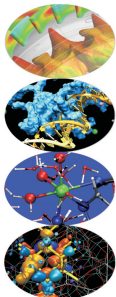


A Fortran Survey

F. Salvatore

CINECA Roma - SCAI Department

Roma, 2014



Le basi

Introduzione

Il mio primo programma Fortran

Tipi intrinseci e derivati

Array e cicli

Input/output

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

Le basi

Introduzione

Il mio primo programma Fortran

Tipi intrinseci e derivati

Array e cicli

Input/output

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

- ▶ Sviluppato negli anni '50: il primo linguaggio di alto livello! (HLL)
- ▶ Rapidamente diffuso nell'area del calcolo scientifico e tecnico:
 - ▶ Le regole del linguaggio permettono un'ottimizzazione efficiente dei calcoli
 - ▶ Molte funzioni e tipi matematici fanno parte del linguaggio
- ▶ Gli standard definiscono le evoluzioni del linguaggio:
 - ▶ Fortran 66: il primo standard di linguaggio in assoluto
 - ▶ Fortran 77: diffusissimo, tuttora molti codici lo seguono
 - ▶ Fortran 90 e poi 95: aggiunge l'allocazione dinamica, l'*array-syntax*, ADT etc.
 - ▶ Fortran 2003: aggiunge programmazione OO e varie altre cose, quasi completamente implementato dai compilatori
 - ▶ Fortran 2008: pubblicato nel 2010 e implementato solo in parte dai compilatori

▶ A favore del C

- ▶ codici Fortran correnti seguono almeno quattro “stili” (legati agli standard) oltre a istruzioni non standard più o meno diffuse
- ▶ il Fortran può essere terribilmente scomodo per applicazioni non di calcolo numerico (GUI, accessi a DB,...)
- ▶ ci sono più programmatori C che non programmatori Fortran
- ▶ lo standard C99 ha soddisfatto alcune esigenze numeriche del C

▶ A favore del Fortran

- ▶ Fortran è più rigido del C: è più facile scrivere codice con buone prestazioni rispetto a quanto accade nel C
- ▶ dal 2003 il Fortran dispone di feature Object Oriented
- ▶ dal 2003 ci sono modalità standard di interoperabilità con C e altri linguaggi
- ▶ il Fortran gestisce meglio del C i tipi definiti dall’utente

Le basi

Introduzione

Il mio primo programma Fortran

Tipi intrinseci e derivati

Array e cicli

Input/output

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

C

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

int main() {
    const int N=10;
    double a[N];
    double avg,var;

    for(int i=0;i<N;i++) { // inizializzazione
        a[i] = i;
    }

    avg = 0.; var = 0.;
    for(int i=0;i<N;i++) { // calcolo
        avg += a[i];
        var += a[i]*a[i];
    }
    avg = avg/N;
    var = var/N - avg*avg;

    printf("media: %lf \n",avg); // output
    printf("varianza: %lf \n", var);
    return 0;
}
```

C

```
// Media e varianza di array
int main() {
    const int N=10;
    ...
    for(int i=0;i<N;i++)
    ...
}
```

Fortran

```
! Media e varianza di array
program avg_var
    implicit none
    integer, parameter :: N=10
    ...
    do i=1,N
    ...
end program avg_var
```

- Il main, uno e uno solo, inizia con l'istruzione `[program nome]` e termina con l'istruzione `end [program nome]`

C

```
// Media e varianza di array
int main() {
    const int N=10;
    ...
    for(int i=0;i<N;i++)
    ...
}
```

Fortran

```
! Media e varianza di array
program avg_var
    implicit none
    integer, parameter :: N=10
    ...
    do i=1,N
    ...
    end program avg_var
```

- ▶ Il main, uno e uno solo, inizia con l'istruzione `[program nome]` e termina con l'istruzione `end [program [nome]]`
- ▶ Seguono in ordine: dichiarazione di variabili e istruzioni

C

```
// Media e varianza di array
int main() {
    const int N=10;
    ...
    for(int i=0;i<N;i++)
    ...
}
```

Fortran

```
! Media e varianza di array
program avg_var
    implicit none
    integer, parameter :: N=10
    ...
    do i=1,N
    ...
    end program avg_var
```

- ▶ Il main, uno e uno solo, inizia con l'istruzione `[program nome]` e termina con l'istruzione `end [program [nome]]`
- ▶ Seguono in ordine: dichiarazione di variabili e istruzioni
 - ▶ `implicit none` evita l'assegnazione automatica dei tipi alle variabili non definite ([i-n] interi, reali le altre)

C

```
// Media e varianza di array
int main() {
    const int N=10;
    ...
    for(int i=0;i<N;i++)
    ...
}
```

Fortran

```
! Media e varianza di array
program avg_var
    implicit none
    integer, parameter :: N=10
    ...
    do i=1,N
    ...
    end program avg_var
```

- ▶ Il main, uno e uno solo, inizia con l'istruzione `[program nome]` e termina con l'istruzione `end [program [nome]]`
- ▶ Seguono in ordine: dichiarazione di variabili e istruzioni
 - ▶ `implicit none` evita l'assegnazione automatica dei tipi alle variabili non definite ([i-n] interi, reali le altre)
- ▶ I commenti dopo il `!` e per proseguire una linea `&`

C

```
// Media e varianza di array
int main() {
    const int N=10;
    ...
    for(int i=0;i<N;i++)
    ...
}
```

Fortran

```
! Media e varianza di array
program avg_var
    implicit none
    integer, parameter :: N=10
    ...
    do i=1,N
    ...
    end program avg_var
```

- ▶ Il main, uno e uno solo, inizia con l'istruzione `[program nome]` e termina con l'istruzione `end [program [nome]]`
- ▶ Seguono in ordine: dichiarazione di variabili e istruzioni
 - ▶ `implicit none` evita l'assegnazione automatica dei tipi alle variabili non definite ([i-n] interi, reali le altre)
- ▶ I commenti dopo il `!` e per proseguire una linea `&`
- ▶ Non servono i `;` a fine istruzione se si va a capo...

C

```
// Media e varianza di array
int main() {
    const int N=10;
    ...
    for(int i=0;i<N;i++)
    ...
}
```

Fortran

```
! Media e varianza di array
program avg_var
    implicit none
    integer, parameter :: N=10
    ...
    do i=1,N
    ...
    end program avg_var
```

- ▶ Il main, uno e uno solo, inizia con l'istruzione **[program nome]** e termina con l'istruzione **end [program [nome]]**
- ▶ Seguono in ordine: dichiarazione di variabili e istruzioni
 - ▶ **implicit none** evita l'assegnazione automatica dei tipi alle variabili non definite ([i-n] interi, reali le altre)
- ▶ I commenti dopo il **!** e per proseguire una linea &
- ▶ Non servono i **;** a fine istruzione se si va a capo...
- ▶ ... e la riga è lunga al massimo 132 colonne

C

```
// Media e varianza di array
int main() {
    const int N=10;
    ...
    for(int i=0;i<N;i++)
    ...
}
```

Fortran

```
! Media e varianza di array
program avg_var
    implicit none
    integer, parameter :: N=10
    ...
    do i=1,N
    ...
    end program avg_var
```

- ▶ Il main, uno e uno solo, inizia con l'istruzione **[program nome]** e termina con l'istruzione **end [program [nome]]**
- ▶ Seguono in ordine: dichiarazione di variabili e istruzioni
 - ▶ **implicit none** evita l'assegnazione automatica dei tipi alle variabili non definite ([i-n] interi, reali le altre)
- ▶ I commenti dopo il **!** e per proseguire una linea &
- ▶ Non servono i **;** a fine istruzione se si va a capo...
- ▶ ... e la riga è lunga al massimo 132 colonne
- ▶ Il Fortran è case InSENSitIVE!

Le basi

Introduzione

Il mio primo programma Fortran

Tipi intrinseci e derivati

Array e cicli

Input/output

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

C

```
const int N=10;  
  
double a[N];  
double avg,var;
```

Fortran

```
integer :: N=10
```

- ▶ `integer` per gli interi

C

```
const int N=10;  
  
double a[N];  
double avg,var;
```

Fortran

```
integer, parameter :: N=10
```

- ▶ **integer** per gli interi
 - ▶ l'attributo **parameter** definisce una costante

C

```
const int N=10;  
  
double a[N];  
double avg,var;
```

Fortran

```
integer :: N=10  
  
real :: a(N)  
real :: avg,var
```

- ▶ **integer** per gli interi
 - ▶ l'attributo **parameter** definisce una costante
- ▶ **real** per i *floating-point* in singola precisione

C

```
const int N=10;  
  
double a[N];  
double avg,var;
```

Fortran

```
integer :: N=10  
  
real :: a(N)  
real :: avg,var
```

- ▶ **integer** per gli interi
 - ▶ l'attributo **parameter** definisce una costante
- ▶ **real** per i *floating-point* in singola precisione
- ▶ Per la doppia precisione:

C

```
const int N=10;  
  
double a[N];  
double avg,var;
```

Fortran

```
integer :: N=10  
  
real*8 :: a(N)  
real*8 :: avg,var
```

- ▶ **integer** per gli interi
 - ▶ l'attributo **parameter** definisce una costante
- ▶ **real** per i *floating-point* in singola precisione
- ▶ Per la doppia precisione:
 - ▶ **real*8** (**non-standard** ma molto usato)

C

```
const int N=10;

double a[N];
double avg,var;
```

Fortran

```
integer :: N=10
integer, parameter :: myk=kind(1.d0)
real(myk) :: a(N)
real(kind(1.d0)) :: avg,var
```

- ▶ **integer** per gli interi
 - ▶ l'attributo **parameter** definisce una costante
- ▶ **real** per i *floating-point* in singola precisione
- ▶ Per la doppia precisione:
 - ▶ **real*8** (**non-standard** ma molto usato)
 - ▶ **real(kind(1.d0))** in cui **kind(1.d0)** restituisce il “kind” del reale in doppia precisione **1.d0**

C

