Fortran 90

- Što je Fortran i zašto Fortran
- Tipovi podataka, operacije
- Polja
- Kontrola tijeka programa
- Struktura fortranskog programa
  - Procedure
  - Moduli
  - Interne procedure
  - Ulaz/izlaz
- Razno

Što je Fortran i zašto Fortran 90

- Programski jezik visoke razine za znanstveno/tehničko računanje
  - Početak u ranim 60-tim godinama, IBM FORmula TRANslation
  - **Jednostavan** i **učinkovit**
  - Odavno i duboko prisutan u znanosti i industriji
  - Razvoj u nekoliko faza (Fortran, Fortran II, Fortran IV, Fortran 66, Fortran 77 (i dalje u širokoj upotrebi), Fortran 90, ..., HPF, ...
- Fortran 90
  - Uključenje modernih spoznaja o programskim jezicima (struktura programa, upravljanje memorijom, ...)
  - Zadržana **učinkovitost** (ali na žalost ne i jednostavnost)
- Numerički najzahtjevniji programi (npr. meteorološki i ocenaografski numerički modeli) su pisani u Fortranu

Tipovi podataka

- FORTRAN je jako tipiziran (strongly typed) I statički tipiziran (statically typed)
- 4 osnovna tipa: **Cjelobrojni** (INTEGER), **realni** (REAL), **logički** (LOGICAL) I **znakovni** (CHARACTER)
- Zašto cjelobrojni i realni?
  - Memorija --> bitovi, byte-ovi, riječi
    1 byte = 8 bitova, 4 byte = 1 riječ (32 bitna arhitektura)
    8 byte = 1 riječ (64 bitna arhitektura)
  - Cijeli broj ima **egzaktan** prikaz u memoriji, cjelobrojne operacije se **mogu egzaktno** provesti u računalu
  - Isto **ne vrijedi** za realne brojeve i operacije koje su, osim toga, složenije od cjelobrojnih
  - Tipovi INTEGER i REAL su odvojeni zbog učinkovitosti
- Logički tip – samo dvije vrijednosti, .TRUE. i .FALSE.
- Znakovni tip – običan tekst (pomoćnog karaktera u num. mat.)

Tipovi podataka, nast.

- **Varijabla** – simboličko ime pridruženo nekoj vrijednosti, pri čemu se vrijednost može mijenjati, 'kućica s imenom'
  - Svaka varijabla ima svoj **tip** (prema tipu vrijednosti) koji se **ne može mijenjati**
  - Varijable je najbolje eksplicitno deklarirati i to naredbom oblika:
    `TIP, eventualni_atributi :: imeVar1, imeVar2, ...`
    (atributi, ovisno o kontekstu, daju dodatne informacije o varijabli)

- Na primjer:

```
INTEGER  :: I, A10=7, xy
INTEGER(KIND=2)  :: I, A1, x5y
REAL  ::  X, y, KgB
REAL(KIND=8), PARAMETER :: pi=3.14159265358979 3D0
LOGICAL :: B, K
CHARACTER(LEN=7)  ::  IME='AnteIvo'
```

- **Konstanta** – fiksna vrijednost, također ima tip
  - Npr. 10, -15, -15., 3.4e3, 7.3d-2, 'pero'

Računske i poredbene operacije

- Računske operacije:
  +, -, *, /, **
  su definirane za svaki (brojčani) tip posebno
- Npr:

```
INTEGER :: I, XY
REAL :: X
I = 4/3       !  I=1        ..... cjelobrojna operacija i cjelobrojni konačni rezultat
X = 4/3.      !  X=1.33333  ..... realna operacija i realni konačni rezultat
XY = 39/10.   !  XY = 3     ..... realna operacija, rezultat pretvoren u cijeli broj
```

- Relacijski operatori:

```
 <      >     <=     >=     ==     /=
.LT.  .GT.  .LE.  .GE.  .EQ.  .NE.        <-- sintaksa iz F77, vrijedi i dalje
```

- Logički operatori:

.NOT. .AND. .OR. .EQV. .NEQV.

