Een korte samenvatting van enkele FORTRAN opdrachten

Inhoud

1 Introductie

Tijdens dit practicum zult u werken met het pakket SEPRAN en met Nedit voor het maken van files. SEPRAN is in FORTRAN geschreven en voor het practicum is het nodig zelf een element subroutine te schrijven in die taal.
Het is niet de bedoeling van deze handleiding om u FORTRAN te leren, slechts die aspecten die nodig zijn voor het practicum worden behandeld. Verder wordt ervan uitgegaan dat u al kennis heeft van pascal.

2 De structuur van een FORTRAN programma

Een FORTRAN programma bestaat uit een hoofdprogramma, eventueel gevolgd door een of meer subprogramma's.
Voor het practicum is het handig deze alle in dezelfde file te zetten.
De structuur van een programma wordt gegeven in het volgende schema:

```
Hoofdprogramma:
    program statement (bijvoorbeeld program SEPCOMP)
    declaraties
    statements
    end
Subprogramma:
    subroutine ...  of function ...
    declaraties
    statements
    end
```

NB. In tegenstelling tot Pascal is recursief programmeren in FORTRAN niet mogelijk. Variabelen die in het hoofdprogramma gebruikt worden, zijn alleen in het hoofdprogramma bekend en niet in de subroutines. Dit geldt ook omgekeerd, variabelen in de subroutines zijn alleen lokaal bekend. Uitwisseling van variabelen tussen hoofdprogramma en subroutines kan dus in principe alleen door de parameterlijst.
De volgende regels zijn van belang:

1. Indien een statement niet op een regel past dan kan de statement op de volgende regel(s) worden vervolgd. Dan moet op het eind van de regel een `&` worden gezet om aan te geven dat de statement doorloopt.

2. Slechts één statement per regel is toegestaan

3. Alle tekst achter het uitroepteken (`!`) wordt beschouwd als commentaar.

3 Datatypen, variabelen en declaraties

Namen van variabelen in FORTRAN bestaan uit letters en cijfers, het eerste symbool moet altijd een letter zijn.
FORTRAN kent als type `integer`, `real`, `double precision`, `complex`, `double complex` en `logical` plus nog wat types die hier niet van belang zijn.
Hoewel FORTRAN formeel niet eist dat alle variabelen expliciet gedeclareerd worden is dat voor het practicum wel verplicht. Men kan dit forceren door als eerste statement in het programma, voor alle andere declaraties op te nemen:

```
      implicit none
```

Dit moet in ieder subprogramma herhaald worden. In SEPRAN moeten alle real variabelen als `double precision` worden gedeclareerd, en alle complex variabelen als `double complex`. Op deze

wijze bestaat ieder reeel getal uit ongeveer 16 decimalen.
Een variabele van het type `logical` is het zelfde als een variabele van het type Boolean in Pascal. Hij kan alleen de waarde `.true.` of `.false.` hebben.
Typische voorbeelden van declaraties zijn:

```
implicit none
integer i, loop, index
double precision alpha, beta, gamma
double complex c1, c2
```

Arrays worden gedeclareerd door in de expliciete declaratie de dimensies mee te geven. Als de ondergrens niet wordt opgegeven dan wordt als ondergrens 1 verondersteld. De grenzen mogen uitsluitend gehele getallen of integer expressies zijn.
Voorbeelden:

```
implicit none
integer i(1:100), j(-3:1,1:2), k(100)
double precision alpha(200,30), beta(-7:-5,1:9,10)
double complex c1(8), c2(5,4)
```

In FORTRAN is een konstante een getal van het type real of integer, of de logische waarde `.true.` of `.false.` of een tekst. Een tekst is rij karakters tussen apostrofes, bijvoorbeeld 'tekst'.
Een real getal wordt aangegeven met een punt, bijvoorbeeld 3.1415 of met een exponent bijv. 3.1415e0 of 3.1415d0. Indien geen exponent of de exponent E wordt gebruik dan is de konstante enkele precisie (7 decimalen). Wordt de D als exponent gebruikt dan is de konstante dubbele precisie (16 decimalen). Voorbeelden zijn: `1d-3` ($=10^{-3}$),
`4.1d7` ($=4.1 * 10^7$).
Een integer getal is een getal zonder punt. Voorbeeld 1014

Arrays worden als volgt gebruikt:

```
x=a(i)
b=g(i,j)+c(i)
```

4 Expressies-volgorde van uitwerking

De volgende tabel geeft een vergelijking tussen operatoren in Pascal en Fortran.

