Marrying FORTRAN to C

M.Valin
CIOB

C FORTRAN interoperability issues

- FORTRAN name mangling
  (compiler specific, may be altered via switches)
- FORTRAN non predictable intrinsic type length
  (may be altered via compiler switches e.g. -i8 -r8 ...)
- Argument passing (by address vs by value)
- Pointer arguments
  (FORTRAN pointers are very complex entities)
- FORTRAN derived types vs C structs
- Hidden arguments (FORTRAN character strings)
- FORTRAN load time libraries and other items
- Proper FORTRAN run time library initialization

ISO_C_BINDING

- FORTRAN 2003 intrinsic MODULE
  USE ISO_C_BINDING
- Provides C type definitions to FORTRAN
- Provides C/FORTRAN pointer interoperability procedures
- Provides argument interoperability
- Provides some derived type and global data interoperability
- BIND(C [, NAME='...']) attribute
  - For subroutines and functions
  - For derived types
  - For global data

Interoperability of intrinsic types

Named constant	C type or types
C_INT	int
C_SHORT	short int
C_LONG	long int
C_LONG_LONG	long long int
C_SIGNED_CHAR	signed char, unsigned char
C_SIZE_T	size_t
C_INT8_T	int8_t
C_INT16_T	int16_t
C_INT32_T	int32_t
C_INT64_T	int64_t
C_INTMAX_T	intmax_t
C_INTPTR_T	intptr_t
C_FLOAT	float
C_DOUBLE	double
C_LONG_DOUBLE	long double
C_FLOAT_COMPLEX	float _Complex
C_DOUBLE_COMPLEX	double _Complex
C_LONG_DOUBLE_COMPLEX	long double _Complex
C_BOOL	_Bool
C_CHAR	char

Inetroperability of pointers

C_LOC(X) is an inquiry function that returns the C address of an object. X is permitted to be
(i) a variable with interoperable type and type parameters that has the TARGET attribute and is either interoperable, an allocated allocatable variable, or a scalar pointer with a target; or
(ii) a nonpolymorpic scalar without length parameters that has the TARGET attribute and is either an allocated allocatable variable, or a scalar pointer with a target.

C_FUNLOC(X) is an inquiry function that returns the C address of a procedure. X is permitted to be a procedure that is interoperable (see Section 5.6) or a pointer associated with such a procedure;

C_ASSOCIATED (C_PTR1[, C_PTR2]) is an inquiry function for object or function pointers. It returns a default logical scalar. It has the value **false** if C_PTR1 is a C **null** pointer or if C_PTR2 is present with a different value; otherwise, it has the value true.

Interoperability of pointers

C_F_POINTER (CPTR, FPTR [, SHAPE])) is a subroutine with arguments

CPTR is a scalar of type C_PTR with INTENT(IN). Its value is the C address of an entity That is interoperable with variables of the type and type parameters of FPTR or Was returned by a call of C_LOC for a variable of the type and type parameters of FPTR. It must not be the C address of a Fortran variable that does not have the TARGET attribute.

FPTR is a pointer that becomes pointer associated with the target of CPTR. If it is an array, its shape is specified by SHAPE.

SHAPE (optional) is a rank-one array of type integer with INTENT(IN). If present, its size is equal to the rank of FPTR. If FPTR is an array, it must be present.

C_F_PROCPOINTER (CPTR, FPTR)) is a subroutine with arguments

CPTR is a scalar of type C_FUNPTR with INTENT(IN). Its value is the C address of a procedure that is interoperable.

FPTR is a procedure pointer that becomes pointer associated with the target of CPTR

Interoperability of derived types

- Type MUST be given the BIND attribute
- Each component
  - MUST have interoperable type
  - MUST NOT be a pointer
  - MUST NOT be allocatable
- example
  - type, bind(c) :: mytype
    integer(C_INT) :: I, J
    real (C_FLOAT) :: S
    end type mytype
  - typedef struct {
    int m,n;
    float r;
    } myctype
  - are interoperable

Interoperability of variables

- SCALAR variable
  - MUST be of interoperable type
  - MUST NOT be a pointer
  - MUST NOT be allocatable
  - character arguments MUST have length 1
- ARRAY variable
  - MUST be of interoperable type
  - MUST have explicit shape or assumed size
- Example
    - INTEGER :: a(1, 3:7, *)
      is interoperable with
      int b[][5][18]
    - CAVEAT: subscripts are REVERSED

