External procedures in New Stanford Pascal

First of all: please excuse possible errors in my English; I am German and not a native English speaker … I will do the best I can.

External procedures are procedures that are compiled separately from the main procedures and linked later. On the mainframe, this is done by a separate linkage editor step (on MVS) or during load time (on CMS) – here every external procedure is compiled separately to 370 machine code. On Windows etc., several PRR files are combined by PCINT during program startup (when PCINT reads and „assembles" the PRR files into an internal in-memory representation).

Because the different parts of such a combined program are compiled seperately, the compiler has no control over the correct definition of interfaces etc.; that is: if the parameters are defined in a compatible way in both parts of the communication. This is left to the programmer. The runtime errors which can occur, if the parameters don't match, may be very hard to find and to repair.

On the mainframe, it is now possible to call external procedures and functions that are written in FORTRAN or ASSEMBLER, too. In fact, it was possible for FORTRAN since 1979, although seldom used – and restricted to some datatypes known to FORTRAN. But now it is possible for ASSEMBLER, too, with almost no restrictions. This is why I decided it would be a good idea to write a paper on this topic and give some examples, how external procedures can be connected to Pascal programs.

External procedures written in Pascal (aka Modules)

Since 2015 ca., it is possible to use the new „MODULE" syntax for external Pascal procedures.

Modules start with the MODULE keyword and have an empty main program (BEGIN END). Modules may have global CONST and TYPE definitions, even global STATIC variables (which are known only inside the module), but no normal „global" variables in the VAR definition section.

Of course, the procedures defined in the modules may have their local variables, as usual, and they can use the predefined types and functions, the local definitions from the module, and the predefined files INPUT, OUTPUT, PRR, PRD, QRR, QRD, which are shared across all modules.

The procedures and functions defined in a module are visible to the linkage editor (that is: known outside the module), if they appear at the top level of the module. If not (level 2 and lower), they are not known to the linker.

You can limit the visibility of a procedure to the module, if you add the LOCAL keyword to the procedure or function definition, for example:

```
local procedure PLOC (x : integer);
begin end;
```

will not be visible outside the module, even if it appears at level 1.

The number of external procedures and functions inside a module is not limited. You can group procedures and functions into modules as you like, depending on the needs of your application.

For example:

the Pascal runtime contains a module PASLIBX (Pascal library extended), which has different external entries. $PASSTR, $PASLIB, $PASMAT and $PASMEM are called by the compiler, when certain functions are needed. But is also contains procedures CMSX and WINX, used to pass commands to CMS and Windows, respectively.

You can find the source code of PASLIBX on the GitHub repository.

Here is an example of an external Pascal module:

```
module $PAS2PAS ;

//**********************************************************************
//$A+
//**********************************************************************

type CHAR20 = array [ 1 .. 20 ] of CHAR ;


procedure PAS2PAS ( X1 : INTEGER ; var X2 : INTEGER ; T1 : CHAR20 ; var
                  T2 : CHAR20 ) ;

   begin (* PAS2PAS *)
     WRITELN ( 'pas2pas: x1 = ' , X1 ) ;
     WRITELN ( 'pas2pas: x2 = ' , X2 ) ;
     WRITELN ( 'pas2pas: t1 = <' , T1 , '>' ) ;
     WRITELN ( 'pas2pas: t2 = <' , T2 , '>' ) ;
     X2 := 5 ;
     T2 := 'string from pas2pas' ;
   end (* PAS2PAS *) ;


function PAS2Pf ( X1 : INTEGER ; x2 : INTEGER ) : integer ;

   begin (* PAS2pf *)
     pas2pf := ( x1 + x2 ) div 2 ;
   end (* PAS2PAS *) ;


begin (* HAUPTPROGRAMM *)

end (* HAUPTPROGRAMM *) .
```

Both the procedure and the function are known to the linkage editor, because they are defined at level 1 and no local keyword was specified (take care: the external names should not exceed 8 characters).

