my file.

SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY

COMPUTING DIVISION

DISTRIBUTED INTERACTIVE COMPUTING NOTE 731

PERQ UNIX IMPLEMENTATION NOTE #35

Issued by
T. Watson

Inter-Language Procedure Calling Conventions

25 November 1982

DISTRIBUTION:
R W Witty
K Robinson
C Prosser
A S Williams
L O Ford
E V C Fielding
T Watson
J C Malone
J M Loveluck
P J Smith
Alan Kinroy
Jim Collis
C J Webb
C Wadsworth
RL Support/PERQ/UNIX Implementation Notes file

1. Introduction

This note describes how to program inter-language communication at the routine level between C, F77 and Pascal on UNIX on the PERQ. C is as defined for V7 unix in [5]. F77 is Fortran as defined for V7 unix in [2] with the changes described in [3]. Pascal is as defined in [1]. The generic term routine is used for Pascal procedures and functions, F77 subroutines and functions and C routines.

Taking the interface between C & F77, C & Pascal and Pascal & F77, in turn, the features of each language that affect the routines defined in one language but called from the other are described.

1.1 Type definition

To make compatible definitions in the three languages it is necessary to know the corresponding representations of C, F77, and Pascal data types. The definitions of these can be found in [6]. There are also include files called Ctype.dfs and mch.dfs (residing in /usr/include/sys in the Accent Unix directory structure on the Perq), defining the basic C types in terms of Pascal, which helps to ensure that the correct types are used.

1.2 Input and output

Pascal, F77 and C each employ a unique method of manipulating files and doing input and output. It is therefore recommended that all I/O be done in the main program body using the routines provided by the language in which the main program is written.

1.3 Fortran argument lists

All arguments of F77 routines are called by reference. This means that an argument in C which corresponds to one in F77 has to be a pointer. Pascal arguments which correspond to ones in F77 have to be var or pointers to the relevant type.

2. The C / F77 interface

To be able to write C routines that call or are called by F77 routines, and vice versa, it is necessary to know the conventions for data representation, argument lists and return values that the compiled code obeys. Also it is necessary to know the way in which F77 common blocks equivalents are defined in C.
Figures 1 and 2 give examples of F77 and C inter-language calling. In the examples in figure 1, i and a are defined as pointers in C, while the other arguments are not. This is because tha name of an array in C is interpreted as a pointer to the array, so need not be explicitly passed as pointers to F77.

The conventions for data representation are given in [6].
Underscores are not now appended to procedure names as they were on the PDP 11/70 Unix.

2.1 Argument lists

In F77 user defined actual parameters are passed by reference, so all corresponding C parameters should be defined as pointers to the relevant type. F77 produces extra arguments for character parameters. These arguments are added on to the end of the argument list in the same order as the arguments that they apply to.

In F77 when an argument of type character is defined an integer typed argument is added to the list. It seems that F77 passes the declared length of the variable, not its current length. Thus if, in figure 1, c were set to "abd", the value of lc would be 10, not 3.

See [3] for other differences between V7 F77 and [2].

2.2 Return values

See section 4.3 of [2] for a description of how F77 subroutines are defined and invoked in C.
Complex results are now passed as structures.
F77 Character-valued functions can not be referenced in C. This is because a C array name is interpreted as a pointer to the array but F77 passes back the array.

2.3 Input and output

It is possible to call C I/O library routines from a F77 main program, but mixing them with F77 I/O routines is undefined.

2.4 Fortran Common blocks and C

A FORTRAN common block, for example,

```
      common/fred/ a, i, x
      real a,x
      integer i
```

may be accessed from a C routine by defining an external structure, thus :

```
      extern struct { float a;
                      int i;
                      float x; } fred;
```

Blank common can be accessed using a structure called _BLNK_. As with FORTRAN *common* statement, the order of the variables in the C *struct* statement defines the order of storage allocation. So the order in the C struct and F77 common block must correspond.

