Fortran C
Breve introduzione

Edmondo Giovannozzi

Interfaccia Fortran C

- Il Fortran 2003 ha definito un modulo intrinsico che fornisce costanti e funzioni per interfacciarsi con il C (ISO_C_BINDING).
- Nel Fortran 2008 ulteriori aggiunte al modulo ISO_C_BINDING.
- Ulteriormente, nel 2012, è stato prodotta una specifica tecnica: "TS 29113 Further Interoperability of Fortran with C".
- Questa verrà incorporata nel nuovo standard Fortran che dovrebbe essere pubblicato nel 2018.

C

- Il C è una sorta di lingua franca.
- Molti linguaggi definiscon una interfaccia vero il C:
  - C++ con extern("C")
  - Matlab con loadlibrary
  - Python con "ctypes"
- Tipicamente serve uno strato di interfaccia:
  - Ad esempio se da Python (con ctypes) o Matlab chiamo una routine in Fortran mi serve una routine che espone una interfaccia C che a sua volta chiama la routine Fortran che mi interessa.
  - Se invece da Fortran voglio chiamare una routine in C mi serve un modulo che dichiara le interfacce delle routine C. E se necessario delle routine che mi possono semplificare la chiamata.
- Non presenterò la parte legata al TS 29113.
- Non sarò esaustivo.

Sommario

- Fortran breve introduzione.
  - Passaggio di argomenti ad una funzione, tipi definiti dall'utente, puntatori.
- C breve introduzione
  - Variabili, puntatori e memoria dinamica.
- Chiamata di una funzione C da Fortran, e viceversa
- Gestione delle stringhe e storage association.
- Uso delle callback in una ipotetica libreria C.

Argomenti ed attributo Value


```
module passing_args_01_mod
implicit none

contains
    subroutine pass_scalar(a, b)
        integer :: a
        integer, value :: b
        a = a + 10
        b = b + 10
        print *,'Nella subroutine', a, b
    end subroutine
end module
!---------------------------------------
program passing_args
use passing_args_01_mod
implicit none
    integer :: a, b

    a = 1
    b = 2

    call pass_scalar(a, b)

    print *,'Nel Programma   ', a, b
end program
```

```
Nella subroutine     11      12
Nel Programma        11       2
```

Un argomento dummy con l'attributo value può essere cambiato all'intero di una funzione senza che questo cambiamento si rifletta nel chiamante.

F77 Array arguments


```
module passing_vectors_a_mod
implicit none
contains
    subroutine array_arguments(a, b, c, n)
        integer :: n
        integer :: a(*)      ! assumed size
        integer :: b(3)      ! explicit shape
        integer :: c(n)      ! explicit shape

        a(1:2) =  1
        b = 2
        c = 3
    end subroutine
end module
!------------------------------
program passing_vectors_a
use passing_vectors_a_mod
implicit none

    integer :: a(2), b(3), c(4)

    call array_arguments(a, b, c, size(c))
    print *,'a', a
    print *,'b', b
    print *,'c', c
end program
```

Questo modo di passare gli array corrisponde al modo usato anche nel Fortran 77.

```
 a        1        1
 b        2        2        2
 c        3        3        3        3
```

Assumed shape e allocatable


```
module passing_vectors_b_mod
implicit none
contains
    subroutine array_arguments(a, b)
        integer :: a(:)                ! assumed shape
        integer,allocatable :: b(:) ! allocatable

        a =  size(a)

        allocate(b(2))
        b = 5
    end subroutine
end module
!-----------------------------------
program passing_vectors_b
use passing_vectors_b_mod
implicit none

    integer :: a(3)
    integer, allocatable :: b(:)

    call array_arguments(a, b)
    print *,'a', a
    print *,'b', b

end program
```

Gli array assumed shape sono stati introdotti con il Fortran 90.

Mentre gli allocatable come argomenti di una procedura sono stati introdotti nel 2001.

