Forelesninger i programmering med Pascal.

Utdrag.

Av Steinar Thorvaldsen, Høgskolen i Tromsø. (steinar@hitos.no)

Brukt som en del av studiet **Informatikk I** (10 vt) i tiden 1986-96.
Lærebok: **Havnen/Quale: "Strukturert programmering med Pascal"**
NKI-forlaget, 3.utgave 1992, Kap 1 til kap 11.2.

Disse forelesningene ble i ulike varianter brukt en del av studiet Informatikk I (10 vt) etter den gamle rammeplanen av 1985. Som verktøy ble Turbo Pascal versjon 3.0, 5.0, 5.5, 7.0 og senere versjon 1.0 og 1.5 for Windows brukt. Turbo Pascal fra Borland i USA var i sin tid et banebrytende programmeringsverktøy som gjorde det mulig å utvikle profesjonell programvare på en liten PC, noe som før den tid for det meste var forbeholdt større datasystemer.

Innhold:

DEL 1: INNLEDNING OM PASCALPROGRAMMERING.

Copyright Steinar Thorvaldsen 13.10.93, Sist endret 6.10.95.

Les Kap. 1 og 2 i læreboka i sammenheng med at du ser på dette:

A. HVORFOR KUNNE EN DEL OM PROGRAMMERING?
Mange informatikkstudenter lurer på om det lenger har noen hensikt å lære seg programmering når man allikevel i praksis må bruke ferdiglagede programmer.

Nå er ikke vår hensikt å utdanne programmerere. Men likevel kan det være nyttig for den vanlige bruker å kunne en del programmering. Programmeringskunnskaper vil øke din forståelse av hvordan datamaskiner og programmer virker, og gjøre deg til en sikrere bruker. Når man vet litt om hva som foregår "under panseret", får man derved en økt trygghet i bruken av datautstyr.
Av og til kan det dessuten være behov for å tilpasse et standardprogram til en spesialoppgave, og da må ofte litt programmering til. Det kan også hende at man må lage småprogrammer til internt bruk.
Dessuten vil jeg også påstå at det er ganske morsomt når man får til et egenprodusert program på sin PC, slik at programmering neppe kan karakteriseres som bortkastet tid i informatikkstudiet.

Må man så kunne mye matematikk for å lære å programmere? Her er svaret et betinget nei. Vi vil ikke benytte særlig mye matematikk. Men den samme type strukturert og ryddig tenkning som i matematikken, vil du nok finne igjen. Så de som ikke har sine sterkeste sider på denne arena, må nok regne med noe ekstrainnsats i Pascal. Samtidig kan ikke de som kan mye matematikk ikke regne med å flyte igjennom som gratispassasjerer.

B. PASCAL.
Det finnes mange programmeinigsspråk. Pascal ble utviklet av professor N. Wirth i Zurich rundt 1970. Han hadde to hovedhensikter med språket:

1. Pascal skulle være et pedagogisk tilrettelagt språk som var lett å lære, men samtidig inneholdt de begreper som var vanlige i alle programmeringsspråk.
2. Pascal skulle være et språk som var sterkt standardisert, det skulle være billig og enkelt å legge inn på ulike typer datamaskiner.

Professor Wirt oppkalte språket sitt etter den franske matematiker, naturforsker og religionsfilosofen Blaise Pascal (1623-62). Pascal er kjent for å ha konstruert en mekanisk regnemaskin da han var 19 år gammel. Men det han er mest kjent for, er boka "Tanker", som er et bredt anlagt forsvarsskrift for kristendommens sannhet.

Turbo-Pascal fra det amerikanske firmaet Borland, kom på markedet i begynnelsen av 1980-tallet og fikk stor utbredelse på mikromaskiner og PC'er. På den gamle norske skolemaksinen TIKI-100 ble versjon 3.0 benyttet. Versjon 5.5 for MS-DOS var den siste versjonen der man tok seg tid til å oversette hjelpedelen av systemet til norsk. En mangel ved versjon 5.5, var at den ikke hadde støtte for bruk av mus. Man måtte bruke piltastene og returknappen istedet. Turbo-Pascal har senere kommet i Windowsversjon (versjon 1.5, engelsk), og systemet har etterhvert blitt meget stor og omfattende. Siste versjon av systemet går under navnet Delphi som er et visuelt og objektorientert programmeringsverktøy på linje med Visual Basic. Men disse nye programmeringsmetodene er ikke pensum på Informatikk I.

Turbo Pascal er et kompilerende språk. Det vil si at programmet blir oversatt til maskinkode (som blir lagret som fil på harddisken), og dermed er klar til kjøring. Derfor kaller vi ofte Turbo Pascal språket for en kompilator. Det motsatte er interpreterende språk, som oversett programkoden til maskinkode etterhvert som programmet blir kjørt.

Lærebøkene som finnes på norsk baserer seg fortsatt på MS-DOS versjonen av Turbo Pascal. Windowsversjonen er mer omfattende, men stort sett lik i menyene, og i tillegg er den fullstendig integrert med windowsmiljøet. Derfor anbefales det å bruke denne.

C. PROBLEMLØSNING.
Mønsterplanen av 1987 omtaler problemløsning som et viktig temafelt å gi elevene trening i. Med problem mener vi her oppgaver som det ikke er noen standard framgangsmåte for å løse. Programmering føyer seg naturlig inn her.

Problemløsning deler man ofte opp i faser, eller en handlingsplan:
(i parantes bak stipuleres en vanlig tidsbruk i prosent)

1. FORSTÅ PROBLEMET (15%)
Ofte kan en presisering være nødvendig.

2. DRØFT LØSNINGSMETODER (25%)
Ta gjerne utgangspunkt i spesialtilfeller.

3. LAG PLAN FOR LØSNINGA (30%)
Dette er en forholdsvis presis oppskrift som vi gjerne kaller en algoritme. Er den korrekt?

4. GJENNOMFØR PLANEN (10%)
Skriv program, f.eks. i Turbo-Pascal.

5. SE TILBAKE! (20%)
Rett gjennværende feil.

Som dere ser er problemløsning en ganske prosessorientert aktivitet. Merk at vi venter ganske lenge med programmeringen.