```
const int N=10;

double a[N];
double avg,var;
```

Fortran

```
integer :: N=10
integer, parameter :: myk=kind(1.d0)
real(myk) :: a(N)
real(kind(1.d0)) :: avg,var
```

- ▶ **integer** per gli interi
 - ▶ l'attributo **parameter** definisce una costante
- ▶ **real** per i *floating-point* in singola precisione
- ▶ Per la doppia precisione:
 - ▶ **real*8** (**non-standard** ma molto usato)
 - ▶ **real(kind(1.d0))** in cui **kind(1.d0)** restituisce il “kind” del reale in doppia precisione **1.d0**
 - ▶ **real(selected_real_kind(12))** in cui **selected_real_kind(12)** restituisce il “kind” di un reale con una precisione di almeno 12 cifre

- ▶ Attenzione alle costanti letterali: `1.2` in C è un **double**, in Fortran è del tipo reale di default (di fatto in singola), quindi come `1.2F`
- ▶ Per usare costanti in doppia precisione, come visto, `1.2d0` oppure modi più raffinati usando i kind definiti (`1.2_myk`)
- ▶ Il Fortran fornisce i tipi intrinseci **logical** per i booleani e **complex** per i complessi, analoghi dei tipi C **bool** e **complex** (solo in C99)
- ▶ Per i caratteri:
 - ▶ **character :: c** definisce una variabile contenente un singolo carattere
 - ▶ **character(len=80) :: s** definisce una variabile contenente fino a 80 caratteri
- ▶ Regole analoghe a quelle del C valgono anche per il Fortran riguardo i *cast* impliciti

- ▶ L'equivalente di **struct** in C è **type**

```
type position
  real :: x, y, z
end type position
```

- ▶ **type(position) :: r** dichiara una variabile di tipo **position**
- ▶ Tipi derivati possono contenere altri tipi derivati
- ▶ L'operatore % permette di accedere alle componenti:
r%y = 0.0; p%v%z=0.0
- ▶ Il nome del tipo è il costruttore di default:
r=position(1.,2.,3.)
- ▶ L'assegnazione = è disponibile intrinsecamente
- ▶ **interface operator(op-name)** permette di definire gli altri operatori

Le basi

Introduzione

Il mio primo programma Fortran

Tipi intrinseci e derivati

Array e cicli

Input/output

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

- ▶ Array monodimensionali si dichiarano con la dimensione tra parentesi tonde **real :: a(N)**
o specificando l'attributo **dimension**
real, dimension(N) :: a,b,c,d
- ▶ Di default le componenti dell'array vanno da 1 a N ma è possibile specificarlo **real :: a(-N/2:N/2-1)**
- ▶ Per accedere all'array si usano sempre parentesi tonde e indici interi
- ▶ I compilatori non fanno *bound checking* né in compilazione né a run-time a meno di usare opzioni specifiche

► **do**

blocco di istruzioni

end do

1. Esegue il blocco di istruzioni per sempre ...
2. ... fino a quando non incontra l'istruzione **exit**, che porta a eseguire il codice dopo l'**end do**
3. ... mai permettere che un ciclo vada avanti all'infinito

► **do while** (*condizione logica*)

blocco di istruzioni

end do

1. Valuta la *condizione logica*
2. Se è falsa, va al 5
3. Esegue il *blocco di istruzioni*
4. Va al passo 1
5. L'esecuzione procede all'istruzione seguente **end do**

C

```
avg = 0.; var = 0.;  
for(int i=0;i<N;i++) {  
    avg += a[i];  
    var += a[i]*a[i];  
}
```

Fortran

```
avg = 0.d0; var = 0.d0;  
do i=1,N  
    avg = avg + a(i);  
    var = var + a(i)*a(i);  
end do
```

► **do** *var* = *init*, *limit* [, *step*]

blocco di istruzioni

end do

1. Pone *step* pari 1, se non specificato
2. Assegna a *var* il valore *init*
3. Calcola $n_{iter} = \max\{0, \lfloor (limit - init + step) / step \rfloor\}$ (**diverso dal C**, l'iteratore non si può modificare!)
4. Se n_{iter} vale zero va a 6
5. Esegue n_{iter} volte il *blocco di istruzioni*, aggiungendo *step* a *var* al termine di ogni *blocco di istruzioni*
6. L'esecuzione procede all'istruzione che segue **end do**

Le basi

Introduzione

Il mio primo programma Fortran

Tipi intrinseci e derivati

Array e cicli

Input/output

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

C

```
printf("numero: %d \n",N);  
scanf("media: %lf \n",&avg);  
printf("media: %lf \n",avg);
```

Fortran

```
write(*,*) N  
read(*,*) avg  
write(*,'(F20.12)') avg
```

- Le istruzioni **write** e **read** hanno due argomenti:

C

```
printf("numero: %d \n",N);  
scanf("media: %lf \n",&avg);  
printf("media: %lf \n",avg);
```

Fortran

```
write(*,*) N  
read(*,*) avg  
write(*,'(F20.12)') avg
```

- ▶ Le istruzioni **write** e **read** hanno due argomenti:
- ▶ Il secondo argomento è il formato
 - ▶ * sta per un formato di default (*compiler dependent*)
 - ▶ altrimenti il formato si specifica con una stringa che inizia e termina con parentesi tonde
 - ▶ **(F20.12)** per un reale con 20 cifre di cui 12 dopo la virgola
 - ▶ **(I4)** per un intero usando al più 4 cifre

C

```
printf("numero: %d \n",N);  
scanf("media: %lf \n",&avg);  
printf("media: %lf \n",avg);
```

Fortran

```
write(*,*) N  
read(*,*) avg  
write(*,'(F20.12)') avg
```

- ▶ Le istruzioni **write** e **read** hanno due argomenti:
- ▶ Il secondo argomento è il formato
 - ▶ * sta per un formato di default (*compiler dependent*)
 - ▶ altrimenti il formato si specifica con una stringa che inizia e termina con parentesi tonde
 - ▶ (F20.12) per un reale con 20 cifre di cui 12 dopo la virgola
 - ▶ (I4) per un intero usando al più 4 cifre
- ▶ Le **read** assegnano le variabili perchè in Fortran di default il passaggio degli argomenti è per riferimento

C

```
FILE *hFile;  
hFile = fopen('par.dat', "r");  
fscanf(hfile, "%d", N);  
fclose(hFile);  
printf("Fine Input\n");
```

Fortran

```
open(unit=15, file='par.dat')  
read(15, *) N  
close(15)  
write(*, *) 'Fine input'
```

- Il primo argomento di **read** e **write** è l'unità su cui scrivere

C

```
FILE *hFile;  
hFile = fopen('par.dat', "r");  
fscanf(hFile, "%d", N);  
fclose(hFile);  
printf("Fine Input\n");
```

Fortran

```
open(unit=15, file='par.dat')  
read(15, *) N  
close(15)  
write(*, *) 'Fine input'
```

- ▶ Il primo argomento di **read** e **write** è l'unità su cui scrivere
- ▶ ***** sta per standard output/input

C

```
FILE *hFile;  
hFile = fopen('par.dat', "r");  
fscanf(hfile, "%d", N);  
fclose(hFile);  
printf("Fine Input\n");
```

Fortran

```
open(unit=15, file='par.dat')  
read(15, *) N  
close(15)  
write(*, *) 'Fine input'
```

- ▶ Il primo argomento di **read** e **write** è l'unità su cui scrivere
- ▶ ***** sta per standard output/input
- ▶ Per leggere e scrivere su file è necessario precollegarlo assegnandogli un'unità tramite l'istruzione **open**

C

```
FILE *hFile;  
hFile = fopen('par.dat', "r");  
fscanf(hfile, "%d", N);  
fclose(hFile);  
printf("Fine Input\n");
```

Fortran

```
open(unit=15, file='par.dat')  
read(15, *) N  
close(15)  
write(*, *) 'Fine input'
```

- ▶ Il primo argomento di **read** e **write** è l'unità su cui scrivere
- ▶ ***** sta per standard output/input
- ▶ Per leggere e scrivere su file è necessario precollegarlo assegnandogli un'unità tramite l'istruzione **open**
- ▶ Non è necessario specificare se l'utilizzo è in lettura o scrittura (ma è possibile farlo con **action='write'**)

C

```
FILE *hFile;  
hFile = fopen('par.dat', "r");  
fscanf(hFile, "%d", N);  
fclose(hFile);  
printf("Fine Input\n");
```

Fortran

```
open(unit=15, file='par.dat')  
read(15, *) N  
close(15)  
write(*, *) 'Fine input'
```

- ▶ Il primo argomento di **read** e **write** è l'unità su cui scrivere
- ▶ ***** sta per standard output/input
- ▶ Per leggere e scrivere su file è necessario precollegarlo assegnandogli un'unità tramite l'istruzione **open**
- ▶ Non è necessario specificare se l'utilizzo è in lettura o scrittura (ma è possibile farlo con **action='write'**)
- ▶ Si può gestire correttamente l'errore aggiungendo agli argomenti di open **status=ierr**

Fortran

```
open(unit=15 , file='campi.dat' , &  
      status=ierr , form='unformatted' )  
if(ierr /= 0) STOP 'Reading parametri.dat failed'  
write(15) a  
close(15)
```

- ▶ Per I/O binario specificare **form='unformatted'** e poi omettere il formato nelle **write** e **read**

Fortran

```
open(unit=15 , file='campi.dat' , &  
      status=ierr , form='unformatted' )  
if(ierr /= 0) STOP 'Reading parametri.dat failed'  
write(15) a  
close(15)
```

- ▶ Per I/O binario specificare **form='unformatted'** e poi omettere il formato nelle **write** e **read**
- ▶ Il Fortran inserisce un *record-marker* tra una scrittura e un'altra per permettere la navigazione nel file con comandi tipo **rewind** e **backspace**

Fortran

```
open(unit=15 , file='campi.dat' , &  
      status=ierr , form='unformatted' , access='stream')  
if(ierr /= 0) STOP 'Reading parametri.dat failed'  
write(15) a  
close(15)
```

- ▶ Per I/O binario specificare **form='unformatted'** e poi omettere il formato nelle **write** e **read**
- ▶ Il Fortran inserisce un *record-marker* tra una scrittura e un'altra per permettere la navigazione nel file con comandi tipo **rewind** e **backspace**
- ▶ Per evitare *record marker* e avere file compatibili con quelli C si può specificare nell'open **access=stream**

Fortran

```

!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
  ! Inizialization of a array
  do i=1,N
    a(i) = i-1
  end do
  ! Calculation
  avg = 0.d0; var = 0.d0;
  do i=1,N
    avg = avg + a(i)
    var = var + a(i)*a(i)
  end do
  avg = avg/N
  var = var/N - avg*avg
  ! Output
  write(*,*) 'average: ',avg
  write(*,' ("variance",F20.12)') var
end program avg_var

```

Fortran

!-Calculate average and variance of array data-