Polja (arrays)

- Fundamentalni pojam (u programiranju i šire)
- Polje je uređeni niz (skup) varijabli **istog tipa** objedinjenih pod istim imenom
- Pojedina varijabla u polju je **element polja**
- Naredbe za **deklaraciju** polja (npr.):

```
REAL, DIMENSION(30)  ::  a,b,c
INTEGER  ::  X(-5:10), iz(7)
REAL(KIND=8), DIMENSION(-3:2,15)  ::  Z
```

- Terminologija:
  - **dimenzija** (dopuštena su 1-dim, 2-dim, ..., 7-dim polja)
  - **rang** polja = broj dimenzija = broj indeksa (drugačije nego u lin. alg.)
  - **raspon** (*extent*) neke dimnezije = broj elemenata u toj dimenziji
  - **veličina** (*size*) = ukupan broj elemenata u polju
  - **oblik** (*shape*) = broj dimenzija i raspon svake od njih
- Redoslijed spremanja elemenata polja u memoriji: Prva dimenzija se mijenja najbrže, potom druga, pa treća, ... (dakle, matrica se sprema **po stupcima**)

Polja (arrays), nast.

- Operacije s i nad poljima
  - Dva polja su **konformna** ako su istog oblika
  - Skalar je konforman sa svakim poljem
  - Osnovne računske i poredbene operacije su, za konformna polja, definirane **po elementima**, npr.:

```
INTEGER, DIMENSION(2,3)  ::  a,b,c
LOGICAL, DIMENSION(2,3)  ::   la
c = a+b          ! c je suma od a i b po elementima
b = a*c          ! b je produkt od a i c po elementima
a = 2            ! svaki elt. od a je jednak 2
la = (a<3)       ! polje vrijednosti .TRUE. ili .FALSE.
```

  - Sve intrinzične (ugrađene) funkcije rade na poljima po elementima, npr.

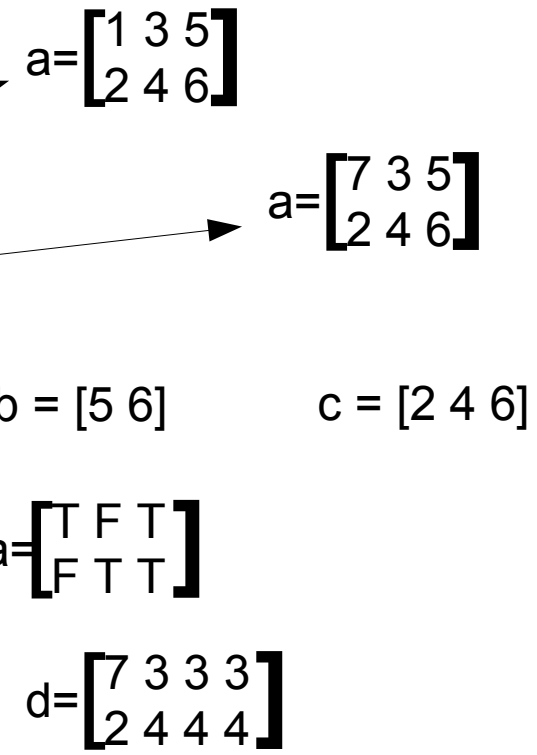
```
a = SQRT(b)
b = TAN(c)
```

  - U pravilu, svaki izraz koji je 'intuitivno' OK je i u Fortarnu 90 OK

Polja (arrays), nast.

- Pojedinačni i vektorski indeksi (a'la Python ili Matlab), npr.

```
PROGRAM test2

REAL, DIMENSION(6) :: a1=(/ 1, 2, 3, 4, 5, 6 /)
REAL :: a(2,3), b(2), c(3), d(2,4)
LOGICAL, DIMENSION(2,3) :: la
INTEGER, DIMENSION(4) :: ind=(/ 1,2,2,2 /)

a = RESHAPE(a1, (/ 2,3 /) )
a(1,1) = 7;  b = a(:,3);  c = a(2,:)
PRINT *,  a;    PRINT *,  b;    PRINT *,  c;

la = (a>3);
PRINT *, la

d = a(:,ind);
PRINT *, d

END PROGRAM test2
```



Polja (arrays), nast.

