SUBROUTINES WITH VARIABLE NUMBER OF ARGUMENTS

BY

UDAY G. GUJAR

TR75-004, APRIL 1975

SUBROUTINES WITH VARIABLE NUMBER OF ARGUMENTS

BY

UDAY G. GUJAR

School of Computer Science
and
Computing Centre
University of New Brunswick
Fredericton, N.B.

TR75-004, April 1975

ABSTRACT

In designing a general purpose subroutine package to solve a class of problems, one often has to write subroutines with a large number of arguments. Though these arguments are required to cover a range of possibilities, many of these arguments have some commonly occurring values. The user is usually burdened with supplying a long list of arguments and making sure that the number and types match. Alternate solution is to write such subroutines in the assembly language so that they could have variable number of arguments. This approach eliminates a large class of program designers who do not and do not want to know the assembler language. This paper describes a facility which enables these program designers to write their routines completely in a higher level language (FORTRAN) and yet enjoy the "luxury" of having variable number of arguments in the calling sequence.

1. INTRODUCTION

The techniques for writing subroutines in an Assembly language callable from higher level languages such as FORTRAN, PL/1, etc. are well-known. If such a routine is to have variable number of arguments, it has to be written in the Assembly language. This eliminates a large class of the users who do not and do not want to know the Assembly language. Further, it is tedious and laborious to code each such routine in the Assembly language just so that it could have the "luxury" of being called with variable number of arguments.

This paper suggests a general method which enables a FORTRAN programmer to write subroutines in FORTRAN with variable number of arguments. The method has been implemented and is in use for the past several years on an IBM 360/50 and an IBM 370/158 computer. It is believed that perhaps a similar technique could be used for the other higher level languages.

2. METHOD

2.1 General Requirements

The method consists of writing one Assembly language routine which is capable of achieving the following:

(a) Get the address of the area of the arguments of the calling routine.

(b) Determine if the number of arguments supplied is less than that given in the definition of the calling routine; if it is go to step (c); otherwise, go to step (d).

(c) Get the default values for the omitted arguments and append the argument list.

(d) Take care of initialization of the calling routine, if any, and return.

The step (d) above is rather ambiguous as stated; it will be clear while going through the next section.

The user who intends to write a subroutine with variable number of arguments simply calls this assembly language subroutine once in the definition of such subroutine.

2.2. Implementation for FORTRAN on IBM 360 and 370 Computers

It is assumed that the reader is familiar with the linkage conventions between FORTRAN and OS/VS ASSEMBLER (see Ref. 1 for details). The following steps are involved in writing a subroutine, let us call it VARARG (Ref. 2), to enable a FORTRAN programmer to write FORTRAN subroutines with variable number of arguments to run on an IBM 360 or an IBM 370 computer:

(a) Pick up the address, say ARGSUP, of the argument list of the calling program.

(b) Build another argument list, say ARGNEW, consisting of all the addresses from ARGSUP.

(c) Append the address list in ARGNEW, if necessary, by the address list of the default values of the omitted arguments. The address list for the default values is supplied through the arguments of VARARG.

(d) Change callers argument list pointer to point to this new appended list - ARGNEW.

(e) Set a software switch to bypass all the above steps and to branch to the step (h) on next entrance to VARARG. (This is necessary to avoid recursive loops because of the next step.)

(f) Branch to the beginning of the calling routine so that all the initialization of the calling routine is redone with this newly supplied appended argument list.

(g) Because of step (f), VARARG is entered again, but because of step (e) a new path is taken which bypasses all the steps and starts at step (h).

(h) Reset the software switch so that the next entry into VARARG will start at step (a).

(i) Return normally to the caller.

It is interesting to note that the above method will work (and has worked) satisfactorily for the nested calls;

i.e., the subroutine A calls B which calls C which in turn calls D and all A,B,C, and D are to have variable number of arguments. The proof is simple and is left to the reader as an exercise.

3. USAGE

The usage has to be considered at two levels:

(a) Programmer level: The person who writes the subroutine which is to have variable number of arguments.

(b) User level: The person who calls the above routine.

3.1 Programmer Level

Let us assume that we want to write a subroutine RTN with M compulsory arguments (M>0) and N optional arguments. Let L (=M+N) be the total number of (maximum possible) arguments. The definition of this subroutine RTN will be as follows:

```
SUBROUTINE RTN(C1,C2,...,CM,/O1/,/O2/,...,/ON/)
DIMENSION  ARG(L)
CALL VARARG(ARG,C2,C3...,CM,D1,D2,...,DN)
 : : :
END
```

Note the following things:

(a) Array ARG which is passed as the first argument of VARARG contains at least L elements. This array is not to be used by the FORTRAN programmer.

(b) The call to VARARG inside the definition of RTN must have at least L arguments the first of which is an array as in note (a) above. The next M-1 arguments are ignored but must be supplied. The remaining N arguments contain the desired default values to be assumed by the optional arguments, if omitted.