Con il Fortran 2003 gli allocatable possono essere anche variabili scalari.

```
a           3           3           3
b           5           5
```

Fortran pointers


```
program fortran_pointers
implicit none

    integer,target :: a(3)
    integer,target :: b(5)
    integer,pointer :: c_ptr(:)
    integer :: i

    a = [(i*10,i=1,3)]
    b = [(i,i=1,5)]

    c_ptr => a(2:3)
    print *,'C',c_ptr

    c_ptr => b(2:4)
    print *,'b prima', b

    c_ptr = 100
    print *,'b dopo ', b
    print *,'Size c_ptr', size(c_ptr)

end program
```

```
C           20           30
 b prima      1      2      3      4      5
 b dopo       1    100    100    100      5
 Size c_ptr           3
```

Con il Fortran 2003 i puntatori possono essere associati anche con variabili scalari.

User Defined Types


```
module user_defined_type_mod
implicit none
    type t_point
        real :: x
        real :: y
    end type
contains
    function norma(p) result(r)
        type(t_point) :: p
        real :: r
        r = sqrt(p%x**2 + p%y**2)
    end function
end module
!-------------------------------------
program user_defined_type
use user_defined_type_mod
implicit none

    type(t_point) :: p

    p = t_point(0.3, 0.4)
    print *,'P:', p
    print *,'Norma:',norma(p)
end program
```

```
 P:   0.30000001       0.40000001
Norma:   0.50000000
```

I tipi definiti dall'utente sonostati introdotti nel fortran 90.

Nel 2001 possono avere componenti di tipo allocatable.

Con il fortran 2003 sono diventati la base per la programmazione orientata agli oggetti.

Fortran ISO_C_BINDING


```
program varainfo
use iso_c_binding
implicit none

    real(c_float) :: singola
    real(c_double) :: doppia
    integer(c_int) :: intero
    integer(c_size_t) :: size_t

    print *,'c_float ', c_float
    print *,'c_double', c_double
    print *,'c_int   ', c_int
    print *,'c_size_t', c_size_t
    print *

    print *,c_sizeof(singola), precision(singola)
    print *,c_sizeof(doppia), precision(doppia)
    print *,c_sizeof(intero), huge(intero)
    print *,c_sizeof(size_t), huge(size_t)

end program
```

```
c_float              4
c_double             8
c_int                4
c_size_t             8

           4           6
           8          15
           4  2147483647
           8  9223372036854775807
```

Tipi principali in C


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

int main(void) {
    float singola;
    double doppia;
    int intero;
    size_t size;

    printf("Size singola %d\n",sizeof(singola));
    printf("Size doppia  %d\n",sizeof(doppia));
    printf("Size intero  %d\n",sizeof(intero));
    printf("Size size_t  %d\n",sizeof(size));

    printf("precisione singola %d\n",FLT_DIG);
    printf("precisione doppia %d\n",DBL_DIG);
}
```