To call the external function and the external procedure from the previous page, the calling program has to use the following code:

```
...

procedure PASCAL_TO_PASCAL ( X1 : INTEGER ; var X2 : INTEGER ; T1 :
                             CHAR20 ; var T2 : CHAR20 ) ;
   EXTERNAL 'PAS2PAS' ;

function PAS_TO_PAS_FUNC ( X1 : INTEGER ; X2 : INTEGER ) : INTEGER ;

   EXTERNAL 'PAS2PF' ;

begin (* HAUPTPROGRAMM *)
  //***************************************************************
  // set variables and test call of PASCAL proc
  //***************************************************************
  I := 42 ;
  N := 42 ;
  C1 := 'Test string 1' ;
  C2 := 'Test string 2' ;
  WRITELN ( 'Values before PAS2PAS call:' ) ;
  WRITELN ( 'i   = ' , I ) ;
  WRITELN ( 'n   = ' , N ) ;
  WRITELN ( 'c1  = <' , C1 , '>' ) ;
  WRITELN ( 'c2  = <' , C2 , '>' ) ;
  PASCAL_TO_PASCAL ( I , N , C1 , C2 ) ;
  WRITELN ( 'Values after PAS2PAS call:' ) ;
  WRITELN ( 'i   = ' , I ) ;
  WRITELN ( 'n   = ' , N ) ;
  WRITELN ( 'c1  = <' , C1 , '>' ) ;
  WRITELN ( 'c2  = <' , C2 , '>' ) ;
  //***************************************************************
  // test call of external functions
  //***************************************************************
  XRES := PAS_TO_PAS_FUNC ( 20 , 30 ) ;
  WRITELN ( 'PAS_TO_PAS_FUNC (20, 30) RETURNS ' , XRES ) ;
end (* HAUPTPROGRAMM *) .
```

It is not necessary to use a longer name for the external procedure and specify an alternate external name on the EXTERNAL clause, but it is possible …

you can use the original short name, if you wish and use only EXTERNAL ;

External procedures written in FORTRAN

If the external procedure or function is written in FORTRAN, you have to specify it on the EXTERNAL definition on the Pascal side.

Example:

```
procedure PASCAL_TO_FORTRAN ( X1 : INTEGER ; var X2 : INTEGER ; T1 :
                              CHAR20 ; var T2 : CHAR20 ) ;

   EXTERNAL FORTRAN 'PAS2FTN' ;

function PAS_TO_FTN_FUNC ( X1 : INTEGER ; X2 : INTEGER ) : INTEGER ;

   EXTERNAL FORTRAN 'PAS2FF' ;
```

Again, the alternate name on the EXTERNAL statement is not needed, if the original name of the procedure or function can be used. The compiler translates those names to uppercase, BTW (but not the alternate names inside the quotes).

Maybe the external FORTRAN names are limited to 6 chars; I could not test it, because I had no FORTRAN compiler available on MVS or CMS. I did my tests using ASSEMBLER subroutines which simulated the FORTRAN behaviour.

FORTRAN has several peculiarities, compared to Pascal and ASSEMBLER:

- all parameters must be passed by reference, no call by value
- the last parameter (address) must have the leftmost bit set
- function results have to be returned in register zero

The Pascal compiler does some additional work, so that the call to FORTRAN subroutines works correctly.

For example, if you look again at the external definition from the previous page:

```
procedure PASCAL_TO_FORTRAN ( X1 : INTEGER ; var X2 : INTEGER ; T1 :
                              CHAR20 ; var T2 : CHAR20 ) ;

   EXTERNAL FORTRAN 'PAS2FTN' ;
```

Because Fortran does not support call-by-value, Pascal copies the X1 value into a temporary variable (a so-called dummy argument – I borrowed this terminology from PL/1) and passes the address of this variable. So the value of the original variable X1 will never change, even if the FORTRAN subroutine decides to change the parameter :-)

X2 is passed as address, no problem (the var symbol tells Pascal to do call-by-reference, too, which perfectly matches the FORTRAN logic)

I don‘t know exactly what happens to T1 and T2, but I guess, both values are passed by address in the same way. It will be a problem anyway for FORTRAN to do something sensible with the two character arrays.

