QuickStart i FORTRAN

for studenter i faget Anvendt Datateknikk

Imran A. Majid
E-post: imran@ipt.ntnu.no

Deklarere variabler

Det finnes en rekke varianter av variable som kan defineres, eller deklareres, i FORTRAN. Variablene kan være heltall, desimaltall, tegn, tekst, tabeller etc. Felles for disse variablene er at de alltid er tilgjengelig til bruk i programkoden, og at data strømmer kontinuerlig gjennom disse variablene. For lange programmeringskoder kan dette imidlertid bli et problem (tenk dere programkode på flere hundre sider med mange hundre variabler…..) Blant annet derfor er tilgjengeligheten til disse variablene i programmeringskoden forskjellig, avhengig av hvor i koden de deklareres og ikke minst hvor i koden de brukes. Det er, med ande ord, mulig å ha flere variabler med samme navn og type, men dog atskilt i isolerte deler i koden. Vi skiller altså mellom globale variabler (som er gjellende for hele programkoden) og lokale variabler (som er gjellende for en spesifikk del av programkoden).

Globale variabler

Disse variablene defineres i begynnelsen av programkoden. Variablene defineres som en bestemt type variabler, men det assosieres vanligvis ingen verdier til dem. Altså, vi oppretter for eksempel en variable som kun kan ta inn heltall, men selve variabelen gis ingen heltall som verdi. En heltalls variabel deklareres ned kodeordet `INTEGER`, og deklareres på følgende måte.

```
INTEGER   A, B, C, D, E
```

Alle heltallsvariablene over kan altså holde et heltall, men ingen av variablene inneholder ved dette tidspunkt noen verdi. (For eldre versjoner av FORTRAN er man nødt til å ”slette” variabelen, eller sette den lik NIL som det heter. Dette ble gjort fordi hver gang en variabel ble deklarert, så ble det opprettet en peker til et tilfeldig sted i maskinens minne som ikke var i bruk. Om dette tilfeldige stedet hadde vært i bruk ved en tidligere anledning, så var det meget mulig at det lå et tall eller et tegn. Dermed kunne den nyopprettede variabelen plutselig inneholde en verdi, og videre gi uriktig input til kalkulasjonen). Ved bruka av samme fremgangsmåte som nevnt ovenfor, deklarerer vi en desimal variable, men her er historien litt annerledes. Det finnes nemlig desimalvariabler som kan ha alt fra en til flere tiltalls desimaler etter komma. Vi skiller her mellom standard antall desimaler og dobbel presisjon. En

Imran A. Majid
E-post: imran@ipt.ntnu.no

desimal variabel kalles i programmeringsspråk for en `REAL` verdi, og defineres på følgende måte:

```
REAL      ALFA, BETA, CHARLIE
REAL*8    DELTA, ECHO
```

Legg merke til at `REAL*8` deklareringen dobler nøyaktigheten, noe som er viktig i kalkulasjoner hvor man ønsker å minimalisere avrundingsfeil. På den andre siden vil en programkode som får en stor mengde tall å knuse, bruke noe lengre tid når desimalvariabler med dobbelpresisjon deklareres.

For både `INTEGER` og `REAL` variabler kan vi tildele henholdsvis heltall- og flyttallsverdier under programmeringsdelen. Verdiene til disse variablene vil endre seg etter alt hvordan vi selv ønsker å bruke dem. Det finnes en type variabel deklarasjon som er en konstant under hele programmeringen. Denne type deklarasjonen kalles `PARAMETER`. Nyttige bruksområder for denne parameteren er eksperimentelle verdier. Det er verdt å merke seg at de aller fleste matematiske konstanter allerede ligger inne programmeringsspråk. Videre er det viktig å legge merke til at den variabler som deklareres under `PARAMETER` seksjonen, behøver ikke å bli typedeklarert. FORTRAN vil automatisk opprette den passende variabeltypen og selve variabelen, enten `REAL` eller `INTEGER`, avhengig hvilken type verdi som brukes. En `PARAMETER` deklareres på følgende måte:

```
PAREMETER PI = 3,1415, E = 2.28
```

For mange kalkulasjoner er det hensiktsmessig å definer to- eller tredimensjonale tabell bestående av en selvbestemt type variabel. Dette kan for eksempel være innlesning av eksperimentelle resultater hvor det er assosiert to eller flere utfall for hvert tidssteg. Selve tabellen deklareres som en variabelen i to eller tre dimensjoner. For eksempel vil følgende deklarasjon opprette en todimensjonal heltall tabell og en tre dimensjonal flyttalls tabell:

Imran A. Majid
E-post: imran@ipt.ntnu.no

```
INTEGER   TABELLA (10,10)
REAL      TABELLB (10,10)
```

For å definere en tre dimensjonal variabel, brukes samme fremgangsmåte som vist over, men definisjonsområdet utvides til tre dimensjoner.

```
INTEGER   TABELLX (10,10,10)
REAL      TABELLY (10,10,10)
```

Det er hensiktsmessig å bruke tabeller når store mengder tall skal beregnes og deretter evalueres før de eventuelt beholdes eller beregnes på nytt. For å eksemplifisere dette, la oss tenke på et todimensjonal reservoar som er delt opp i 11 blokker (`RESERVOAR (1,11)`). Vi får oppgitt en metning for blokk 6 (`RESERVOAR (1,6) = 0.65`) og samtidig får vi oppgitt at trykket i endene er et gitt tall (`RESERVOAR (1,1) = 150` og `RESERVOAR (1,11) = 200`) . Vi bruker en viss analytisk likning og løser, med de initiale betingelsene, for hele reservoaret. Om løsningen for blokk 1 og 11 ikke gir de riktige trykkene, må vi beregne på nytt, men denne gangen må vi gjerne justere noen av faktorene og gjerne redusere eller øke konvergerings kriterium. Her gir todimensjonale tabeller oss friheten til å enkelt allokere beregnede verdier, undersøke og deretter eventuelt eksportere til en fil (vi kommer tilbake til eksportering til filer litt senere). Ikke minst er det, med hensyn til minnebruk og bergningshastighet, raskere å bruke tabeller sammenlignet med enkelte variabler.

Det kan godt hende at en ønsker å deklarere 20 variabler av samme type. For å effektivisere deklarerings seksjonen, er mulig å bruke en spansk vri. `IMPLICIT` lar brukeren defineres variabler slik at alle variabelnavn mellom for eksempel K og Z skal være heltall, og alle variabelnavn mellom A og J skal være flyttall. Følgende er framgangsmåten:

```
IMPLICIT REAL (A-J)
IMPLICIT INTEGER (K-Z)
```


Imran A. Majid
E-post: imran@ipt.ntnu.no

Ofte er det nødvendig å be brukeren skrive en tekst som input. Det kan være enkle ting som ”JA” og ”NEI” til spørsmål som stilles under kjøringen, til den avanserte siden hvor programmet lese inn store mengder tekst for deretter å analysere noe forubestemt. Tekst lese inn som tegn, og i programmeringens verden kalles det for `CHARACTER`. En tekststreng er en variable som kan ta inn alt fra et til flere antall tegn. Det er viktig å legge merke til at en tekststreng kan ta inn bokstaver, tall, tegn og symbol som input uten å ”kjenne” det igjen eller kunne se forskjelle mellom bokstaver, tall, tegn og symbol. Altså; det går ikke an å lese inn et tall i en `CHARACTER` variabel, for deretter å bruke den i en kalkulasjon. (Dette er en sannhet med modifikasjoner. Det gå an å konvertere fra en tekststreng til en tallvariabel, men dette er utenfor pensum). En `CHARACTER` med en bestemt lengde deklareres på følgende måte:

```
CHARACTER A*3, B*5, C*15, D*6
```

Lokale variable

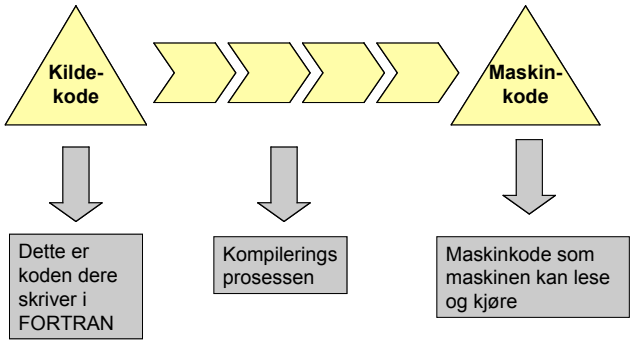
Disse variablene defineres i en underrutine eller subrutine av programkoden. En subrutine er en programdel som er ”selvstendig”, altså en programkode som kan ligge i en annen fil en selve programmeringsfilen. Fordelene ved å stykke opp programmeringsfilen er mange. For lange koder er det ofte slik at samme type operasjon gjentar seg. I stedet for å repetere denne operasjonen i koden like mange ganger som den skal anvendes, vil det være en fordel å flytte den koden ut av hovedfilen, og heller kalle den hver gang den trengs. Vi kaller denne programkoden for et bibliotek.

