



Denne guide er oprindeligt
udgivet på Eksperten.dk

Fortran 77 for C/C++/Java/C# programmører

**Denne artikel vil give en kort introduktion til programmerings
sproget Fortran 77.**

**Den forudsætter nogen erfaring med mindst et af sprogene C/C++/Java/C#, fordi
artiklen forklarer hvordan man gør tingene i Fortran 77 ikke hvordan man
programmerer.**

Skrevet den **02. Feb 2009** af **arne_v** i kategorien **Programmering / Andre** | ★★★★★

Historie:

V1.0 - 20/07/2008 - original

Standarder

Fortran er det ældste high level language.

Første version udkom i 1957.

Sproget er standardiseret via ANSI/ISO:

- Fortran 66
- Fortran 77
- Fortran 90
- Fortran 95
- Fortran 2003

De traditionelle store platforme for Fortran var forskellige IBM systemer
og VMS VAX (oprindeligt Digital - idag ejet af HP).

Artiklen vil beskrive Fortran 77. Fortran 77 var den sidste meget udbredte
Fortran version. De nyere versioner har stået i skyggen af nyere sprog.

Mine eksempler vil gå lidt udover Fortran 77 standarden og inkludere
de mest gængse extensions. Jeg vil teste eksemplerne med GCC g77 på Windows
og HP Fortran på VMS Alpha.

Artiklen giver nok mest hvis man har en Fortran compiler man kan lege med
mens man læser artiklen.

Windows: hent GCC MinGW

Linux: GCC skulle gerne være installeret

generel program opbygning

Fortran programmer har en helt anderledes opbygning end nyere programmerings
sprog. Formatet er ikke free format men er linie orienteret og positions afhængigt.

position 1-5 bruges til numeriske labels

position 6 bruges til continuation markering
(et tegn betyder at linien er en fortsættelse af forrige linie)

Position 7-72 bruges til kode.

Position 72- er kommentarer.

(I gamle dage skrev man linienumre derude, fordi så kunne man sortere hulkortene efter dem)

Et program starter med PROGRAM statement og slutter med END statement.

Så er vi vist klar til helloworld.for:

```
PROGRAM HELLOWORLD
WRITE(*,*) 'Hello world !'
END
```

WRITE(*,*) vil blive forklaret senere i artiklen.

Man ser at al koden starter i position 7.

Ofte skriver man Fortran kode i ren uppercase, men det er ikke nødvendigt.
(da Fortran blev lavet var der computer systemer som kun havde store bogstaver)

Kommentarer laves ved at skrive et C i position 1. Det er valgfrit hvilket tegn man bruger som continuation markering i position 6, men det anbefales at bruge +.

Nu kan vi lave en helloworld.for som viser disse features:

```
C
C Dette er et hello world program i Fortran
C
    PROGRAM HELLOWORLD
    WRITE(*,*)
+      'Hello world !'
    END
```

Bemærk at spaces er insignificante i Fortran.

```
MYVAR=1
```

kan skrives som:

```
M Y V A R = 1
```

men det bør man naturligvis aldrig gøre.

simple typer og konstanter

Fortran 77 standarden definerer data typerne:

- INTEGER
- LOGICAL
- CHARACTER
- REAL
- DOUBLE PRECISION
- COMPLEX

Men det er de facto standard at man kan kvalificere typerne med at angive deres størrelse i bytes.

Så:

C/C++/Java/C#	Fortran
-----	-----
short	INTEGER*2
int	INTEGER*4
long	INTEGER*8
bool/boolean	LOGICAL*4
string/String	CHARACTER*n
float	REAL*4
double	REAL*8
-	REAL*16
-	COMPLEX*8
-	COMPLEX*16

Bemærk at CHARACTER i Fortran er en fast længde streng ikke en variabel længde streng som i de andre sprog.

Den opmærksomme læser kan nu begynde at se hvorfor Fortran er et oplagt sprog til tal knuseri i floatng point og komplekse tal, mens at behandling af tekst er noget besværligt.

