Le module : **ISO_C_BINDING**

Les buts poursuivis sont de : Pouvoir appeler une fonction C à partir d'un programme Fortran
(exemples des appels systemes, graphismes, …)
Pouvoir créer en Fortran des procédures utilisables dans une fonction C
(exemples de procédures de calcul, …)
Pouvoir utiliser des variables globales entres procédures C et Fortran, et
Rendre compatibles les types dérivés ou les énumérations entre les deux langages.

```
USE, INTRINSIC :: ISO_C_BINDING
USE, INTRINSIC :: ISO_C_BINDING , ONLY : .....          (mieux !)
```

Les types « communs » entre C et Fortran :

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_INT_LEAST8_T	int_least8_t
C_INT_LEAST16_T	int_least16_t
C_INT_LEAST32_T	int_least32_t
C_INT_LEAST64_T	int least64_t
C_INT_FAST8_T	int_fast8_t
C_INT_FAST16_T	int_fast16_t
C_INT_FAST32_T	int_fast32_t
C_INT_FAST64_T	int_fast64_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

Caractères « communs » entre C et Fortran :

C_NULL_CHAR	null character	CHAR(0)	'\0'
C_ALERT	alert	ACHAR(7)	'\a'
C_BACKSPACE	backspace	ACHAR(8)	'\b'
C_FORM_FEED	form feed	ACHAR(12)	'\f'
C_NEW_LINE	new line	ACHAR(10)	'\n'
C_CARRIAGE_RETURN	carriage return	ACHAR(13)	'\r'
C_HORIZONTAL_TAB	horizontal tab	ACHAR(9)	'\t'
C_VERTICAL_TAB	vertical tab	ACHAR(11)	'\v'

NOTE 15.3 : The value of NEW LINE(C NEW LINE) is C NEW LINE (13.7.85).

Les types d'interopérabilité entre C et Fortran :

Pour des variables interopérables :
C_PRT dont une valeur particulière (!) est : C_NULL_PTR .

Pour des fonctions interopérables :
C_FUNPTR dont une valeur particulière (!) est : C_NULL_FUNPTR

Procédures d'interopérabilité entre C et Fortran :

```
Type (C_PRT) Function C_LOC ( X )

Type (C_FUNPRT) Function C_FUNLOC ( X )
```

Subroutine `C_F_POINTER ( CPRT , FPTR [ , SHAPE ] )`

Subroutine `C_F_PROCPOINTER ( CPRT , FPTR )`

(logical function) `C_ASSOCIATED ( CPRT )`
faux ssi `CPRT` est `NULL` !

(logical function) `C_ASSOCIATED ( CPRT1 , CPTR2 )`

Parmi les mots clés (très) utilisés (voir les exemples), on trouvera :

```
BIND(C)
BIND(C,Name=<name>)
INTERFACE …… END INTERFACE
IMPORT ……
VALUE
```

Remarques : L'interopérabilité ne doit pas interférer avec les mécanismes de Fortran et/ou de C en ce qui concerne :
- Les allocartions/désallocations (`ALLOCATE` , `DEALLOCATE` , `malloc` , `free`) ;
- Les entrées/sorties (pourvues qu'elles n'opérent pas sur les mêmes fichiers !) ;
- Les entrées/sorties standards e.g. du type `stdin` , `stdout` , `stderr` ;
- L'initialisation des données internes au programme.

EXEMPLES

```
typedef struct
{
   int m , n ;
   float r ;
} myctype
```

```
USE, INTRINSIC :: ISO_C_BINDING
TYPE, BIND(C) :: MYFTYPE
   INTEGER(C_INT) :: I , J
   REAL(C_FLOAT) :: S
END TYPE MYFTYPE
```

```
INTERFACE
   FUNCTION FUNC(I, J, K, L, M), BIND(C)
      USE, INTRINSIC :: ISO_C_BINDING
      INTEGER(C_SHORT) :: FUNC
      INTEGER(C_INT), VALUE :: I
      REAL(C_DOUBLE) :: J
      INTEGER(C_INT) :: K, L(10)
      TYPE(C_PTR), VALUE :: M
   END FUNCTION FUNC
END INTERFACE
```

is interoperable with the C function prototype

```
short func (int i, double * j, int * k, int l[10], void * m) ;
```

the binding label of C_SUB is "c_sub", and the binding label of C_FUNC is "C_funC".

```
SUBROUTINE C_SUB, BIND(C)
...
END SUBROUTINE C_SUB

INTEGER(C_INT) FUNCTION C_FUNC(), BIND(C, NAME="C_funC")
   USE, INTRINSIC :: ISO_C_BINDING
   ...
END FUNCTION C_FUNC
```

```
void copy(char in[], char out[]) ;
```