- Posebne funkcije za polja:
  - SIZE(*array*,*dim*) ..... vraća **raspon** (extent) polja *array* u zadanoj dimenziji *dim,* ili ukupan broj elemenata ako je *dim* izostavljen
  - LBOUND(*array*,*dim*) .... vraća **donju granicu** polja *array* u zadanoj dimenziji *dim*
  - UBOUND(*array*,*dim*) .... kao i gore, ali za **gornju granicu**
  - MAXVAL(*array*) .... vraća vrijednost **maksimalnog** elementa polja *array*
  - MINVAL(*array*) .... kao I gore, ali za **minimalni** element
  - SUM(*array*) .... vraća *sumu* svih elemenata polja *array*
  - PRODUCT(*array*) .... vraća *produkt* svih elemenata polja array

Napomena: U prve tri funkcije argument *dim* se može izostaviti, a u ostale dodati (slično kao u Pythonu ili Matlabu).

Kontrola tijeka programa

- Grananje: IF-THEN-ELSE

```
IF <logicki uvjet> THEN          # zaglavlje
    <blok naredbi>               # tijelo
END IF


IF <uvjet1> THEN
      <blok 1>
ELSE IF <uvjet2> THEN
      <blok 2>
         …
ELSE
      <blok_alt>
END IF
```

Kontrola tijeka programa, nast.

- Grananje: SELECT CASE

```
SELECT CASE  <izraz>                    ! <izraz> je izraz tipa integer, character
CASE <slučaj 1>                                ili logical, ali NE i real
    <naredbe tijela>

CASE <slučaj 2>
    <naredbe tijela>

      .....

END SELECT
```

Kontrola tijeka programa, nast.

- Ponavljanje (petlja): DO

```
DO  brojac = prvi, zadnji, korak
        ...
    <blok naredbi>
        ...
 END DO
```

Varijabla 'brojac' ima poseban status i ne smije se izravno mijenjati unuta do petlje. Ona se mijena automatski, tako da se u svakom koraku dodaje korak.

Struktura fortranskog programa

- Programska jedinica je fortranski program koji se može prevoditi (compilirati) sam za sebe (i dati strojni, objektni kod)
- Programske jedinice (blokovi) su:
  - glavni (main) program
  - funkcijski potprogram
  - subroutine potprogram

  } procedure
  - modul (module)
  - block-data
- Jedna datoteka može sadržavati više programskih jedinica.
  Čitav program se može nalaziti u više datoteka

Struktura fortranskog programa, nast.

- Opća struktura svake programske jedinice:

  linija identifikacije (zadaje vrstu i ime jedinice)
  deklaracijske naredbe
  izvršne naredbe
  završna linija (END)

- Ključni koncept: Jedna programska jedinica **nikada ne mora** poznavati detalje drugih jedinica
- Varijable deklarirane u nekoj programskoj jedinici su **lokalne** za tu jedinicu, tj. 'vide' se samo unutar te jedinice, ali zato ...
- Postoje **načini i pravila** kako pojedine programske jedinice komuniciraju
- → Olakšano pisanje, debugiranje, održavanje; moguća **opetovana upotreba** programa, npr. kroz biblioteke (*libraries*)

Procedure

- **Procedure** - zajedničko ime za function i subroutine potprograme
- **Potprogram** je programska cjelina koja obavlja neki 'zaokruženi' zadatak s jasno definiranim **ulaznim** i **izlaznim** veličinama

- ! Primjer poziva **funkcijskog** potprograma

```
PROGRAM glavni
IMPLICIT none
INTEGER :: n,suma,suma_prvih
n=10
suma = suma_prvih(n)  ! Poziv funkcije
PRINT *, 'SUMA=',suma
END PROGRAM glavni
```

```
FUNCTION suma_prvih(k)
IMPLICIT none
INTEGER :: k,suma_prvih,i
suma_prvih=0
DO i=1,k
  suma_prvih = suma_prvih+i
ENDDO
END FUNCTION suma_prvih
```

- ! Primjer poziva **subroutine** potprograma

```
PROGRAM glavni
IMPLICIT none
INTEGER :: n,suma
n=10;
CALL suma_prvih(n,suma)  ! Poziv subroutine
PRINT *, 'SUMA=',suma
END PROGRAM glavni
```