I slike koder, som er selvstendige programkoder, kan variabler defineres på samme måte som de deklareres i hovedfilen. Disse variablene som deklareres i biblioteket, er kun anvendbare i selve biblioteket og ikke av hovedfilen. Av den grunn kalles de for lokale variabler. Derimot kan biblioteket bruke de variablene som er deklarert i hovedfilen, bedre kjent som globale variabler. Senere vil ytterligere forklaring rundt både hva vi mener med en underrutine og hvordan bibliotek lages og brukes.

Imran A. Majid
E-post: imran@ipt.ntnu.no

Kompilator

En kompilator er et program som oversetter kildekoden (som vi skriver) til maskinkode (som PC'en forstår). En PC forstår ikke et programmeringsspråk, uansett hvilket språk det måtte være. Derfor må enhver kildekode kompileres.



Kompilator kommandoen for FORTRAN er, for den som er logget inn på petra maskinene ved IPT, som følgende:

```
petra1:~:(134)$   xlf   kode.f  -o  prog
** kode === End of Compilation 1 ===
1501-510 Compilation successful for file kode.f.
```

xlf	Kompileringskommando
kode.f	FORTRAN kildekode som skal kompileres
-o	En opsjon som forteller at kildekoden skal skrives ut til en fil
prog	Navnet til den ferdigkompilerte, kjørbare filen

I eksemplet over ser vi at kildekoden til kode.f er kompilert .

Under kompileringen sjekker kompilatoren selve koden - tegn for tegn, linje for linje. Hvis kompilatoren finner en feil, uansett om den er på et tegn eller ti linjer, vil den melde ifra om feil og deretter avslutte kompileringen. Altså må koden være absolutt fri for syntaksfeil for å kunne bli oversatt il maskinkode. Syntaks er dermed ortografien og grammatikken til selve programmeringsspråket. Kompilatoren bryr seg ikke om verdiene i de deklarerte variablene er verdier som er riktige eller gale, men den bryr seg som hvordan variabelen er deklarert og hvordan variablene brukes gjennom programkoden.

Ingen regel er uten unntak, og det gjelder her også. I kompileringsprosessen, slik forklart over, vil kun feil som påvirker programmeringsmetodikk identifiseres og

Imran A. Majid
E-post: imran@ipt.ntnu.no

påpekes. Kompilatoren er ikke intelligent og kan ikke avdekke potensielle problemer som kan fremkomme når selve programmet kjøres.
Et eksempel er brøk med null i nevner. En slik operasjon, som er matematisk udefinert, vil medføre at programmet avbrytes – selv om kompilering gikk greit. For de tilfeller hvor det er sannsynelig at et slikt problem kan fremkomme, er det lurt å ha en løkke som sjekker nevneren før beregningen utføres. Et programmeringsmetode som tar hensyn til potensielle problemer, kalles for robust programmering.

Det finnes også egne Debugging verktøy for å finne feil i programkode. Et debuggingsverktøy vil kjøre programmet, linje for linje, og hele tiden angi hva innholdet i ulike variabler er. På denne måten kan man avdekke problemer i kildekoden som eller ville har vært meget tungt å avdekke.

