

Razne dodatne fortranske naredbe

« Numeričke metode »

Ivo Batistić

Fizički odsjek, PMF
Sveučilište u Zagrebu

predavanja 2005/2006

Pregled predavanja

Mjerenje vremena - DATE_AND_TIME

Generiranje slučajnih brojeva - RANDOM_NUMBER

Zadavanje početnih vrijednosti podacima - DATA

Zajednički podaci - COMMON blok

Zajednički podaci - MODULE

Izbor numeričkog modela - SELECTED_{INT,REAL}_KIND

Konverzije među numeričkim podacima

Operacije na bitovima

Dodatne funkcije koje nisu dio jezika - IARGC i GETARG

Mjerenje vremena - DATE_AND_TIME

- ▶ U fortranu 90 postoji procedura za dobivanje podataka o datumu i satu koja se oslanja na datum i sat samog računala.
- ▶ Procedura se zove DATE_AND_TIME.
- ▶ Procedura ima četiri argumenta:
 1. argument je niz od 8 slova: 'CCYYMMDD', gdje CC označava stoljeće (npr. 21, YY godinu, MM mjesec i DD dan u mjesecu.
 2. argument je niz od 10 slova u obliku 'HHMMSS.SSS', gdje HH označava sat, MM minute, SS.SSS sekunde i dijeliće sekundi.
 3. argument je zona, niz od 5 slova, oblika '±HHMM' što bi trebalo predstavljati razliku između sata na računalu i UTCa (vrijeme po Greenwichu).
 4. argument je niz od 8 cijelih brojeva, koje predstavljaju redom: godinu, mjesec, dan, razliku lokalnog i UTC vremena u minutama, sat, minute, sekunde i milisekunde.

Mjerenje vremena - DATE_AND_TIME

```
PROGRAM ispis_date_time

INTEGER, DIMENSION(8):: date_time
CHARACTER (LEN = 8) :: date
CHARACTER (LEN = 10) :: time
CHARACTER (LEN = 5) :: zone

CALL DATE_AND_TIME (date,time,zone, date_time)
PRINT *,date_time

END PROGRAM
```

Program ispisuje:

2006	5	15	120	9
36	47	967		

Još jedan primjer - množenje matrica

U ovom primjeru služimo se procedurom `DATE_AND_TIME` da bi provjerili vrijeme izvršavanja množenja matrica na dva različita načina.

```
PROGRAM matrix_mult
INTEGER, PARAMETER          :: nn = 1000
INTEGER, DIMENSION(8)      :: date_time
CHARACTER (LEN = 12), DIMENSION(3) :: clk
REAL(KIND=8), DIMENSION(nn,nn)  :: aa,bb,cc
REAL(KIND=8)                :: dd,t1,t2,t3

CALL RANDOM_NUMBER(aa)      ! popunimo matrice
CALL RANDOM_NUMBER(bb)     ! slučajnim brojevima

....
```

Još jedan primjer - množenje matrica

```
....  
cc = 0.0  
CALL DATE_AND_TIME (clk(1),clk(2),clk(3),date_time)  
t1 = 3600.0*date_time(5)+60.0*date_time(6)+REAL(date_time(7)) &  
      +date_time(8)/1000.0  
  
DO i=1,nn  
DO j=1,nn  
  DO k=1,nn  
    cc(j,i) = cc(j,i) + aa(j,k)*bb(k,i)  
  END DO  
END DO  
END DO  
  
CALL DATE_AND_TIME (clk(1),clk(2),clk(3),date_time)  
t2 = 3600.0*date_time(5)+60.0*date_time(6)+REAL(date_time(7)) &  
      +date_time(8)/1000.0  
PRINT *, "Obicno mnozenje = ", t2-t1
```

Još jedan primjer - množenje matrica

```
....  
cc = 0.0  
CALL DATE_AND_TIME (clk(1),clk(2),clk(3),date_time)  
t1 = 3600.0*date_time(5)+60.0*date_time(6)+REAL(date_time(7)) &  
      +date_time(8)/1000.0  
  
DO i=1,nn  
DO k=1,nn  
  dd = bb(k,i)  
  DO j=1,nn  
    cc(j,i) = cc(j,i) + aa(j,k)*dd  
  END DO  
END DO  
END DO  
  
CALL DATE_AND_TIME (clk(1),clk(2),clk(3),date_time)  
t2 = 3600.0*date_time(5)+60.0*date_time(6)+REAL(date_time(7)) &  
      +date_time(8)/1000.0  
PRINT *, "Poboljsano mnozenje = ", t2-t1
```