	Pascal	Fortran
rekenkundige operatoren		
- machtsverheffen		**
- vermenigvuldiging	*	*
- deling	/	/
- gehele deling	div, mod	
- optelling	+	+
- aftrekking	-	-
relatie operatoren		
- kleiner	<	<
- kleiner of gelijk	<=	<=
- groter	>	>
- groter of gelijk	>=	>=
- gelijk	=	==
- niet gelijk	<>	/=
logische operatoren		
- niet	not	.not.
- en	and	.and.
- of	or	.or.

De volgorde waarin expressies worden uitgevoerd is als volgt:

1. **
2. * en /
3. + en -
4. <, <=, ==, /=, >= en >
5. .not.
6. .and.
7. .or.

Binnen deze niveau's vindt de evaluatie van links naar rechts plaats.
De volgorde kan worden gewijzigd door het gebruik van haakjes.
Voorbeeld:

```
program volgorde
implicit none
integer n, nmax
logical konv, uitkomst
konv = .true.
n = 5
nmax = 20
uitkomst = .not. konv .and. 2*n .lt. nmax
print *, 'uitkomst = ', uitkomst
end
```

Als we voorgaand programma uitvoeren dan wordt het resultaat van de expressie als volgt geëvalueerd:

1. 2*n, resultaat r1 = 10
2. r1 .lt. nmax, resultaat r2 = .true.
3. .not. konv, resultaat r3 = .false.

4. r3 .and. r2, resultaat: .false.

Het programma print dus: uitkomst = .false.

5 Standaard funkties

De volgende standaard funkties (intrinsic functions in Fortran) zijn beschikbaar:
`exp, log` (=ln), `log10` (= ^{10}log), `sin, cos, tan, atan` (=arctan)
`sqrt, abs, min, max`
Er zijn uiteraard meer standaard funkties.
Als het argument enkele precisie is, is het resultaat ook enkele precisie. Een dubbele precisie argument geeft een dubbele precisie resultaat.
Dus exp(1.) is enkele precisie en exp(1d0) is dubbele precisie.

6 Toekenningsopdracht

De assignment statement in fortran heeft de vorm

```
variabele = expressie
```

Als de variabele van het type LOGICAL is, moet de expressie logisch zijn. Als de variabele van het type integer is moet de expressie van het type integer zijn of er wordt afgerond in het geval van een real expressie. Als de variabele van het type real is mag de expressie zowel van het type real als integer zijn.
Deling van twee integers (ook in expressies) is in feite heling; het resultaat is integer en wordt niet afgerond maar afgekapt.
Voorbeelden

```
7/3 wordt 2
3d0*(1/2) wordt 0
```

Bij machtsverheffen maakt het verschil of de exponent real of integer is. Bij een integer exponent wordt de machtsverheffing als herhaaldelijke vermenigvuldiging uitgevoerd, bij een real exponent m.b.v. een logarthme en exponent funktie. Het gevolg is bijvoorbeeld dat -2d0**2 gelijk is aan 4, maar 2d0 ** 2d0 een foutmelding geeft.