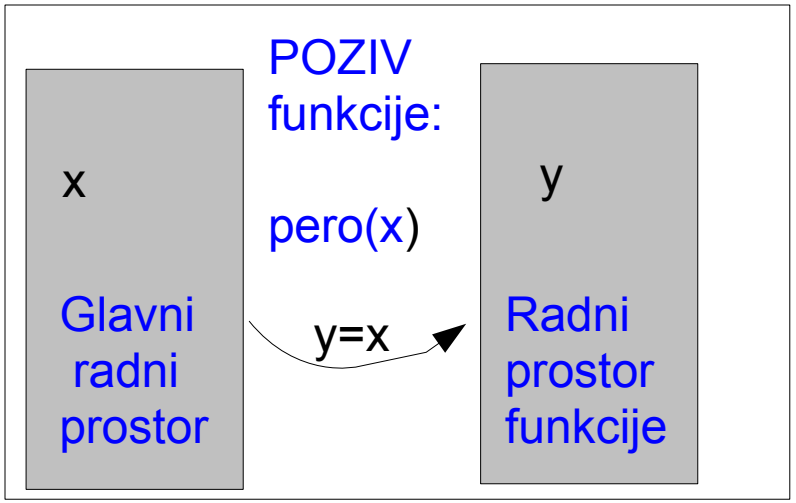
```
SUBROUTINE suma_prvih(k,s)
IMPLICIT none
INTEGER, INTENT(IN) :: k
INTEGER, INTENT(OUT) :: s
INTEGER :: i
s=0
DO i=1,k
  s = s+i
ENDDO
END SUBROUTINE suma_prvih
```

Prenošenje argumenata u Matlabu i Fortranu

(digresija)

- Matlab

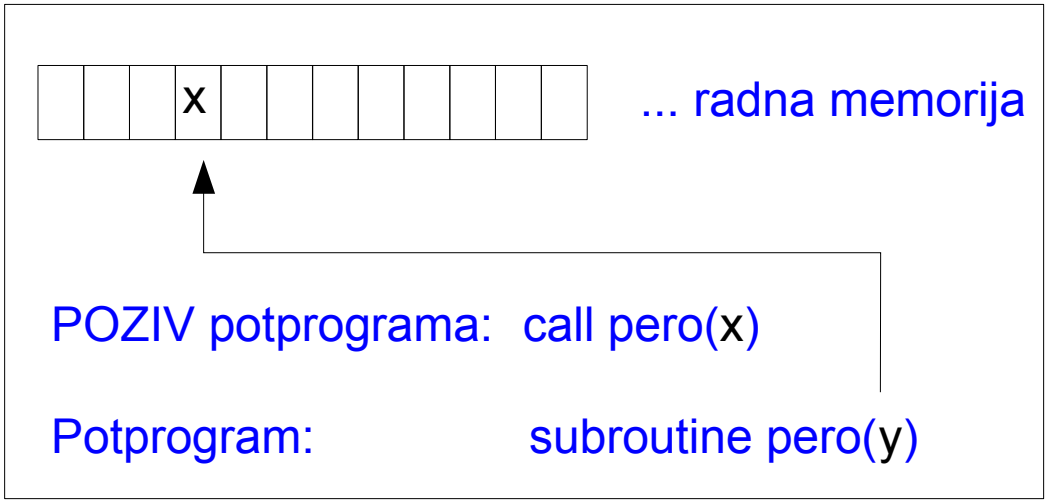
```
function  pero(y)
  y = 2
  return
```



- x se kopira
- funkcija radi s **kopijom**
- izvorni x ostaje netaknut
- prenose se vrijednosti (by value)

- Fortran

```
subroutine  pero(y)
  y = 2
  return
  end
```



- y je fiktivna varijabla (zapravo pokazivač)
- u trenutku poziva y pokazuje na x
- funkcija radi **izravno** s x (iako ga zove y)
- u gornjem primjeru x poprima vrijednost 2
- prenose se adrese (by reference)

Procedure, nast.

- U pozivnom programu zadani su **realni (stvarni) argumenti**
  U potprogramu nalaze se **fiktivni (engl. *dummy*) argumenti**

- **Broj, redoslijed i tip** argumenata se **moraju podudarati** !!!