```
Size singola 4
Size doppia  8
Size intero  4
Size size_t  8
precision singola 6
precision doppia 15
```

Puntatori C

Dichiarazioni

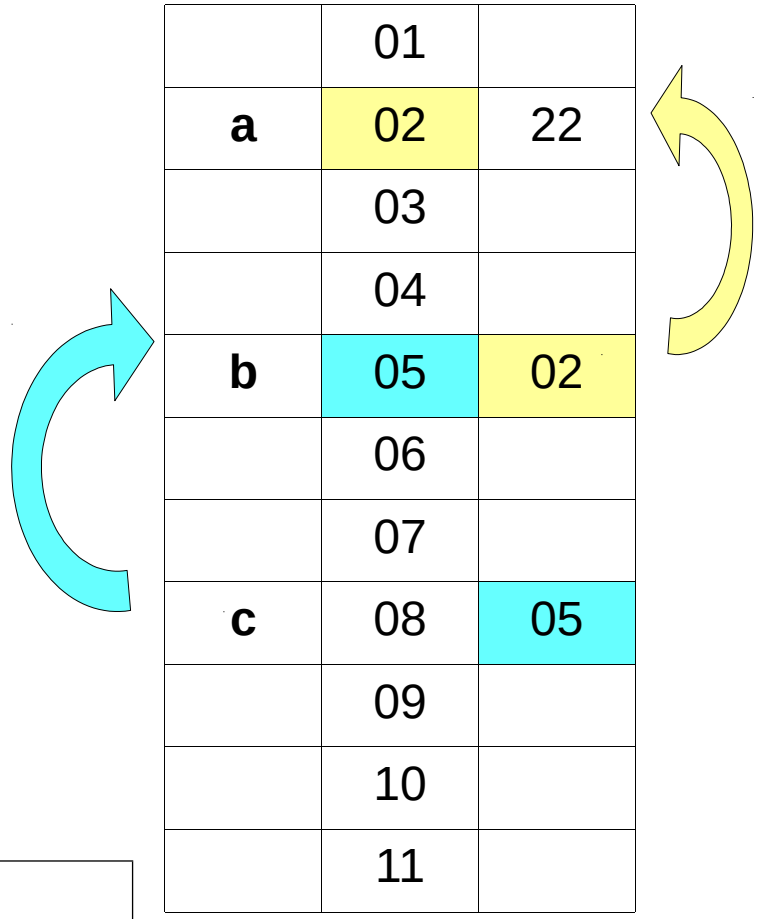
`int  a;`	**a** variabile intera
`int * b;`	**b** puntatore a variabile intera
`int ** c;`	**c** puntatore a puntatore a variabile intera

Assegnazione

`b = & a;`	assegno a **b** l'indirizzo di **a**
`c = & b;`	assegno a **c** l'indirizzo di **b**

Riferimento

`* b;`	recupero il valore all'indirizzo puntato da **b**, ovvero quello che sta in **a**.
`** c;`	doppio riferimento, prima trovo dove punta **c** (a **b**) e poi dove punta **b** (ad **a**) e quindi ne prendo il valore



	01	
a	02	22
	03	
	04	
b	05	02
	06	
	07	
c	08	05
	09	
	10	
	11	

Puntatori C (2)


```c
#include <stdio.h>

int main(void) {
    int a;
    int * b;
    int **c;

    a = 22;

    b = & a;
    c = & b;

    printf("a:   %d\n", a);
    printf("*b:  %d\n", *b);
    printf("**c: %d\n", **c);

    printf("b:  %p  %p\n",  b, &a);
    printf("c:  %p  %p\n",  c, &b);
    printf("*c: %p\n",  *c);

}
```

```
a:   22
*b:  22
**c: 22
b:  0x7ffe2a9e4e04  0x7ffe2a9e4e04
c:  0x7ffe2a9e4df8  0x7ffe2a9e4df8
*c: 0x7ffe2a9e4e04
```

Argomenti per valore o puntatore


```
#include <stdio.h>

int sommacambia(int aa, int *bb) {
    aa = aa + 10;
    *bb = *bb + 100;
    return aa + *bb;
}

int main(void) {
    int a, b, c;

    a = 1;
    b = 2;
    c = sommacambia(a, &b);

    printf("a: %d \n", a);
    printf("b: %d \n", b);
    printf("c: %d \n", c);
}
```

```
a: 1
b: 102
c: 113
```

Nel C le funzioni passano gli argomenti sempre per valore. Quindi se uno vuole passare un puntatore lo deve fare esplicitamente.

Memoria dinamica C


```c
#include <stdio.h>
#include <stdlib.h>

void carray(int a[], int n, int **b) {

    *b = (int *) calloc(n, sizeof(int));

    for (int i=0; i<n; i++) (*b)[i] = a[i] + 10;
}

void freearr(int **b) {
    free(*b);
}
int main(void) {
    int a[5];
    int *b;

    for (int i=0; i<5; i++) a[i] = i;

    carray(a, 5, &b);
    for (int i=0; i<5; i++) {
        printf("a: %d   b: %d\n",a[i], b[i]);
    }
    freearr(&b);
}
```

```
a: 0   b: 10
a: 1   b: 11
a: 2   b: 12
a: 3   b: 13
a: 4   b: 14
```

Fortran chiama C