7 Herhalingsopdrachten

- De herhalingsopdracht

  ```
  Voor i := begin (stap) eind doe
     statements
  ```

  heeft in Fortran de vorm:

  ```
  do i = begin, eind, stap
     statements
  end do
  ```

  begin, stap en eind moeten van het type integer zijn.
  Als stap = 1, dan mag hij worden weggelaten.
  Voorbeeld:

```
som = 0
do i = 1, 10
   read *, getal
   som = som + getal
end do
```

- herhalingsopdracht

  ```
  Zolang Voorwaarde doe
     statements
  ```

  heeft in Fortran de vorm:

  ```
  do while ( Voorwaarde )
     statements
  end do
  ```

  Voorbeeld:

  ```
  som = 0
  read *, getal
  do while ( getal .ge. 0 )
     som = som + getal
     read *, getal
  end do
  ```

- keuzeopdrachten de `if then else` constructie in fortran heeft een van de volgende vormen:

  ```
  if ( Voorwaarde ) then
     statements
  end if
  ```

  of

  ```
  if ( Voorwaarde1 ) then
     statements
  else if ( Voorwaarde2 ) then
     statements
  else
     statements
  end if
  ```

  Voorbeelden:

  ```
  if ( i == 0 ) then
     a = a+1
     print *, a
  end if
  ```

  of

  ```
  if ( i == 0 ) then
     read *, a
     print *, a
  else
     read *, b
     print *, b
  end if
  ```

of

```
if ( i == 1 ) then
   a = a+1
   print *, a
else if ( i == 2 ) then
   b = b+1
   print *, b
else
   c = c+1
   print *, c
end if
```

8 Subprogramma's

Een subprogramma in FORTRAN kan zijn:

- Een subroutine
- Een function

Een subroutine in FORTRAN is een zelfstandig programma dat vanuit het hoofdprogramma of vanuit een ander subprogramma kan worden aangeroepen door een CALL statement. De statement `call piet` in het hoofdprogramma roept de subroutine met de naam `piet` aan. De subroutine zelf begint met het woord `subroutine`, dan de naam van de subroutine, hier `piet`, en dan tussen haakjes de formele parameters. De subroutine-tekst eindigt met het woord `end`. (Opmerking: in het voorbeeld hierboven heeft de subroutine `piet` geen parameters).
De parameters worden gebruikt in de zin van 'call by reference'. 'Call by value' bestaat niet in Fortran.

De variabelen in een subroutine die niet in de parameterlist (of common block) voorkomen zijn lokaal, dus alleen in de subroutine bekend. Waarden worden vanuit het aanroepende programma (hoofdprogramma of subprogramma) naar de subroutine overgedragen via het parameter mechanisme. Dit geldt ook andersom (overdragen van waarden van subroutine naar aanroepend programma).
Subroutines mogen ook andere Fortran subroutines aanroepen, maar niet recursief.
In FORTRAN bestaat ook nog het FUNCTION subprogramma. De function begint met het woord `function`, dan volgt de naam van de function en tussen haakjes de formele parameters. De function eindigt net als de subroutine met het woord `end`. Binnen de function subroutine moet de naam van de function een waarde krijgen.

Declaraties binnen een subroutine of function hebben zowel betrekking op de lokale variabelen, de variabelen in de parameter lijst en op de naam van de function. De function moet ook gedeclareerd worden in het aanroepende programma (of subprogramma).
Arrays in de parameterlijst moeten in het subprogramma worden gedeclareerd als arrays. Voor vaste arrays gelden dezelfde regels als in het hoofdprogramma. Daarnaast mogen in de parameterlijst ook nog variabele arrays voorkomen waarvan de lengte met een integer expressie wordt aangegeven. Hierbij mogen variabelen uit de parameterlist worden gebruikt.
Daarnaast kan voor variabele lengte arrays ook nog de optie `*` worden gebruikt voor de bovengrens in de laatste dimensie van het array. (NB. alleen voor formele parameters van functions en subroutines).
Voorbeeld:

```
double precision a(3:*), b(*), c(1:n,*), d(4,5,*)
```

