Proseduurit

Proseduuri voi olla

1) Funktio, joka palauttaa jonkin arvon:

```
real function sinc(x)
real x
sinc = sin(x)/x
end

...
y = sinc(1.5)
```

2) Aliohjelma, jonka toiminta perustuu sivuvaikutuksiin: aliohjelma muuttaa parametrejaan tai globaaleja muuttujia, tulostaa jotakin jne.

Aliohjelmaa kutsutaan `call`-lauseella.

```
subroutine store(x, n)
real x(:)
integer n
integer i
do i=1,n
  write(6,*) i,x(i)
end
return
end

...
call store (table,n)
```

Proseduurin paikalliset muuttujat

- vain aliohjelman käytössä; kutsuva ohjelma ja muut aliohjelmat eivät pääse käsiksi
- dynaaminen varaus: tila varataan pinosta joka kutsukerralla erikseen (nykyisin oletus useimmissa ympäristöissä); aliohjelmasta poistuttaessa muuttuja tuhoutuu.
- staattinen varaus: tila varataan vain kerran kiinteältä muistialueelta; kun aliohjelmaa kutsutaan uudestaan, staattisesti varatulla muuttujalla on yhä vanha arvo.

```
real, save :: x
```

- taulukkoja ja isoja tietorakenteita ei yleensä talleteta pinoon; pinossa vain osoitin tietorakenteeseen
- jos usein kutsuttavassa aliohjelmassa suuria dynaamisesti varattavia taulukoita, toiminta hidastuu

Globaalit muuttujat

- paikallisten muuttujien lisäksi aliohjelma näkee aliohjelmille yhteisiä muuttujia
- Pascal, C: globaalit muuttujat määräytyvät ohjelman kirjoitusasun perusteella
- Fortran 77: globaalit muuttujat sijoitettava common-alueelle, joka määriteltävä kaikissa sitä käyttävissä aliohjelmissa
- Fortran 90: molemmat em. vaihtoehdot

```
program esimerkki
real x, y, z, u
x=0.1
y=1.0
u=f(x)
write (*,'(4F10.4)') x,y,z,u

contains

real function f(x)
  real x, y
  y=x**2
  z=sin(x)
  f=z * y
end function

end program esimerkki
```

Tulostus:

```
  0.1000   1.0000   0.0998   0.0010
```

Pääohjelman muuttujat `x`, `y`, `z`, `u` ovat globaaleja.

Funktion `y` on paikallinen, joten sen muuttaminen ei vaikuta kutsuvan ohjelman muuttujaan `y`.

Funktion sisällä ei ole määritelty muuttujaa `z`, joten se viittaa globaaliin muuttujaan `z`. Tämä on funktion sivuvaikutus, eikä välttämättä toivottu.

Proseduurien parametrit (argumentit)

- Välittävät tietoa aliohjelman ja kutsuvan ohjelman välillä
- Arvoparametri: kutsuva ohjelma laskee parametrin arvon, jonka aliohjelma sijoittaa omaan paikalliseen muuttujaan
    - todellinen parametri voi olla muuttuja, vakio tai lauseke
    - aliohjelma voi käyttää parametria tavallisen paikallisen muuttuja tavoin
    - vaikka aliohjelma muuttaisi parametria, muutos ei vaikuta kutsuvan ohjelman muuttujiin
    - ei voi välittää tietoa aliohjelmasta kutsuvaan ohjelmaan
    - C:ssä kaikki parametrit arvoparametreja
- Viiteparametri: välitetään todellisen parametrin osoite
    - parametrin muuttaminen muuttaa myös kutsuvan ohjelman muuttujaa $\Rightarrow$ vaarallinen!
    - Fortranissa kaikki parametrit viiteparametreja
    - Fortran 90: voidaan määritellä, mihin suuntaan parametrilla välitetään tietoa

```
x=1
call sub(x)
write(6,*) x  ! 2
call sub(1)   ! ??
call sub(x+y)  ! ??

subroutine sub(x)
real x
x=x+1
end

subroutine sub (x)
real, intent(in) :: x
x=x+1    ! kielletty!
end
```

Intent-attribuutti:

`in`: aliohjelma ei saa muuttaa parametrin arvoa; todellinen argumentti voi olla muuttuja, vakio tai lauseke.

`out`: todellinen argumentti saa olla vain muuttuja; aliohjelman on annettava sille arvo, mutta muuttujaa ei saa käyttää esim. sijoituslauseen oikealla puolella.

`inout`: ei käyttörajoituksi, mutta todellisen argumentin oltava muuttuja.

Intent-attribuutti puuttuu: ei mitään rajoituksia.

Positionaalinen parametri: paikka määrää, mitä muodollista parametria todellinen parametri vastaa.

Avainsanaparametri (keyword): järjestys vapaa, parametri tunnistetaan nimen perusteella.

```
subroutine huu(x,y,z)
real x, y, z
...
end

real a, b, c
call huu(a, b, c)
call huu(a, z=b, y=1.0)
call huu(z=b, x=a, y=c)
call huu(x=a, b, c)     ! kielletty
```

Valinnainen parametri voi puuttua:

```
subroutine haa(x,y,z)
real x
real, optional :: y,z
real zz=42.0
..
if (present(z)) zz=z

end
```

Sopii tilanteeseen, jossa muuttujalla yleensä järkevä oletusarvo.

```
call haa(a)
call haa(a,0.1)
call haa(a,b,c)
call haa(a,z=1.5)
call haa(x=a,z=1.5,y=1.0)
```

Funktiot

Funktio palauttaa laskemansa arvon.

Funktiolla voi olla myös sivuvaikutuksia; niitä tulisi välttää.

Funktiolle on aina määriteltävä tyyppi.

```
real function sinc(x)
real, intent(in) :: x
if (abs(x) < epsilon)
  sinc=1.0
else
  sinc = sin(x)/x
endif
end
```

Funktion arvo voi olla mikä tahansa muuttujatyyppi

```
real x(3),y(3)
x(:) = (/ 1, 2, 3 /)
y=inv(x,3)

...

function inv(x,n)
 real, dimension(:), intent(in) :: x
 real, dimension(size(x)) :: inv
 integer n
 inv=x(n:1:-1)
end
```

Funktio voi olla rekursiivinen:

```
recursive function fibonacci(n)
integer fibonacci, n
if (n<=2)
  fibonacci=1
else
  fibonacci=fibonacci(n)+fibonacci(n-1)
endif
end
```

Melko vähän käyttöä numeerisissa tehtävissä.

Sopii esimerkiksi puiden käsittelyyn:

```
program tree
integer, parameter :: n=5
integer left(n), right(n), data(n)
left(:)  = (/ 2, 4, 5, 0, 0 /)
right(:) = (/ 3, 0, 0, 0, 0 /)
data(:)  = (/ 10, 20, 30, 40, 50 /)
writetree (1)

contains
recursive subroutine writetree(n)
 integer n
 if (left(n) > 0) writetree(left(n))
 write (6,*) data(n)
 if (right(n) > 0) writetree(right(n))
end
end program
```

Yksinkertainen lajittelu:

```
real x(n)

do i=1,n-1
  do j=i+1,n
    if (x(j) < x(i)) then
      t=x(i) ; x(i)=x(j) ; x(j)=t
    end if
  end do
end fo
```

Ajankäyttö verrannollinen n^2. Ei kannata, kun on parempiakin menetelmiä.

Quicksort (rekursiivinen) yleensä nopea, mutta huono jo valmiiksi järjestyksessä olevalle aineistolle.

Kekolajittelu (heapsort) takaa, että aika korkeintaan $n \log n$.