```
module clibnostra_mod
use iso_c_binding
implicit none

interface
! int sommacambia(int aa, int *bb)
    function sommacambia(aa, bb) result(r) bind(C,name="sommaCambia")
        import C_INT
        integer(C_INT),value :: aa
        integer(C_INT) :: bb
        integer(C_INT) :: r
    end function
! void carray(int a[], int n, int **b)
    subroutine carray(a, n, b) bind(C,name="cArray")
        import C_INT, C_PTR
        integer(C_INT) :: a(*)
        integer(C_INT),value :: n
        type(C_PTR) :: b
    end subroutine
! void freearr(int **b)
    subroutine freearr(b) bind(C,name="freeArr")
        import C_PTR
        type(C_PTR) :: b
    end subroutine
end interface

end module
```

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

int sommaCambia(int aa, int *bb) {
    aa = aa + 10;
    *bb = *bb + 100;
    return aa + *bb;
}

void cArray(int a[], int n, int **b) {
    *b = (int *) calloc(n, sizeof(int));
    for (int i=0; i<n; i++) (*b)[i] = a[i] + 10;
}

void freeArr(int **b) {
    free(*b);
}
```

Fortran C scalari


```
program fortran_C_scalars
use  clibnostra_mod
implicit none

    integer :: a, b, c

    a = 1
    b = 2
    c = sommacambia(a, b)

    print *,'A:',a
    print *,'B:',b
    print *,'C:',c


end program
```

```
 A:           1
 B:         102
 C:         113
```

```
int sommaCambia(int aa, int *bb) {
    aa = aa + 10;
    *bb = *bb + 100;
    return aa + *bb;
}
```

Fortran C array


```
program fortran_C_arrays
use  clibnostra_mod
implicit none

    integer :: a(5)
    integer :: i
    type(C_PTR) :: p
    integer, pointer :: b(:)

    a = [(i,i=1,5)]

    call carray(a, size(a), p)
    call c_f_pointer(p, b, shape(a))

    print "(A,5I4)",'A', a
    print "(A,5I4)",'B', b

    call freearr(p)


end program
```

```
A   1   2   3   4   5
B  11  12  13  14  15
```

```
void cArray(int a[], int n, int **b) {
    *b = (int *) calloc(n, sizeof(int));
    for (int i=0; i<n; i++) (*b)[i] = a[i] + 10;
}

void freeArr(int **b) {
    free(*b);
}
```

Calling Fortran from C


```
module fromCtoFrotran_mod
use iso_c_binding
implicit none
contains

! int sommaCambiaFortran(int aa, int *bb, double vv[], int n);
    function sommaCambia(aa, bb, vv, n) result(r) bind(C,name="sommaCambiaFortran")
        integer(C_INT),value :: aa
        integer(C_INT) :: bb
        real(C_DOUBLE) :: vv(*)
        integer(C_INT),value :: n
        integer(C_INT) :: r

        aa = aa + 10;
        bb = bb + 100;
        r = aa + bb;

        vv(1:n) = sqrt(vv(1:n)) + r

    end function
end module
```

```
 1
 102
v(0): 113.000000
v(1): 114.000000
v(2): 114.414214
v(3): 114.732051
v(4): 115.000000
```

```
#include <stdio.h>

int sommaCambiaFortran(int aa, int *bb,
double vv[], int n);

int main(void){
    int a,b;
    double v[10];
    int n;
    n = 10;
    a = 1;
    b = 2;

    for(int i=0;i<n; i++) v[i] = i;
    sommaCambiaFortran(a, &b, v, n);

    printf(" %d \n",a);
    printf(" %d \n",b);
    for(int i=0;i<n; i++) {
        printf("v(%d): %f \n",i,v[i]);
    }
}
```

ISO_C_BINDING

Principali tipi		
Tipo Fortran	kind	C type
INTEGER	C_INT	int
INTEGER	C_SHORT	short int
INTEGER	C_LONG	long int
INTEGER	C_SIZE_T	size_t
REAL	C_FLOAT	float
REAL	C_DOUBLE	double
CHARACTER	C_CHAR	char

tipi derivati come argomenti di funzioni		
Fortran	C	Passaggio
integer(C_INT), value ::	int	intero
integer(C_INT) ::	int *	puntatore ad intero
type(C_PTR), value ::	void *	puntatore generico
type(C_PTR) ::	void **	puntatore a puntatore generico