Generiranje slučajnih brojeva

- ▶ Slučajni brojevi imaju ogromnu važnost u računalnim metodama. Mnogobrojne numeričke metode bazirane su upravo na dobivanju skupa brojeva koji su nasumično izabrani unutar nekog intervala. Tipično te metode imaju ime *Monte Carlo*, iako su one međusobno vrlo različite.
- ▶ U fortranu 90 slučajne brojeve možemo dobiti koristeći proceduru **`RANDOM_NUMBER`**
- ▶ Procedura `RANDOM_NUMBER(a)` upisuje slučajne brojeve u argument. Argument može biti realni broj ili niz realnih brojeva.
- ▶ Upisani realni brojevi su između 0.0 i 1.0.
- ▶ Upisani realni brojevi trebaju uvijek biti različiti i jednoliko raspoređeni unutar (0.0 ,1.0) intervala.

Generiranje slučajnih brojeva

- ▶ Dobiveni realni brojevi ipak **nisu sasvim slučajni** nego su dio niza brojeva koji se generiraju jedan iz drugog, s tim da je period ponavljanja jako velik.
- ▶ Na generirani niz brojeva može se utjecati procedurom **RANDOM_SEED** koja ima obavezno jedan od tri opcionalna argumenta s imenima SIZE, PUT i GET.

```
PROGRAM pet_slucajnih_brojeva
```

```
REAL, DIMENSION(5) :: x
```

```
CALL RANDOM_NUMBER(x)
```

```
CALL RANDOM_SEED
```

```
CALL RANDOM_NUMBER(x)
```

```
PRINT *, x
```

```
END PROGRAM
```

Premjer generiranja istih i različitih slučajnih brojeva

```
PROGRAM isti_razliciti
  REAL          :: rnd
  INTEGER       :: duzina
  INTEGER, ALLOCATABLE :: iseed(:)
  REAL, DIMENSION(5)  :: xxx
  CALL RANDOM_SEED (SIZE=duzina)
  ALLOCATE(iseed(duzina))
  CALL RANDOM_SEED(GET=iseed)
  iseed = 0
  CALL RANDOM_NUMBER(xxx)
  PRINT *, 'Uvijek isti:'
  PRINT *, xxx
  CALL RANDOM_SEED
  CALL RANDOM_NUMBER(xxx)
  PRINT *, 'Uvijek razliciti:'
  PRINT *, xxx
  DEALLOCATE(iseed)
END PROGRAM
```

Inače nasumične brojeve možemo generirati i iz `DATE_AND_TIME` procedure čitajući tisućinke sekunde.

Osim toga dostupan je i veliki broj raznih drugih algoritama za generiranje slučajnih brojeva. U tom slučaju, broj tisućinki sekunde dobar je izbor za početni nasumični broj u nizu brojeva koji se generira.

Primjer - volumen jedinične sfere

```
PROGRAM MC_integracija
  REAL    :: x,y,z
  REAL    :: pi
  INTEGER :: i,j,n=1000000
  pi = 4.0*ATAN(1.0)
  j = 0
  DO i=1,n
    CALL RANDOM_NUMBER(x)
    CALL RANDOM_NUMBER(y)
    CALL RANDOM_NUMBER(z)
    IF ((x*x+y*y+z*z) < 1.0) j=j+1
  END DO
  PRINT *, 'Izracunat volumen = ', 8*REAL(j)/REAL(n)
  PRINT *, 'Tocan rezultat    = ', 4.0*pi/3.0
END PROGRAM
```

Program ispisuje:

```
Izracunat volumen =      4.185976
Tocan rezultat    =      4.188790
```

Zadavanje početnih vrijednosti podacima

- ▶ Podacima možemo zadati početne vrijednosti prilikom deklaracije varijabli.
- ▶ Početne se vrijednosti mogu zadati i pomoću naredbe **DATA**.
- ▶ DATA naredba može se nalaziti bilo gdje u programu, ali se tipično stavlja na početak iza deklaracija.
- ▶ Argumenti DATA naredbi su lista varijabli iza koje dolazi lista vrijednosti zatvorena između dvije kose crte.

```
REAL                :: a
CHARACTER(LEN=10)   :: b

DATA a,b           / 1.5,'znakovi' /
```