Vi laver nu et simpelt eksempel med variabel erklæringer og nogle værdi tildelinger.

```
PROGRAM VARASGN
  INTEGER*2 V1
  INTEGER*4 V2
  INTEGER*8 V3
  LOGICAL*4 V4
  CHARACTER*16 V5
  REAL*4 V6
  REAL*8 V7
C    REAL*16 V8
  COMPLEX*8 V9
```

```

COMPLEX*16 V10
V1=123
V2=123
V3=123
V4=.TRUE.
V5='Dette er en test'
V6=123.456
V7=123.456D0
C    V8=123.456Q0
V9=(1.2,3.4)
V10=(1.2D0,3.4D0)
WRITE(*,*) V1
WRITE(*,*) V2
WRITE(*,*) V3
WRITE(*,*) V4
WRITE(*,*) V5
WRITE(*,*) V6
WRITE(*,*) V7
C    WRITE(*,*) V8
WRITE(*,*) V9
WRITE(*,*) V10
END

```

(REAL*16 ikke understøttet af g77)

Bemærk at det er valgfrit om man vil erklære variable i Fortran. Ikke erklærede variable får type efter første bogstav i variabel navnet:

- navne der starter med I-N er INTEGER
- navne der starter med andet er REAL

Denne feature kan disables med IMPLICIT NONE statement. Det anbefales at bruge denne option !

Man kan erklære flere variable af samme type i samme statement.

Man kan erklære konstanter via PARAMETER statement.

Vi ser lige de nye features:

```

PROGRAM VARASGN
IMPLICIT NONE
INTEGER*4 A,B,C,ZERO
PARAMETER(ZERO=0)
A=1
B=2
C=3
WRITE(*,*) A,B,C,ZERO
END

```

Der mangler stadig to features nemlig COMMON og DATA statements, men de kommer

under subroutiner/funktioner og arrays.

operatorer

Fortran har alle de normale operatorer og basale funktioner.

C/C++/Java/C#	Fortran
-----	-----
+	+
-	-
*	*
/	/
%	MOD funktion
pow/Pow funktion	**
abs/fabs/Abs funktion	ABS funktion
floor/Floor funktion	AINF funktion
round/Round funktion	NINT funktion
min/Min funktion	MIN funktion
max/Max funktion	MAX funktion
==	.EQ.
!=	.NE.
<	.LT.
<=	.LE.
>	.GT.
>=	.GE.
&&	.AND.
	.OR.

Derudover har Fortran en hel stribe matematiske funktioner:

EXP, LOG, LOG10, SQRT
COS, SIN, TAN, ACOS, ASIN, ATAN for radian
COSD, SIND, TAND, ACOSD, ASIND, ATAND for grader

Bemærk at disse funktioner eksisterer for alle præcisioner af floating point.

Fortran har ogsaa nogle enkelte funktioner til brug på characters:

LEN(S) giver længden af S
INDEX(S,'ABC') finde 'ABC' i S
S(3:4) laver en substring med bogstav 3-4

Vi tager et eksempel:

```
PROGRAM OPS
INTEGER*4 N,M
```

```

REAL*8 X
CHARACTER*16 S
N=7
M=5
WRITE(*,*) N+M,N-M,N*M,N/M,MOD(N,M)
WRITE(*,*) N.EQ.M,N.NE.M,N.LT.M,N.GT.M
X=2.5
WRITE(*,*) SQRT(X**2),LOG(EXP(X)),LOG10(10**X)
WRITE(*,*) ACOS(0.0),ASIN(1.0)
C   WRITE(*,*) COSD(90.0),SIND(90.0)
S='Dette er en test'
WRITE(*,*) LEN(S),INDEX(S,'er'),S(7:8)
END

```

(trigonometriske funktioner med grader er ikke understøttet af g77)

conditional statements

Fortran har 5 forskellige varianter af IF statement.

De første 3 er ret simple og svarer til if i andre sprog:

```
IF(betingelse) singlestatement
```

```

IF(betingelse) THEN
    multistatements
ENDIF

```

```

IF(betingelse) THEN
    multistatements
ELSE
    multistatements
ENDIF

```

Så har man en til at hoppe til 3 forskellige steder alt efter om en værdi er negativ, nul eller positiv:

```
IF(value) negativlabel,nulllabel,positivlabel
```

Og så har man en til at hoppe til forskellige labels alt efter om en værdi er 1,2,3,4,... (svarer til en switch som kun kan tage fortløbne integer værdier):

GOTO(label1,label2,label3,label4) value

Der er vist brug for et eksempel:

```
PROGRAM COND
INTEGER*4 V
V=2
C single statement IF
  IF(V.GT.1) WRITE(*,*) 'V > 1'
C block statement IF ELSE
  IF(V.GT.1) THEN
    WRITE(*,*) 'V > 1'
  ELSE
    WRITE(*,*) 'V <= 1'
  ENDIF
C arithmetic IF
  IF(V) 100,200,300
100  WRITE(*,*) 'V < 0'
    GOTO 400
200  WRITE(*,*) 'V = 0'
    GOTO 400
300  WRITE(*,*) 'V > 0'
400  CONTINUE
C computed GOTO
  GOTO (500,600,700,800) V
500  WRITE(*,*) 'V = 1'
    GOTO 900
600  WRITE(*,*) 'V = 2'
    GOTO 900
700  WRITE(*,*) 'V = 3'
    GOTO 900
800  WRITE(*,*) 'V = 4'
    GOTO 900
900  CONTINUE
END
```

(der bruges også simple GOTO statements, men det er vel indlysende hvordan de fungerer)

Den opmærksomme læser har nok opdaget nu, at Fortran kan blive noget forfærdeligt spaghetti kode. Det kan imidlertid laves rimeligt struktureret, hvis man vil.