- U trenutku poziva procedure fiktivni argumenti se putem memorijske adrese 'vezuju' na stvarne argumente (tj. na njihove **položaje** u memoriji, *by reference*)

- FUNCTION: Poziva se direktno imenom, ima tip i izravno se koristi u izrazima

- SUBROUTINE: Poziva se naredbom CALL, rezultate vraća preko argumenata

Procedure i polja

- Ako su realni i fiktivni argumenti **polja** procedura mora nešto znati o obliku (*shape*) polja. (Primjer s matricom 3x4)
- To je moguće pomoću:
  - **Eksplicitno zadanih** polja (*explicit size*)
    - Npr. naredba REAL :: a(mm,nn)
      u potprogramu zadaje polje eksplicitnog oblika, pri čemu su mm i nn **fiktivni** argumenti, **poznati u trenutku poziva**
  - Polja **pretpostavljenog oblika** (*assumed shape*)
    - Npr. naredba REAL DIMENSION(:,:) :: kGb
      u potprogramu zadaje 2-dim. polje kGb pretpostavljenog oblika.
    - Zahtijeva **eksplicitno sučelje** (*explicit interface*)
    - Oblik (shape) odgovarajućeg realnog argumenta (polja) se dobiva pomoću funkcije SHAPE ili UBOUND

Procedure i polja, nast.

- Primjer potprograma s **eksplicitno zadanim** poljem

```
PROGRAM glavni
    IMPLICIT none
    INTEGER, PARAMETER :: m=3,n=5
    INTEGER :: k,j
    REAL :: a(m,n)=-3, prod
    k=2
    CALL produkt_retka(a,k,m,n,prod)
    PRINT *, 'redak ',k,' glasi:'
    PRINT *, (a(k,j),j=1,n)
    PRINT *, 'Produkt=',prod
    END PROGRAM glavni
```

```
SUBROUTINE produkt_retka(aa,k,mm,nn,produkt)
    IMPLICIT none
    INTEGER, INTENT(IN) :: mm,nn,k
    REAL :: aa(mm,nn), produkt
    INTEGER :: j
    produkt=1
    DO j=1,nn
      produkt=produkt*aa(k,j)
    ENDDO
    END SUBROUTINE produkt_retka
```

Ispis na monitoru:

```
redak           2  glasi:
  -3.000000      -3.000000      -3.000000      -3.000000      -3.000000
 Produkt=  -243.0000
```

- **Automatska** polja (*automatic arrays*) su polja eksplicitnog oblika, zadana kao gore, ali koja **nisu fiktivni argumenti.**

Procedure i polja, nast.

- Primjer potprograma s poljem **pretpostavljenog oblika**

```
PROGRAM glavni
    IMPLICIT none
    INTEGER, PARAMETER :: m=3,n=5
    INTEGER :: k,j
    REAL :: a(m,n)=-3, prod
    k=2
    CALL produkt_retka(a,k,prod)
    PRINT *, 'redak ',k,' glasi:'
    PRINT *, (a(k,j),j=1,n)
    PRINT *, 'Produkt=',prod
    END PROGRAM glavni
```

```
SUBROUTINE produkt_retka(aa,k,produkt)
    IMPLICIT none
    INTEGER :: k
    REAL :: aa(:,:), produkt
    INTEGER :: j
    nn = UBOUND(aa,2)
    produkt=1
    DO j=1,nn
      produkt=produkt*aa(k,j)
    ENDDO
    END SUBROUTINE produkt_retka
```

Ispis na monitoru:

```
redak           2  glasi:
  -3.000000      -3.000000      -3.000000      -3.000000      -3.000000
 Produkt=  -243.0000
```

PAZI !!! U gornjem primjeru **nedostaje** sučelje !!! Potrebno ga je eksplicitno navesti kroz *interface block*, ili potprogram **uključiti kroz modul**.

Moduli

- Posebna programska jedinica (*program block*) koja olakšava ustrojavanje složenih (kompliciranih) programa
- Tri namjene:
  - Omogućava **globalni pristup** varijablama i poljima deklariranim u modulu
  - Definira **eksplicitno sučelje** za sve procedure definirane kroz modul
  - Podržava druge napredne koncepte (složeni tipovi, objektno orjentirano programiranje)
- Struktura modula

```
MODULE ime_modula
deklaracijske naredbe
CONTAINS
   procedure
END MODULE ime_modula
```

Moduli, nast.