DATA naredba

- ▶ DATA naredba može sadržavati i više lista varijabli iza kojih dolazi lista vrijednosti u kosim crtama.
- ▶ Ako većem broju varijabli pridjeljujemo istu vrijednost, lista vrijednosti se može skratiti koristeći **broj ponavljanja** i znak **množenja** (zvjezdicu '*')
- ▶ Između liste varijabli i liste vrijednosti **nema** zareza
- ▶ Različite liste varijabli sa svojim vrijednostima se međusobno razdavaju zarezima ako su u istoj DATA naredbi.

```
INTEGER, DIMENSION(3,3) :: a,b
```

```
DATA a(1,:) / 1,2,2 /, a(2,:) / 2,1,2 /, a(3,:) / 2,2,1 /  
DATA b      / 1, 3*0, 1, 3*0, 1 /
```

DATA naredba

Kod iniciranja nizova, lista varijabli se također može skratiti koristeći implicitnu DO petlju.

```
INTEGER, DIMENSION(5,5) :: b

DATA ((b(i,j),b(j,i), j=i+1,5), i=1,5) / 20*0 /
DATA (b(i,i), i=1,5) / 5*1 /

PRINT '(5i)',b
```

Ispis:

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

DATA naredba

U podprogramima varijable inicirane s DATA naredbom čuvaju vrijednosti između uzastopnih poziva. Isto vrijedi za varijable inicirane kod deklaracije. DATA naredba se izvršava **samo jednom**, kod pokretanja programa.

```
PROGRAM data_test
  INTEGER :: i
  DO i=1,5
    CALL procA ! 1,2,3,4,5
    CALL procB ! 1,2,3,4,5
    CALL procC ! 1,1,1,1,1
  END DO
END PROGRAM
SUBROUTINE procB
  INTEGER :: i = 0
  i = i+1
  PRINT *, 'procB : ', i
END SUBROUTINE
```

```
SUBROUTINE procA
  INTEGER :: i
  DATA i / 0 /
  i = i+1
  PRINT *, 'procA : ', i
END SUBROUTINE
SUBROUTINE procC
  INTEGER :: i
  i = 0
  i = i+1
  PRINT *, 'procC : ', i
END SUBROUTINE
```

DATA naredba

DATA naredba se može koristiti i kod iniciranja izvedenih vrsta varijabli.

```
PROGRAM data_type
  IMPLICIT NONE
  TYPE ocjena
    CHARACTER (LEN=10) :: student
    INTEGER             :: ocjena
  END TYPE ocjena
  TYPE(ocjena), DIMENSION(100) :: numericke
  INTEGER                       :: i
  DATA numericke(1) / ocjena('Bolonjic',5) /
  DATA numericke(2) / ocjena('Studic',3) /
  DO i=1,2
    PRINT '(a10,":",i2)',numericke(i)
  END DO
END PROGRAM
```

```
Bolonjic   : 5
Studic    : 3
```

Zajednički podaci

- ▶ Programske cjeline (glavni program, podprogrami, funkcije,) mogu imati zajedničke podatke *uskladištene* u **COMMON** bloku.
- ▶ Prije Fortrana 90 to je bio jedini način kako je to bilo moguće postići.
- ▶ U Fortranu 90 preferirani način čuvanja zajedničkih podataka su **moduli**.
- ▶ *Common* blok se zadaje s listom (prethodno deklariranih) varijabli koja slijedi iza naredbe COMMON.
- ▶ Između liste varijabli i naredbe COMMON može biti navedeno i ime *common* bloka zatvoreno između dvije kose crte.
- ▶ Imenovanih *common* blokova može biti proizvoljno puno, ali samo jedan bez imena.

Zajednički podaci

- ▶ *Common* blok je globalni entitet, ne postoje *common* blokovi koji postoje samo unutar jedne programske cjeline.
- ▶ Jedna varijabla se **ne može** pojavljivati u dva različita *common* bloka.
- ▶ Varijable u *common* bloku zauzimaju memoriju računala po redu kako su izlistane.
- ▶ Varijable u *common* bloku u različitim programskim jedinicama mogu imati različita imena ali moraju biti istog tipa.
- ▶ Dvije ili više uzastopnih varijabli u *common* bloku koje su istog tipa mogu u drugoj programskoj jedinici činiti niz.