D. OPPGAVER
Programmering lærer du ikke ved å ligge på sofaen og lese en bok.
Her må du opp og sette deg ned med blyant og maskin. Praktisk trening er veldig viktig innen programmeringen. Det er også viktig at du tidlig blir kjent med maskinen og Turbo-Pascal's muligheter, slik at du kan lette senere arbeide. Dette oppnås kun ved å sitte ved maskinen og taste / programmere. Jeg vil derfor oppfordre alle til å bruke mye tid ved maskinen, spesielt i begynnerfasen. Dette vil på lang sikt tjene seg inn ti-dobbelt, både når det gjelder forståelse og tidsbruk.

Oppgavene er i første omgang små og velavgrenset slik at studenten kan konsentrere seg om mindre deler uten for mange forstyrrende elementer.

Oppgaver senere i kurset bygger på det som skal være lært i tidligere deler. Derfor vil jeg oppfordre deg til å bruke tid på å utføre alle oppgavene, spesiellt i startfasen !!

Oppgavene vil også etterhvert invitere til strukturert problemløsning. Dette skjer ved at oppgavene ikke går ut på å lage et helt program, men kun en funksjonell del. Slike deler blir senere satt sammen til hele programmer.

Et viktig poeng er dette: Tenk strukturert !!
Finn ut hva du vil/må gjøre for å løse oppgaven, tenk så på kun en av disse delene om gangen, og programmer del for del. Sørg for å la koden være oversiktlig, og bruk kommentarer i programmet. Hvis du får problemer, ikke forsøk ulike ting i panikk, tenk planmessig først, prøv siden.

E. TIPS OM FILNAVN.
Programmene som vi skal kjøre i Turbo-Pascal, må skrives/lagres av en tekstbehandler som benytter såkalt standard txt-format på filene. Det betyr at filene ikke kan inneholde formateringskoder som stor skrift o.l. Istedet bruker vi da tabulatorinnrykk for å redigere programmene slik at de da er mest mulig oversiktlige. Eksempler på tekstbehandlere som bruker dette tekstformate på filene, er Notisblokka i Windows, tekstbehandleren i Turbo Pascal og Winix. Tekstbehandleren i Works kan også lagre filer i standard Tekst-format (velg: Fil. Lagre Som og klikk på liste over Filtyper). Men den VANLIGSTE tekstbehandleren å bruke i Turbo-Pascal, er den som følger med integrert i Pascal-systemet. Jeg anbefaler at du bruker denne, selv om du altså i tillegg kan få overført en programtekst via modem og Winix for lagring på harddisken. Senere henter du denne direkte inn i Turbo-pascal sin tekstbehandler for kjøring og videre arbeide.

Det er ofte behov for å sammenligne oppgaveløsningene med hverandre. Det vil derfor være lurt å holde seg til en fast mal når det gjelder filnavn. Hvis i tillegg alle kjenner denne malen og holder seg til den samme, vil det være lettere å hjelpe hverandre / se på andres filer. Vi vil derfor foreslå følgende:

Gi oppgavene navn etter nummer. Oppgave **1.1.** kan lettest gi filnavnet **nr11.pas.**(Legg merke til at jeg ikke skriver punktumet mellom tallene). Filnavnene vil da være enkle og ikke kunne misforstås, og de viser til oppgavenummer, slik at det er lett å finne tilbake til besvarelsen av alle oppgaver. Hvis det er flere versjoner av oppgave 1.1, kan man slenge på noen forbokstaver etter nummeret, eks.: **nr11st.pas** .

Jeg oppfordrer dere til å skrive programmene oversiktlig. Lag innrykk (tabulator) for å bedre leseligheten. Dette gjør det mye lettere for læreren / andre / deg selv å holde oversikten.

F. PROGRAMUTVIKLINGSMILJØET I TURBO PASCAL.
De fleste av dere har lagt inn Turbo-Pascal slik at den kan startes som et symbol i Windows. Hvis ikke du har det slik, så kan du starte pascal i Windows sin filbehandler.

Ved oppstart får du fram et skjermvindu, med en Windows-meny i toppen, og et tekstbegandlingsfelt under. Programmet styres som et standard Windowsprogram.Når du har skrivet inn programmet ditt i tekstvinduet, må maskinen oversette det til maskinkode for at det skal bli kjørbart. Programmet må kompileres. Dette gjør du ved å velge **Compile** i toppmenyen, og deretter **Compile**, **Make** eller **Build** i undermenyen. Programmet blir da kompilert og lagret på en fil som har samme fornavn som kodefila (dvs. .pas - fila som du har skrive inn i tekstbehandleren), men med etternavnet .exe .

Compile oversett den aktuell .pas - fil vi jobber med.

Make oversetter alle filer som er endret siden siste kompilering.

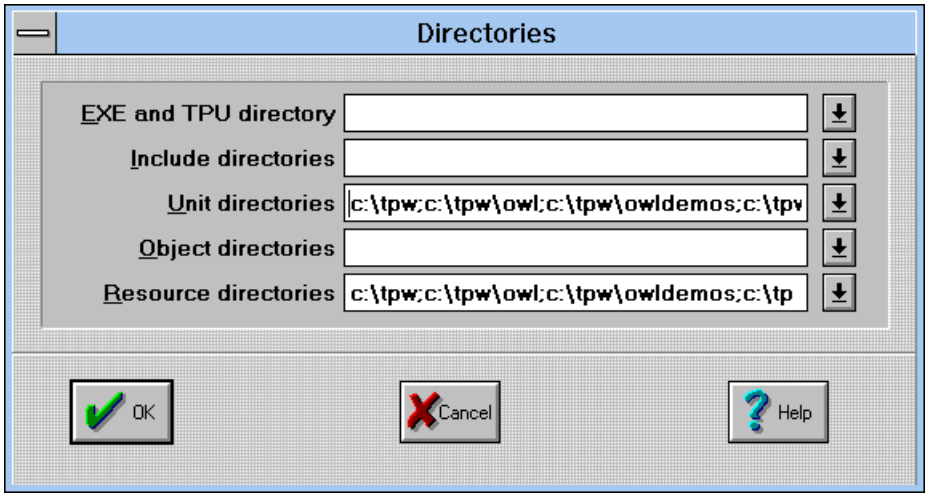
Build oversetter alle filene som er i bruk i programmet.