Please look at the ASSEMBLER source code for PAS2FTN which is shown at the next page. The FORTRAN compiler uses the normal linkage conventions, which is generated by Pascal, too, in this case (this is NOT true, BTW, for Pascal-to-Pascal communications).

```
PAS2FTN  TITLE 'ASSEMBLER SUBROUTINE CALLABLE FROM PASCAL'
*
**************************************************************
*        Pascal prototype =
*
*        procedure PASCAL_TO_FORTRAN ( X1 : INTEGER ;
*                                      var X2 : INTEGER ;
*                                      T1 : CHAR20 ;
*                                      var T2 : CHAR20 ) ;
*
*           EXTERNAL FORTRAN 'PAS2FTN' ;
*
*        because the subroutine is specified as
*        EXTERNAL FORTRAN, all parameters are passed
*        by reference, and the leftmost bit is set
*        on the last parameter address
**************************************************************
*
PAS2FTN  CSECT
         STM   R14,R12,12(R13)
         LR    R11,R15             LOAD BASE REGISTER
         USING PAS2FTN,R11
         LA    R15,SAVEAREA
         ST    R15,8(R13)
         ST    R13,4(R15)
         LR    R13,R15
*
**************************************************************
*        fetch parameter addresses
**************************************************************
*
         LM    R2,R5,0(R1)
*
**************************************************************
*        work on parameters
**************************************************************
*
         LA    R6,8
         ST    R6,0(R2)            will change dummy arg only
         LA    R6,8
         ST    R6,0(R3)            set X2 to 7
*
```

```
         LTR   R5,R5
         BP    R5POS
         MVC   0(20,R5),=CL20'Last Addr. negative'
         B     OK
R5POS    DS    0H
         MVC   0(20,R5),=CL20'Last Addr. positive'
OK       DS    0H
*
**************************************************************
*        exit (return to caller)
**************************************************************
*
EXIT     DS    0H
         L     R13,4(R13)
         LM    R14,R12,12(R13)
         XR    R15,R15
         BR    R14
         EJECT
*
**************************************************************
*        definitions
**************************************************************
*
         DS    0D
SAVEAREA DS    18F
*
**************************************************************
*        REGISTER ASSIGNMENTS
**************************************************************
*
         REGEQU
*
         END
```

Note: External functions in FORTRAN work very much the same way, but they have to put the function result into register zero before returning to the Pascal caller.

See the ASSEMBLER example for an external Pascal function (I decided to implement the same technique there – I did not want to use register 15, because the compiler wants the entry point of called functions staying there for optimization reasons).

External procedures written in ASSEMBLER

If the external procedure or function is written in ASSEMBLER, you have to specify it on the EXTERNAL definition on the Pascal side.

Example:

```
procedure PASCAL_TO_ASSEMBLER ( X1 : INTEGER ; var X2 : INTEGER ; T1 :
                                CHAR20 ; var T2 : CHAR20 ) ;

   EXTERNAL ASSEMBLER 'PAS2ASM' ;

function PAS_TO_ASM_FUNC ( X1 : INTEGER ; X2 : INTEGER ) : INTEGER ;

   EXTERNAL ASSEMBLER 'PAS2AF' ;
```

Again, the alternate name on the EXTERNAL statement is not needed, if the original name of the procedure or function can be used. The compiler translates those names to uppercase, BTW (but not the alternate names inside the quotes).

Pascal generates the same sequence for the parameters as it would do it for a Pascal-to-Pascal communication.

That is:

- by-value parameters are „real“ by-value parameters
  (much like in Mainframe C)
- they are inserted „as values“ into the register 1 based parameter list
- the last parameter (which may be a value, not an address) cannot have the leftmost bit set, of course

but:

- different from Pascal-to-Pascal communication, the compiler generates Standard OS linkage in this case
- function results are expected in register zero (which is also different from Pascal-to-Pascal linkage)

BTW:

when I have some more time, I will try out some ASSEMBLER programs which use the Pascal stack for their local variables and their save areas and probably also use functions of the Pascal runtime … should be no big deal.

Let‘s look again at the Pascal „prototype“ of the external ASSEMBLER procedure:

```
procedure PASCAL_TO_ASSEMBLER ( X1 : INTEGER ; var X2 : INTEGER ; T1 :
                              CHAR20 ; var T2 : CHAR20 ) ;

   EXTERNAL ASSEMBLER 'PAS2ASM' ;
```

this means, that the parameter list of the procedure looks like follows:

- offset 0: X1 (value)
- offset 4: X2 (address)
- offset 8: T1 (character 20, value)
- offset 28: T2 (address of character 20)

the parameter list ends at offset 32.