Such a function may be invoked from Fortran as follows :

```
USE, INTRINSIC :: ISO_C_BINDING,                    &
                     ONLY: C_CHAR, C_NULL_CHAR
INTERFACE
   SUBROUTINE COPY(IN, OUT), BIND(C)
      IMPORT C_CHAR
      CHARACTER(KIND=C_CHAR), DIMENSION(*) :: IN, OUT
   END SUBROUTINE COPY
END INTERFACE

CHARACTER(LEN=10, KIND=C_CHAR) ::                                    &
              DIGIT_STRING = C_CHAR_'123456789' // C_NULL_CHAR
CHARACTER(KIND=C_CHAR) :: DIGIT_ARR(10)

CALL COPY(DIGIT_STRING, DIGIT_ARR)
PRINT '(1X, A1)', DIGIT_ARR(1:9)
END
```

Example of Fortran calling C

C Function Prototype:

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

Fortran Modules:

```
MODULE FTN_C_1
   USE, INTRINSIC :: ISO_C_BINDING
END MODULE FTN_C_1

MODULE FTN_C_2
   INTERFACE
      INTEGER (C_INT) FUNCTION C_LIBRARY_FUNCTION                    &
                     (SENDBUF, SENDCOUNT, RECVCOUNTS),               &
                     BIND(C, NAME='C_Library_Function')
         USE FTN_C_1
         IMPLICIT NONE
         TYPE (C_PTR), VALUE :: SENDBUF
         INTEGER (C_INT), VALUE :: SENDCOUNT
         TYPE (C_PTR), VALUE :: RECVCOUNTS
      END FUNCTION C_LIBRARY_FUNCTION
   END INTERFACE
END MODULE FTN_C_2
```

Fortran Calling Sequence:

```
USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_INT, C_FLOAT, C_LOC
USE FTN_C_2
...
REAL (C_FLOAT), TARGET :: SEND(100)
INTEGER (C_INT) :: SENDCOUNT
INTEGER (C_INT), ALLOCATABLE, TARGET :: RECVCOUNTS(100)
...
ALLOCATE( RECVCOUNTS(100) )
...
CALL C_LIBRARY_FUNCTION(C_LOC(SEND), SENDCOUNT, &
C_LOC(RECVCOUNTS))
...
```

Example of C calling Fortran

Fortran Code:

```
SUBROUTINE SIMULATION(ALPHA, BETA, GAMMA, DELTA, ARRAYS), BIND(C)
   USE, INTRINSIC :: ISO_C_BINDING
   IMPLICIT NONE
   INTEGER (C_LONG), VALUE                 :: ALPHA
   REAL (C_DOUBLE), INTENT(INOUT)          :: BETA
   INTEGER (C_LONG), INTENT(OUT)           :: GAMMA
   REAL (C_DOUBLE),DIMENSION(*),INTENT(IN) :: DELTA
   TYPE, BIND(C) :: PASS
      INTEGER (C_INT) :: LENC, LENF
      TYPE (C_PTR)         :: C, F
   END TYPE PASS
   TYPE (PASS), INTENT(INOUT) :: ARRAYS
   REAL (C_FLOAT), ALLOCATABLE, TARGET, SAVE :: ETA(:)
   REAL (C_FLOAT), POINTER :: C_ARRAY(:)
   ...
   ! Associate C_ARRAY with an array allocated in C
   CALL C_F_POINTER (ARRAYS%C, C_ARRAY, (/ARRAYS%LENC/) )
   ...
   ! Allocate an array and make it available in C
   ARRAYS%LENF = 100
   ALLOCATE (ETA(ARRAYS%LENF))
   ARRAYS%F = C_LOC(ETA)
   ...
END SUBROUTINE SIMULATION
```