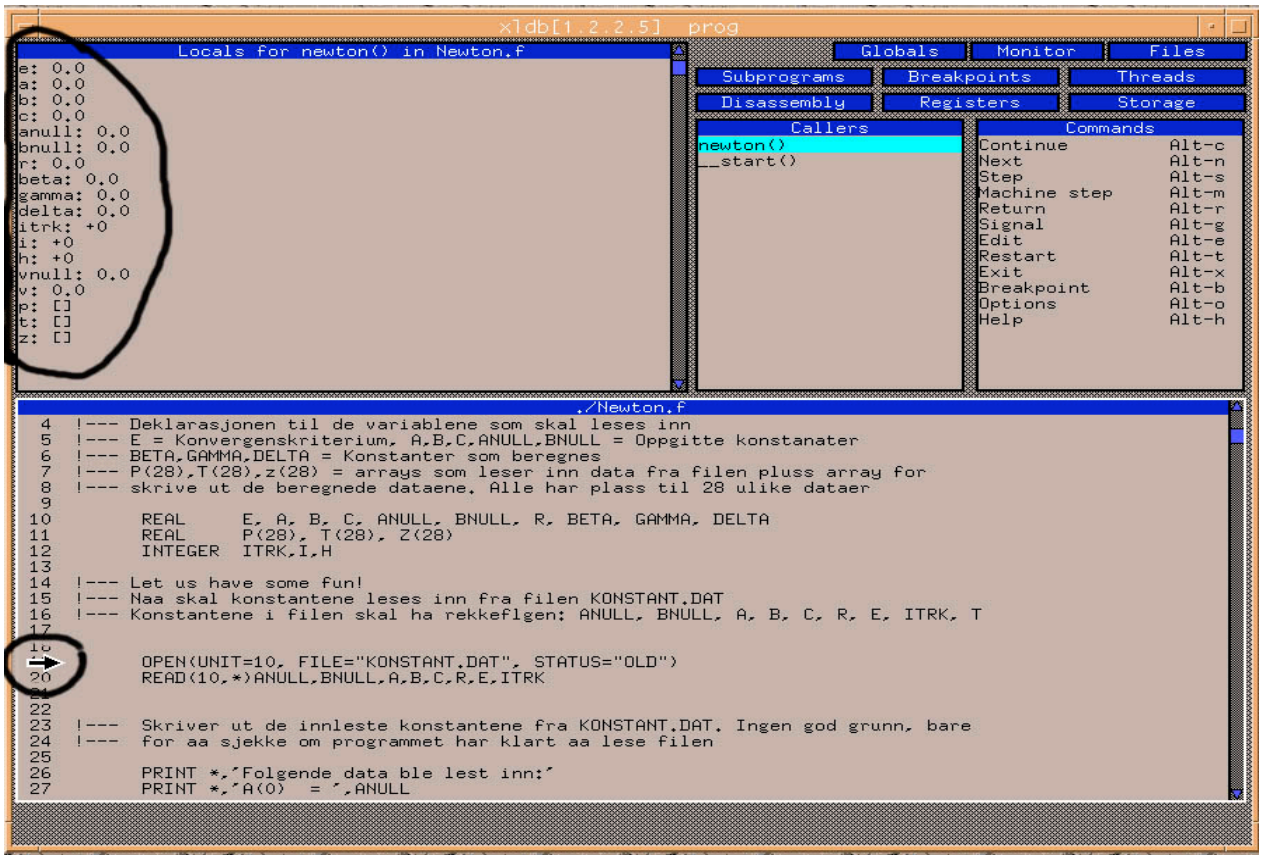
For at et debugging program skal fungere, er det imidlertid viktig at selve kompileringen er gjennomført. Slik vi husker fra tidligere, så er det viktig å konvertere kildekoden til et maskinspråk som PC'en kan skjønne. Når en kildekode ført er kompilert, så er det ikke mulig å få tilbake kildekoden. Hvis opsjonen for debugging skal brukes, er det nødvendig å legge inn ekstra informasjon i maskinkoden slik st debugging programmet kan hente opp de nødvendige data samt selve kildekoden. For å inkludere debugging opsjonen, kompileres kildekoden med opsjonen –g og startes med en xldb kommando:

```
petra1:~ xlf  -g  kildekode.f  -o  prog
petra1:~  xldb prog
```

Programmer er ganske selvforklarende, og det anbefales å bruke litt tid for å sette seg inn i det. Effektivt bruk av programmet kan spare mange timer bak skjermen for å finne mystiske feil som kompilatoren ikke oppdater.

I skjermbildet under er alle variablene listet opp i øverste venstre hjørne. I delvinduet under er selve kildekoden listet opp, men en pil som angir hvilken linje som for øyeblikket debugges. Debuggeren beveger seg nedover, linje for linje, ved at brukere klikker Next i listet øverst til høyre.