```

program avg_var
    implicit none
    integer, parameter :: N=10
    integer, parameter :: myk = kind(1.d0)
    integer :: i
    real(myk) :: a(N)
    real(myk) :: avg,var
    ! Inizialization of a array
    do i=1,N
        a(i) = i-1
    end do
    ! Calculation
    avg = 0.d0; var = 0.d0;
    do i=1,N
        avg = avg + a(i)
        var = var + a(i)*a(i)
    end do
    avg = avg/N
    var = var/N - avg*avg
    ! Output
    write(*,*) 'average: ',avg
    write(*,' ("variance",F20.12)') var
end program avg_var

```

Fortran

```
!-Calculate average and variance of array data-
program avg_var
    implicit none
    integer, parameter :: N=10
    integer, parameter :: myk = kind(1.d0)
    integer :: i
    real(myk) :: a(N)
    real(myk) :: avg,var
!   Inizialization of a array
    do i=1,N
        a(i) = i-1
    end do
!   Calculation
    avg = 0.d0; var = 0.d0;
    do i=1,N
        avg = avg + a(i)
        var = var + a(i)*a(i)
    end do
    avg = avg/N
    var = var/N - avg*avg
!   Output
    write(*,*) 'average: ',avg
    write(*,' ("variance",F20.12)') var
end program avg_var
```

Fortran

```
!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
!  Inizialization of a array
do i=1,N
  a(i) = i-1
end do
!  Calculation
avg = 0.d0; var = 0.d0;
do i=1,N
  avg = avg + a(i)
  var = var + a(i)*a(i)
end do
avg = avg/N
var = var/N - avg*avg
!  Output
write(*,*) 'average: ',avg
write(*,' ("variance",F20.12)') var
end program avg_var
```

Fortran

```

!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
!  Inizialization of a array
do i=1,N
  a(i) = i-1
end do
!  Calculation
avg = 0.d0; var = 0.d0;
do i=1,N
  avg = avg + a(i)
  var = var + a(i)*a(i)
end do
avg = avg/N
var = var/N - avg*avg
!  Output
write(*,*) 'average: ',avg
write(*,' ("variance",F20.12)') var
end program avg_var

```

Fortran

```
!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
!  Inizialization of a array
do i=1,N
  a(i) = i-1
end do
!  Calculation
avg = 0.d0; var = 0.d0;
do i=1,N
  avg = avg + a(i)
  var = var + a(i)*a(i)
end do
avg = avg/N
var = var/N - avg*avg
!  Output
write(*,*) 'average: ',avg
write(*,' ("variance",F20.12)') var
end program avg_var
```

Fortran

```
!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
!  Inizialization of a array
do i=1,N
  a(i) = i-1
end do
!  Calculation
avg = 0.d0; var = 0.d0;
do i=1,N
  avg = avg + a(i)
  var = var + a(i)*a(i)
end do
avg = avg/N
var = var/N - avg*avg
!  Output
write(*,*) 'average: ',avg
write(*,' ("variance",F20.12)') var
end program avg_var
```

Fortran

```
!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
!  Inizialization of a array
do i=1,N
  a(i) = i-1
end do
!  Calculation
avg = 0.d0; var = 0.d0;
do i=1,N
  avg = avg + a(i)
  var = var + a(i)*a(i)
end do
avg = avg/N
var = var/N - avg*avg
!  Output
write(*,*) 'average: ',avg
write(*,' ("variance",F20.12)') var
end program avg_var
```


Fortran

```
!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
!  Inizialization of a array
do i=1,N
  a(i) = i-1
end do
!  Calculation
avg = 0.d0; var = 0.d0;
do i=1,N
  avg = avg + a(i)
  var = var + a(i)*a(i)
end do
avg = avg/N
var = var/N - avg*avg
!  Output
write(*,*) 'average: ',avg
write(*,' ("variance",F20.12)') var
end program avg_var
```

Fortran

```
!-Calculate average and variance of array data-
program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg,var
!  Inizialization of a array
do i=1,N
  a(i) = i-1
end do
!  Calculation
avg = 0.d0; var = 0.d0;
do i=1,N
  avg = avg + a(i)
  var = var + a(i)*a(i)
end do
avg = avg/N
var = var/N - avg*avg
!  Output
write(*,*) 'average: ',avg
write(*,' ("variance",F20.12)') var
end program avg_var
```

- ▶ Copiare la versione Fortran del codice di calcolo di media e varianza su un file `media_varianza.f90`
- ▶ Compilare
`gfortran media_varianza.f90 -o media_varianza`
- ▶ Eseguire `./media_varianza`
- ▶ Controllare che i risultati siano corretti

Le basi

Unità di programma

Funzioni e subroutine

Moduli

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

Le basi

Unità di programma

Funzioni e subroutine

Moduli

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

C

```
double avg_var_fun(int N, double *a, double *var) {

    double avg;
    avg = 0.;
    *var = 0.;
    for(int i=0;i<N;i++) {
        avg += a[i];
        *var += a[i]*a[i];
    }
    avg = avg/N;
    *var = *var/N - avg*avg;
    return avg;
}

// main program...
int main() {
    const int N=10;
    double a[N];
    double avg,var;
    ...
    avg = avg_var_fun(N, a, &var);
    ...
}
```

- ▶ Il Fortran fornisce due tipi di procedure: funzioni e subroutine

- ▶ Il Fortran fornisce due tipi di procedure: funzioni e subroutine
- ▶ Le funzioni
 - ▶ restituiscono una variabile (anche un array)
 - ▶ il nome della funzione va dichiarato come variabile a cui assegnare il valore da ritornare
 - ▶ il comando `return` permette di uscire dalla funzione prima della fine
 - ▶ è buona pratica evitare *side effects* (modifica degli argomenti passati)

- ▶ Il Fortran fornisce due tipi di procedure: funzioni e subroutine
- ▶ Le funzioni
 - ▶ restituiscono una variabile (anche un array)
 - ▶ il nome della funzione va dichiarato come variabile a cui assegnare il valore da ritornare
 - ▶ il comando **return** permette di uscire dalla funzione prima della fine
 - ▶ è buona pratica evitare *side effects* (modifica degli argomenti passati)
- ▶ Le subroutine non restituiscono nulla
 - ▶ si chiamano con il comando **call nome-subroutine**
 - ▶ a differenza delle funzioni non possono essere usate in espressioni
 - ▶ corrispondono alle funzioni che restituiscono **void** del C

- ▶ Gli argomenti di una procedura si dicono
 - ▶ parametri formali (*dummy arguments*): nel contesto delle dichiarazioni della procedura
 - ▶ parametri effettivi (*actual arguments*): nel contesto del programma che invoca la procedura
- ▶ In Fortran di default le procedure passano gli argomenti per riferimento, *actual* e *dummy arguments* condividono la stessa area di memoria (diverso dal C!)
- ▶ È possibile specificare l'utilizzo che si intende fare delle variabili passate usando l'attributo **intent** per i parametri formali

```
real, intent(in) :: a  
real, intent(out) :: b  
real, intent(inout) :: c
```

Fortran

```

function avg_var_fun(N, a, var)
  implicit none
  integer, parameter :: myk = kind(1.d0)
  integer :: i, N
  real(myk) :: avg_var_fun, avg, var, a(N)
  avg = 0.d0;      var = 0.d0
  do i=1,N
    avg = avg + a(i); var = var + a(i)*a(i)
  end do
  avg = avg/N;    var = var/N - avg*avg
  avg_var_fun = avg
end function avg_var_fun

program avg_var
  implicit none
  integer, parameter :: N=10
  integer, parameter :: myk = kind(1.d0)
  integer :: i
  real(myk) :: a(N)
  real(myk) :: avg, var, avg_var_fun
  ...
  avg = avg_var_fun(N,a,var)
  ...
end program avg_var

```

Fortran

```
interface
  function avg_var_fun(a, var)
    integer, parameter :: myk = kind(1.d0)
    real(myk) :: avg_var_fun
    real(myk), intent(in) :: a(:)
    real(myk), intent(out) :: var
  end function avg_var_fun
end interface
```

- ▶ Programmi o procedure possono dichiarare tramite **interface** le procedure chiamate con i loro argomenti
 - ▶ necessarie per costrutti Fortran avanzati
 - ▶ assicurano il controllo sui tipi a *compile-time*
 - ▶ permettono di passare le funzioni come argomento di altre funzioni
 - ▶ permettono l'*overloading* delle procedure e degli operatori
- ▶ Svolgono in parte la funzione dei prototipi del C

Fortran

```

interface
  function avg_var_fun(a, var)
    integer, parameter :: myk = kind(1.d0)
    real(myk) :: avg_var_fun
    real(myk), intent(in) :: a(:)
    real(myk), intent(out) :: var
  end function avg_var_fun
end interface
  
```

- ▶ Specificando l'interfaccia è possibile passare vettori come *assumed-size*
 - ▶ nella procedura (e nell'interfaccia) si dichiara un vettore di lunghezza imprecisata scrivendo `a(:)`
 - ▶ la lunghezza viene quindi presa dal parametro attuale...
 - ▶ ... e si può estrarre con la funzione intrinseca `size`
- ▶ È possibile dichiarare variabili locali di lunghezza dipendente dai parametri effettivi: `real::work(size(a)) !automatic object`

- ▶ Il Fortran ha oltre 100 funzioni intrinseche
 - ▶ Funzioni di conversione di tipo: `int`, `real`, `cmplx`, ...
 - ▶ Funzioni matematiche: `sin`, `exp`, `log`, `max`, `mod`, ...
 - ▶ Funzioni di *inquiry*: `huge`, `digits`, `associated`, ...
 - ▶ Molte altre per gestione stringhe, bit, tipi, random, array, *reduce*, *timing*, ...
- ▶ L'interfaccia è esplicita
- ▶ Le funzioni matematiche sono generiche cioè lavorano correttamente con diversi tipi che ricevono come argomenti
- ▶ Molte funzioni sono `elemental` ovvero si applicano ad array elemento per elemento e restituiscono un array
- ▶ Possono essere usate nelle inizializzazioni, ad esempio:
`real(kind(1.d0)), parameter :: pi=acos(-1.d0)`

Le basi

Unità di programma

Funzioni e subroutine

Moduli

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

Fortran