Bruken av Make eller Build gjelder kun hvis man jobber med større prosjekter, og det har ingen praktisk betydning hva for kompileringsmetode du i første omgang bruker.

For å kjøre det ferdig kompilerte programmet, velger du **Run** i menyraden, og dernest **Run** i undermenyen. Dersom du ikke har oversatt programmet med Compile, før du velger Run, blir programmet automatisk kompilert før det startes.

Pascal avslutter du ved å velge File/Exit som i standard Windows.

Vær oppmerksom på valget **Options**/Directories i menyraden:



Her kan du spesifisere de kataloger der Turbo Pascal skal plassere og søke etter filer og ressurser. Velg **Help** knappen for nærmere informasjon.

Turbo-Pascal inneholder en godt utbygd hjelpefunksjon. Det er laget hjelp både om betydningen av menyvalg, og bruken av selve programmeringsspråket. Ved å trykke på tasten F1 får du fram generell hjelp, med muligheit for søking på spesielle ord. Ved trykk på <ctrl> F1 vises spesifik hjelp på det ordet som skrivemerket står ved i editoren. Du kan også få fram hjelpeteksten ved å velge **Help** i toppmenyen.

Denne hjelpefunksjonen er viktig å bruke flittig når du programmerar i Turbo Pascal. Hjelpen inneholder like mye informasjon som brukermanualen som følger med.

G. EKSEMPLER.

I tekstbehandleren kan vi skrive inn følgende lille programstubb:

```
Program Test1;
Uses
 WinCrt;
begin
```

```
 writeln('Dette skal gå bra!');
end.
```

For å skrive inn dette programmet må du velge File i menyraden, og deretter New i undermenyen. Det vises da et tekstvindu på skjermen, der du kan skrive inn programkoden. Denne kodan må du lagre på fil ved å velge File/Save As.... Programkoden lagrar du på ei fil med valgfritt fornavn, men den bør ha etternavnet '.pas', for å vise at det er pascalkode som er lagret på fila. For å kompilere programmet velger du Compile, og for å kjøre det må du velge Run.

Et program starter alltid med det reserverte ordet Program, i tillegg til et valgfritt navn som her er Test1.
Alle instruksjoner (dvs. nesten hver linje) avsluttes med semikolon.
Det som skal utføres i programmet står mellom de reserverte ordene begin og end. Programmet avsluttes alltid med end. (End med punktum!)
Writeln er det vi kaller en innebygd prosedyre i Pascal. Denne prosedyra skriv ut på skjermen, det som blir spesifisert i parantesane (etter prosedyra). I dette tilfellet teksten: Dette skal gå bra!. All tekst som skal skrives ut, må stå i enkle hermetegn (dråper).

Uttrykket **Uses WinCrt** må tas med for at Turbo Pascal automatisk skal opprette et vindu der utskriften kan plasseres. Dette er spesielt for Windowsversjonen av Pascal, og alle programmene som står i læreboka må ha med denne ekstra linja for å fungere under Windows!!! WinCrt er det vi kaller en unit, eller eit bibliotek. Så snart dette biblioteket er tatt med i et program fører det til at vi får tilgang på ulike innebygde kommandoer som er definert i denne uniten.

Korte bemerkninger som skrives inn i programmet for vår egen hukommelses skyld kalles **kommentarer**. Kommentarer kan skrives inne i programmet mellom tegnene { og }, eller mellom (* og *). I disse kommentarene kan du skriva inn den tekst du selv ønsker. Kommentarene blir hoppet over ved kompileringen av programmet.

Eksperimenter med systemet. Som øvelse kan du åpne fila PRAT.PAS som jeg sender med nedendor sammen med denne del 1 av Pascal.
Inspiser koden skrevet inn i fila. Velg så Run i menyen. Da oversettes prat.pas først til maskinkode (compileres), deretter lagres den på harddisken som PRAT.EXE, og tilslutt kjøres denne maskinkoden på maskina.

Se om du kan forklare hva programmet gjør steg for steg:

```
PROGRAM PRAT;

Uses WinCrt;
VAR forst, sist : char;

Begin
    clrscr;
    writeln ('GOD DAG! Mitt navn er Mystisk Box,');
    writeln ('men du kan bare kalle meg M.B.');
    writeln;
    writeln ('Vær vennlig å svare ved å bruke tastaturet, og ');
    writeln ('husk endelig å avslutte hvert svar ved å trykke RETUR');
    writeln;
    writeln ('La oss begynne med begynnelsen, som de sier.');
    writeln ('Fortell meg hva du heter til fornavn:');
    readln (forst);
    writeln ('Så fint navn du har! Jeg ville like å bli kjent med deg');
    writeln ('Hva er etternavnet ditt?');
    readln (sist);
    writeln ('Så koselig å treffe deg   ',forst,'.',sist,'.');
    writeln;
    writeln ('Det er så ensomt inni denne boksen her...');
    writeln ('fortell meg litt om utsiden?');
    writeln ('Regner det ute nå?');
    readln;
    writeln ('Det var interesant');
    writeln;
    writeln (' Uff da ! Nå får jeg mye arbeid som må gjøres igjen..');
    writeln (' Ka e klokka blitt?');
    readln;
```

```
    writeln (' Tida har gått fra meg. Det blir hardt å ta igjen dette');
    writeln (' Jeg må bare skynde meg');
    writeln (' Ha det da   ',forst,'.',sist,'.');
    writeln (' Det var helt topp å prate med deg!');
     readln;
END.
```

DEL 2: PASCALPROGRAMMERING: DATATYPER.

Av Steinar Thorvaldsen, Info1-Tromsø. 16.10.95.