COMMON blok

```
PROGRAM zajednistvo
  INTEGER          :: i,j
  REAL             :: a
  REAL, DIMENSION(5) :: b
  COMMON /zajedno_smo_jaci/ i,j, a,b
  i = 1; j = 2
  a = SQRT(2.0)
  b = 0.0
  b(3) = 1.0
  CALL clan1
  PRINT *,a
END PROGRAM

SUBROUTINE clan1
  INTEGER, DIMENSION(2) :: z
  REAL, DIMENSION(6)    :: w
  COMMON /zajedno_smo_jaci/ z,w
  PRINT *,w
  w(1) = SQRT(3.0)
END SUBROUTINE
```

The diagram illustrates the sharing of variables between two COMMON blocks. In the first COMMON block, the variables `i, j` are circled in red, and `a, b` are circled in green. In the second COMMON block, `z, w` are circled in red and green respectively. A red arrow points from the red circle in the first block to the red circle in the second block, indicating that `i` and `j` are shared with `z` and `w`. A green arrow points from the green circle in the first block to the green circle in the second block, indicating that `a` and `b` are shared with `z` and `w`.

COMMON block

Common blok u C-programu se pojavljuje kao struktura imena koje ima podvučenu crtu na kraju.

FORTRAN:

```
INTEGER      :: i
COMPLEX      :: c
REAL (KIND=8) :: d

COMMON /com/ i,c,d
```

C-program

```
extern struct {
    int i;
    struct {float real, imag;} c;
    double d;
} com_;
```

Moduli umjesto COMMON bloka

```
PROGRAM zajednistvo
  USE zajedno_smo_jaci
  IMPLICIT NONE
  i = 1; j = 2
  a = SQRT(2.0)
  b = 0.0
  b(3) = 1.0
  CALL clan1
  PRINT *,a
END PROGRAM
SUBROUTINE clan1
  USE zajedno_smo_jaci
  IMPLICIT NONE
  PRINT *,a,b
  a = SQRT(3.0)
END SUBROUTINE
```

```
MODULE zajedno_smo_jaci
  IMPLICIT NONE
  INTEGER :: i,j
  REAL :: a
  REAL, DIMENSION(5) :: b
END MODULE
```

U Fortranu 90 umjesto *common* bloka, preferirani način čuvanja zajedničkih podataka između različitih programskih cjelina su **MODULE** (globalne) programske cjeline.

Za razliku od *common* bloka

- ▶ kod modula nije potrebno ponovo deklarirati tipove varijabli i zadati im imena. Varijable se mogu odmah koristiti s imenima kakva su im zadana u modulu.
- ▶ Upotrebu modula potrebno je odmah navesti kao **prvu liniju** u programskoj cjelini.
- ▶ Osim podataka, moduli mogu sadržavati i zajedničke funkcije i/ili potprograme, te definicije novih tipova varijabli.
- ▶ Moduli mogu sadržavati nizove naznačene kao `ALLOCATABLE` koji se inače ne mogu nalaziti u *common* bloku.

Primjer

```
PROGRAM zajednistvo
  USE zajednicki_podaci
  IMPLICIT NONE
  a = SQRT(2.0)
  b = 0.0
  b(3) = 1.0
  CALL clan1
  ALLOCATE (f(100))
  f = (/ (i, i=1,100) /)
  CALL clan1
  DEALLOCATE(f)
  CALL clan1
END PROGRAM
```

```
SUBROUTINE clan1
  USE zajednicki_podaci
  IMPLICIT NONE
  PRINT *,a
  a = SQRT(a*a+1)
  IF (ALLOCATED(f)) THEN
    PRINT *,'f = ',SUM(f)
  ELSE
    PRINT *,'f nije alociran'
    PRINT *,'ali b = ',SUM(b)
  END IF
END SUBROUTINE

MODULE zajednicki_podaci
  IMPLICIT NONE
  REAL :: a
  REAL, DIMENSION(5) :: b
  REAL,ALLOCATABLE,DIMENSION(:) :: f
END MODULE
```

Moduli

- ▶ Programske cjeline koje koriste module mogu koristiti druga imena od onih specificiranih u modulu. To se postiže dodatnim argumentima kod USE naredbe, gdje treba navesti `<novo_ime> => <staro_ime>`.

```
USE zajednicki_podaci, novo => b
```