```
module statistics
  implicit none
  integer, parameter :: myk = kind(1.d0)
contains
  subroutine init(a)
    real(myk) :: a(:) ; integer i
    do i=1,size(a)
      a(i) = i-1
    enddo
  end subroutine init
  function avg_var_fun(a, var)
    real(myk) :: avg_var_fun, avg, var, a(:)
    integer :: N, i
    N = size(a)
    avg = 0.; var = 0.
    do i=1,N
      avg = avg + a(i)
      var = var + a(i)*a(i)
    enddo
    avg = avg/N
    var = var/N - avg*avg
    avg_var_fun = avg
  end function avg_var_fun
end module statistics
```


- ▶ I moduli permettono di organizzare efficacemente codici
- ▶ Inserendo una procedura dentro un modulo l'interfaccia viene automaticamente generata in compilazione
- ▶ Per chiamare le procedure di un modulo, occorre usare il modulo tramite il comando **use nome-modulo** da porre prima sia di **implicit none** che di tutte le altre dichiarazioni
 - ▶ **use statistics, only : average** dichiara solo la funzione **average** del modulo **statistics**
 - ▶ **use statistics, only : media=>average** inoltre rinomina la funzione **average** in **media**
- ▶ Un modulo può contenere variabili che condividono la memoria con tutti i programmi che lo usano
 - ▶ In Fortran *antico* a tal fine si usavano le **common**

Fortran

```
program avg_var
  use statistics
  implicit none
  integer, parameter :: N=10
  real(myk) :: a(N)
  real(myk) :: avg,var

!  SUBROUTINE CALL: Initialization of a array
  call init(a)

!  FUNCTION CALL: Calculation
  avg = avg_var_fun(a,var)

!  Output
  write(*,*) 'average: ',avg, ' ; variance: ', var

end program avg_var
```

- ▶ Scrivere e testare le tre possibili evoluzioni del codice che calcola media e varianza
 - ▶ versione che utilizza una funzione per eseguire il calcolo vero e proprio
 - ▶ versione che utilizza una funzione e specifica l'interfaccia di tale funzione al chiamante
 - ▶ versione che utilizza un modulo all'interno del quale é inserita la funzione
- ▶ A che serve specificare l'interfaccia? In che modo i moduli aiutano?

Le basi

Unità di programma

Array e allocazione dinamica

Array multidimensionali

Array-syntax

Memoria dinamica

Puntatori

Fortran + C = *iso_c_binding*

Bibliografia

- Formula discreta su griglia 2D con condizioni al contorno di Dirichelet

$$\begin{cases} f(x_{i+1,j}) + f(x_{i-1,j}) - 2f(x_{i,j}) + \\ f(x_{i,j+1}) + f(x_{i,j-1}) - 2f(x_{i,j}) = 0 & \forall x_{i,j} \in (a, b)^2 \\ f(x_{i,j}) = \alpha(x_{i,j}) & \forall x_{i,j} \in \partial[a, b]^2 \end{cases}$$

- Soluzione iterativa con il metodo di Jacobi

$$\begin{cases} f_{n+1}(x_{i,j}) = \frac{1}{4} [& f_n(x_{i+1,j}) + f_n(x_{i-1,j}) + \\ & f_n(x_{i,j+1}) + f_n(x_{i,j-1})] & \forall n > 0 \\ f_0(x_{i,j}) = 0 & \forall x_{i,j} \in (a, b)^2 \\ f_n(x_{i,j}) = \alpha(x_{i,j}) & \forall x_{i,j} \in \partial[a, b]^2, & \forall n > 0 \end{cases}$$

Le basi

Unità di programma

Array e allocazione dinamica

Array multidimensionali

Array-syntax

Memoria dinamica

Puntatori

Fortran + C = *iso_c_binding*

Bibliografia

C

```
int main() {
    unsigned n=100, n2=n+2, maxIter=100000, i, j, iter=0;
    double T[n2][n2], Tnew[n2][n2]; // array di array
    double tol=0.0001, var = DBL_MAX, top = 100.0;
    for (i=0; i<=n; ++i)
        for (j=0; j<=n; ++j)
            T[i][j] = 0.0;
    for (i=1; i<=n; i++) {
        T[n+1][i] = i * top / (n+1); T[i][n+1] = i * top / (n+1);
    }
    while(var > tol && iter <= maxIter) {
        ++iter;    var = 0.0;
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j) {
                Tnew[i][j] = 0.25*( T[i-1][j] + T[i+1][j] +
                                     T[i][j-1] + T[i][j+1] );
                var = fmax(var, fabs(Tnew[i][j] - T[i][j]));
            }
        if(iter%100==0)printf("iter: %8u, var=%12.4lE\n",iter,var);
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j)
                T[i][j]=Tnew[i][j];
    }
    ...
}
```

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer             :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)           :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1;    var = 0.d0
    do j = 1, n
      do i = 1, n
        Tnew(i,j) = 0.25d0*( T(i-1,j) + T(i+1,j) + &
                             T(i,j-1) + T(i,j+1) )
        var = max(var, abs( Tnew(i,j) - T(i,j) ))
      end do
    end do
    if (mod(iter,100)==0) &
      write(*,"(a,i8,e12.4)") ' iter, variation:', iter, var
    T(1:n,1:n) = Tnew(1:n,1:n)
  end do
end do
end program laplace

```


C

```

int main() {
    unsigned n=100, n2=n+2, maxIter=100000, i, j, iter=0;
    double T[n2][n2], Tnew[n2][n2]; // array di array
    double tol=0.0001, var = DBL_MAX, top = 100.0;
    for (i=0; i<=n; ++i)
        for (j=0; j<=n; ++j)
            T[i][j] = 0.0;
    for (i=1; i<=n; i++) {
        T[n+1][i] = i * top / (n+1); T[i][n+1] = i * top / (n+1);
    }
    while(var > tol && iter <= maxIter) {
        ++iter;    var = 0.0;
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j) {
                Tnew[i][j] = 0.25*( T[i-1][j] + T[i+1][j] +
                                    T[i][j-1] + T[i][j+1] );
                var = fmax(var, fabs(Tnew[i][j] - T[i][j]));
            }
        if(iter%100==0)printf("iter: %8u, var=%12.4lE\n",iter,var);
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j)
                T[i][j]=Tnew[i][j];
    }
    ...
}

```

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer             :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)           :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1;    var = 0.d0
    do j = 1, n
      do i = 1, n
        Tnew(i,j) = 0.25d0*( T(i-1,j) + T(i+1,j) + &
                             T(i,j-1) + T(i,j+1) )
        var = max(var, abs( Tnew(i,j) - T(i,j) ))
      end do
    end do
    if (mod(iter,100)==0) &
      write(*,"(a,i8,e12.4)") ' iter, variation:', iter, var
    T(1:n,1:n) = Tnew(1:n,1:n)
  end do
end program laplace

```

- ▶ Si possono definire array fino a 7 dimensioni

- ▶ Si possono definire array fino a 7 dimensioni
- ▶ Se non specificato l'indice inferiore è 1, ma può essere specificato diversamente: `real :: T(0:n+1, -n:n)`

C

```

int main() {
    unsigned n=100, n2=n+2, maxIter=100000, i, j, iter=0;
    double T[n2][n2], Tnew[n2][n2]; // array di array
    double tol=0.0001, var = DBL_MAX, top = 100.0;
    for (i=0; i<=n; ++i)
        for (j=0; j<=n; ++j)
            T[i][j] = 0.0;
    for (i=1; i<=n; i++) {
        T[n+1][i] = i * top / (n+1); T[i][n+1] = i * top / (n+1);
    }
    while(var > tol && iter <= maxIter) {
        ++iter;    var = 0.0;
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j) {
                Tnew[i][j] = 0.25*( T[i-1][j] + T[i+1][j] +
                                    T[i][j-1] + T[i][j+1] );
                var = fmax(var, fabs(Tnew[i][j] - T[i][j]));
            }
        if(iter%100==0)printf("iter: %8u, var=%12.4lE\n",iter,var);
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j)
                T[i][j]=Tnew[i][j];
    }
    ...
}

```

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer             :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)            :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1;    var = 0.d0
    do j = 1, n
      do i = 1, n
        Tnew(i,j) = 0.25d0*( T(i-1,j) + T(i+1,j) + &
                             T(i,j-1) + T(i,j+1) )
        var = max(var, abs( Tnew(i,j) - T(i,j) ))
      end do
    end do
    if (mod(iter,100)==0) &
      write(*,"(a,i8,e12.4)") ' iter, variation:', iter, var
    T(1:n,1:n) = Tnew(1:n,1:n)
  end do
end do
end program laplace

```

- ▶ Si possono definire array fino a 7 dimensioni
- ▶ Se non specificato l'indice inferiore è 1, ma può essere specificato diversamente: `real :: T(0:n+1, -n:n)`
- ▶ Gli elementi sono referenziati specificando la lista di indici:
`T(n+1, n+1) = top`

- ▶ Si possono definire array fino a 7 dimensioni
- ▶ Se non specificato l'indice inferiore è 1, ma può essere specificato diversamente: `real :: T(0:n+1, -n:n)`
- ▶ Gli elementi sono referenziati specificando la lista di indici:
`T(n+1, n+1) = top`
- ▶ Si può inoltre referenziare un sottoinsieme di un array in un colpo solo: `T(start:end[:step], :) = 0.d0`