Løkker

Fortran har standard kun en enkelt løkke type nemlig DO løkken som svarer til hvad man i andre sprog kalder en for løkke (i den simple udgave af for løkken):

```
      DO label counter=start,end  
        multistatements  
label CONTINUE
```

```
      DO label counter=start,end,step  
        multistatements  
label CONTINUE
```

Men de fleste Fortran 77 dialekter har også en DO WHILE løkke:

```
      DO WHILE expression  
        multistatement  
      ENDDO
```

Lad os se et eksempel:

```
      PROGRAM LOOPS  
      INTEGER*4 I  
      DO 100 I=1,5  
        WRITE(*,*) I  
100    CONTINUE  
      DO 200 I=1,5,2  
        WRITE(*,*) I  
200    CONTINUE  
      I=1  
      DO WHILE(I.LE.5)  
        WRITE(*,*) I  
        I=I+2  
      ENDDO  
      END
```

Hvis disse løkker ikke er tilstrækkelige (fordi man skal bruge do while eller en kompleks for løkke), så bruger man bare IF og GOTO.

subroutiner/funktioner

I Fortran skelner man mellem subroutiner (void functions) og funktioner (som returnerer en værdi).

Syntaxen ses nemmest med et eksempel:

```
      PROGRAM SUB  
      INTEGER*4 I
```



```

      DO 100 I=1,10
        CALL TESTFAC(I)
100   CONTINUE
      END
C
      SUBROUTINE TESTFAC(N)
      INTEGER*4 N
      INTEGER*4 FAC
      EXTERNAL FAC
      WRITE(*,*) N,FAC(N)
      RETURN
      END
C
      INTEGER*4 FUNCTION FAC(N)
      INTEGER*4 N
      INTEGER*4 RES,I
      RES=1
      DO 100 I=1,N
        RES=RES*I
100   CONTINUE
      FAC=RES
      RETURN
      END

```

EXTERNAL bruges til at understrege at en funktion ikke er en af de indbyggede funktioner.

En vigtig detalje er at standard Fortran 77 ikke understøtter rekursion og at lokale variable kan være på heap og dermed bevare deres værdier mellem kald som default (man kan tvinge dem til det med SAVE statement). De fleste Fortran compilere idag putter dog lokale variable på stack som default og kan dermed tillade rekursion.

Fortran har ikke rigtige globale variable som man kender det fra C/C++ eller kan simulere i Java/C# med public static fields.

Fortran har en ret avanceret export/import metode kaldet common blokke til mere selektivt at expose og bruge data mellem forskellige subroutiner/funktioner.

```
COMMON /commonnavn/var1,var2,var3
```

Vi tager et lille eksempel:

```

PROGRAM COMM
INTEGER*4 I,J,K
COMMON /B1/I
COMMON /B2/J
COMMON /B3/K

```

```

I=123
J=456
K=65537
CALL SUB1
CALL SUB2
CALL SUB3
END

```

C

```

SUBROUTINE SUB1
INTEGER*4 I
COMMON /B1/I
WRITE(*,*) I
RETURN
END

```

C

```

SUBROUTINE SUB2
INTEGER*4 J
COMMON /B2/J
WRITE(*,*) J
RETURN
END

```

C

```

SUBROUTINE SUB3
INTEGER*2 K1,K2
COMMON /B3/K1,K2
WRITE(*,*) K1,K2
RETURN
END

```

Nummer 3 illustrer en grim lille detalje. Common blokke er reelt bare at sted i memory, der er ikke noget type check af at de to parter opfatter det sted i memory som samme type.

For at gøre det lidt nemmere at styre, så understøtter de fleste Fortran 77 dialekter bruge af INCLUDE statement.

```

INTEGER*4 I,J
COMMON /B/I,J

```

```

PROGRAM COMM
INCLUDE 'comm2.inc'
I=123
J=456
CALL SUB1
CALL SUB2
END

```

C

```

SUBROUTINE SUB1

```

```

      INCLUDE 'comm2.inc'
      WRITE(*,*) I
      RETURN
      END
C
      SUBROUTINE SUB2
      INCLUDE 'comm2.inc'
      WRITE(*,*) J
      RETURN
      END

```

IO

Fortran IO er lidt speciel men faktisk ret kraftfuldt.