Imran A. Majid
E-post: imran@ipt.ntnu.no



Programmering

Etter å ha deklarert de nødvendige variablene, begynner selve programmeringen, altså den delen hvor de deklarerte variablene skal brukes. Vi ønsker å sette opp en kommunikasjon mellom selve programkoden og brukeren. Med andre ord, en interaksjon som fører til at en resultat oppnås. Av ren pedagogisk hensyn, velges det her å gå igjennom et par eksempler for å illustrere hvordan programmering foregår.

Eksempel I – Løsning av annengradslikning

Med utgangspunkt i at alle er familiær med løsningsmetoden for en annengradslikning, deklareres de nødvendige variablene.

I begynnelsen av hver programmeringsfil, gis det et navn som identifiserer filen. Dette navnet gis med kommandoen `PROGRAM` som den første setningen i koden, etterfulgt av et navn som kun inneholder bokstaver. Videre defineres tre variable for

Imran A. Majid
E-post: imran@ipt.ntnu.no

de respektive koeffisientene i en annengradslikning, samt to variabler for de to løsningene som skal kalkuleres.

```
PROGRAM ANNEGRAD
REAL  A,B,C,D, LOS1, LOS2
```

Programmet skal spørre brukeren om å taste inn koeffisientene `A, B` og `C`. Først da kan den gå videre med å beregningene. For å få koeffisientene, bør en beskjed skrives til skjermen hvor de respektive koeffisientene etterspørres. For å skrive beskjeden eller en etterspørselen til skjermen, bruker vi følgende kommando:

```
WRITE (*,*) ' Vennligst tast inn koeffisienten A: '
```

Legg merke til at teksten skrives med et anførselstegn både før og etter. Kommandoen `WRITE` skriver beskjeden ut på skjermen, og parentesen med to. Den første stjernen angir enheten beskjed skal skrives til. Denne enheten kan enten være skjerm (som er satt opp som standard), en fil eller en printer. Den andre stjernen angir hvilket format beskjed skal skrives ut på. (Formatering er en egen seksjon som senere kommer tilbake til).

Det er nå brukeres tur til å taste inn verdien til koeffisienten `A`. I programkoden er det en ny linje som tar seg av innlesning.

```
READ (*,*) A
```

Programmet leser inn den inntastene verdien, og legger den inn i variabelen `A`. På denne måten lese de resterende verdiene inn, en etter en.

```
WRITE (*,*) ' Vennligst tast inn koeffisienten B: '
READ (*,*) B
WRITE (*,*) ' Vennligst tast inn koeffisienten C: '
READ (*,*) C
```


Imran A. Majid
E-post: imran@ipt.ntnu.no

Nå har programmet fått den nødvendig informasjon den trenger for å løse annengradslikning. Fra matematikken vet vi at den finnes både reelle og komplekse løsninger på en annengradslikning. Vi ser bort fra å løse likningen på det komplekse plan (som fremkommer hvis tallet under kvadratroten bli null). En enkel sjekk kan avsløre dette før vi går videre. Her ønsker vi at programmet skal foreta et valg: hvis tallet under kvadratroten er negativ, så skal videre beregning avbrytes og brukeren skal opplyses om at likningen har en kompleks løsning. Hvis tallet under kvadratroten er null eller et positivt tall, skal beregningen gjennomføres og de reelle løsningene skrives ut på skjermen.

For å gjennomføre valget mellom å løse eller og ikke løse likningen, bruker vi en `IF-THEN-ELSE` setning. Det er her best å studere koden for å forstå mekanismen i denne setningen samt introdusere både matematiske funksjoner og logiske operasjoner. Vi beregner determinanten og tester den med en `IF-THEN-ELSE`-SETNING.

```
D = B**2 - 4*A*C
IF ( D .LT. 0) THEN
      WRITE (*,*) 'Likningen har kompleks løsning.'
ELSE
      LOS1 = (-B + SQRT(D)) / (2*A*C)
      LOS2 = (-B - SQRT(D)) / (2*A*C)
ENDIF
```

Logiske relasjoner er funksjoner som brukes for å evaluere to variable ut ifra en logisk relasjon. I slike evalueringer er svaret enten sann eller usann. I eksemplet over brukes `.LT.` kommandoen, som står for ”less then”, altså blir det logiske relasjonen: hvis D er mindre enn null. Det finnes en rekke logiske relasjoner i FORTRAN som er verdt å legge merke til.