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer              :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)             :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1;    var = 0.d0
    do j = 1, n
      do i = 1, n
        Tnew(i,j) = 0.25d0*( T(i-1,j) + T(i+1,j) + &
                             T(i,j-1) + T(i,j+1) )
        var = max(var, abs( Tnew(i,j) - T(i,j) ))
      end do
    end do
    if (mod(iter,100)==0) &
      write(*,"(a,i8,e12.4)") ' iter, variation:', iter, var
    T(1:n,1:n) = Tnew(1:n,1:n)
  end do
end do
end program laplace

```

- ▶ Si possono definire array fino a 7 dimensioni
- ▶ Se non specificato l'indice inferiore è 1, ma può essere specificato diversamente: `real :: T(0:n+1, -n:n)`
- ▶ Gli elementi sono referenziati specificando la lista di indici:
`T(n+1, n+1) = top`
- ▶ Si può inoltre referenziare un sottoinsieme di un array in un colpo solo: `T(start:end[:step], :) = 0.d0`
- ▶ Il costruttore di default per gli array è: `(/ ... /)`
 - ▶ può contenere una sequenza di valori: `1, 2, 4`
 - ▶ che può anche essere definita da un *loop* implicito: `(expr, indice=inizio, fine [, incremento])`

- ▶ Si possono definire array fino a 7 dimensioni
- ▶ Se non specificato l'indice inferiore è 1, ma può essere specificato diversamente: `real :: T(0:n+1, -n:n)`
- ▶ Gli elementi sono referenziati specificando la lista di indici:
`T(n+1, n+1) = top`
- ▶ Si può inoltre referenziare un sottoinsieme di un array in un colpo solo: `T(start:end[:step], :) = 0.d0`
- ▶ Il costruttore di default per gli array è: `(/ ... /)`
 - ▶ può contenere una sequenza di valori: `1, 2, 4`
 - ▶ che può anche essere definita da un *loop* implicito: `(expr, indice=inizio, fine [, increment])`
- ▶ La sequenza degli *extent* di un array è detta *shape*, e.g. se `T` è `real :: T(3, 2:5)` allora:
 - ▶ `shape(T)` restituisce l'array `(/3, 4/)`
 - ▶ `size(T)` restituisce 12

- ▶ Si possono definire array fino a 7 dimensioni
- ▶ Se non specificato l'indice inferiore è 1, ma può essere specificato diversamente: `real :: T(0:n+1, -n:n)`
- ▶ Gli elementi sono referenziati specificando la lista di indici:
`T(n+1, n+1) = top`
- ▶ Si può inoltre referenziare un sottoinsieme di un array in un colpo solo: `T(start:end[:step], :) = 0.d0`
- ▶ Il costruttore di default per gli array è: `(/ ... /)`
 - ▶ può contenere una sequenza di valori: `1, 2, 4`
 - ▶ che può anche essere definita da un *loop* implicito: `(expr, indice=inizio, fine [, increment])`
- ▶ La sequenza degli *extent* di un array è detta *shape*, e.g. se `T` è `real :: T(3, 2:5)` allora:
 - ▶ `shape(T)` restituisce l'array `(/3, 4/)`
 - ▶ `size(T)` restituisce 12
- ▶ Per definire un array di costanti potremo usare l'attributo `parameter`

Le basi

Unità di programma

Array e allocazione dinamica

Array multidimensionali

Array-syntax

Memoria dinamica

Puntatori

Fortran + C = *iso_c_binding*

Bibliografia

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer             :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)           :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1;    var = 0.d0
    do j = 1, n
      do i = 1, n
        Tnew(i,j) = 0.25d0*( T(i-1,j) + T(i+1,j) + &
                             T(i,j-1) + T(i,j+1) )
        var = max(var, abs( Tnew(i,j) - T(i,j) ))
      end do
    end do
    if (mod(iter,100)==0) &
      write(*,"(a,i8,e12.4)") ' iter, variation:', iter, var
    T(1:n,1:n) = Tnew(1:n,1:n)
  end do
end do
end program laplace

```

- ▶ Array multidimensionali (*rank*>1) e sue sezioni possono essere usati in espressioni

```
Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + T(2:n+1,1:n) + &  
                          T(1:n,0:n-1) + T(1:n,2:n+1) )
```

- ▶ Array multidimensionali ($rank > 1$) e sue sezioni possono essere usati in espressioni

```
Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + T(2:n+1,1:n) + &
                          T(1:n,0:n-1) + T(1:n,2:n+1) )
```

- ▶ Gli operatori e gran parte delle procedure intrinseche possono essere applicate ad array

```
var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
```


- ▶ Array multidimensionali ($rank > 1$) e sue sezioni possono essere usati in espressioni

```
Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + T(2:n+1,1:n) + &
                          T(1:n,0:n-1) + T(1:n,2:n+1) )
```

- ▶ Gli operatori e gran parte delle procedure intrinseche possono essere applicate ad array

```
var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
```

- ▶ Le operazioni vanno intese elemento per elemento

- ▶ Array multidimensionali ($rank > 1$) e sue sezioni possono essere usati in espressioni

```
Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + T(2:n+1,1:n) + &  
                          T(1:n,0:n-1) + T(1:n,2:n+1) )
```

- ▶ Gli operatori e gran parte delle procedure intrinseche possono essere applicate ad array

```
var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
```

- ▶ Le operazioni vanno intese elemento per elemento
 - ▶ senza un ordine specificato

- ▶ Array multidimensionali ($rank > 1$) e sue sezioni possono essere usati in espressioni

```
Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + T(2:n+1,1:n) + &  
                          T(1:n,0:n-1) + T(1:n,2:n+1) )
```

- ▶ Gli operatori e gran parte delle procedure intrinseche possono essere applicate ad array

```
var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
```

- ▶ Le operazioni vanno intese elemento per elemento
 - ▶ senza un ordine specificato
 - ▶ e come se l'intera espressione a destra venisse valutata prima dell'assegnazione

- ▶ Array multidimensionali ($rank > 1$) e sue sezioni possono essere usati in espressioni

```
Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + T(2:n+1,1:n) + &
                          T(1:n,0:n-1) + T(1:n,2:n+1) )
```

- ▶ Gli operatori e gran parte delle procedure intrinseche possono essere applicate ad array

```
var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
```

- ▶ Le operazioni vanno intese elemento per elemento

- ▶ senza un ordine specificato
- ▶ e come se l'intera espressione a destra venisse valutata prima dell'assegnazione

- ▶ Le espressioni devono essere conformi

```
real :: t(1,1), s, u(4,4), v(4,4), w(4,4)
t = s ! it's right
s = t ! it's wrong
w = s*u+v+2.3 ! it's OK
w = u+v(1:2,1:2) ! it's wrong
```

- ▶ Array multidimensionali ($rank > 1$) e sue sezioni possono essere usati in espressioni

```
Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + T(2:n+1,1:n) + &
                        T(1:n,0:n-1) + T(1:n,2:n+1) )
```

- ▶ Gli operatori e gran parte delle procedure intrinseche possono essere applicate ad array

```
var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
```

- ▶ Le operazioni vanno intese elemento per elemento

- ▶ senza un ordine specificato
- ▶ e come se l'intera espressione a destra venisse valutata prima dell'assegnazione

- ▶ Le espressioni devono essere conformi

```
real :: t(1,1), s, u(4,4), v(4,4), w(4,4)
t = s ! it's right
s = t ! it's wrong
w = s*u+v+2.3 ! it's OK
w = u+v(1:2,1:2) ! it's wrong
```

- ▶ Espressioni non conformi possono non essere rilevate anche abilitando il *bound checking*

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer             :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)           :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0*( T(0:n-1,1:n) + T(2:n+1,1:n) + &
                           T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    T(1:n,1:n) = Tnew(1:n,1:n)
    if (mod(iter,100)==0) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  ...
  open(10, file="results")
  write(10, "(i8,i8,e18.9)") (( i, j, T(i,j), i=1,n), j=1,n)
  close(10)
end program laplace

```

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer             :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)           :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0*( T(0:n-1,1:n) + T(2:n+1,1:n) + &
                           T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    T(1:n,1:n) = Tnew(1:n,1:n)
    if (mod(iter,100)==0) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  ...
  open(10, file="results")
  write(10, "(i8,i8,e18.9)") (( i, j, T(i,j), i=1,n), j=1,n)
  close(10)
end program laplace

```

- ▶ **if** (*condizione logica*) **then**
blocco di istruzioni
end if
 - ▶ Esegue il *blocco di istruzioni* solo se la *condizione logica* è vera
 - ▶ Operatori relazionali: == (uguale), /= (diverso), >, <, >=, <=
 - ▶ Per condizioni che si applicano ad una singola istruzione si può usare la forma su un'unica linea
if (*condizione logica*) *istruzione*

- ▶ **if** (*condizione logica*) **then**
blocco di istruzioni
end if
 - ▶ Esegue il *blocco di istruzioni* solo se la *condizione logica* è vera
 - ▶ Operatori relazionali: == (uguale), /= (diverso), >, <, >=, <=
 - ▶ Per condizioni che si applicano ad una singola istruzione si può usare la forma su un'unica linea
if (*condizione logica*) *istruzione*
- ▶ Forme complete prevedono **else** e/o **else if** all'interno di
if () **then/end if**

- ▶ **if** (*condizione logica*) **then**
blocco di istruzioni
end if
 - ▶ Esegue il *blocco di istruzioni* solo se la *condizione logica* è vera
 - ▶ Operatori relazionali: == (uguale), /= (diverso), >, <, >=, <=
 - ▶ Per condizioni che si applicano ad una singola istruzione si può usare la forma su un'unica linea
if (*condizione logica*) *istruzione*
- ▶ Forme complete prevedono **else** e/o **else if** all'interno di **if () then/end if**
- ▶ L'analogo dello **switch** del C in Fortran è il **select case**

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0), n = 100
  integer             :: maxIter = 100000, i, j, iter = 0
  real(dp), dimension(0:n+1,0:n+1) :: T, Tnew
  real(dp)           :: tol = 1.d-4, var = 1.d0, top = 100.d0
  T(0:n,0:n) = 0.d0
  T(n+1,1:n) = (/ (i, i=1,n) /) * (top / (n+1))
  T(1:n,n+1) = (/ (i, i=1,n) /) * (top / (n+1))
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0*( T(0:n-1,1:n) + T(2:n+1,1:n) + &
                             T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    T(1:n,1:n) = Tnew(1:n,1:n)
    if (mod(iter,100)==0) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  ...
  open(10, file="results")
  write(10, "(i8,i8,e18.9)") (( i, j, T(i,j), i=1,n), j=1,n)
  close(10)
end program laplace

```

- ▶ **if** (*condizione logica*) **then**
blocco di istruzioni
end if
 - ▶ Esegue il *blocco di istruzioni* solo se la *condizione logica* è vera
 - ▶ Operatori relazionali: == (uguale), /= (diverso), >, <, >=, <=
 - ▶ Per condizioni che si applicano ad una singola istruzione si può usare la forma su un'unica linea
if (*condizione logica*) *istruzione*
- ▶ Forme complete prevedono **else** e/o **else if** all'interno di **if () then/end if**
- ▶ L'analogo dello **switch** del C in Fortran è il **select case**
- ▶ I *loop* impliciti possono essere annidati
 - ▶ e usati in istruzioni di I/O

Le basi

Unità di programma

Array e allocazione dinamica

Array multidimensionali

Array-syntax

Memoria dinamica

Puntatori

Fortran + C = *iso_c_binding*

Bibliografia

C