To refer to a variable defined in a F77 common block from C, the form "name.variable" must be used. For example:

```
      fred.a = fred.x + c;
```

Figure 1: F77 calling C

In a F77 routine:

```
      integer i, j(10)
      real a, b(7)
      character *10 c
      character *5 d
       ...
      call jim (i, a, c, j, b, d)
       ...
```

A C routine:

```
jim (i, a, c, j, b, d, lc, ld)
int *i, j[];
float *a, b[];
char c[], d[];
int lc, ld, iii, jjj;
{
*i = j[3] + j[2];
b[2] = (*a) + b[3];
c[iii] = d[jjj];
}
```

Figure 2: C calling F77

In a C routine:

```
extern int jim();
int i, j[10];
float a, b[7];
char c[10];
char d[5];
  ...
jim(&i, &a, c, j, b, d, 10, 5);
  ...
```

A F77 routine:

```
        integer function jim (i,a,c,j,b,d)
        integer i, j(10), iii, jjj
        real a, b(7)
        character *(*)c , d
           ...
        i = j(4) + j(3)
        b(3) = a + b(4)
        c(iii) = d(jjj)
        jim = 5
        end
```

3. C / Pascal interface

In order to call Pascal routines from C, or C routines from Pascal it is necessary to know about the conventions for data representation, routine names, return values and argument lists. Figure 3 gives an example of a C program which calls Pascal and figure 4 gives an example of a Pascal main program which calls C.

3.1 Data representation

Equivalent type mappings for data representation and arguments can be found in [6].

3.2 Routine names

The Pascal compiler converts all routine names and variables to uppercase. So the name of every routine that is defined in C but called in Pascal, or defined in Pascal but referenced in C has to be written in uppercase in the C source file. For example:

```
        in C:
           int FUNN (a,b)
           int a,b;
            {
             return(a);
            }

         in Pascal:
           var  i,j,k:long;
           begin
              i := funn(j,k);
```

The Pascal compiler needs to have all routines and variables defined , see [1], so there has to be a dummy Pascal module created to satisfy these references when C is called from Pascal. This dummy module must have the same module name as the C module (see pcc(1) for how to name the C module), and must export the routines in the same order as they are defined in the C module. The type of each parameter in the three modules must match the corresponding parameter type in the other two modules.
Use the include files /usr/include/sys/Ctype.dfs and /usr/include/sys/mch.dfs to get the correct basic type mappings [4]. See figures 3 and 4 for examples of C and Pascal inter-language calling.

3.3 Return values

All C routines return a value, whose type can easily be translated into the equivalent Pascal function definition or call. The type correspondances are defined in [6].

3.4 Argument list

The arguments of a C routine are called by value so to use a var argument in Pascal the equivalent C one has to be a pointer of the corresponding type. There is a problem here because Pascal uses word pointers and C character pointers are byte pointers. So the character pointers will need to be adjusted, as described in [7] and P_MAKESTRING() in <pstring.h> in [4]. Note that an array name in C is interpreted as a pointer to the array so the PASCAL equivalent must be defined as a pointer, or a var argument.

Figure 3: A main C program which calls routines from a Pascal module

From a C program:

```
 ...
int c[2];
int i,j;
char b[3];
extern int CADD();
 ...
 i = CADD (i,j,c,b);
 ...
```

From the Pascal module:

```
exports
type
{$include mch.dfs}
{$include Ctype.dfs}

type iarr = array [0..1] of C_int;
     carr = packed array [0..2] of C_char;
     ptr_iarr = ^iarr;
     ptr_carr = ^carr;

function cadd( i : C_int;
               j : C_int;
               c : ptr_iarr;
               b : ptr_carr ): C_int;
 ...
private

function cadd( i : C_int;
               j : C_int;
               c : ptr_iarr;
               b : ptr_carr ): C_int;
begin
     b^[3] := 'a';
     cadd := i;
end;
 ...
```

Figure 4: A Pascal main program, C module and
          corresponding Pascal dummy module

Pascal program:

```
program ptest2 (input,output);

imports ctest1 from ctest1;
      ...
type arr = array [0..1] of long;
var i,j,k : long;
    ca : ^arr;
begin
      ..
     k := cadd (i, j, ca);
      ..

end.
```

C module:

```
int CADD (i, j, c)
int i, j;
int c[2];
{
 return(i + j);
}
```

Pascal dummy module:

```
module ctest1;

exports

type
 {$include mch.dfs}
 {$include ctype.dfs}

function cadd(i:C_int;
              j:C_int;
              pc : C_ptr_array ):C_int;
private
```

4. Pascal / F77 interface

To be able to write Pascal routines that call or are called by F77 routines, and vice versa, it is necessary to know the conventions for procedure names, type mappings, argument lists and return values. See figures 5 and 6 for examples of Pascal and F77 inter-language calling.