Les Kap. 3 og 4 i læreboka i sammenheng med at du ser på dette:

A. VARIABLE.

I Turbo Pascal kan vi definere det som kalles **variable**. Disse variablene er lagringsplasser i maskinhukommelsen (RAM) for ulike typer data. Pascal særmerker seg som dataspråk ved at det krever at alle variable defineres med navn og type før vi kan ta fatt på selve det programmet som skal utføres. I første omgang kan dette virke tungvint, men i neste omgang gir det oss den naturlige kontroll med de data som skal bearbeides. Vi skal i starten arbeide med følgende typer variable:

Navn:	**Tillatt verdi:**	**Størrelse i RAM:**
Integer	-32768..32767	(2 bytes)
Longint	-2147483648..2147483647	(4 bytes)
real	2.9e-39..1.7e38	(6 bytes)
char	1 tegn(karakter)	(1 byte)
String[N]	samling av tegn	(N+1 bytes)

En variabel må alltid defineres å tillhøre en spesiell type, f.eks. integer. Dette avgjør at det i denne konkrete variabelen kun kan lagres heltall og ikke f.eks. en bokstav(char) eller et desimaltall (real). Ved definering av variable bestemmer en altså innholdet av variabelen. Vi kan også konvertere data mellom to variabeltyper, f.eks. mellom real og string. Til dette brukes egne konverteringsfunksjoner (se læreboka kap 10.1 og 12.2).

B. DEFINISJON AV VARIABLE

Variable blir definert/deklarert etter det reserverte ordet **VAR** ved programmets begynnelse. Her gis de variable **navn** i tillegg til **type**. Typen spesifiserer hva som kan lagres i variabelen.
Den delen av programmet der de variable blir definert kalles gjerne for **deklarasjonsdelen** til programmet.
Delen som kommer senere (mellom **begin** og **end**.) kalles ofte for **instruksjonsdelen** i programmet.

```
Program Test2;
Uses
  WinCrt;

VAR
    AntallStud        : Integer;
    GjHoyde           : Real;
    Svar              : Char;
    Navn, Adresse  : String[25];
begin
    ....
end.
```

I programeksemplet over er det definert fem ulike variable:
Den første har navnet AntallStud og i denne variabelen kan det lagres heltall.
Den andre har navnet GjHoyde og kan lagre desimaltall.
Den tredje har navnet Svar og kan lagra **ett** tegn.

Variabel nummer fire og fem er like, og begge kan lagre en streng (en tekst) på maksimalt 25 tegn. Som du ser kan vi definere flere like variable, ved å liste de opp med komma mellom.

For navngivning av variable gjelder en del regler:

- Første tegn i navnet skal være en bokstav.
- Æ, Ø, Å er ikke tillatt.
- Tall er tillatt inne i variabelnavnet, men ikke i starten.
- Store og små bokstaver er tillatt.

Det skilles ikke mellom små og store bokstaver i variabelnavn. Det betyr at du gjerne kan definere en variabel som heter **Svar**, og senere referere til den under navnet **svar**.

Det er viktig å gi de variable du skal bruke gode navn som forteller noe om hva de inneholder.

C. TILORDNING AV VERDI TIL VARIABLE.

Variable kan gis **verdi** (innhold) gjennom en såkalt tilordnings-setning i programmet. **:=** er tilordningsoperasjon og leses gjerne **"settes lik"**. Uttrykket

```
AntallStud := 23;
```

fører til at tallet 23 blir lagret i variabelen med navnet AntallStud.

```
Program Test2;
Uses
  WinCrt;

VAR
    AntallStud        : Integer;
    GjHoyde           : Real;
    Svar              : Char;
    Navn, Adresse   : String[25];

begin
    AntallStud     := 23;
    GjHoyde        := 1.75;
    Svar           := 'J';
    Navn           := 'Blaise Pascal';
end.
```

Alle variable kan selvfølgelig også tilordnes verdi ved at den leses inn fra **tastaturet**. Dette utføres ved bruk av de innebygde kommandoene Read eller Readln. I praksis fører bruk av disse kommandoene til at programmet stopper opp og venter på at brukeren skal skrive inn noe fra tastaturet, og avslutte dette med å trykke Retur-tasten. Den verdi eller tekst som brukaren har skrevet inn blir automatisk lagret i variabelen som er står i parantesen etter kommandoen Read eller Readln:

```
begin
   Readln(AntallStud);
   Readln(GjHoyde);
   Read(Svar);
   Read(Navn);
end.
```

Readln skifter linje (flytter skrivemerket til starten av neste linje) etter at brukeren har skrive inn data og trykket Retur-tasten.
Read skifter ikke linje etter input fra tastaturet.

D. KONSTANTER.

Konstanter bruker vi til å lagre faste verdier gjennom hele programmet. Forskjellen mellom konstanter og variable er at variablene kan endre verdi når programmet kjøres, mens konstantene forblir de samme gjennom heile programmet. Konstantdefinisjonene kommer som regel før variabeldefinisjonene, og konstanter kan ikke tilordnes nye verdier under kjøring av programmet

Konstantdeklarasjonen skal skrives i definisjonsdelen av programmet:

```
CONST
Moms = 23;
Fartsgrense = 80;
DagerIJanuar = 31;
Forfatter = 'Steinar Thorvaldsen';
```

Fordelen med bruk av konstanter er at dersom en konstantverdi som f.eks. momsen blir endret, er det nok å oppdatere denne verdien **en** plass i programmet. Dette gjør at endriner av programmet blir lett å gjennomføre. Ved bruk av navn på konstanter blir det også enklere å forstå programmet siden verdien er navngitt.

E. PROGRAMEKSEMPEL.

```
PROGRAM Skattetrekk;

USES    WinCrt;

VAR     Navn : String[25];
        Inntekt, Skatteprosent, Netto : Real;

Begin
        Write ('Hva heter du? ');
        Readln (Navn);
        Writeln;
        Write ('Skriv inntekt og skatteprosent med blank imellom: ');
        Readln (Inntekt,Skatteprosent);
        Netto := Inntekt - (Inntekt*Skatteprosent)/100;
        Writeln;
        Writeln ('$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$');
        Writeln;
        Writeln ('Nettoinntekten til ',Navn,' er ',Netto:8:2);
        Writeln;
        Writeln ('$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$');
End.
```

DEL 3: VALG i PASCAL: IF og CASE.

Av Steinar Thorvaldsen, Info1-Tromsø. 03.11.93. Endret 09.11.95

Les Kap. 5 i læreboka i sammenheng med at du ser på dette:

I pascal har vi to setninger som kan brukes når vi skal få et program til å foreta et valg eller en test: IF-setningen og CASE-setningen. IF-setningen er gunstig å bruke når det er TO valgtilfeller man skal skille mellom i programmet. I tillegg er IF-setningen mer generell enn CASE-setningen når det gjelder betingelsene som danner grunnlag for valget. Men ofte kan samme valgsituasjon programmeres både med IF og Case.