```

int main() {
    double *T, *Tnew, *Tmp;
    double tol, var = DBL_MAX, top = 100.0;
    unsigned n, n2, maxIter, i, j, iter = 0;
    printf("Enter mesh size, max iterations and tolerance: ");
    scanf("%u ,%u ,%lf", &n, &maxIter, &tol);
    n2 = n+2;
    T = calloc(n2*n2, sizeof(*T));
    Tnew = calloc(n2*n2, sizeof(*T));
    init_and_set_bc(T,top,'linear');
    while(var > tol && iter <= maxIter) {
        ++iter;    var = 0.0;
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j) {
                Tnew[i*n2+j] = 0.25*( T[(i-1)*n2+j] + T[(i+1)*n2+j]
                    + T[i*n2+(j-1)] + T[i*n2+(j+1)] );
                var = fmax(var, fabs(Tnew[i*n2+j] - T[i*n2+j]));
            }
        Tmp=T; T=Tnew; Tnew=Tmp;
    }
    free(T); free(Tnew);
    ...
    return 0;
}

```

C

```
int main() {
    double *T, *Tnew, *Tmp;
    double tol, var = DBL_MAX, top = 100.0;
    unsigned n, n2, maxIter, i, j, iter = 0;
    printf("Enter mesh size, max iterations and tolerance: ");
    scanf("%u ,%u ,%lf", &n, &maxIter, &tol);
    n2 = n+2;
    T = calloc(n2*n2, sizeof(*T));
    Tnew = calloc(n2*n2, sizeof(*T));
    init_and_set_bc(T,top,'linear');
    while(var > tol && iter <= maxIter) {
        ++iter;    var = 0.0;
        for (i=1; i<=n; ++i)
            for (j=1; j<=n; ++j) {
                Tnew[i*n2+j] = 0.25*( T[(i-1)*n2+j] + T[(i+1)*n2+j]
                    + T[i*n2+(j-1)] + T[i*n2+(j+1)] );
                var = fmax(var, fabs(Tnew[i*n2+j] - T[i*n2+j]));
            }
        Tmp=T; T=Tnew; Tnew=Tmp;
    }
    free(T); free(Tnew);
    ...
    return 0;
}
```

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0)
  integer              :: n, maxIter, i, j, iter = 0
  real (dp), dimension(:, :), allocatable :: T, Tnew
  real (dp)            :: tol, var = 1.d0, top = 100.d0
  write(*,*) 'Enter mesh size, max iterations and tolerance:'
  read(*,*)  n, maxIter, tol
  allocate (T(0:n+1,0:n+1), Tnew(0:n+1,0:n+1))
  call init_and_set_bc(T, top, 'linear')
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + &
      T(2:n+1,1:n) + T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    T(1:n,1:n) = Tnew(1:n,1:n)
    if( mod(iter,100) == 0 ) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  deallocate(T,Tnew)
end program laplace

```


Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0)
  integer              :: n, maxIter, i, j, iter = 0
  real (dp), dimension(:, :), allocatable :: T, Tnew
  real (dp)            :: tol, var = 1.d0, top = 100.d0
  write(*,*) 'Enter mesh size, max iterations and tolerance:'
  read(*,*)  n, maxIter, tol
  allocate (T(0:n+1,0:n+1), Tnew(0:n+1,0:n+1))
  call init_and_set_bc(T, top, 'linear')
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + &
      T(2:n+1,1:n) + T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    T(1:n,1:n) = Tnew(1:n,1:n)
    if( mod(iter,100) == 0 ) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  deallocate(T, Tnew)
end program laplace

```

- ▶ L'attributo *allocatable* permette di dichiarare un array specificando solo il suo *rank*, ad esempio
real, allocatable :: T(:, :)
- ▶ Lo *statement allocate* esegue l'allocazione della memoria e definisce gli *extent*, ad esempio: **allocate (T(3, 5) [, stat=integer_var])**
 - ▶ Se l'allocazione fallisce il programma termina
 - ▶ Specificando la clausola opzionale **stat=integer_var** il programma non termina e la variabile *integer_var* assume valore nullo se l'allocazione ha avuto successo, altrimenti un valore positivo
- ▶ Lo *statement deallocate* permette di liberare la memoria precedentemente riservata all'array
- ▶ La funzione **allocated()** permette di controllare lo stato di una array *allocatable*

Le basi

Unità di programma

Array e allocazione dinamica

Array multidimensionali

Array-syntax

Memoria dinamica

Puntatori

Fortran + C = *iso_c_binding*

Bibliografia

- ▶ L'attributo *pointer* permette di dichiarare una variabile da usare come alias di un altro oggetto
 - ▶ un puntatore è associato ad un dato oggetto tramite l'istruzione di assegnazione di puntatore =>
 - ▶ se un puntatore è associato ad una variabile questa dovrebbe avere l'attributo **target**
- ▶ Gli operatori operano sui puntati senza bisogno di dereferenziare, ad esempio
 - ▶ l'istruzione di assegnazione = equivale all'assegnazione dei puntati
- ▶ Lo stato di un puntatore può essere
 - ▶ associato, se è associato ad un *target*
 - ▶ disassociato, se => `null()`
 - ▶ indefinito, altrimenti
- ▶ La funzione **associated(ptr)** permette di controllare lo stato di un puntatore a meno che questo sia indefinito

- ▶ Si possono dichiarare puntatori multidimensionali
 - ▶ si deve specificare solo il *rank*
 - ▶ e il tipo deve essere lo stesso del *target*
- ▶ Come per gli array si possono usare sottoinsiemi
- ▶ È inoltre possibile allocare un puntatore di cui si sia specificato *rank* e tipo
 - ▶ La funzione `allocate()` crea un array senza nome, associato al puntatore, di tipo e *rank* del puntatore
 - ▶ La funzione `deallocate()` libera la memoria riservata dal puntato e disassocia il puntatore
- ▶ L'allocazione dei puntatori è diversa dagli array e può causare *memory leak*:

```

real, dimension(:, :), pointer :: p
real :: a
!...
allocate(p(n, m))
p=1
a=p(1, 1)
allocate(p(n, m))
! ...
  
```

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0)
  integer             :: n, maxIter, i, j, iter = 0
  real (dp), dimension(:, :), pointer :: T, Tnew, Tmp=>null()
  real (dp)           :: tol, var = 1.d0, top = 100.d0
  write(*,*) 'Enter mesh size, max iterations and tollerance:'
  read(*,*)  n, maxIter, tol
  allocate (T(0:n+1,0:n+1), Tnew(0:n+1,0:n+1))
  call init_and_set_bc(T, top, 'linear')
  Tnew = T
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + &
      T(2:n+1,1:n) + T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    Tmp =>T; T =>Tnew; Tnew => Tmp;
    if( mod(iter,100) == 0 ) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  ...
  deallocate (T, Tnew)
  nullify(Tmp)
end program laplace

```

Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0)
  integer             :: n, maxIter, i, j, iter = 0
  real (dp), dimension(:, :), pointer :: T, Tnew, Tmp=>null()
  real (dp)           :: tol, var = 1.d0, top = 100.d0
  write(*,*) 'Enter mesh size, max iterations and tollerance:'
  read(*,*)  n, maxIter, tol
  allocate (T(0:n+1,0:n+1), Tnew(0:n+1,0:n+1))
  call init_and_set_bc(T, top, 'linear')
  Tnew = T
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + &
      T(2:n+1,1:n) + T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    Tmp =>T; T =>Tnew; Tnew => Tmp;
    if( mod(iter,100) == 0 ) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  ...
  deallocate (T, Tnew)
  nullify(Tmp)
end program laplace

```

Fortran

```
program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0)
  integer              :: n, maxIter, i, j, iter = 0
  real (dp), dimension(:, :), pointer :: T, Tnew, Tmp=>null()
  real (dp)            :: tol, var = 1.d0, top = 100.d0
  write(*,*) 'Enter mesh size, max iterations and tollerance:'
  read(*,*)  n, maxIter, tol
  allocate (T(0:n+1,0:n+1), Tnew(0:n+1,0:n+1))
  call init_and_set_bc(T, top, 'linear')
  Tnew = T
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + &
      T(2:n+1,1:n) + T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    Tmp =>T; T =>Tnew; Tnew => Tmp;
    if( mod(iter,100) == 0 ) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  ...
  deallocate (T, Tnew)
  nullify(Tmp)
end program laplace
```


Fortran

```

program laplace
  implicit none
  integer, parameter :: dp=kind(1.d0)
  integer              :: n, maxIter, i, j, iter = 0
  real (dp), dimension(:, :), pointer :: T, Tnew, Tmp=>null()
  real (dp)           :: tol, var = 1.d0, top = 100.d0
  write(*,*) 'Enter mesh size, max iterations and tollerance:'
  read(*,*)  n, maxIter, tol
  allocate (T(0:n+1,0:n+1), Tnew(0:n+1,0:n+1))
  call init_and_set_bc(T, top, 'linear')
  Tnew = T
  do while (var > tol .and. iter <= maxIter)
    iter = iter + 1
    Tnew(1:n,1:n) = 0.25d0 * ( T(0:n-1,1:n) + &
      T(2:n+1,1:n) + T(1:n,0:n-1) + T(1:n,2:n+1) )
    var = maxval(abs( Tnew(1:n,1:n) - T(1:n,1:n) ))
    Tmp =>T; T =>Tnew; Tnew => Tmp;
    if( mod(iter,100) == 0 ) write(*,"(a,i8,e12.4)") &
      ' iter, variation:', iter, var
  end do
  ...
  deallocate (T, Tnew)
  nullify(Tmp)
end program laplace

```

- ▶ Scrivere e testare due possibili evoluzioni del codice che evolve l'equazione di Laplace
 - ▶ versione che utilizza memoria dinamica tramite **allocatable**.
 - ▶ versione che utilizza memoria dinamica tramite **pointers**
- ▶ Provare a enucleare pregi e difetti di ognuna, anche dal punto di vista delle performance

Le basi

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Fortran chiama C

Modulo *iso_c_binding*

C chiama Fortran

Bibliografia

C