Recuperare puntatori

Puntatori (come variabile in generale)		
Fortran	C	Passaggio
type(C_PTR)	void *	puntatore a dati
type(C_FUNPTR) ::	void(*)(void)	puntatore a procedura

Funzioni per gestire i puntatori in Fortran	
procedura	Nota
PX = C_LOC(X)	recupera PX il puntatore C di X
PX = C_FUNLOC(X)	PX puntatore C alla procedura X
C_ASSOCIATED(PX)	controlla se il puntatore C PX è nullo. Con due argomenti controlla se puntano allo stesso oggetto
C_F_POINTER(CPTR,FPTR,[shape])	Trasforma un puntatore C in un puntatore Fortran.
C_F_PROCPOINTER(CPTR,FPTR)	Trasforma il puntatore C di una procedura C in un puntatore ad una procedura Fortran.

C_LOC ⟺ C_F_POINTER

C_FUNLOC ⟺ C_F_PROCPOINTER

In generale si può usare C_LOC o C_FUNLOC con tipi e procedure non interoperabili (per le quali non esiste un corrispondente C).
E' vietato però per variabili polimorfiche.

Fortran string


```
module cstring_fortran_mod
use iso_c_binding
implicit none

interface
! void print_string(char * str)
    subroutine print_string(str) bind(c,name="print_string")
        import C_CHAR
        implicit none
        character(kind=C_CHAR) :: str(*)

    end subroutine
end interface
end module
!-----------------------------------------------
program cstring_fortran
use cstring_fortran_mod
use iso_c_binding
implicit none

    call print_string("Ciao Come stai?"//C_NULL_CHAR)

end program
```

```
#include <stdio.h>
void print_string(char * str){
    printf("string: %s\n",str);
}
```

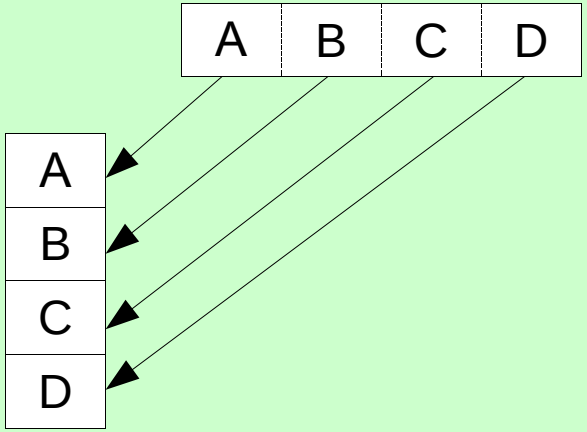
string: Ciao Come stai?

Storage association

Permette di avere delle associazioni in base a come sono meorizzati gli array in memoria. In particolare riguardo agli array character(kind=C_CHAR)

Argomento attuale (nella chiamata):
CHARACTER(LEN=4,kind=C_CHAR) CH

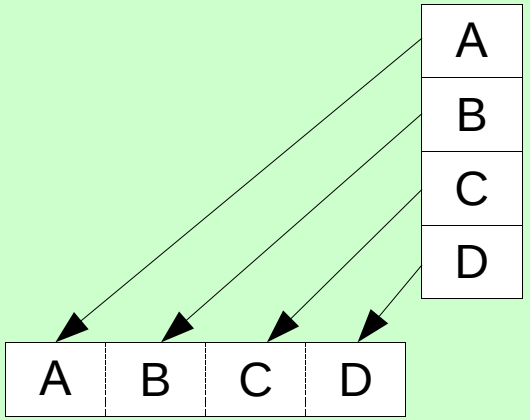
Argomento dummy (nella routine):
CHARACTER(kind=C_CHAR) CH(4)



Argomento attuale (nella chiamata):
CHARACTER(kind=C_CHAR) CH(4)