A. IF-setningen.

IF-setningen har følgende struktur:

```
If (betingelse) then
Utføres dersom betingelsen er SANN
else
Utføres dersom betingelsen er USANN
```

Else-delen av valget trenger ikke alltid være med og er valgfri. Det vil si at en kan la deler av programmet bli utført dersom betingelse er sann, og ellers ingen ting. Legg merke til at det **ikke** er semikolon på avslutningen av linja framfor else. Dette er et unntak fra regelen om at alle instruksjoner skal avsluttes med semikolon, og viser at IF-setningen skal oppfattes som en helhet.

Anta som et eksempel at vi skal lage et program som skal lese inn to heltall fra tastaturet og skrive dem ut med det minste først.

```
Program MinsteFoerst; { Oppgave 5.6 i læreboka}
      (* Programmet leser inn to heltall fra tastaturet og skriver dem
         så ut, med det minste først. *)
uses Wincrt;
var
   Heltall1,
   Heltall2 : Integer;

Begin (* hovedprogram *)
   Write ('Oppgi første heltall : ');
   readln (Heltall1);
   write ('Oppgi andre heltall : ');
   readln (Heltall2);

      (*------ Utskrift starter ------*)
   If Heltall1 < Heltall2 then
      writeln (Heltall1,'    ',Heltall2)
   else
      writeln (Heltall2,'    ',Heltall1);
   readln;
end.
```

Her er det bare to muligheter, noe som passer godt med IF-setningen.
Det tilfellet at begge tallene er like, kan klassifiseres under begge mulighetene if/else.

IF-setningen kan brukes på utallige måter, f.eks til sorteringsopperasjoner. Her følger et eksempel hvor vi finner fram til det største tallet av et utvalg på tre tall. Merk teknikken med å starte med å anta at det første tallet er størst, og så justere seg inn etterhvert:

```
Program FinnerStoersteTall; { Oppgave 5.7 i læreboka noe endret}
      (* Programmet leser inn tre heltall fra tastaturet og finner frem
         det største tallet, og skriver det ut. *)
uses Wincrt;

var
   Heltall1,
   Heltall2,
   Heltall3,
   DetStoerste : Integer;

Begin (* hovedprogram *)
   Write ('Oppgi første heltall : ');
   readln (Heltall1);
   write ('Oppgi andre heltall : ');
   readln (Heltall2);
   write ('Oppgi tredje heltall : ');
   readln (Heltall3);

               (*------ Vi finner det største ------*)
   DetStoerste := Heltall1;
   IF Heltall2 > DetStoerste then DetStoerste := Heltall2;
   IF Heltall3 > DetStoerste then DetStoerste := Heltall3;
               (*------ Utskrift starter ------*)

   write ('Det største av tallene var: ');
   writeln (DetStoerste);
end.
```

B. CASE-setningen.

Denne setningen er ofte oversiktlig og grei å programmere når vi velger mellom fler enn to alternativer. I et case-valg kan vi sette opp mange alternative utganger av valget. Testen i case-valget går i korthet ut på at innholdet av en variabel skal sjekkast, og ulike muligheter for variabelens innhold blir satt opp.

Case valget har slik struktur:

```
case variabel of
 alternativ1 : .... ;
 alternativ2 : .... ;
 alternativ3 : .... ;
else
 .... ;
end;
```

Else-delen er valgfri i case-setningen.
Case må avsluttes med end; (uten begin i starten).

Vi illustrerer case-setningen best med eksempler:

```
Program  Kalkulator;
      (* Programmet  ber om to tall og en regningsart. Programmet
         utfører  så valgte regningsart mellom  de to tallene.  *)
uses Wincrt;
var
   Tall1,
   Tall2,
   Svar         : Real;
   Regningsart  : Char;

begin  { Hovedprogram }
   Svar := 0;
   write ('Oppgi første tall : ');
   readln (Tall1);
   write ('Oppgi andre tall : ');
   readln (tall2);
   write ('Velg regningsart (+, -, *, /) : ');
   readln (Regningsart);

            (*------ Vi beregner svaret  ------*)
   Case Regningsart of
      '+' : Svar := Tall1 + Tall2;
      '-' : Svar := Tall1 - Tall2;
      '*' : Svar := Tall1 * Tall2;
      '/' : Svar := Tall1 / Tall2;
    end;
            (*------  Vi skriver så ut på 1 linje  !!! ------*)
    writeln;
    writeln (Tall1:2:2,' ',Regningsart,' ',Tall2:2:2,' = ',Svar:2:2);
    readln;
end.
```

Dette programmet vil ikke utføre noen beregning hvis du taster inn noe annet enn en av de 4 regningsarter i variabelen Regningsart. Og hvis variabelen Tall2 er 0 og regningsart er /, fås en stygg programfeil. På disse to punktene kan altså programmet forbedres... Se læreboka side 99 og 100.

Merk at den eller de variable som brukes som betingelseskriterium for valg i CASE-setningen, må være av typen **Character**(Tegn) eller **Integer**(Heltall). I eksemplet var variabelen Regningsart av typen Char.
Variable av typen Real(Desimaltall) vil ikke gå her da denne er av en mer kontinuerlig type som er vanskelig å teste på likhet.

Man kan la den variable være innenfor et intervall til hvert av valgen i CASE-setningen. Man kan også bruke ELSE til slutt for å dekke opp alle andre muligheter som ikke er brukt før. Hvis det skal skje mer enn én ting ved et valg, rammer man inn instruksjonene med begin end; :

Et eksempel:

```
Program Kriger;
(* Programmet leser inn et årstall, og skriver ut melding hvis
det er verdenskrig i denne tiden *)

var
   aar: Integer;

Begin (* Hovedprogram *)
   Write ('Oppgi et årstall i vårt århundre:  ');
   Readln (aar);

   Case aar of
     1914..1918: Writeln ('Den første verdenskrig pågikk');
     1940..1945: Begin
                    Writeln ('Den andre verdenskrig pågikk');
                    Writeln ('Over 50 mill. mennesker ble drept');
                 End;
     Else  Writeln ('Ingen verdenskriger');
   End; (*Case*)
   Writeln;
   Readln;
END.
```

C. Oppgaver til neste samling:

Oppgave 5.3 , 5.4 og 5.5 i læreboka side 100.
I oppgave 5.5 kan det lønne seg å regne vekten i gram slik
at en case-setning kan brukes.