- **Varijable** i **procedure** iz modula se mogu učiniti 'vidljivima', tj.koristiti, u bilo kojoj programskoj jedinici (osim BLOCK DATA) putem naredbe

  USE ime_modula

```
PROGRAM glavni
USE prvi_modul

IMPLICIT none
INTEGER :: k,j
REAL :: a(m1,n1)=-3, prod

k=2
CALL produkt_retka(a,k,prod)
PRINT *, 'redak ',k,' glasi:'
PRINT *, (a(k,j),j=1,n1)
PRINT *, 'Produkt=',prod

END PROGRAM glavni
```

```
MODULE prvi_modul

  INTEGER, PARAMETER :: m1=3, n1=5

CONTAINS

  SUBROUTINE produkt_retka(aa,k,produkt)
  IMPLICIT none
  INTEGER, INTENT(IN) :: k
  REAL :: aa(:,:), produkt
  INTEGER :: j
  produkt=1
  nn = UBOUND(aa,2)
  DO j=1,nn
    produkt=produkt*aa(k,j)
  ENDDO
  END SUBROUTINE produkt_retka

END MODULE prvi_modul
```

Moduli, nast.

- Jedinice koje sadrže module treba prevesti (*compile*) **prije** nego se prevedu jedinice koje te module koriste

- Rezime, bez ulaženja u 'napredne koncepte'
  - Moduli omogućuju jednostavno organiziranje (grupiranje) varijabli i funkcija u logički (prirodno) ustrojene skupine
  - Time je značajno olakšan i razvoj, ispravljanje te održavanje složenih programa (koji se sastoje od stotina programskih jedinica)

Unutarnje (*internal*) procedure

- Svaka programska jedinica (a ne samo moduli) može putem naredbe CONTAINS sadržavati **interne** potprograme tipa function i subroutine
- Takvi potprogrami se mogu pozivati **samo** iz programskog bloka (jedinice) u kojem se nalaze
- **Dostupne** su im sve varijable definirane u programskom bloku 'domaćinu' (npr. varijabla pi u donjem primjeru, lijevo, te varijable pi, r u primjeru, desno)

```
PROGRAM pero
IMPLICIT none
REAL(KIND=4), PARAMETER :: pi = ASIN(1.)*2
REAL :: r,o_k

r=10.;
CALL opseg_kruga(r,o_k)
print *,'r=',r, 'opseg=',o_k

CONTAINS
  SUBROUTINE opseg_kruga(rr,opseg)
  IMPLICIT none
  REAL :: rr,opseg
  opseg=2*pi*rr
  END SUBROUTINE opseg_kruga

END PROGRAM pero
```

```
PROGRAM pero
IMPLICIT none
REAL(KIND=4), PARAMETER :: pi = ASIN(1.)*2
REAL :: r,o_k

r=10.;
CALL opseg_kruga
print *,'r=',r, 'opseg=',o_k

CONTAINS
  SUBROUTINE opseg_kruga
  o_k=2*pi*r
  END SUBROUTINE opseg_kruga

END PROGRAM pero
```

Ulaz/izlaz (I/O)

- Ulaz: naredba read
- Izlaz: naredbe write i print (print je ekviv. sa write na standardni izlaz (ekran))

- Način čitanja ili pisanja:
  - *Zadan listom varijabli (list-directed)*
  - *Zadan formatom (formatted)*

- List-directed
  - read *, lista_varijabli

    read *, x,i1,i2
  - print *, lista_varijabli

    print *, x,i1,i2
  - kod unosa (read), vrijednosti se odvajaju bjelinom ili zarezom
  - služi za unos tipkovnicom ili ispis na ekran manjeg broja vrijednosti, obično prilikom pisanja i testiranja programa

Ulaz/izlaz (I/O), nast.

- Formatted
  - read(*u,fmt*) lista_varijabli, npr. read(5, '(f5.0, 2i2)') x,i1,i2
  - print *fmt*, lista_varijabli, npr. print '(f5.0, 2i2)', x,i1,i2
  - write(*u,fmt*) lista_varijabli, npr. write(7, '(f5.0, 2i2)') x,i1,i2
- Pri tom je:
  - ***u*** = logički broj datoteke iz koje se čita ili u koju se piše
  - broj *u* se pridjeljuje datoteci pomoću naredbe

    open(unit=*u*, file=*ime_datoteke*), npr. open(unit=1,file='pero.txt')
  - read(*, …) podrazumijeva standardni ulaz, write(*,...) standardni izlaz
  - **fmt** = niz znakova u navodncima i oblim zagradama, ili varijabla tipa character koja sadrži format (pravilo) za konverziju
  - fmt = * zapravo znači *list-directed* ulaz ili izlaz
  - npr. sljedeće je isto:

    read(*,*) a,b,c

    read *, a,b,c

Ulaz/izlaz (I/O), nast.