- ▶ Programske cjeline ne moraju koristiti sve raspoložive podatke iz modula, već samo dio njih izlistan iza naznake `ONLY`.

```
USE zajednicki_podaci, ONLY : a,b
```

- ▶ Varijable, novi tipovi podataka, funkcije i podprogrami mogu se deklarirati javnim ili privatnim.

```
MODULE zajednicki_podaci
  INTEGER, PRIVATE   :: i,j
  REAL, PUBLIC       :: a
  REAL, DIMENSION(5) :: b   ! PUBLIC !
END MODULE
```

- ▶ Ako podaci nisu eksplicitno naznačeni da su privatni onda su javni.
- ▶ Javni znači dostupni drugim programskim cjelinama koje koriste taj modul.
- ▶ Ako definicija modula započinje s PRIVATE onda su svi podaci po pretpostavci privatni, osim onih koji su eksplicitno naznačeni da su javni.
- ▶ Privatni podaci dostupni su svim funkcijama i podprogramima unutar modula, pa i onima koje su naznačeni kao javni. Uporaba privatnih podataka moguća je posredno kroz javno dostupne funkcije i podprograme.

Izbor numeričkog modela

Specifikacija vrste realnog broja (ili cijelog broja) ovisna je o prevoditelju. Kod Intelovih prevoditelja `REAL(kind=8)` znači dvostruku preciznost, ali kod drugih prevoditelja dvostruka preciznost se postiže s `REAL(kind=2)` (npr. Silverfrostov ili NAGov prevoditelj).

- ▶ Naredbe **`SELECTED_INT_KIND`** i **`SELECTED_REAL_KIND`** pomažu da izaberemo vrst cijelih i realnih brojeva prema prema potrebama problema koji se razmatra.
- ▶ Ako nam trebaju cijeli brojevi unutar područja $\pm 10^p$, tada cjelobrojna funkcija `SELECTED_INT_KIND(p)` vraća podatak koju vrstu cijelih brojeva treba upotrebiti.
- ▶ Ako na računalu (i prevoditelju) ne postoji vrst cijelih brojeva tražene veličine, tada funkcije `SELECTED_INT_KIND(p)` vraća negativni broj.

```
INTEGER, PARAMETER :: i8 = SELECTED_INT_KIND(10)
INTEGER(KIND=i8)    :: i = 10_i8**10
```

SELECTED_REAL_KIND

Funkcija `SELECTED_REAL_KIND(p, e)` vraća vrstu realnih brojeva koji zadovoljavaju uvjete preciznosti p decimalnih mijesta, te da se eksponent može mjenjati od/do $10^{\pm e}$:

```
PROGRAM izbor
  INTEGER, PARAMETER :: i8 = SELECTED_INT_KIND(10)
  INTEGER, PARAMETER :: r8 = SELECTED_REAL_KIND(15,30)
  INTEGER (kind=i8) :: i = 8_i8**10
  REAL (kind=r8) :: r = 1.2345678901234567890123456789_r8
  PRINT *, 'tip cijelih brojeva = ', i8
  PRINT *, 'tip realnih brojeva = ', r8
END PROGRAM
```

```
tip cijelih brojeva =      8      INTELov prevoditelj
tip realnih brojeva =      8
```

```
tip cijelih brojeva =  4
tip realnih brojeva =  2      NAGov prevoditelj
```

Primjer - SELECTED_{INT,REAL}_KIND

```
PROGRAM vrste_podataka

  INTEGER :: i,j
  PRINT *, 'Cijeli brojevi (eksponent,tip)'
  DO i = 1,20
    PRINT *, i, SELECTED_INT_KIND(i)
  END DO

  PRINT *
  PRINT *, 'Realni brojevi (decimale,eksponent,tip)'
  j = 20
  DO i = 1,40
    PRINT *, i, j, SELECTED_REAL_KIND(i, j)
  END DO

END PROGRAM
```

Pomoću ovog programa možemo ustanoviti koje vrste podataka (i preciznosti) pojedini prevoditelj nudi.

Ostale funkcije za ispitivanje brojeva

Funkcija	Tipovi podataka	Opis
KIND(a)	cijeli, realni	numerička oznaka podvrste
HUGE(a)	cijeli, realni	najveći broj unutar podvrste
TINY(a)	realni	najmanji broj unutar podvrste
EPSILON(a)	realni	broj zanemariv naspram jedan
PRECISION(a)	realni	točnost u broju decimalnih mjesta
SPACING(a)	realni	apsolutni razmak između brojeva
RRSPACING(a)	realni	relativni razmak između brojeva
NEAREST(a,b)	realni	najbliži broj
...