De to primære IO statements er READ og WRITE.

De findes i mange varianter bl.a.:

```

      READ(*,*) var1,var2,...
      READ(n,*) var1,var2,...
      READ(*,formatlabel) var1,var2,...
      READ(n,formatlabel) var1,var2,...
      READ(UNIT=n,FMT=formatlabel) var1,var2,...
      WRITE(*,*) var1,var2,...
      WRITE(n,*) var1,var2,...
      WRITE(*,formatlabel) var1,var2,...
      WRITE(n,formatlabel) var1,var2,...
      WRITE(UNIT=n,FMT=formatlabel) var1,var2,...

```

UNIT er et fil nummer. * betyder console. 5 er console input og 6 er console output per en meget gammel tradition.

En formatlabel består af:

```
label FORMAT(formatinfo)
```

hvor format info består af et antal komma separerede formate angivelser af formen:

nX = n mellemrum

Iw = integer med bredden w

Fw.d = floating point med bredden w og antal decimaler d

A = character

Aw = character med bredden w

'xxx' = konstant tekst streng

nHxxx = konstant tekst streng

n(...) = n forekomster af underformat

\$ = undlad lineskift for output (ikke standard men almindelig understøttet)

Bemærk at per meget gammel tradition så bruges første kolonne i output til printer styring:

1 = side skift

mellemrum = normal ny linie

Det er faktisk lidt komplekst, så vi tager lige et eksempel:

```
PROGRAM IO
INTEGER*4 I,N
INTEGER*4 FAC
EXTERNAL FAC
WRITE(6,1000)
READ(5,1100) N
DO 100 I=1,N
    WRITE(6,1200) I,FAC(I)
100  CONTINUE
1000  FORMAT(1X,'Indtast max. værdi: ', $)
1100  FORMAT(I2)
1200  FORMAT(1X,I2,1X,I8)
END
C
INTEGER*4 FUNCTION FAC(N)
INTEGER*4 N
INTEGER*4 RES,I
RES=1
DO 100 I=1,N
    RES=RES*I
100  CONTINUE
FAC=RES
RETURN
END
```

Og et eksempel som er en lille smule mere avanceret:

```
PROGRAM IO
INTEGER*4 I,N
INTEGER*4 FAC
EXTERNAL FAC
WRITE(6,1000)
READ(5,1100) N
DO 100 I=1,N
    WRITE(6,1200) 'I=',I,' FAC(I)=',FAC(I),
+               'I=',N+I,' FAC(I)=',FAC(N+I)
100  CONTINUE
1000  FORMAT(1X,'Indtast max. værdi: ', $)
1100  FORMAT(I2)
1200  FORMAT(1X,2(A2,I2,1X,A7,I8,1X))
END
```

```

C
      INTEGER*4 FUNCTION FAC(N)
      INTEGER*4 N
      INTEGER*4 RES,I
      RES=1
      DO 100 I=1,N
        RES=RES*I
100    CONTINUE
      FAC=RES
      RETURN
      END

```

Skal man bruge filer, så skal man udover READ og WRITE også bruge OPEN og CLOSE statements.

Det kan vi illustrere med et lille fil copy program:

```

      PROGRAM FCOPY
      INTEGER*4 IX
      CHARACTER*80 LINE
      OPEN(UNIT=1,FILE='ind.txt',STATUS='OLD')
      OPEN(UNIT=2,FILE='ud.txt',STATUS='NEW')
100    READ(UNIT=1,FMT=1000,END=300) LINE
      IX=80
200    IF(LINE(IX:IX).EQ.' ') THEN
      IX=IX-1
      GOTO 200
    ENDIF
      WRITE(UNIT=2,FMT=1100) LINE(1:IX)
      GOTO 100
300    CLOSE(UNIT=2)
      CLOSE(UNIT=1)
1000   FORMAT(A)
1100   FORMAT(A)
      END

```

En speciel feature med Fortran IO er at standarden supporterer unformatted IO og direct access til fixed length files, hvilket faktisk er et helt lille database system.