```

int init(int, double*);
double avg_var_fun(int , double*, double*);

int main(int argc, char *argv[]) {
    int N, i;
    double *a, avg, var;
    if(argc != 2){
        fprintf(stderr, "Usage: %s size\n", argv[0]); exit(1); }
    N=atoi(argv[1]);
    a = (double*)malloc(N*sizeof(double));
    init(N,a);
    avg = avg_var_fun(N,a, &var);
    printf("average: %lf ; variance: %lf \n",avg, var);
    return 0;
}

int init(int N, double* a) {
    for(int i=0;i<N;i++) a[i] = i; return 0;
}

double avg_var_fun(int N, double* a, double* var) {
    double avg=0.; *var = 0.;
    for(int i=0;i<N;i++) { avg += a[i]; *var += a[i]*a[i]; }
    ...
    return avg;
}

```

Fortran

```

module statistics
  implicit none
  integer, parameter :: myk = kind(1.d0)
contains
  subroutine init(a)
    real(myk) :: a(:)
    integer :: i
    do i=1,size(a)
      a(i) = i-1
    enddo
  end subroutine init
  function avg_var_fun(a, var)
    real(myk) :: avg_var_fun,avg,var,a(:)
    integer :: N, i
    N = size(a)
    avg = 0.; var = 0.
    do i=1,N
      avg = avg + a(i)
      var = var + a(i)*a(i)
    end do
    ...
    avg_var_fun = avg
  end function avg_var_fun
end module statistics
    
```

Fortran

```
program avg_var
  use statistics
  implicit none
  integer :: N
  real(myk), allocatable :: a(:)
  real(myk) :: avg,var
  character(len=128) :: command
  character(len=80) :: arg

  call get_command_argument(0,command)
  if (command_argument_count() /= 1) then
    write(0,*) 'Usage:', trim(command), ' size'
    stop
  else
    call get_command_argument(1,arg)
    read(arg,*) n
  endif

  allocate(a(n))
  call init(a)
  avg = avg_var_fun(a,var)
  write(*,*) 'average: ',avg, ' ; variance: ', var

end program avg_var
```

Le basi

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Fortran chiama C

Modulo *iso_c_binding*

C chiama Fortran

Bibliografia



- Può essere utile chiamare una funzione C da un programma Fortran



- ▶ Può essere utile chiamare una funzione C da un programma Fortran
 - ▶ soprattutto se la funzione non è semplice da riscrivere



- ▶ Può essere utile chiamare una funzione C da un programma Fortran
 - ▶ soprattutto se la funzione non è semplice da riscrivere
 - ▶ oppure nel caso di una funzione di libreria per la quale non si possiede il sorgente

- ▶ Può essere utile chiamare una funzione C da un programma Fortran
 - ▶ soprattutto se la funzione non è semplice da riscrivere
 - ▶ oppure nel caso di una funzione di libreria per la quale non si possiede il sorgente
- ▶ Supponiamo di voler chiamare la funzione C

```
double avg_var_fun(int n, double a[], double *var) {  
    double avg = 0.; *var = 0.;  
    for(int i=0;i<n;i++) {  
        avg += a[i]; *var += a[i]*a[i];  
    }  
    avg = avg/n ; *var = *var/n - avg*avg;  
    return avg;  
}
```

dal Fortran

```
avg = avg_var_fun(n,a,var)
```

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Così si ottiene però un *undefined reference error* dovuto a un problema di *name mangling*:

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Così si ottiene però un *undefined reference error* dovuto a un problema di *name mangling*:
 - ▶ il compilatore Fortran modifica il nome degli *objects* rispetto ai nomi del sorgente

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Così si ottiene però un *undefined reference error* dovuto a un problema di *name mangling*:
 - ▶ il compilatore Fortran modifica il nome degli *objects* rispetto ai nomi del sorgente
 - ▶ tipicamente aggiunge un underscore ma dipende dall'implementazione e con i moduli le cose peggiorano

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Così si ottiene però un *undefined reference error* dovuto a un problema di *name mangling*:
 - ▶ il compilatore Fortran modifica il nome degli *objects* rispetto ai nomi del sorgente
 - ▶ tipicamente aggiunge un underscore ma dipende dall'implementazione e con i moduli le cose peggiorano
 - ▶ prima del Fortran 2003, si scrivevano le funzioni C usando nomi che includessero tali suffissi

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Così si ottiene però un *undefined reference error* dovuto a un problema di *name mangling*:
 - ▶ il compilatore Fortran modifica il nome degli *objects* rispetto ai nomi del sorgente
 - ▶ tipicamente aggiunge un underscore ma dipende dall'implementazione e con i moduli le cose peggiorano
 - ▶ prima del Fortran 2003, si scrivevano le funzioni C usando nomi che includessero tali suffissi
- ▶ L'altra difficoltà per linkare sorgenti C e Fortran è il *matching* tra i tipi delle variabili

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Così si ottiene però un *undefined reference error* dovuto a un problema di *name mangling*:
 - ▶ il compilatore Fortran modifica il nome degli *objects* rispetto ai nomi del sorgente
 - ▶ tipicamente aggiunge un underscore ma dipende dall'implementazione e con i moduli le cose peggiorano
 - ▶ prima del Fortran 2003, si scrivevano le funzioni C usando nomi che includessero tali suffissi
- ▶ L'altra difficoltà per linkare sorgenti C e Fortran è il *matching* tra i tipi delle variabili
 - ▶ non è garantito che `int` corrisponda a `integer` ...

- ▶ Sorgenti C e Fortran devono essere compilati separatamente e poi linkati

```
user@caspur$> gcc -c stat_libc.c
user@caspur$> gfortran -c stat_mainf.f90
user@caspur$> gfortran stat_libc.o stat_mainf.o -o stat_mainf
```

- ▶ Così si ottiene però un *undefined reference error* dovuto a un problema di *name mangling*:
 - ▶ il compilatore Fortran modifica il nome degli *objects* rispetto ai nomi del sorgente
 - ▶ tipicamente aggiunge un underscore ma dipende dall'implementazione e con i moduli le cose peggiorano
 - ▶ prima del Fortran 2003, si scrivevano le funzioni C usando nomi che includessero tali suffissi
- ▶ L'altra difficoltà per linkare sorgenti C e Fortran è il *matching* tra i tipi delle variabili
 - ▶ non è garantito che **int** corrisponda a **integer** ...
 - ▶ ... e **float** a **real**

Le basi

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Fortran chiama C

Modulo *iso_c_binding*

C chiama Fortran

Bibliografia

- ▶ Fortran 2003 fornisce un modulo intrinseco per “interoperare”
col C: **module iso_c_binding**

- ▶ Fortran 2003 fornisce un modulo intrinseco per “interoperare” col C: **module iso_c_binding**
- ▶ Procedure, variabili e tipi possono essere dichiarate interoperabili a *compile time*

- ▶ Fortran 2003 fornisce un modulo intrinseco per “interoperare” col C: **module iso_c_binding**
- ▶ Procedure, variabili e tipi possono essere dichiarate interoperabili a *compile time*
- ▶ **iso_c_binding** contiene costanti contenenti i *kind* da usare per i tipi intrinseci, e.g.
`integer(c_int)` corrisponde a `int`

- ▶ Fortran 2003 fornisce un modulo intrinseco per “interoperare” col C: **`module iso_c_binding`**
- ▶ Procedure, variabili e tipi possono essere dichiarate interoperabili a *compile time*
- ▶ **`iso_c_binding`** contiene costanti contenenti i *kind* da usare per i tipi intrinseci, e.g.
`integer(c_int)` corrisponde a `int`
- ▶ Alcuni di questi

| Type | Named constant | C type or types |
|------------------------|-------------------------------|------------------------------|
| <code>integer</code> | <code>c_int</code> | <code>int</code> |
| | <code>c_short</code> | <code>short int</code> |
| <code>real</code> | <code>c_float</code> | <code>float</code> |
| | <code>c_double</code> | <code>double</code> |
| <code>complex</code> | <code>c_float_complex</code> | <code>float _Complex</code> |
| | <code>c_double_complex</code> | <code>double _Complex</code> |
| <code>logical</code> | <code>c_bool</code> | <code>_Bool</code> |
| <code>character</code> | <code>c_char</code> | <code>char</code> |

- ▶ Usando i tipi interoperabili

- ▶ Usando i tipi interoperabili
 - ▶ variabili scalari, a parte i puntatori, interoperano

- ▶ Usando i tipi interoperabili
 - ▶ variabili scalari, a parte i puntatori, interoperano
 - ▶ array interoperano se di *shape* esplicita (o *assumed size*):
allocatable o *pointers* non interoperano direttamente col C!

- ▶ Usando i tipi interoperabili
 - ▶ variabili scalari, a parte i puntatori, interoperano
 - ▶ array interoperano se di *shape* esplicita (o *assumed size*):
allocatable o *pointers* non interoperano direttamente col C!

- ▶ Per array multidimensionali, le regole sono abbastanza intuitive

- ▶ Usando i tipi interoperabili
 - ▶ variabili scalari, a parte i puntatori, interoperano
 - ▶ array interoperano se di *shape* esplicita (o *assumed size*):
allocatable o *pointers* non interoperano direttamente col C!

- ▶ Per array multidimensionali, le regole sono abbastanza intuitive
 - ▶ `integer(c_int) :: f_arr(18, 3:7, 4)`
corrisponde a
`int c_arr[4][5][18]`

- ▶ Usando i tipi interoperabili
 - ▶ variabili scalari, a parte i puntatori, interoperano
 - ▶ array interoperano se di *shape* esplicita (o *assumed size*):
allocatable o *pointers* non interoperano direttamente col C!

- ▶ Per array multidimensionali, le regole sono abbastanza intuitive
 - ▶ `integer(c_int) :: f_arr(18,3:7,4)`
corrisponde a
`int c_arr[4][5][18]`
 - ▶ `integer(c_int) :: f_arr(18,3:7,*)`
corrisponde a
`int c_arr[][5][18]`

- ▶ Il *name mangling* viene gestito dall'attributo (Fortran 2003)
bind

- Il *name mangling* viene gestito dall'attributo (Fortran 2003)

bind

Fortran

```
function avg_var_fun(n, a, var) bind(c)
```


- ▶ Il *name mangling* viene gestito dall'attributo (Fortran 2003)

bind

Fortran