- Formati za učitavanje:

  I*w* učitaj sljedećih *w* znakova kao cijeli broj

  F*w.d* -"- -"- *w* -"- kao realni broj s *d* decimala (pazi!)

  E*w.d* isto kao prethodno

  A*w* učitaj sljedećih *w* znakova kao niz znakova (character)

  A

  *n*X preskoči sljedećih n znakova

- Formati za ispis:

  I*w* ispiši cijeli broj u sljedećih *w* znakova

  F*w.d* -"- realni broj u sljedećih *w* mjesta s *d* decimala

  E*w.d* -"- realni broj u sljedećih w mjesta s d decimala koristeći znanstvenu notaciju

  A*w* ispiši niz znakova (character) u sljedećih *w* znakova

  A

  *n*X preskoči sljedećih n mjesta

Ulaz/izlaz (I/O), nast.

- Napomene
  - Broj ispred formata znači ponavljanje, npr. ‘(3I2, 4X, 5F7.3)’
  - Svako izvršavanje naredbe read zahvaća (čita) jedan redak ulazne datoteke; iduće izvršavanje učitava sljedeći redak
  - Analogno vrijedi i za naredbu write

Primjer:

```
PROGRAM pero
IMPLICIT none
REAL :: x
INTEGER :: i1,i2
!
print *, 'unesi tri broja1', 17
read *, x,i1,i2
print *, x,i1,i2
print '(a3,f5.2,15x,2i3)', 'aa',x,i1,i2
write(*, '(a3,f5.2,15x,2i3)') 'aa',x,i1,i2
!
write(*,'(a)') 'unesi tri broja prema formatu (f5.0, 2i2)'
read(*,'(f5.0, 2i2)') x,i1,i2
write(*,*) x,i1,i2
!
END PROGRAM pero
```

Ostalo

- F90 podržava **slobodni format**
  - Na istoj liniji naredbe se odvajaju s ';'
  - Ključne riječi se odvajaju **barem** jednom bjelinom
  - Produljenje naredbe u novi red se postiže znakom '&'
- F90 ne razlikuje mala i velika slova (iako se mogu koristiti radi preglednosti)
- Imena varijabli
  - grade se iz znakova A-Z, a-z, 0-9, _
  - počinju slovom, max. duljina = 31
- Preporuča se koristiti IMPLICIT none, nakon čega treba sve varijable **eksplicitno deklarirati**.
- U protivnom, ako tip varijable nije eksplicitno zadan, varijable čija imena počinju s I,J,K,L,M,N su cjelobrojne, a sve ostale realne (povijesno naslijeđe)
- **Preskočeno**: I/O, složeni tipovi, grananja, petlje, BLOCK DATA, generičke funkcije, ...

Prevođenje pomoću GNU prevodioca

- gfortran pero.f90 ivo.f90
  prevodi i povezuje (*link*-a); Rezultat je izvršni program a.out
- gfortran -o ante pero.f90 ivo.f90
  isto kao gore, ali izvršni program se zove ante
- gfortran -c pero.f90 ivo.f90
  prevodjenje bez povezivanja (*link*-anja); rezultat su objektne datoteke pero.o i ivo.o
- gfortran -o ante -O3 -L /moja_biblioteka pero.f90 ivo.f90
  prevodi pero.f90 i ivo.f90, optimizira na nivou 3 (-O3), uključuje, ako treba, već prevedene procedure iz imenika /moja_biblioteka (-L) i proizvodi izvršni program ante
- man gfortran
  daje detaljni opis opcija

- Gfortran,ako se ne zatraži drugačije, automatski poziva linker (ld → man ld)
- Gfortran podržava sve opcije kao i gcc (GNU C i C++ prevoditelj → man gcc)

- Kako si olakšati prevođenje?
  - Primitivan način: Spremiti naredbu za prevođenje u izvršnu datoteku, npr. ff.sh, po potrebi je editirati i izvršavati
  - Pravi način: Koristiti make, ...