4.1 Procedure names

All routines defined or referenced in Pascal have their names converted to uppercase, but all F77 programs are converted to lowercase by default. To ensure that the Pascal names are matched correctly with the call or definition in F77, the names of the Pascal routines referenced from F77 have to be in uppercase as well. Details of how to cancel the lowercase conversion in F77 are given in [2].

When calling F77 from Pascal there must be a dummy Pascal module matching the F77 subprogram declarations, to satisfy the Pascal compiler. This dummy module must have the same module name as the F77 subprogram (see f77(1) for how to name a F77 subprogram) and must export the routines in the same order as they are defined in the F77 subprogram. The types of each parameter in the three modules must match the corresponding parameters type in the other two modules or subprograms. Use the include files /usr/include/sys/Ctype.dfs and /usr/include/sys/mch.dfs on Perq Unix to get the correct basic type mappings [4].

4.2 Type mappings

Equivalent type mappings for parameters can be found in [6]. See figure 5 for some examples.

4.3 Argument lists

F77 produces extra variables for _character_ parameters and results. These arguments are added on to the end of the argument list in the same order as the arguments that they apply to.

In F77 when an argument of type _character_ is defined an _integer_ typed argument is added to the list. It seems that F77 passes the declared length of the variable, not its current length. Thus if, in figure 6, _b_ were set to "abd", the value of _lena_ would be 4, not 3.

See [3] for other differences between Perq Unix F77 and [2].

4.4 Return values

All F77 subroutines used in F77 that are defined external to the F77 module are changed into functions that return a F77 _integer_. The equivalent definition in Pascal is a function returning a Pascal _long_.

Figure 5: Pascal calling F77

In the dummy Pascal module:

```
exports

type pa4 = packed array [0..3] of char;
     pa5 = packed array [0..4] of char;
     complex = record
                 x : real;
                 y : real
                end;


function fun(var a : pa4;
             var b : pa5;
             alen : long;
             blen : long
             ) : long;

function fun2 ( var c : complex; var d : real) : complex;

private
```

In Pascal main program:

```
imports ftest10 from ftest10;

var i : long;
    a : pa4;
    b : pa5;
    c,j : complex;
    d : real;

begin
     i := fun(a,b,4,5);
     j := fun2(c , d);
end.
```

In the F77 module:

```
        integer function FUN (a,b)
        character*4 a
        character*5 b
        FUN = 5
        end

        complex function FUN2 (c,d)
        complex c
        real d
        FUN2 = (1.2, 3.4)
        end
```

Figure 6: F77 calling Pascal

f77 program:

```
       ...
      integer I, J, M
      character *4 A, K
       ...
      call PADD (I,J,M)
       ...
      K = PADD2 (I,A,J)

      stop
      end
```

Pascal module:

```
module ptest3;

exports

type arr = packed array [0..3] of char;

function padd (var i : long;
               var j : long;
               var k : long):long;


function padd2 (var i : long;
                var a : arr;
                var j : long;
                lena :long):arr;

private

function padd (var i : long;
               var j : long;
               var k : long):long;

begin
     k := i + j;
end;

function padd2 (var i:long;
                var a : arr;
                var j : long;
                lena :long):arr;

begin
     padd2 := a;
end.
```

REFERENCES

1. PERQ : Pascal Guide, Second Edition (1), International Computers Limited , June 1981.

2. S.I. FELDMAN AND P.J.WEINBERGER, "A Portable Fortran 77 Compiler", Unix Programmer's Manual, Seventh Edition, Volume 2B, Bell Laboratories (January 1979).

3. E. FIELDING, "Perq Unix Implementation Note - Differences between the PDP11 Fortran 77 - and Perq Unix Fortran 77", DCS Note, Rutherford Appleton Laboratory (December 1982). [to be published].

4. C. PROSSER, "Perq Unix Implementation Note # 30 - Changes to UNIX Include Files", DCS Note # 726, Rutherford Appleton Laboratory (November 1982).

5. D.M. RITCHIE, "The C Programming Language - Reference Manual", Unix Programmer's Manual, Seventh Edition, Volume 2A (January 1979).

6. A.S. WILLIAMS, "Perq Unix Implementation Note # 25 - Type mapping between C, Fortran 77, and Pascal", DCS Note # 608, Rutherford Appleton Laboratory (May 1982).

7. A.S. WILLIAMS, "Perq Unix Implementation Note # 22 - Mapping of C data types for Q-code", DCS Note # 605, Rutherford Appleton Laboratory (May 1982).

$