DEL 4: LØKKER i PASCAL: FOR, WHILE OG REPEAT.

Av Steinar Thorvaldsen, Info1-Tromsø. 16.11.93 Endret 23.11.95

Les Kap. 6 og 7 i læreboka i sammenheng med at du ser på dette:

Løkker brukes til å å gjenta deler av et program inntil en betingelse er oppfylt, eller et gitt antall gjennomløp er utført.
I Pascal kan slike gjentakelser gjøres på **tre** ulike måter, men i prinsipp er det det samme som skjer, og ofte er det valgfritt hva for løkke-setning som kan brukes.

De 3 løkketypene vi har er:

For...to....do
While....do
Repeat.....until

I programmering er det ofte behov for å bruke maskina til å gjenta ting mange ganger (løkker). I f.eks. oppgave 4.3 c som vi har sett på tidligere, var det 5 varepriser som skulle leses inn. Dette er det naturlig å gjøre ved hjelp av én read-setning som gjentas 5 ganger:

```
Program Varepriser;  { Oppgave 4.3 c med løkke }
        (* Programmet leser inn 5 varepriser. Disse summeres, og
           kontantrabatt fratrekkes. *)
Uses Wincrt;
var
   Pris,
   Summen : Real;
   PrisNummer : Integer;

begin { Hovedprogram }
   Pris := 0;
   Summen := 0;
   For PrisNummer := 1 to 5 do
      begin
         write ('Oppgi ',PrisNummer,'. pris : ');
         readln (Pris);
```

```
        Summen := Summen + Pris;
        writeln ('    Summen så langt : ',Summen:7:2);
      end;
        (*------ Vi trekker så fra rabatten ------*)
   Summen := Summen * 0.95;
   writeln;
   writeln ('ENDELIG SUM Å BETALE: ',Summen:7:2);
   readln;
end.
```

La oss så se på et eksempel som kan programmeres med hver av løkke-setningene. Det eksemplet vi velger, er en velkjent aktivitet fra skolestua: Å skrive ut gangetabeller.

A. FOR-løkke

```
Program SkriverToGangen;
        (* Programmet skriver ut 2-gangen ved hjelp av en for-løkke.*)
Uses WinCrt;

var
   Telletall :Integer;

begin { Hovedprogram }
   clrscr;
   writeln ('2-gangen.');
   writeln;
   For Telletall := 1 to 10 do
      writeln (Telletall:3,' * 2 =',Telletall*2:3);
   Readln;
end.
```

For-løkka har følgende form:

```
      For tellevariabel := startverdi to stoppverdi do
            begin
                  ....
                  ....
            end;
```

For-løkka har et fast antall gjennomløp, som er bestemt når løkka starter. Programbiten som blir utført for hvert gjennomløp, er setningene mellom begin og end.
Det er viktig å være klar over at tellevariabelen i løkka automatisk blir oppdatert for hvert gjennomløp. Denne økingen av tellevariabelens verdi treng altså ikke programmeres.

B. REPEAT-løkke

```
Program SkriverTreGangen;
        (* Programmet skriver ut 3-gangen ved hjelp av Repeat-løkke.*)
Uses WinCRT;

var
   Telletall :Integer;

begin { Hovedprogram }
   clrscr;
   writeln ('3-gangen.');
   writeln;
   Telletall := 1;
   repeat
      writeln (Telletall:3,' * 3 =',Telletall*3:3);
      Telletall := Telletall + 1;
   until Telletall > 10;
   readln;
end.
```

Repeat løkka har følgende form:

```
      Repeat
            ....
            ....
      until (betingelse = True);
```

Repeat-until løkka repeterer instruksjonene mellom repeat og until, helt til betingelsen blir sann.
Merk at i denne løkka trenger vi ikke å knytte flere instruksjoner sammen med begin og end. De reserverte ordene **repeat** og **until** fungerer i praksis på samme måte i denne type løkke.
Programmet må sørge for at betingelsen blir sann, for å stoppe løkka.

C. WHILE-løkke

```
Program SkriverFireGangen;
        (* Programmet skriver ut 4-gangen ved hjelp av While-løkke.*)
Uses WinCRT;

var
   Telletall :Integer;

begin { Hovedprogram }
   clrscr;
   writeln ('4-gangen.');
   writeln;
   Telletall := 1;
   While Telletall <= 10 do
      begin
         writeln (Telletall:3,' * 4 =',Telletall*4:3);
         Telletall := Telletall + 1;
      end;
   readln;
end.
```

While løkka har følgende form:

```
      While (betingelse  =  True) do
            begin
                  ....
                  ....
            end;
```

While-løkka repeterer instruksjonene, så lenge betingelsen er sann. Programmet må selv sørge for at betingelsen blir usann, for å stoppe løkka. Det er ingen automatisk oppdatering av tellevariable i while-løkker, slik som det er i for-løkker.

Merk at tellevariabelen "Telletall" initieres ved i dette tilfellet å settes lik 1 som startverdi. Dette er viktig fordi variabelen har en tilfeldig verdi etter at den er definert i var-setningen. For å få et bestemt antall gjennomløp av løkka, må denne variabelen altså gis en startverdi.
For det andre må tellevariabelen økes inne i løkka (Telletall := Telletall + 1), for at testen i løkka (Telletall <= 10) skal bli usann, og dermed stoppe løkka. Dersom denne økingen av variabelen "Telletall" blir uteglemt, vil løkka ikke stoppe, fordi testen i starten av løkka vil alltid være sann.

Forskjellen mellom while og repeat- løkkene er at en repeat løkke alltid må ha minst ett gjennomløp, for heile tatt å komme fram til testen som kan avslutte løkka. Ei while løkke treng ikke ha noe gjennomløp i det heile tatt. Dette skjer dersom testen i løkka i utgangspunktet er usann.

D. Oppsummering:
-Skal mer enn en instruksjon utføres i for- eller while-løkkene, må de knyttes sammen i en blokk med begin og end.
-For-løkka har automatisk oppdatering av tellevariabelen.
-I while- og repeat-løkkene må programmet sørge for å stoppe løkkene ved å oppdatere variable som er med i betingelsen.
-Repeat-løkka vil alltid ha minst ett gjennomløp, mens While- og For-løkkene ikke er tvunget til å ha gjennomløpninger.