`.LT.`	Less then / mindre enn
`.LE.`	Less then or equal / mindre enn eller lik
`.EQ.`	Equal / lik
`.NE.`	Not equel / ikke lik

Imran A. Majid
E-post: imran@ipt.ntnu.no

.GE. Greater or equal / større enn eller lik

.GT. Greater then / større enn

For tilfellet hvor den logiske relasjonen gir en sann Verdi, altså er determinanten mindre enn null, skrives det ut en beskjed til skjermen. For tilfellet hvor den logiske operasjonen er usann, gjennomføres beregningen. I en IF-THEN-ELSE setning er det alltid en logisk relasjon som skal valueres.

I beregningen brukes en forhåndsdefinert funksjon med navnet SQRT, som beregner kvadratroten av et tall. Det finnes en rekke funksjoner som er forhåndsdefinert i FORTRAN og kan brukes fritt. Blant disse kan funksjoner som SIN, COS, TAN nevnes. I tillegg til disse standard matematiske funksjoner, finnes det et stort antall spesialiserte funksjoner. Disse finnes i et bibliotek ved navnet NAG.

Det er viktig å avslutte en IF-THEN-ELSE setning ved kommandoen ENDIF, dette fordi ved nøstede IF-THEN-ELSE setninger så kan argumentet fra en setning lett havne over i den andre (altså, maskinen vil kun kjøre den koden som den blir bedt om å kjøre, men det er meget mulig at brukeren under innstastingen av nøstede setninger mener noe, mens maskinen tolker det som noe annet).

```
WRITE (*,*) LOS1, LOS2
END
```

Programmet kan nå skrive ut løsningene til skjermen.

For å markere at programmeringen er over, brukes END. Om det står noe tekst/kode etter END, vil den bli neglisjert av kompilatoren.

Utskrift til skjermen vil gå tapt. Det vil si at utskriften ikke vil bli lagret noen sted, og etter hvert som mer tekst og nye kommandoer kommer opp, vil resultatene bli borte. For eksemplet over vil det være relativt lett å gjenskap resultatene ved å kjøre programmet på nytt. Vanligvis er ikke dette så lett at programmene kan kreve lang kjøretid, og resultatene i bøtter og spann. Vi ønsker å lagre resultatene i en fil som senere kan brukes, enten for plotting eller som videre input til et annet program.

Imran A. Majid
E-post: imran@ipt.ntnu.no

I FORTRAN åpnes filer på følgende måte:

```
OPEN (UNIT = 10, FILE = "RESULTATETER.DATA", STATUS = "OLD")
```

`Unit` – Dette er identifikasjonen til filen. Videre i programmer vil filen bli henvist til dette nummeret ved lesing og skriving.
`File` – Her angis filnavnet. Være klar over at filen må ligger i samme katalog hvor selve kildekoden ligger.
`Status` – Når en fil skal brukes som input, velges denne opsjonen `OLD`. Er tilfellet slik at det skal opprettes en ny fil, velges opsjonen `NEW`. For det tilfellet hvor vi ikke vet om filen eksisterer eller ikke, settes denne opsjonen til `UNKNOWN`.

En modifikasjon av eksemplet over kan være at vi ønsker å skrive røttene til filen ut i en fil i stedet for skjermen. Vi trenger da både å lage en ny fil, og deretter skrive resultatene til filen. I koden under er de nødvendige endringene market med rødt. Husk at det er viktig å lukke filen før programmet avsluttes. Om det ikke blir gjort, er det fare for at data ikke skrives fra maskinens minne inn på selve filen, og at filen enten forblir tom eller inneholder delvise resultater.

Imran A. Majid
E-post: imran@ipt.ntnu.no

```
PROGRAM ANNEGRAD
REAL  A,B,C,D, LOS1, LOS2

OPEN (UNIT=10, FILE=LOSNING.DATA", STATUS="NEW")

WRITE (*,*) ' Vennligst tast inn koeffisienten A: '
READ (*,*) A
WRITE (*,*) ' Vennligst tast inn koeffisienten B: '
READ (*,*) B
WRITE (*,*) ' Vennligst tast inn koeffisienten C: '
READ (*,*) C

D = B**2 - 4*A*C
IF ( D .LT. 0) THEN
      WRITE (11,*) 'Likningen har kompleks løsning.'
ELSE
      LOS1 = (-B + SQRT(D)) / (2*A*C)
      LOS2 = (-B - SQRT(D)) / (2*A*C)
ENDIF
WRITE (11,*) LOS1, LOS2

CLOSE(UNIT=10)
END
```

Vi vil nå avansere programmet ytterligere, og ønsker at brukeren skal taste inn koeffisientene `A, B` og `C` i en fil som heter INPUT.DATA. Denne filen ligger i den samme katalogen som kildekoden. Her ligger faktorene `A, B` og `C` på hver sin linje, og disse faktorene kan være enten heltall eller flyttall.