Konverzije među numeričkim podacima

- ▶ Realne brojeve pretvaramo u cijele naredbama

```
i = INT (a,KIND=i4)
```

```
j = INT (a,KIND=i8)
```

- ▶ Realne brojeve pretvaramo u **najbliže** cijele naredbama

```
i = NINT (a,KIND=i4) !! zaokruživanje
```

```
j = NINT (a,KIND=i8)
```

- ▶ Cijele brojeve pretvaramo u realne

```
a = REAL(i,KIND=r4)
```

```
b = REAL(i,KIND=r8)
```

Operacije na bitovima cijelih brojeva

Funkcija	Opis
BTEST (i,j-1)	Provjerava j-ti bit BTEST(8,3) -> da
IAND (i,j)	AND na bitovima IAND(10,7) -> 2
IOR (i,j)	OR na bitovima IOR(10,7) -> 15
IEOR (i,j)	ekskluzivni OR na bitovima IEOR(10,7) -> 13
IBCLR(i,j-1)	brisanje j-ti bit IBCLR(10,1) -> 8
IBSET(i,j-1)	postavlja j-ti bit IBSET(10,2) -> 14

Operacije na bitovima cijelih brojeva

Funkcija	Opis
IBITS(i,j-1,l)	izdvaja bitove dužine l IBITS(1234567,1,3) -> 3
ISHFT(i,j)	pomicanje bitova za j mjesta ISHFT(7,1) -> 14, ISHFT(7,-1) -> 3
ISHFC(i,j,k)	cirkularni pomak bitova za j mjesta dužine k ISHFTC(7,-1,8) -> 131

Ostale funkcije

Postoji još mnoštvo drugih funkcija.

- ▶ One koje operiraju na nizovima: izravnavaju nizove, spajaju nizive, rade pomicanje članova niza u pojedinim dimenzijama, cirkularno ili ne, itd.
- ▶ Transferiraju podatke kao bitove između sasvim različitih podataka
- ▶

IARGC i GETARG

Svi prevoditelji, osim standardnih fortranskih naredbi, imaju i dodatne naredbe koje su implementirane ili kao biblioteke ili moduli. Tipično njihova implementacija povezana je operacijskim sustavom. Od zanimljivijih obaraditi ćemo **IARGC** i **GETARG**

- ▶ Funkcija IARGC sadrži broj argumenata/opcija na komandnoj liniji kod pokretanja izvršnog programa.
- ▶ Podprogram GETARG kopira sadržaj pojedinih argumenata u varijablu tipa CHARACTER.

Primjer

```
PROGRAM komandna_linija
!  
! ifort -fpp -DINTEL prog.F90 ....  
!  
#ifdef NAG  
USE F90_UNIX_ENV, ONLY : IARGC,GETARG  
#endif  
#ifdef PGI  
INCLUDE 'lib3f.h'  
#endif  
  
INTEGER :: i,imx  
CHARACTER (LEN=80) :: arg  
REAL :: a=1.0, b=2.0  
.....  
  
END PROGRAM
```

Primjer

```
.....
  imx = IARGC()
  i = 1
  DO WHILE ( i <= imx )
    CALL GETARG(i,arg)
    SELECT CASE (arg(1:2))
      CASE ('-a','-A')
        i = i + 1
        CALL GETARG(i,arg)
        READ (arg,'(g)') a
      CASE ('-b','-B')
        i = i + 1
        CALL GETARG(i,arg)
        READ (arg,'(g)') b
    END SELECT
    i = i + 1
  END DO
  PRINT *, 'a = ', a
  PRINT *, 'b = ', b
END PROGRAM
```

```
promot> ./komandna
```

```
a = 1.000000
```

```
b = 2.000000
```

```
promot> ./komandna -a 1.2345
```

```
a = 1.234500
```

```
b = 2.000000
```

```
promot> ./komandna -B -1.2E-5
```

```
a = 1.000000
```

```
b = -1.2000000E-05
```

```
promot> ./komandna -A 0.0 -b 23.0
```

```
a = 0.0000000E+00
```

```
b = 23.000000
```