DEL 5: PROSEDYRER i PASCAL.

Av Steinar Thorvaldsen, Info1-Tromsø. 24.11.93 endret 5.1.96

Les Kap. 8 i læreboka i før du ser på dette:

Prosedyrer blir det neste naturlige steg på programmeringskunstens vei. Saken er nemlig at programmer fort har en tendens til å bli stor uoversiktlige, og her er det prosedyrene kommer inn som hjelp. Oppdeling av programmer i prosedyrer er altså svaret på problemene med å holde **OVERSIKT**. Dette er det vesentligste problem vi står overfor når det gjelder større programmeringsoppgaver. For å oppnå dette bruker vi prosedyrer til å bryte programmet ned i mindre og avgrensede deler, og dermed øke oversikten og lesbarheten i programmet.

I tillegg åpner prosedyredefinisjoner for gjennbruk av programkoden, ved at en prosedyre kan kalles opp flere ganger, uten at selve koden i prosedyren trenger å bli gjentatt. En viktig begrunnelse for bruken av prosedyrer er derfor at det vi programmerer på denne måten blir mer **GENERELT** anvendelig.

Med en **prosedyre** mener vi en avgrenset programbit som utfører en bestemt oppgave (ofte kalt et delprogram). Litt forenklet kan vi si at vi navngir en naturlig samling av instruksjoner slik at disse kan kalles opp ved hjelp av navnet til prosedyren. Kallet skjer i hovedprogram eller i en annen prosedyre

Egentlig har vi allerede brukt flere prosedyrer. Tenk på en setning som

```
WRITE (Tall);
```

Dette er en standardprosedyre i Pascal og tilhører slik Pascal sitt standard ordforråd. Det vi skal gjøre er å utvide dette ordforrådet ved selv å definere NYE ord (prosedyrer). Dermed gjør vi programmet vårt mer og mer "ordrikt" og rikere på begreper for å beskrive og løse de problemstillinger vi arbeider med.

I læreboka eksemplifiseres prosedyrer med tegning av ulike typer figurer på skjermen, og det vises hvordan dette kan programmeres på en oversiktlig og fornuftig måte ved hjelp av prosedyrer. For ikke å gjenta dette, skal jeg hente mine eksempler fra et annet felt som er velkjent for enhver lærer:

Programeksempel.
Studer programeksemplet med regneoppgaver under. Legg spesielt merke til at programmet er delt opp i to prosedyrer, og hvordan disse er kalt opp i hovedprogrammet. Hovedprogrammet blir dermed kort og lettforståelig. Egentlig er det nok å lese hovedprogrammet for å vite hva programmet gjør.

```
PROGRAM addisjonsprove;     (* Nr6.18 i oppgavesett 4  *)

USES WinCrt;
VAR AntallRiktigSvar:Integer;


Procedure GiOppgaver;
VAR Telletall,Tall1,Tall2,RiktigSvar,ElevensSvar:Integer;
```

```
Begin
writeln('Her får du en addisjonsprøve på 20 oppgaver.');
writeln('Etter å ha svart på alle, får du vite hvor mange riktige svar');
writeln('du hadde.');

Randomize;
FOR Telletall:=1 TO 20 DO      (* 20 oppgaver *)
  BEGIN
   Tall1:=Random(20)+1;
   Tall2:=Random(20)+1;
   Riktigsvar:=Tall1+Tall2;
   writeln;
   writeln;
   writeln(Tall1:12);
   writeln('     + ',Tall2:5);
   writeln('     _______');
   write('          ');
   READLN(ElevensSvar);
   writeln;
   IF ElevensSvar=RiktigSvar THEN
    BEGIN
     writeln('RIKTIG!!!');
     AntallRiktigSvar:=AntallRiktigSvar+1;
    END
   ELSE
    BEGIN
     writeln('Beklager, du svarte feil.');
     writeln('Riktig svar er: ',RiktigSvar);
    END;
  END;
writeln('Du har nå svart på alle oppgavene.');
End;  {--------------Procedure GiOppgaver------------}

Procedure GiKarakter;
Var Karakter:String (.2.);

Begin
CASE AntallRiktigSvar OF
     0.. 2: Karakter := 'Lg';
     3.. 6: Karakter := 'Ng';
     7..13: Karakter := 'G';
     14..17:Karakter := 'Mg';
     18..20:Karakter := 'Sg';
 END;

 writeln;
 writeln('Du fikk ',AntallRiktigSvar,' riktige svar.');
 writeln('Det gir deg karakteren ',Karakter,'.');
End;   {--------------Procedure GiKarakter------------}



BEGIN {Hovedprogram}
 AntallRiktigSvar:=0;
 Clrscr;
 GiOppgaver;
 GiKarakter;
END.  {--------------Hovedprogram------------}
```

Merk at alle prosedyrer må skrives inn før første begin i hovedprogrammet. Har du bruk for egne variable inne i prosedyrene, skal disse defineres i en VAR-setning før første begin i prosedyren. Disse variablene kalles **lokale**. Variable som er definert i hovedprogrammet kan fritt benyttes i alle prosedyrene. Slike variable kalles **globale**.

Som oppsummering kan vi merke oss at definisjonen av prosedyrer er svært lik den definisjonen vi kjenner for et program. Forskjellene er:
- Definisjonen av en prosedyre starter med det reserverte ordet **procedure**, i steden for program.
- Prosedyren skal avsluttes med end; (end med semikolon), og ikke end. (end med punktum).

DEL 6: PROSEDYREPARAMETERE og FUNKSJONER. SAMMENDRAG.

Av Steinar Thorvaldsen, Info1-Tromsø. 9.12.93 Oppdatert 17.1.96

Lærebok: Havnen/Quale: "Strukturert programmering med Pascal"
NKI-forlaget, 3.utgave 1992.

Les Kap. 9.1 - 10 i læreboka i før du ser på dette:
Merk et par trykkfeil:
Avsnitt 9.3 side 178: nederst skal det stå **tall** to steder istedenfor t.
Avsnitt 9.3 side 182: midt på skal det stå :
slutt, **riktig**: Boolean; som 2. linje bak VAR i hovedprogrammet.