Det er her viktig å legge merke til et par ting. FORTRAN er et høynivå programmeringsspråk. Det vil si at vi i kildekoden ikke trenger å fortelle hvordan kildekoden skal lese inn data, punkt for punkt, men heller forteller hva den skal lese etter. Vi forteller her at programmet skal lete etter og lese inn tre heltall eller flyttall. Disse kunne enten stå etter hverandre – separert med et eller flere mellomrom, eller stå på hver sin linje. FORTRAN skjønner dette og vil lese inn tallene riktig.

Imran A. Majid
E-post: imran@ipt.ntnu.no

Videre vil et heltall, som leses inn I en flyttall variabel, automatisk bli konvertert til et flyttall. Altså vil tallet 1 bli lest inn som 1.00000

Slik leser vi inn faktorene `A, B` og `C` fra filen INPUT.DATA

```
PROGRAM ANNEGRAD
REAL  A,B,C,D, LOS1, LOS2

OPEN (UNIT=10, FILE="LOSNING.DATA", STATUS="NEW")
OPEN (UNIT=11, FILE="INPUT.DATA", STATUS="OLD")

READ (11,*) A, B, C

D = B**2 - 4*A*C
IF ( D .LT. 0) THEN
      WRITE (11,*) 'Likningen har kompleks løsning.'
ELSE
      LOS1 = (-B + SQRT(D)) / (2*A*C)
      LOS2 = (-B - SQRT(D)) / (2*A*C)
ENDIF
WRITE (11,*) LOS1, LOS2

CLOSE (UNIT=10)
CLOSE (UNIT=11)
END
```

FORMAT setning

Utskrift av et flyttall kan være ganske "stygg", det vil at den inneholde mange desimaler. Hvis flere flyttall skrives ut samtidig, vil det være vanskelig for leseren å holde oversikten. Vi ønsker gjerne at utskriften skal følge et gitt oppsett.

Med en `FORMAT` setning kan brukeren angi hvordan hvert eneste tegn i utskriften skal brukes, hvordan tallene skal se ut, antall mellomrom og hvor en eventuell tekst skal skrives.

Imran A. Majid
E-post: imran@ipt.ntnu.no

En `FORMAT` setning defineres med en tilordningsetikett og selve setningen. Tilordningsetiketten skal være unik for hver `FORMAT` setning. For heltall brukes indeksen I, for flyttall F, for tekst X og for mellomrom X.

Etter indeksen I for heltall, angis et tall om forteller hvor mange plasser heltallet kan brukes. For flyttall brukes indeksen F.

Imran A. Majid
E-post: imran@ipt.ntnu.no

Funksjoner og Subrutiner

For de tilfeller hvor en og samme operasjon, for eksempel løsning av en likning, gjentas, kan det være hensiktmessig å lage en selvstendig kode for denne oppgaven. Avhengig av hvilke type oppgave vi ønsker å løse, kan vi enten lage en funksjon eller en subrutine. Funksjonen eller subrutinen vil bli kalt opp hver gang det er behov for å løse oppgaven. Man kan kanskje tenke at vi faktoriserer ut den gjentakbare delen av hovedprogrammet, og lager et delprogram av det. Dette delprogrammet tar inn en rekke verdier fra hovedprogrammet, gjør det den skal gjøre med de verdiene og sender svar tilbake til hovedprogrammet.

Funksjon

En funksjon defineres enten etter at `END` er skrevet i hovedprogrammet, eller i en egen fil. Vi skal her konsentrere oss om å skrive funksjoner i samme filen som hovedprogrammet. En funksjon kan ta inn en eller flere variabler, men kan kun returnere en verdi.

