Mixed Language Programming, Fortran 2003 and C

Bo Einarsson

January 19, 2009

Introduction

At a conference in Kyoto 1995 I held a lecture (Einarsson, 1995) about the use of routines in C from Fortran, and the opposite. It was then necessary to make this somewhat differently on systems from various manufacturers.

Since Fortran 2003 has standardized mixing of Fortran and C, I have transformed those examples into the standard, not entirely successfully in the last case.

I have used Sun Fortran 95 8.2 2005/10/13, which contains parts of Fortran 2003. There is a minor error in the Sun implementation, in the standard

```
subroutine sub(f, b)  bind(c)
```

has to be written with a comma in order to avoid compilation error:

```
subroutine sub(f, b) , bind(c)
```

A similar remark is valid for functions.

With this modern method we have to declare in the Fortran routine all interchange variables to be of the type used in C, see (Metcalf et al., 2004, Table 14.1). In order to avoid doing this I find it simpler to check that the C-type and the Fortran-type agree, that is that the two have the same `KIND`. For this purpose I have written the following simple program, which is easy to modify. On Sun I only got a problem with logical (boolean) variables. If such are used you either have to declare them not to be of the standard kind (if they are to be used also in C), or convert them in a communication routine.

```
Program CandF
! Program to test cooperation between C and Fortran
use, intinsic :: iso_c_binding
IF (c_int == KIND(1)) THEN
   WRITE (*,*) 'The KIND of c_int =    ', c_int, &
 ' agrees with INTEGER = ', KIND(1)
ELSE
```

```
   WRITE (*,*) 'The KIND of c_int =    ', c_int, &
 ' does not agree with INTEGER = ', KIND(1)
END IF

IF (c_float == KIND(1.0)) THEN
   WRITE (*,*) 'The KIND of c_float =  ', c_float, &
 ' agrees with REAL = ', KIND(1.0)
ELSE
   WRITE (*,*) 'The KIND of c_float =  ', c_float, &
 ' does not agree with REAL = ', KIND(1.0)
END IF

IF (c_double == KIND(1.0D0)) THEN
   WRITE (*,*) 'The KIND of c_double = ', c_double, &
 ' agrees with DOUBLE PRECISION = ', KIND(1.0D0)
ELSE
   WRITE (*,*) 'The KIND of c_double = ', c_double, &
 ' does not agree with DOUBLE PRECISION = ', KIND(1.0D0)
END IF

IF (c_bool == KIND(.TRUE.)) THEN
   WRITE (*,*) 'The KIND of c_bool =   ', c_bool, &
 ' agrees with LOGICAL = ', KIND(.TRUE.)
ELSE
   WRITE (*,*) 'The KIND of c_bool =   ', c_bool, &
 ' does not agree with LOGICAL = ', KIND(.TRUE.)
END IF

IF (c_char == KIND('A')) THEN
   WRITE (*,*) 'The KIND of c_char =   ', c_char, &
 ' agrees with CHARACTER = ', KIND('A')
ELSE
   WRITE (*,*) 'The KIND of c_char =   ', c_char, &
 ' does not agree with CHARACTER = ', KIND('A')
END IF

IF (c_float_complex == KIND((1.0, 1.0))) THEN
   WRITE (*,*) 'The KIND of c_float_complex =  ', c_float_complex, &
 ' agrees with COMPLEX = ', KIND((1.0, 1.0))
ELSE
   WRITE (*,*) 'The KIND of c_float_complex =  ', c_float_complex, &
 ' does not agree with COMPLEX = ', KIND((1.0, 1.0))
END IF

END Program CandF
```

Running this program on Sun gave

```
 The KIND of c_int =     4  agrees with INTEGER =  4
 The KIND of c_float =   4  agrees with REAL =  4
 The KIND of c_double =  8  agrees with DOUBLE PRECISION =  8
 The KIND of c_bool =    1  does not agree with LOGICAL =  4
 The KIND of c_char =    1  agrees with CHARACTER =  1
 The KIND of c_float_complex =   4  agrees with COMPLEX =  4
```

1 Use of a subroutine and a function written in Fortran from a program in C

We here describe a case where first a program written in Fortran and later a routine in C both call a subroutine and a function in Fortran. Both the main program and the subroutine calls the function.

The main program in Fortran is in the file `f2sam.f` while both the subroutine and the function are in the file `sam.f`.

```
% cat f2sam.f
        program f2sam
        external f
        integer f
        character*7 s
        integer b(3)
        call sam(f, b(2), s)
        write(6,10) b(2), f(real(b(2))), s
   10   format(i5,i5,10x,a7)
        stop
        end program f2sam

% cat sam.f90
        subroutine sam(f, b, s)
        external f
        character*7 s
        integer b, f
        x = 1.3
        s = 'Bo G E '
        b = f(x)
        end subroutine sam
        integer function f(x)
        f=3*x**3
        return
        end function f
```

Running these files give the result `6 648 Bo G E`.

We now wish to be able to call the file `sam.f` with the subroutine `sam` and the function `f` from C *without* any changes in these two! We therefore first write a communication file `c_sam.f90` in Fortran

```
        subroutine c_sam(c_f, b, s) , bind(c)
        use, intrinsic :: iso_c_binding
        implicit none
        external c_f
        character(len=7)  :: s
        integer (c_int)   :: b, c_f
        real (c_float)    :: x
        integer, external :: f
        call sam(f,b,s)
        end subroutine c_sam

        integer (c_int) function c_f(x) , bind(c)
        use, intrinsic :: iso_c_binding
        implicit none
        real(c_float)     :: x
        integer, external :: f
        c_f=f(x)
        return
        end function c_f
```

and then a main program in C which calls these communication routines `c_sam` and `c_f`, who in turn call the "pure" Fortran routines `sam` and `f`.

```
% cat c2sam.c
#include <stdio.h>
#include <math.h>

/*     C call of Fortran routines via communication routines */

       int c_f(float *);
       int c_sam(int (*c_f)(), int *, char *);
int main()
{
       char s[7];
       int b[3];
       float x;

       c_sam(c_f, &b[1], s);
       x = b[1];
       printf(" %d %d %s \n ", b[1], c_f(&x), s);
       return 0;
}
```

Running with

```
f95 -c c_sam.f90 sam.f90
cc -c c2sam.c
cc c2sam.o c_sam.o sam.o
a.out
```

give the result `6 648 Bo G E` as before.