Det illustreres nok også bedst med et eksempel:

```

      PROGRAM DIRCRE
      INTEGER*4 I
      CHARACTER*80 BUF
      OPEN(UNIT=1,FILE='db.dat',STATUS='NEW',
+        FORM='UNFORMATTED',ACCESS='DIRECT',RECL=80)
      DO 100 I=1,100

```

```

        WRITE(BUF,1000) I
        WRITE(UNIT=1,REC=I) BUF
100    CONTINUE
        CLOSE(UNIT=1)
1000   FORMAT('Dette er record ',I3)
        END

```

```

        PROGRAM DIRLOOKUP
        INTEGER*4 R,IX
        CHARACTER*80 BUF
        OPEN(UNIT=1,FILE='db.dat',STATUS='OLD',
+          FORM='UNFORMATTED',ACCESS='DIRECT',RECL=80)
        WRITE(6,1000)
        READ(5,1100) R
        READ(UNIT=1,REC=R) BUF
        IX=80
100    IF(BUF(IX:IX).EQ.' ') THEN
        IX=IX-1
        GOTO 100
        ENDIF
        WRITE(6,1200) BUF(1:IX)
        CLOSE(UNIT=1)
1000   FORMAT(1X,'Indtast record number: ',I3)
1100   FORMAT(I3)
1200   FORMAT(1X,A)
        END

```

arrays

Fortran understøtter naturligvis også arrays.

Faktisk har Fortran en af de bedste understøttelser af multi dimensionelle arrays.

Syntaxen for erlæring af arrays er simpel:

```

type navn(dim)
type navn(dim1,dim2)

```

Og syntaxen for brug af elementer er:

```

navn(index)
navn(index1,index2)

```

Bemærk at i Fortran starter array index med 1 ikke med 0 !

Lad os først se et simpelt eksempel:

```
PROGRAM ARR
INTEGER*4 N
PARAMETER(N=10)
INTEGER*4 I
REAL*8 X(N)
DO 100 I=1,N
    X(I)=SQRT(1.0D0*I)
100 CONTINUE
DO 200 I=1,N
    WRITE(*,*) I,X(I)
200 CONTINUE
END
```

Og lad os så se et multi dimensionelt eksempel med brug af subroutine/funktioner:

```
PROGRAM MULTIARR
INTEGER*4 N,M
PARAMETER(N=10,M=10)
INTEGER*4 I,J
REAL*8 X(N,M)
REAL*8 MXSUM
EXTERNAL MXSUM
DO 200 I=1,N
    DO 100 J=1,M
        X(I,J)=I*J
100 CONTINUE
200 CONTINUE
WRITE(*,*) MXSUM(X,N,M)
END

C
REAL*8 FUNCTION MXSUM(X,N,M)
INTEGER*4 N,M
REAL*8 X(N,M)
INTEGER*4 I,J
REAL*8 RES
RES=0
DO 200 I=1,N
    DO 100 J=1,M
        RES=RES+X(I,J)
100 CONTINUE
200 CONTINUE
MXSUM=RES
END
```

Til sidst skal lige vises initialisering af arrays med DATA statement og implied loop i output:

```

PROGRAM MOREARR
INTEGER*4 N,M
PARAMETER(N=10,M=10)
INTEGER*4 I,J
REAL*8 X(N,M)
DATA X/1*1.0d0,2*2.0d0,4*3.0d0,8*4.0d0,16*5.0d0,32*6.0d0,37*7.0/
DO 100 I=1,N
    WRITE(6,1000) (X(I,J),J=1,M)
100  CONTINUE
1000 FORMAT(1X,10(F4.2,1X))
END

```

Når man studerer output ser man at Fortran gemmer 2 dimensionelle arrays kolonnevis og ikke rækkevis som de fleste andre sprog.

afsluttende bemærkninger

Denne artikel er naturligvis ikke en komplet gennemgang af alle features i Fortran 77, men det skulle dække alle de mest almindelige features til at man kan komme igang.

Ikke dækkede features inkluderer:

- statement function
- assigned GOTO
- EQUIVALENCE
- BACKSPACE
- ENTRY
- BLOCK DATA

De er nemme at opgoogle information på, hvis man har brug for det.

Kommentar af simonvalter d. 19. Oct 2008 | 1

Kommentar af jih d. 20. Jul 2008 | 2

Kommentar af blacktiger d. 28. Aug 2008 | 3

Du skriver at fortran lige siden 77 har stået i skyggen af nyere sprog. Det vil de færreste sikkert være uenige i, men jeg synes lige det er værd at påpege fortran (desværre?) lever i bedste velgående hvis man omgås folk i de rigtige kredse. F.eks. er fortran næsten de facto sproget hvis man laver astrofysik.

Kommentar af psycosoft-funware d. 20. Jul 2008 | 4

virkelig god og gennearbejdet artikel, meget interessant læsning :) keep up the good work!