Belangrijk is dat de grenzen van de arrays in het hoofdprogramma **exact** hetzelfde moeten zijn als in de subroutine. Deze restrictie geldt niet voor de bovengrens van de laatste dimensie.
Voorbeeld:

```
program voorbeeld
implicit none
double precision a(100), x1(100), y1(100), gemiddelde, v
integer n, i
read *, n
read *, (a(i),i=1,n)
v = gemiddelde(a,n)
read *, n
read *, (x1(i),i=1,n)
call copy ( x1, y1, n )
end

function gemiddelde ( a, n )
implicit none
integer n, j
double precision a(n), gemiddelde, som
som = 0d0
do j = 1, n
   som = som + a(j)
end do
gemiddelde = som / n
end

subroutine copy ( x, y, n )
implicit none
integer n, i
double precision x(*), y(*)
do i = 1, n
   y(i) = x(i)
end do
end
```

NB. De FORTRAN compiler test de arraygrenzen niet. Overschrijden van de arraygrenzen heeft meestal onverwachte gevolgen met onbegrijpelijke foutmeldingen.

Een function wordt aangeroepen door het gebruik van de naam met eventuele parameters, in een expressie.
Voorbeeld:

```
v = gemiddelde(a,100)
```

Een subroutine wordt aangeroepen door een call statement.
Voorbeeld:

```
call copy ( x1, y1, 100 )
```

9 Invoer en uitvoer

FORTRAN kent vele faciliteiten voor lezen van, of schrijven naar respectievelijk invoer- en uitvoermedia. Slechts het hoogst noodzakelijke wordt hier besproken.
De leesopdracht luidt:

```
read *, <variabele list>
```

De <variabele list> is een rij van variabelen gescheiden door komma's. De in te lezen getallen moeten worden gescheiden door èèn of meer spaties, een komma of einde van een regel. Elke nieuwe READ opdracht leest van een nieuwe regel.
Voorbeeld

```
read *, a, b, c
```

De <variabele list> kan ook een zgn. implied do-statement zijn.
Voorbeeld

```
read *, (a(i), i = 1, 100)
```

De variabelen a(1) t/m a(100) worden ingelezen.

De schrijfopdracht luidt:

```
read *, <output list>
```

De <output list> is een rij van variabelen gescheiden door komma's.
Voorbeeld

```
print *, 'a = ', a
```

De <output list> kan ook weer een implied do-statement zijn.
Voorbeeld

```
print *, (a(i), i = 1, 100)
```

De implied do-statement wordt gebruikt bij het lezen en schrijven van een array.
Voorbeeld

```
program voorbeeld
implicit none
integer i, j
double precision a(1:100,-2:2)
do i = 1, 100
   read *, (a(i,j),j=-2,2)
end do
print *, 'Het array A is:'
do i = 1, 100
   print *, (a(i,j),j=-2,2)
end do
```

De invoerfile kan er als volgt uitzien:

```
1 2 4 2 1         eerste regel in de file (= eerste rij van a)
1 2 4 2 1         tweede regel in de file (= tweede rij van a)
  .
1 2 4 2 1         vierde regel in de file (= vierde rij van a)
  .
  .
  .
```

De uitvoer wordt dan:

```
Het array A is:
 1 2 4 2 1
 1 2 4 2 1
   .
 1 2 4 2 1
   .
   .
   .
```

10 Enige veel voorkomende fouten

- De parameterlijst in de aanroep van een subroutine is incorrect. Dit kan zijn een verkeerd aantal parameters of parameters van het verkeerde type. De meldingen van de compiler zijn meestal onduidelijk of worden soms geheel achterwege gelaten.
  Bijvoorbeeld: een integer (meestal 0 of 1) wordt als parameter meegegeven i.p.v. een real of double precision. I.p.v. 0 of 1 moet dan 0d0 of 1d0 worden meegegeven.

- Arraygrenzen worden overschreden. Hierop volgt geen of een zeer onduidelijke foutmelding. Dus steeds zelf goed checken.
  Typische foutmeldingen zijn bijvoorbeld:

  ```
  Segmentation fault - core dumped
  Bus error
  ```

- Deling van 2 integers is in feite een heling