2 Use from Fortran of a matrix assigned values in C

In this case we have a matrix where one element is assigned a value in a C routine, and we wish to use that value in a Fortran program. We of course have to accept the the indices are in reversed order, and that those in C have to be reduced by one compared with those in Fortran. The C program `mlp4.c` looks like

```
#include <stdio.h>
#include <math.h>

/*      Matrix management */

void
p(a,i,j)
int *i, *j, a[3][2];    /* Order 3 2 */
{
a[*j-1][*i-1] = 99;     /* Indices reduced by 1 */
}
```

while the main program in Fortran `mlp3.f90` is

```
      program mlp3
      use, intrinsic :: iso_c_binding
      interface
        subroutine p(a,i,j) , bind(c)
        use iso_c_binding
          integer (c_int) :: a(2,3)
          integer (c_int) :: i, j
        end subroutine p
      end interface
      integer (c_int) ::  a(2,3) ! Order 2 3
      call p(a,1,3)
      write (6,10) a(1,3)
 10   format (1x,i9)
      stop
      end program mlp3
```

These are run with

```
cc  -c mlp4.c
f95 -c mlp3.f90
f95 mlp3.o mlp4.o
./a.out
```

with the result `99`.

3 Use of Fortran `COMMON` in a C program

As in the first example 1 we wish to use a Fortran routine both from Fortran and C, therefore we once again need a communication routine. The `COMMON` block is assigned its values in the routine `sam` in the file `mlp2.f`. In Fortran we thus have a main program in the file `mlp0.f90`

```
program mlp0
implicit none
integer i
real r
common /name/ i, r
call sam
write(*,*) i, r
end program mlp0
```

which together with the routine `sam` in the file `mlp2.f`

```
      subroutine sam()
      common /name/ i, r
      i = 786
      r = 3.2
      return
      end subroutine sam
```

give the result `786 3.2`.

If we **only** wish to use the routine `sam` from C it is quite simple, we just modify `sam` to the file `mlp2.f90`

```
      subroutine sam(), bind(c)
      use, intrinsic    :: iso_c_binding
         integer(c_int) :: i
         real(c_float)  :: r
         common /com/ i, r
         bind(c) :: /com/
      i = 786
      r = 3.2
      return
      end subroutine sam
```

and use the C program in the file `mlp1.c`

```
#include <stdio.h>
#include <math.h>

/*      Use of a COMMON Block from Fortran */

struct  {
   int i; float r;
} com;

main()
{
       sam();
       printf(" %d %f \n ",com.i,com.r);
}
```

and run with

```
cc  -c mlp1.c
f95 -c mlp2.f90
cc     mlp1.o mlp2.o
a.out
```

and obtain the correct result.

Since it is much better if we do not have to change the original Fortran routine `sam` in the file `mlp2.f` to the one in the file `mlp2.f90`, we have to create another communications routine `mlp2_c.f90`. At my tests on Sun it is evident that it does not work with letting the `COMMON` block be used from both C and Fortran (except the Fortran routine in which it is assigned its values). I therefore in the following use the two names `name` and `com` on what is really the same `COMMON` block. Running the program in Fortran can now be done as earlier.

At the use from C you first use the communication routine in the file `mlp2_c.f90`

```
      subroutine sam_c(), bind(c)
      use, intrinsic    :: iso_c_binding
          implicit none
          integer(c_int) :: i
          real(c_float)  :: r
          common  /com/  i, r
          bind(c)        :: /com/
          integer        :: i1
          real           :: r1
          common  /name/ i1, r1
      call sam()
      i = i1
```

```
      r = r1
      return
      end subroutine sam_c
```

Here there are two `COMMON` blocks, one with the name `name` aimed at Fortran and one with the name `com` aimed at C. Since the block is assigned its values in Fortran, those values have to be copied into the variables in the `COMMON` block aimed at C. If the C program in the file `mlp1.c` now is changed to call the Fortran routine `sam_c` (instead of as earlier the Fortran routine `sam`), using the file with the name `mlp5.c`, the program is run with

```
f95 -c mlp2.f
f95 -c mlp2_c.f
cc  -c mlp5.c
f95 mlp5.o mlp2.o mlp2_c.o  (or cc mlp5.o mlp2.o mlp2_c.o)
a.out
```

and gives the correct result. If you wish to assign new values in C for the `COMMON` variables an additional communication routine has to be created.

All the files mentioned here are available for testing at
`http://www.mai.liu.se/~boein/vita/code/`.

References

Einarsson, B. (1995). Mixed Language Programming, Part 4, Mixing ANSI-C with Fortran 77 or Fortran 90; Portability or Transportability? In *Current Directions in Numerical Software and High Performance Computing*, International Workshop, Kyoto, Japan. IFIP WG 2.5. `http://www.nsc.liu.se/~boein/ifip/kyoto/einarsson.html`.

Metcalf, M., Reid, J., and Cohen, M. (2004). *Fortran 95/2003 explained*. Oxford University Press. ISBN 0-19-852693-8.