Interoperability of procedures

- FORTRAN procedure MUST have
  - explicit interface
  - BIND attribute
  - interoperable arguments
  - interoperable scalar result if function
- FORTRAN procedure SHOULD be given explicit C name
  - BIND(C, NAME='External_Name')
  - default name is not reliable
    (external name should be different if case is ignored)
- FORTRAN procedure MAY have VALUE arguments
  (VALUE attribute for the argument type)
  - must correspond to a non pointer C argument
  - except if it is C_PTR (corresponds to a C pointer)

FORTRAN calling C example

```
int C_Library_Function(void* sendbuf, int sendcount, int
*recvcounts)

USE ISO_C_BINDING
IMPLICIT NONE

INTERFACE
INTEGER (C_INT) FUNCTION C_LIBRARY_FUNCTION &
(SENDBUF, SENDCOUNT, RECVCOUNTS) BIND(C, NAME='C_Library_Function')
TYPE (C_PTR), VALUE :: SENDBUF
INTEGER (C_INT), VALUE :: SENDCOUNT
TYPE (C_PTR), VALUE :: RECVCOUNTS
END FUNCTION C_LIBRARY_FUNCTION
END INTERFACE

INTEGER L
REAL(C_FLOAT), TARGET :: SEND(100)
INTEGER(C_INT) :: SENDCOUNT
INTEGER(C_INT), ALLOCATABLE, TARGET :: RECVCOUNTS(:)
. . .
ALLOCATE(RECVCOUNTS(100))
. . .
L = C_LIBRARY_FUNCTION(C_LOC(SEND), SENDCOUNT,C_LOC(RECVCOUNTS))
. . .
```

FORTRAN to C (a real example)

```
program rrbx2ppm   ! FORTRAN and C both writing to stdout
use iso_c_binding
implicit none

interface
  integer(C_INT) function c_write(fd,buf,count) BIND(C,name='write')
  use iso_c_binding
  integer (C_INT),  value ::  fd,  count
  type (C_PTR), value :: buf
  end function c_write
end interface
 . . . .
integer nbr,nr,lcp,ncp,i
integer, pointer, dimension(:,:) :: arr
byte, pointer, dimension(:,:) :: rgb
 . . . .
open(unit=1,file=trim(infile),action='READ',form='UNFORMATTED')
read(1)nbr,nr,lcp,ncp
allocate(arr(lcp,max(3,ncp)))  ! allocate a raster (ncp color planes)
allocate(rgb(3,nbr))
write(6,'(A,/,A,A,/,2I8,/,I8)')'P6','#',trim(comment),nbr,nr,255
call flush(6)
 . . . .
  i=c_write(1,c_loc(   rgb(1,1)  ) , nbr*3)
 . . . .
close(1)
 . . . .
```

C main() using FORTRAN libraries

- 2 itchy problems
  - Properly initializing the FORTRAN runtime library
  - Locating all the ingredients necessary to successfully build an executable file (a.out)
- Use the FORTRAN compiler to build the executable
  - It knows where all the relevant libraries and other necessary items are located
- Use a FORTRAN main program calling the C main
  - A FORTRAN program knows how to initialize the FORTRAN runtime properly

C main() using FORTRAN libraries

- 1 potential problem left
  - My C main program uses argc / argv
- Use fmain2cmain in a dummy FORTRAN main
  - program bidon
  - USE ISO_C_BINDING
  - interface
  - subroutine my_c_main() BIND(C,NAME='my_c_main')
  - end subroutine my_c_main
  - end interface
  - call fmain2cmain(my_c_main)
  - stop
  - end
- Rename main(int argc,char**argv) to my_c_main(int argc,char**argv)

C main() example

```
program bidon
use iso_c_binding
interface
 subroutine my_c_main() BIND(C,NAME='my_c_main')
 end subroutine my_c_main
end interface
call fmain2cmain(my_c_main)
stop
end
my_c_main(int argc,char**argv)
{
while(argc--){
  printf(" %s \n",*argv++);
}
printf("Hello World\n");
}
```

r.compile -src my_f_main.f90 -o a.out -obj my_c_main.o -librmn rmnbeta_011

./a.out
./a.out
Hello World

./a.out a b c d
./a.out
a
b
c
d
Hello World