```
function avg_var_fun(n, a, var) bind(c)
```

- ▶ è interoperabile con

- ▶ Il *name mangling* viene gestito dall'attributo (Fortran 2003)

bind

Fortran

```
function avg_var_fun(n, a, var) bind(c)
```

- ▶ è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```

- ▶ Il *name mangling* viene gestito dall'attributo (Fortran 2003)

bind

Fortran

```
function avg_var_fun(n, a, var) bind(c)
```

- ▶ è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```

- ▶ Si può anche specificare una *binding label* (di default è la versione *lower-case* del codice Fortran), i.e. il nome da usare in C

- ▶ Il *name mangling* viene gestito dall'attributo (Fortran 2003)

bind

Fortran

```
function avg_var_fun(n, a, var) bind(c)
```

- ▶ è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```

- ▶ Si può anche specificare una *binding label* (di default è la versione *lower-case* del codice Fortran), i.e. il nome da usare in C

Fortran

```
function avg_var_fun(n, a, var) bind(c, name="AVG_VAR_FUN")
```

- ▶ Il *name mangling* viene gestito dall'attributo (Fortran 2003)

bind

Fortran

```
function avg_var_fun(n, a, var) bind(c)
```

- ▶ è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```

- ▶ Si può anche specificare una *binding label* (di default è la versione *lower-case* del codice Fortran), i.e. il nome da usare in C

Fortran

```
function avg_var_fun(n, a, var) bind(c, name="AVG_VAR_FUN")
```

- ▶ È richiesta l'interfaccia esplicita!

Fortran

```
interface
  function avg_var_fun(n,a,var) bind(c,name="avg_var")
    use iso_c_binding
    integer(c_int), value :: n
    real(c_double) :: a(*)
    real(c_double) :: var
    real(c_double) :: avg_var
  end function avg_var_fun
end interface
```

è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```

- ▶ Fortran di default passa gli argomenti per riferimento mentre C per valore

Fortran

```
interface
  function avg_var_fun(n,a,var) bind(c,name="avg_var")
    use iso_c_binding
    integer(c_int), value :: n
    real(c_double) :: a(*)
    real(c_double) :: var
    real(c_double) :: avg_var
  end function avg_var_fun
end interface
```

è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```


- ▶ Fortran di default passa gli argomenti per riferimento mentre C per valore
- ▶ L'attributo `value` specifica di passare le variabili per valore e quindi di interoperare con le variabili C

```
integer(c_int), value :: n
```

Fortran

```
interface
  function avg_var_fun(n,a,var) bind(c,name="avg_var")
    use iso_c_binding
    integer(c_int), value :: n
    real(c_double) :: a(*)
    real(c_double) :: var
    real(c_double) :: avg_var
  end function avg_var_fun
end interface
```

è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```

- ▶ Fortran di default passa gli argomenti per riferimento mentre C per valore
- ▶ L'attributo **value** specifica di passare le variabili per valore e quindi di interoperare con le variabili C

```
integer(c_int), value :: n
```

- ▶ Altrimenti un *dummy argument* in Fortran corrisponde a un parametro formale di tipo puntatore e può interoperare con il parametro formale dereferenziato

Fortran

```
interface
  function avg_var_fun(n, a, var) bind(c, name="avg_var")
    use iso_c_binding
    integer(c_int), value :: n
    real(c_double) :: a(*)
    real(c_double) :: var
    real(c_double) :: avg_var
  end function avg_var_fun
end interface
```

è interoperabile con

C

```
double avg_var_fun(int n, double a[], double *var)
```

- ▶ Fortran di default passa gli argomenti per riferimento mentre C per valore
- ▶ L'attributo `value` specifica di passare le variabili per valore e quindi di interoperare con le variabili C

```
integer(c_int), value :: n
```

- ▶ Altrimenti un *dummy argument* in Fortran corrisponde a un parametro formale di tipo puntatore e può interoperare con il parametro formale dereferenziato
- ▶ Un array Fortran interopera naturalmente con un array C (che è di fatto un puntatore)

Fortran

```
interface
  function avg_var_fun(n,a,var) bind(c,name="avg_var")
    use iso_c_binding
    integer(c_int), value :: n
    real(c_double) :: a(n)
    real(c_double) :: var
    real(c_double) :: avg_var
  end function avg_var_fun
end interface
```

è interoperabile con

C

```
double avg_var_fun(int n, double a[n], double *var)
```

- Notare che nel *main* Fortran l'array *a* può essere un array *allocatable*

- ▶ Fortran di default passa gli argomenti per riferimento mentre C per valore
- ▶ L'attributo **value** specifica di passare le variabili per valore e quindi di interoperare con le variabili C

```
integer(c_int), value :: n
```

- ▶ Altrimenti un *dummy argument* in Fortran corrisponde a un parametro formale di tipo puntatore e può interoperare con il parametro formale dereferenziato
- ▶ Un array Fortran interopera naturalmente con un array C (che è di fatto un puntatore)
- ▶ Funzioni C che restituiscono **void** corrispondono a subroutine

- ▶ Per gestire i puntatori in modo esplicito il modulo `iso_c_binding` definisce il tipo `C_PTR`: `type(C_PTR)`

- ▶ Per gestire i puntatori in modo esplicito il modulo **iso_c_binding** definisce il tipo **C_PTR**: **type (C_PTR)**
- ▶ Tuttavia i puntatori Fortran hanno una semantica più ricca dei puntatori C

- ▶ Per gestire i puntatori in modo esplicito il modulo **iso_c_binding** definisce il tipo **C_PTR**: **type (C_PTR)**
- ▶ Tuttavia i puntatori Fortran hanno una semantica più ricca dei puntatori C
 - ▶ array multidimensionali

- ▶ Per gestire i puntatori in modo esplicito il modulo **iso_c_binding** definisce il tipo **C_PTR**: **type (C_PTR)**
- ▶ Tuttavia i puntatori Fortran hanno una semantica più ricca dei puntatori C
 - ▶ array multidimensionali
 - ▶ memoria non contigua

- ▶ Per gestire i puntatori in modo esplicito il modulo **iso_c_binding** definisce il tipo **C_PTR**: **type (C_PTR)**
- ▶ Tuttavia i puntatori Fortran hanno una semantica più ricca dei puntatori C
 - ▶ array multidimensionali
 - ▶ memoria non contigua
- ▶ Per questo motivo **iso_c_binding** fornisce funzioni di collegamento

- ▶ Per gestire i puntatori in modo esplicito il modulo **iso_c_binding** definisce il tipo **C_PTR**: **type (C_PTR)**
- ▶ Tuttavia i puntatori Fortran hanno una semantica più ricca dei puntatori C
 - ▶ array multidimensionali
 - ▶ memoria non contigua
- ▶ Per questo motivo **iso_c_binding** fornisce funzioni di collegamento
- ▶ **c_loc(x)** ritorna un puntatore C al contenuto della variabile **x**
- ▶ **c_f_pointer(cptr, fptr[, shape])** fa l'operazione inversa, scrivendo il risultato in un puntatore Fortran **fptr**
 - ▶ Un argomento opzionale **shape** come **(/n/)** oppure **(/1, m, n/)** definisce la **shape**

- Per lasciare immutato il *main* Fortran e le funzioni C è possibile usare dei *wrapper* che hanno il compito di chiamare le funzioni C

Fortran - Interfacce

```

module statistics
  use iso_c_binding
  integer, parameter :: myk = kind(1.d0)
  interface
    function init_c(n,a) bind(c,name='init')
      use iso_c_binding
      integer(c_int) :: init_c
      integer(c_int), value :: n
      real(c_double) :: a(n)
    end function init_c
  end interface
contains
  subroutine init(a)
    real(kind(1.d0)) :: a(:)
    integer :: i,n
    n = size(a)
    i = init_c(n,a)
  end subroutine init
end module statistics

```

Le basi

Unità di programma

Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Fortran chiama C

Modulo *iso_c_binding*

C chiama Fortran

Bibliografia

- L'interfaccia deve essere esplicita e dichiarata con l'attributo **bind**. I *wrapper* evitano di modificare le procedure Fortran che si vogliono chiamare

Fortran - Wrapper

```

module statistics_c
  use iso_c_binding
  use statistics, only : init,  avg_var_fun
  implicit none
contains
  function init_c(n,a) bind(c, name='init')
    integer(c_int) :: init_c
    integer(c_int), value :: n
    real(c_double) :: a(n)
    call init(a)
    init_c = 0
  end function init_c
  function avg_var_fun_c(n, a, var) bind(c,name='avg_var_fun')
    integer(c_int), value :: n
    real(c_double) :: avg_var_fun_c,avg,var,a(n)
    avg_var_fun_c=avg_var_fun(a,var)
  end function avg_var_fun_c
end module statistics_c

```


- ▶ Se si usa il compilatore C, si possono ottenere degli *undefined reference* a causa di alcune chiamate Fortran
 - ▶ soluzione: linkare esplicitamente le *run-time libraries* del Fortran, ad esempio con il compilatore GNU

```
user@caspur$> gcc -lgfortran procedures.o main.c
```

- ▶ Se si usa il compilatore Fortran con GNU funziona mentre con Intel si ottiene *multiple definition of 'main'*
 - ▶ la *multiple definition of 'main'* si risolve linkando con l'opzione **-nofor-main**

```
user@caspur$> ifort procedures.o main.o -nofor-main
```

Le basi

Unità di programma



Array e allocazione dinamica

Fortran + C = *iso_c_binding*

Bibliografia

-  J3 US Fortran Standards Committee
<http://www.j3-fortran.org/>
-  ISO WG5 Committee
<http://www.nag.co.uk/sc22wg5/>
-  Fortran 2003 Standard Final Draft
Search Internet for `n3661.pdf`
-  M. Metcalf, J. Reid, M. Cohen
Fortran 95/2003 Explained Oxford University Press, corrected ed., 2008
-  M. Metcalf, J. Reid, M. Cohen
Modern Fortran Explained, Oxford University Press, 4th ed., 2011
-  Adams, J.C., Brainerd, W.S., Hendrickson, R.A., Maine, R.E., Martin, J.T., Smith, B.T.
The Fortran 2003 Handbook, Springer, 2009

Esempi di Codici in Fortran avanzato

-  Parallel Sparse Basic Linear Algebra Subroutines
www.ce.uniroma2.it/psblas/index.html
-  Portable Fortran Interfaces to the Trilinos C++ Package
trilinos.sandia.gov/packages/fortrilinos/