For de tilfeller vi skal bruke funksjoner, vil det dreie seg om et heltall eller flyttall. Det er nødvendig å definere funksjonen til den type variable som skal returneres. La oss eksemplifisere dette. I et større program ønsker vi å sende koeffisientene til en annengradslikning til en funksjon. Denne funksjonen skal løse annengradslikning og deretter returnere den roten som er størst. For tilfellet hvor begge løsningene er like, altså x1 = x2, skal funksjonen fortsatt kun sende en verdi sendes. Altså, vi ”faktoriserer” ut metodikken for løsning av en annengradslikning til et delprogram.

Når vi kaller opp en funksjon, gjør vi dette ved å tilordne resultatet vi skal få fra funksjonen i en definert variabel. I eksemplet under er dette den første røde linjen som er merket i rødt.

Imran A. Majid
E-post: imran@ipt.ntnu.no

```
PROGRAM ANNEGRAD
REAL  LOSNING, A, B, C

......
......
LOSNING  = ANNENGRAD (A, B, C)
......
......
END


REAL FUNCTION ANNENGRAD (A, B, C)
REAL D, LOS1, LOS2

D = B**2 - 4*A*C
IF ( D .LT. 0) THEN
      WRITE (11,*) 'Likningen har kompleks løsning.'
ELSE
      LOS1 = (-B + SQRT(D)) / (2*A*C)
      LOS2 = (-B - SQRT(D)) / (2*A*C)
ENDIF

IF (LOS1 .GT. LOS2) THEN
      ANNENGRAD = LOS1
ELSE ANNENGRAD = LOS2

RETURN
END
```

Funksjons deklarasjonen er som gitt over angir type format verdien som sendes tilbake skal ha. Selve funksjonen inneholder samme programmeringsmetodikk som ellers. Om nødvendig, skjer det en deklarasjon av lokale variabler. Videre er fullt mulig å brukeglobale variabler, men det er ikke mulig for en funksjon å endre disse variablene.

Funksjonens beregninger avsluttes ved at den beregnede verdien settes lik funksjonsnavnet. Dette sørger for at resultatet kan overføres til hovedprogrammet.

Imran A. Majid
E-post: imran@ipt.ntnu.no

Selve funksjonen avsluttes ved `RETURN` og deretter `END`, der den første kommandoen returnerer resultatet til hovedprogrammer, og den siste kommandoen avslutter funksjonen og flytter oss tilbake til hovedprogrammet. Husk at når hovedprogrammet kommer til en funksjonsoppkalling, stopper hovedprogrammet og beveger seg ned til funksjonen. Her fullfører den funksjonen før den returnerer til hovedprogrammet (med resultatet) og fortsetter nedover koden.

Subrutine

En subrutinene fungerer i prinsippet på samme måte som en funksjon, men har en del ekstra funksjonalitet. La oss se på en modifisert versjon av programmet over.

```
PROGRAM ANNEGRAD
REAL  LOSNING, A, B, C, LOS1, LOS2

......
......
CALL ANNENGRAD (A, B, C, LOS1, LOS2)
......
......
END


SUBROUTINE ANNENGRAD (A, B, C, LOS1, LOS2)
REAL D

D = B**2 - 4*A*C
IF ( D .LT. 0) THEN
      WRITE (11,*) 'Likningen har kompleks løsning.'
ELSE
      LOS1 = (-B + SQRT(D)) / (2*A*C)
      LOS2 = (-B - SQRT(D)) / (2*A*C)
ENDIF

RETURN
END
```

Det første vi legger merke til er at måten å aktivere subrutinen på. Ved kommandoen `CALL` sender vi en rekke variabler til en predefinert subrutine. Subrutinen tar inn en

Imran A. Majid
E-post: imran@ipt.ntnu.no

rekke variabler, men vi tilordner ikke resultatet til noen variabel. En subrutinene tar inn en rekke variabler, foretar nødvendige beregninger og sender en eller flere resultater til hovedprogrammet gjennom de samme variablene. I tilfellet over sender vi koeffisientene `A, B` og `C` (som inneholder innleste verdier) til subrutinen samt to variabler `LOS1` og `LOS2`. Subrutinen bruker variablene `A, B` og `C`, og sender løsningen gjennom `LOS1` og `LOS2` tilbake til hovedprogrammet.

Programmeringsmetodikken er den sammen som for hovedprogrammet, og subrutinen avsluttes på samme måte som en funksjonen, bortsett fra at et resultat ikke tilordnes gjennom navnet til subrutinen men gjennom variabler.

Imran A. Majid
E-post: imran@ipt.ntnu.no