C Struct Declaration

```
struct pass {int lenc, lenf; float* f, c};
```

C Function Prototype:

```
void simulation(long alpha, double *beta, long *gamma,
         double delta[], pass *arrays);
```

C Calling Sequence:

```
simulation(alpha, &beta, &gamma, delta, &arrays);
```

Example of calling C functions with non-interoperable data

The C prototype of this procedure is

```
void ProcessBuffer(void *buffer, int n_bytes);
```

with the corresponding Fortran interface

```
USE, INTRINSIC :: ISO_C_BINDING

INTERFACE
   SUBROUTINE PROCESS_BUFFER(BUFFER,N_BYTES), BIND(C,NAME="ProcessBuffer")
      IMPORT :: C_PTR, C_INT
      TYPE(C_PTR), VALUE :: BUFFER ! The ``C address'' of the array buffer
      INTEGER(C_INT), VALUE :: N_BYTES ! Number of bytes in buffer
   END SUBROUTINE PROCESS_BUFFER
END INTERFACE
```

This may be done using C LOC ...:

```
REAL(R_QUAD), DIMENSION(:), ALLOCATABLE, TARGET :: QUAD_ARRAY
...
CALL PROCESS_BUFFER(C_LOC(QUAD_ARRAY), INT(16*SIZE(QUAD_ARRAY),C_INT))
   ! One quad real takes 16 bytes on this processor
```

Example of opaque communication between C and Fortran

```
USE, INTRINSIC :: ISO_C_BINDING
   ! Assume this code is inside a module

TYPE RANDOM_STREAM
   ! A (uniform) random number generator (URNG)
CONTAINS
   PROCEDURE(RANDOM_UNIFORM), DEFERRED, PASS(STREAM) :: NEXT
   ! Generates the next number from the stream
END TYPE RANDOM_STREAM

ABSTRACT INTERFACE
   ! Abstract interface of Fortran URNG
   SUBROUTINE RANDOM_UNIFORM(STREAM, NUMBER)
       IMPORT :: RANDOM_STREAM, C_DOUBLE
       CLASS(RANDOM_STREAM), INTENT(INOUT) :: STREAM
       REAL(C_DOUBLE), INTENT(OUT) :: NUMBER
   END SUBROUTINE RANDOM_UNIFORM
END INTERFACE

TYPE :: URNG_STATE ! No BIND(C), as this type is not interoperable
   CLASS(RANDOM_STREAM), ALLOCATABLE :: STREAM
END TYPE URNG_STATE

! Initialize a uniform random number generator:
SUBROUTINE INITIALIZE_URNG(STATE_HANDLE, METHOD), &
                BIND(C, NAME="InitializeURNG")
   TYPE(C_PTR), INTENT(OUT) :: STATE_HANDLE
      ! An opaque handle for the URNG
   CHARACTER(C_CHAR), DIMENSION(*), INTENT(IN) :: METHOD
      ! The algorithm to be used

   TYPE(URNG_STATE), POINTER :: STATE
      ! An actual URNG object

   ALLOCATE(STATE)
      ! There needs to be a corresponding finalization
      ! procedure to avoid memory leaks, not shown in this example
      ! Allocate STATE%STREAM with a dynamic type depending on METHOD
    ...
    STATE_HANDLE=C_LOC(STATE)
      ! Obtain an opaque handle to return to C
END SUBROUTINE INITIALIZE_URNG

! Generate a random number:
SUBROUTINE GENERATE_UNIFORM(STATE_HANDLE, NUMBER), &
              BIND(C, NAME="GenerateUniform")
   TYPE(C_PTR), INTENT(IN), VALUE :: STATE_HANDLE
      ! An opaque handle: Obtained via a call to INITIALIZE_URNG
   REAL(C_DOUBLE), INTENT(OUT) :: NUMBER

   TYPE(URNG_STATE), POINTER :: STATE
      ! A pointer to the actual URNG

   CALL C_F_POINTER(CPTR=STATE_HANDLE, FPTR=STATE)
      ! Convert the opaque handle into a usable pointer
   CALL STATE%STREAM%NEXT(NUMBER)
      ! Use the type-bound procedure NEXT to generate NUMBER
END SUBROUTINE GENERATE_UNIFORM
```