DS    0F                                                    *
P1       DS    AL4                    ADDRESS PARAMETER 1          *
P2       DS    AL4                    ADDRESS PARAMETER 2          *
P3       DS    AL4                    ADDRESS PARAMETER 3          *
P4       DS    AL4                    ADDRESS PARAMETER 4          *
P5       DS    AL4                    ADDRESS PARAMETER 5          *
P6       DS    AL4                    ADDRESS PARAMETER 6          *
P7       DS    AL4                    ADDRESS PARAMETER 7          *
PARMSEND DS    0C                     END OF PARAMETER LIST        *
NUMPARMS EQU   (PARMSEND-P1)          NUMBER OF PARMS IN LIST      *
*--------------------------------------------------------------------*
PARMLEN  EQU   NUMPARMS*4             TOTAL LENGTH OF PARAMETER LIST
MYPPA    CEEPPA
         CEECAA
         CEEDSA
R0       EQU   0
R1       EQU   1
R2       EQU   2
```

Passing a variable number of parameters in COBOL

```
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END   ,                    PHYSICAL END OF THE PROGRAM
```