Argomento dummy (nella routine):
CHARACTER(len=4,kind=C_CHAR) CH**(1)**

Attenzione nella subroutine ho un array



Passing C string to Fortran


```
module fortran_string_mod
implicit none

contains
    subroutine fortran_string(ch)
        character(*) :: ch
        print *,'Fortran :>',ch,'<'
        print *,'Fortran len(ch):', len(ch)
        ch = 'Tutto bene'
    end subroutine fortran_string
!-----------------------------------------
    subroutine fortran_string_c(ch,n) bind(C)
        use iso_c_binding
        integer(C_INT),value :: n
        character(kind=C_CHAR) :: ch(n)

        call internal_sub(ch)

    contains
        subroutine internal_sub(x_ch)
            character(kind=C_CHAR,len=n) :: x_ch(1)

            call fortran_string(x_ch(1))

        end subroutine internal_sub
    end subroutine
end module
```

```
#include <stdio.h>
#include <string.h>
void fortran_string_c(char ch[], int n);

int main(void) {
    char ch[] = "Ciao come stai?";
    int n;
    n = strlen(ch);

    printf("Len: %d\n",n);
    printf("%s\n",ch);

    fortran_string_c(ch, n);

    printf("%s\n",ch);

}
```

```
Len: 15
Ciao come stai?
 Fortran :>Ciao come stai?<
 Fortran len(ch):          15
Tutto bene
```

Passing C string to Fortran


```
module fortran_string_mod
implicit none

contains
    subroutine fortran_string(ch)
        character(*) :: ch
        print *,'Fortran :>',ch,'<'
        print *,'Fortran len(ch):', len(ch)
        ch = 'Tutto bene'
    end subroutine fortran_string
!-----------------------------------------
    subroutine fortran_string_c(ch,n) bind(C)
        use iso_c_binding
        integer(C_INT),value :: n
        type(C_PTR),value :: ch
        character(len=n),pointer :: x_ch

        call c_f_pointer(ch, x_ch)

        call fortran_string(x_ch)

    end subroutine
end module
```

```
#include <stdio.h>
#include <string.h>
void fortran_string_c(char ch[], int n);

int main(void) {
    char ch[] = "Ciao come stai?";
    int n;
    n = strlen(ch);

    printf("Len: %d\n",n);
    printf("%s\n",ch);

    fortran_string_c(ch, n);

    printf("%s\n",ch);

}
```

```
Len: 15
Ciao come stai?
 Fortran :>Ciao come stai?<
 Fortran len(ch):          15
Tutto bene
```

Variabili C globali e struct


```
module c_struct_and_extern_mod
use iso_c_binding
implicit none

type, bind(C) :: x_point
    integer(C_INT) :: i
    real(C_DOUBLE) :: x
    real(C_DOUBLE) :: y
end type

real(C_DOUBLE),bind(C, name="Alpha") :: alpha

interface
    ! void  peralpha(Xpoint *p)
    subroutine peralpha(p) bind(C)
        import  x_point
        type(x_point) :: p
    end subroutine
end interface

end module
```

```
program c_struc_extern
use c_struct_and_extern_mod
implicit none
    type(x_point) :: p
    p % i = 2
    p % x = 3.2
    p % y = 4.5
    alpha = 101.0

    call peralpha(p)

    print *,'p%i:', p % i
    print *,'p%x:', p % x
    print *,'p%y:', p % y
end program
```

```
typedef struct {
    int i;
    double x;
    double y;
} Xpoint;

double Alpha = 4.5;

void  peralpha(Xpoint *p) {
    p->x = p->x * Alpha;
    p->y = p->y * Alpha;
    p->i = p->i * Alpha;
}
```

```
 p%i:           202
 p%x:   323.20000481605530
 p%y:   454.50000000000000
```