(c) When the above routine RTN is called by the user, the arguments O_1, O_2,...,O_N may be omitted by the user. The omitted arguments will assume the corresponding default values D_1, D_2,...,D_N supplied to VARARG inside the definition of RTN.

(d) M is assumed to be a non-zero positive quantity. This limitation was imposed by earlier versions of FORTRAN compilers. For the latest compilers, a trivial modification in VARARG will allow M to be equal to zero.

(e) Each optional argument is enclosed in slashes; i.e., it is referred to by location (see Ref. 4). This is imposed by the FORTRAN compiler. However, this is not a serious drawback since these slashes are to be punched only once while defining the subroutine and are not required at the "user level".

(f) Usually CALL VARARG will be the first executable statement, but not necessarily so. It is sufficient (and necessary) for the programmer to understand that the statements up to and including the call to VARARG are executed twice.

3.2 User Level

The user calls the subroutine RTN just like he calls any other FORTRAN subroutine except he has the option of omitting up to N arguments at the end. There is only one rule that the arguments can only be omitted from the right hand end. This rule can best be explained by an example. Consider a routine RTN with the following "nominal" calling sequence:

```
CALL  RTN(B,I,A[,Y,C,Z])
```

the square brackets are included to indicate that Y, C and Z are the optional arguments. (Needless to say that these square brackets are not to be punched.) The following are the valid calling sequences:

(a) `CALL  RTN(BB,II,AA)`

Y, C and Z will have the default values.

(b) `CALL  RTN(BB,II,AA,YY)`

C and Z will have the default values.

(c) `CALL  RTN(BB,II,AA,YY,CC)`

Z will have the default value.

(d) `CALL  RTN(BB,II,AA,YY,CC,ZZ)`

default values will not be used at all.

Note that if a non-default value for C is desired, a value for Y must be specified even if the default value for Y is suitable. Similarly, in order to be able to specify a value for Z, both Y and C must be specified in that order.

4. EXAMPLE

Several routines have been developed at the author's installation which make use of this facility. The plotting package (Ref. 3) written by the author makes use of this idea extensively. Three routines will be given in this section, both at the programmer and user level. See Reference 3 for more details on the use of these routines.

4.1 Subroutine CIRCLE

4.1.1 Programmer Level

```
      SUBROUTINE CIRCLE(XC,YC,RADIUS,/FROMTH/,TOTH/)
      DIMENSION  ARGS(5)
      CALL VARARG(ARGS,YC,RADIUS,0.,6.283185)
C....
C.... CODING TO DRAW AN ARC OF A CIRCLE FROM AN ANGLE
C.... 'FROMTH' TO 'TOTH' WITH (XC,YC) AS THE CENTRE AND
C.... 'RADIUS' AS THE RADIUS.
        .  .  .
        .  .  .
        .  .  .
        .  .  .
      END
```

4.1.2. User Level

```
1  CALL CIRCLE(3.,4.,5.)
2  CALL CIRCLE(3.,4.,4.,3.14159)
3  CALL CIRCLE(3.,4.,3.,3.14159/2.,3.14159)
```

The statement 1 will draw a full circle with the centre at (3.,4.) and radius of 5 units, while the statement 2 will draw a half circle of radius 4 units. The statement 3 will draw one fourth of the circle of radius 3 units occupying the second quadrant.

4.2 Subroutine RECT

4.2.1 Programmer Level

```
      SUBROUTINE RECT(X,Y,XL,/YL/,/THETA/)
      COMMON  /P611G/FACTOR
      DIMENSION ARGS(5)
      DFLTYL=XL
      IF (DFLTYL.LT.0)DFLTYL=-XL*FACTOR
      CALL VARARG(ARGS,Y,XL,DFLTYL,0.)
C....
C.... STATEMENTS TO DRAW A RECTANGLE WHOSE "LEFT HAND
C.... CORNER" IS AT X,Y, WHOSE WIDTH IS XL UNITS, HEIGHT
C.... IS YL UNITS AND WHICH IS TILTED BY AN ANGLE THETA.
        .  .  .
        .  .  .
        .  .  .
        .  .  .
      END
```

An interesting feature introduced in this example is that the default value for the argument YL is a known function of the argument XL.

4.2.2 User's Level

```
CALL RECT(3.,4.,1.)
CALL RECT(3.,4.,1.,1.,3.14159)
CALL RECT(3.,4.,-2.)
```

4.3 Subroutine NMBR

4.3.1 Programmer Level

```
      SUBROUTINE NMBR(X,Y,NUMBER,FORMAT,/THETA/,
     *  /HEIGHT/,/ILIM/,/ISTART/,/IINCR/,/XRET/,/YRET/)
      DIMENSION ARGS(11)
      CALL VARARG(ARGS,Y,NUMBER,FORMAT,0.,.1,1,1,1,
     *  XX,YY)
C.... STATEMENTS TO PLOT AS MANY NUMBERS AS YOU WANT
C.... ACCORDING TO ANY DESIRED FORMAT AT ANY SPECIFIED
C.... ANGLE WITH ANY DESIRED HEIGHT
         .  .  .
         .  .  .
         .  .  .
         .  .  .
   1  XRET = RETURN VALUE OF X CO-ORDINATE
   2  YRET = RETURN VALUE OF Y CO-ORDINATE
      RETURN
      END
```