You have to take this into account, when writing the external procedure in ASSEMBLER. Look here:

```
PAS2ASM  TITLE 'ASSEMBLER SUBROUTINE CALLABLE FROM PASCAL'
*
**************************************************************
*        Pascal prototype =
*
*        procedure PAS2ASM ( X1 : INTEGER ;
*                            var X2 : INTEGER ;
*                            T1 : CHAR20 ;
*                            var T2 : CHAR20 ) ;
*
*           EXTERNAL ASSEMBLER 'PAS2FTN' ;
**************************************************************
*
PAS2ASM  CSECT
         STM   R14,R12,12(R13)
         LR    R11,R15             LOAD BASE REGISTER
         USING PAS2ASM,R11
         LA    R15,SAVEAREA
         ST    R15,8(R13)
         ST    R13,4(R15)
         LR    R13,R15
```

```
*
***************************************************************
*        fetch parameters
***************************************************************
*
         L     R2,0(R1)            = X1 (by value)
         L     R3,4(R1)            = ADDR of X2 (by addr)
         LA    R4,8(R1)            = ADDR of T1 (by value)
         L     R5,28(R1)           = ADDR of T2 (by addr)
*
***************************************************************
*        work on parameters
***************************************************************
*
         LA    R6,7
         ST    R6,0(R3)            set X2 to 7
*
         MVC   0(20,R5),=CL20'String from PAS2ASM'   set T2
*
***************************************************************
*        exit (return to caller)
***************************************************************
*
EXIT     DS    0H
         L     R13,4(R13)
         LM    R14,R12,12(R13)
         XR    R15,R15
         BR    R14
         EJECT
*
***************************************************************
*        definitions
***************************************************************
*
         DS    0D
SAVEAREA DS    18F
*
***************************************************************
*        REGISTER ASSIGNMENTS
***************************************************************
*
         REGEQU
*
         END
```

External functions written in ASSEMBLER

The external function in the sample program has this prototype:

```
function PAS_TO_ASM_FUNC ( X1 : INTEGER ; X2 : INTEGER ) : INTEGER ;

   EXTERNAL ASSEMBLER 'PAS2AF' ;
```

so we simply have to fetch two parameters (integers) from the parameter list (by value) and put the result into register zero. The result in this case is meant to be (X1 + X2) DIV 2.

Look here:

```
PAS2AF   TITLE 'ASSEMBLER FUNCTION CALLABLE FROM PASCAL'
*
**************************************************************
*        Pascal prototype =
*
*        function PAS_TO_ASM_FUNC ( X1 : INTEGER ;
*                                   X2 : INTEGER ) : INTEGER ;
*
*           EXTERNAL ASSEMBLER 'PAS2AF' ;
**************************************************************
*
PAS2AF   CSECT
         STM   R14,R12,12(R13)
         LR    R11,R15             LOAD BASE REGISTER
         USING PAS2AF,R11
         LA    R15,SAVEAREA
         ST    R15,8(R13)
         ST    R13,4(R15)
         LR    R13,R15
*
**************************************************************
*        fetch parameters
**************************************************************
*
         L     R2,0(R1)            = X1 (by value)
         L     R3,4(R1)            = X2 (by value)
```

```
*
***************************************************************
*        work on parameters
***************************************************************
*
         AR    R2,R3
         SRA   R2,1
         LR    R0,R2                function result in R0
*
***************************************************************
*        exit (return to caller)
***************************************************************
*
EXIT     DS    0H
         L     R13,4(R13)
         LM    R14,R15,12(R13)      restore regs
         LM    R1,R12,24(R13)       but omit R0 (function result)
         XR    R15,R15
         BR    R14
         EJECT
*
***************************************************************
*        definitions
***************************************************************
*
         DS    0D
SAVEAREA DS    18F
*
***************************************************************
*        REGISTER ASSIGNMENTS
***************************************************************
*
         REGEQU
*
         END
```

The only interesting problem is: how to change the returning sequence so that the value in register zero is not destroyed on return.

I hope you like the new „external procedure“ features of New Stanford Pascal; please send comments and suggestions to

berndoppolzer@yahoo.com

or

bernd.oppolzer@t-online.de