Callback procedura interna


```
module c_function_pointer_mod
use iso_c_binding
implicit none

abstract interface
    !double (*Integrand)(double x)
    function integrand(x) result(r) bind(C)
        import C_DOUBLE, C_PTR
        implicit none
        real(C_DOUBLE), value :: x
        real(C_DOUBLE) :: r
    end function
end interface

! double quadrature(Integrand f, double xmin, double xmax)
interface
    function quadrature(f, xmin, xmax) result(r) bind(C)
        import C_DOUBLE,C_PTR, integrand
        implicit none
        procedure(integrand) :: f
        real(C_DOUBLE), value :: xmin, xmax
        real(C_DOUBLE) :: r
    end function
end interface
end module
```

```
program quadrature_main
use c_function_pointer_mod
implicit none

    real(C_DOUBLE) :: r, xmin, xmax, a
    xmin = 0; xmax = 1;  a = 1.0
    r = quadrature(flocal, xmin, xmax)
    print*,'Result: ' , r

contains
    function flocal(x) result(r) bind(C)
        real(C_DOUBLE), value :: x
        real(C_DOUBLE) :: r

        r = x + a
    end function
end program
```

```
typedef  double (*Integrand)(double x);

double quadrature(Integrand f, double xmin, double xmax){
    double ymin, ymax;
    ymax = f(xmax);
    ymin = f(xmin);
    return (ymax + ymin)/2/(xmax-xmin);
}
```

Result: 1.5000000000000000

Dipende dalla possibilità di definire con bind(C) una procedura interna.

Callback User Defined Type


```
module c_function_pointer_cintf_mod
use iso_c_binding
implicit none

abstract interface
    !double (*Integrand)(double x, void * fdata)
    function integrand(x, fdata) result(r) bind(C)
        import C_DOUBLE, C_PTR
        implicit none
        real(C_DOUBLE), value :: x
        type(C_PTR),value :: fdata
        real(C_DOUBLE) :: r
    end function
end interface

! double quadrature(Integrand f, double xmin, double xmax, void * fdata)
interface
    function quadrature(f, xmin, xmax, fdata) result(r) bind(C)
        import C_DOUBLE,C_PTR, integrand
        implicit none
        procedure(integrand) :: f
        real(C_DOUBLE), value :: xmin, xmax
        type(C_PTR),value :: fdata
        real(C_DOUBLE) :: r
    end function
end interface
end module
```

```
typedef  double (*Integrand)(double x, void * fdata);

double quadrature(Integrand f, double xmin, double xmax, void * fdata){
    double ymin, ymax;
    ymax = f(xmax, fdata);
    ymin = f(xmin, fdata);
    return (ymax + ymin)/2/(xmax-xmin);
}
```

Callback User Defined Type (2)


```
module myquad_mod
use c_function_pointer_cintf_mod
implicit none

    type t_mydata
        real(C_DOUBLE) :: a, b
    end type

contains
    function flocal(x, fdata) result(r) bind(C)
        real(C_DOUBLE), value :: x
        real(C_DOUBLE) :: r
        type(C_PTR),value :: fdata
        type(t_mydata),pointer :: fdata_fptr

        call c_f_pointer(fdata, fdata_fptr)

        r = fdata_fptr%a * x + fdata_fptr%b
    end function
end module
```

```
program quadrature_main
use myquad_mod
implicit none

    real(C_DOUBLE) :: r, xmin, xmax
    type(t_mydata),target :: fdata

    xmin = 0; xmax = 1;

    fdata % a = 2.0
    fdata % b = 3.0

    r = quadrature(flocal, xmin, &
              xmax, c_loc(fdata))

    print*,'Result: ' , r

end program
```

La variabile di tipo t_mydata non ha bisogno di essere interoperabile. In effetti nell'esempio non lo è visto che manca il bind(C).

Conclusione

- Lo standard Fortran 2003 ha portato molte novità tra cui la interoperabilità con il C.
- Altre novità riguardano la programmazione orientata agli oggetti. Nessuno potrà più dire che il Fortran non è un linguaggio che permette la programmazione Object Oriented.
- Il Fortran 2008 ha chiarificato ed esteso diverse caratteristiche del Fortran precedente. In più ha introdotto i coarray per la programmazione parallela ed submodule.
- Il Fortran 2015, estenderà l'interazione con il C, e migliorerà la programmazione parallela dei coarray.