It is not necessary to go into the detailed discussion of the arguments of NMBR. The last two arguments XRET and YRET are of particular interest. These are the variables into which the co-ordinates of the plotting pencil are returned to the user. The default "values" supplied in the subroutine VARARG are therefore some other variables (and not constants). Note the statements numbered 1 and 2 which refer to XRET and YRET and not to the default variables XX and YY. VARARG assures the proper switching between XRET and XX (and YRET and YY) depending upon whether XRET is supplied by the user or not.

4.3.2 User Level

It should suffice to say that the user may omit the last 7 arguments. If the arguments in the positions of XRET and/or YRET are supplied they must be variables.

5. CONCLUDING REMARKS

The facility described in this paper is simple, straightforward and easy to use. The design and implementation of a generalized subroutine package to solve a class of problems is simplified by such a facility.

The amount of memory required for the routine VARARG is quite small (126 decimal bytes). It would be interesting to know how this method will work for other languages

and on the computers of other manufacturers. It is felt that the same routine will work with PL/1 without any modifications; however, the author has not varified this.

Needless to say that it is a simple matter to incorporate this facility into the language itself and build it into the compiler. Whether or not this should be done is best left to the judgement of the readers, compiler writers, computer linguists and the others.

REFERENCES

1. IBM System/360 Operating System: FORTRAN IV (G and H) Programmer's Guide, Form GC28-6817-2, pp. 146-155.

2. Gujar, U.G., 'VARARG - a facility to write FORTRAN subroutines with variable number of arguments in FORTRAN', CL286, Computing Centre Library Programs, University of New Brunswick, 1972.

3. Gujar, U.G., 'Computer Plotting', Computing Centre, University of New Brunswick, 212 pages, 1972, second printing Dec. 1974.

4. IBM System/360 System/370 FORTRAN IV language, form GC28-6515-8, p. 100.

APPENDIX

Listing of the Subroutine VARARG

```
 1 VARARG   FENTER
 2+VARARG   CSECT
 3+         USING *,15
 4+         B     12(0,15)
 5+         DC    AL1(6)
 6+         DC    CL6'VARARG'
 7+         STM   14,12,12(13)
 8+         LR    12,15
 9+         DROP  15
10+         USING VARARG,12
11+***** LOWEST LEVEL SUBROUTINE OR ENTRY GENERATED.

12 SWITCH   NOP   ONOFF          'NOP' FIRSTTIME, 'B' SECOND TIME
13          L     13,4(13)       ADDR. OF CALLER SAVE AREA
14          L     10,24(13)      GET HIS REG. 1
15          L     9,0(1)         BUILD ARG. LIST IN THIS ARRAY
16          LR    8,9            REMEMBER THIS IMPORTANT ADDRESS
17          LA    7,4            WE HAVE TO ADD 4 TOO MANY TIMES
18 NEXT     MVC   1(3,9),1(10)   GET HIS ARG.
19          MVI   0(9),0         MAKE SURE THAT THERE IS NO X'80'
20          TM    0(10),X'80'    IS THIS HIS LAST ARG.?
21          BO    DEFAULTS       IF YES, BRANCH TO GET DEFAULTS
22          TM    0(1),X'80'     IN CASE VARARG HAS LESS ARGS.
23          BO    FINI           DING DONG DING DONG, GOT YOU
24          AR    1,7            SKIP THIS DEFAULT(!)
25          AR    10,7           POINT TO HIS NEXT ARG.
26          AR    9,7            UPDATE OUR NEW ARG. LIST POINTER
27          B     NEXT           CHURN SOME MORE
28 DEFAULTS TM    0(1),X'80'     WAS IT LAST ARG. FOR VARARG?
29          BO    FINI           IF YES, YIPPI I I I
30          MVC   4(4,9),4(1)    GET A DEFAULT FROM VARARG-ARGS.
31          AR    1,7            POINT TO NEXT DEFAULT VALUE
32          AR    9,7            UPDATE OUR DINGY POINTER
33          B     DEFAULTS       WE MUST GET ALL THE DEFAULTS
34 FINI     MVI   SWITCH+1,X'F0' 'B' IN SWITCH TO AVOID RECURRSION
35          MVI   0(9),X'80'     INDICATE THAT THIS IS LAST ARG.
36          ST    8,24(13)       POINT HIS REG 1 TO NEW ARG. LIST
37          LM    14,12,12(13)   RESTORE HIS REGS.
38          BR    15             RECALL THE CALLING ROUTINE
39 ONOFF    MVI   SWITCH+1,0     RESTORE 'NOP' IN 'SWITCH'
40          LM    14,12,12(13)
41          BR    14             NORMAL RETURN
42          END
```