Som før nevnt har programmer fort en tendens til å bli uoversiktlige. Da er det prosedyrene kommer inn som hjelp, ved at vi kan dele opp hovedproblemet i delproblemer. Deretter programmeres hvert delproblem som en prosedyre. For å bruke et slagord om denne framgangsmåten så kan vi beskrive metoden som **"Splitt og hersk"**.
(Merk at slagordet har en annen mening i programmerings-sammenheng enn i politikken.)

Egentlig er en slik framgangsmåte den helt naturlige i alle typer strukturert problemløsning. Bare tenk på det å bake en pizza. Her er delproblemene:

-Lage bunn
-Lage fyll
-Kombiner bunn og fyll.

I Pascal settes hoveprogrammet sammen av delprosedyrene, mens hver delprosedyre programmeres for seg før første Begin i hovedprogrammet. Dermed blir hovedprogrammet kort, og detaljene overlates til delprosedyrene. Ofte programmeres og testes hovedprogrammet først, mens prosedyrene programmeres senere (selv om de altså skal skrives inn før hovedprogrammet).

Den jobben en delprosedyre skal gjøre, kan være av mange slag. Det kan være å gi en bestemt tone i høytaleren, skrive noe til skjermen, foreta en innlesning av variable eller gjøre en beregning.

Vi benytter **parametre** til å overføre data inn i prosedyrer og funksjoner. Parameter defineres på samme måte som en variabel, dvs. ved å definere et parameternavn og en datatype. Forskjellen er at parametre defineres i parentes etter prosedyrenavnet, og ikke under det reserverte ordet VAR i deklarasjonsdelen av prosedyren.

Av spesiell interesse er den forandring en prosedyre kan gjøre med en eller flere variable. Disse variable kalles prosedyrens **parametere**. I Pascal er variabelkontrollen grundig, og vi har to hovedtyper parametere: Verdi-parametere og Var-parametere.

A. Var-parametere.
Disse kalles gjerne også INN/UT-parametere til prosedyren. I prosedyrehodet skrives de med Var foran slik:

```
Procedure Adder ( VAR x, y : Integer);
Begin
 ........;
End;
```

Dette betyr at de endringer som skjer med de to parametre i prosedyren Adder, vil tilbakeføres UT til hovedprogrammet eller det program som kalte opp prosedyren. Det lages så og si en midlertidig toveis kobling mellom de to

variablene i hovedprogrammet, og de to i prosedyren. Bruk av variabelparameter er en måte å føre informasjon inn i en prosedyre, samtidig som en sikrer seg tilbakemelding om endringer av disse parametrene.

Eksempel:

```
Program VarParameter;

Uses WinCrt;
Var a, b : Integer;

Procedure Adder ( VAR x, y : Integer);
Begin
       x := x + 1;
       y := y + 1;
       Writeln ('I prosedyren er tallene: ', a, b :4),

End;

BEGIN (*Hovedprogram*)
       a := 5;
       b := 10;
       Writeln ('Starttall i Hovedprog.: ', a, b :4);
       Adder (a , b);
       Writeln ('Sluttall  i Hovedprog.: ', a, b :4);
       Readln;
END.
```

Merk at i prosedyren Adder, er de to variablene (her kalt x og y), **generelle representanter** for de to parametrene den inneholder. Vi kan kalle prosedyren med de variabelnavn vi vil på disse to parameterplassene, forutsatt at vi holder oss innenfor parametertypen Integer:

```
Adder (tall1, tell2);
Adder (a, b);
Adder (x, y);   osv.
```

Ved kjøring gir dette programmet utskriften:

```
Starttall i Hovedprog.: 5  10
I prosedyren er tallene:  6   11
Sluttall  i Hovedprog.: 6  11
```

Variabelparameter kan bare kalles opp i en prosedyre ved å bruke **variable** i kallet. Det følgende oppkall er derfor ulovlig, og forårsaker kompileringsfeil:

```
Adder (10, 20);
```

Grunnen til dette er at når variabelparameteren (som "x") skal returnere sin verdi, må denne verdien kunne tilordnes en variabel.

B. Verdi-parametere.

Disse kalles gjerne også INN-parametere til prosedyren. I prosedyrehodet skrives de uten noe foran slik:

```
Procedure Adder ( x, y : Integer);
Begin
      ....
End;
```

Når prosedyren skal kjøres, må den kalles opp med samme antall parameter som i definisjonen, f.eks..:

```
Adder (1000, rente);
```

Det som skjer i praksis, er at prosedyren "Adder" startes, og tallet 1000 blir tilordnet parameteren x og innholdet i rente tilordnes y. Disse parametrene kan så brukes inne i prosedyren. Kaller vi opp prosedyren med et annet tall

som parameter, vil det være denne verdien som tilordnes x og y, og blir brukt inne i prosedyren.

Dette betyr at de endringer som skjer med de to parametre inne i prosedyren Adder, ikke vil tilbakeføres til hovedprogrammet eller det program som kalte opp prosedyren. Det lages kun en **kopi** av de to variablene og prosedyren jobber med kopiene til den er ferdig. Da slettes kopiene, og de har ingen direkte følger for de tilsvarende variable i hovedprogrammet. Verdiparametere brukes ofte til mellomregninger som vi vet ikke må klusse til noe med variable i hovedprogrammet. Hvis en prosedyre har to eller flere parametere, skal disse skilles med semikolon. Forskjellen på definisjonen av variabel- og verdiparameter er kun at variabelparameter har med det reserverte ordet VAR i definisjonen.

Eksempel:

```
Program VerdiParameter;

Uses Wincrt;
Var a, b : Integer;

Procedure Adder (  x, y : Integer);
Begin
       x := x + 1;
       y := y + 1;
       Writeln ('I prosedyren er tallene: ', a, b :4),

End;

BEGIN (*Hovedprogram*)
       a := 5;
       b := 10;
       Writeln ('Starttall i Hovedprog.: ', a, b :4);
       Adder (a , b);
       Writeln ('Sluttall  i Hovedprog.: ', a, b :4);
       Readln;
END.
```

Her kan vi også kalle prosedyren med de variabelnavn vi vil på disse to parameterplassene, forutsatt at vi holder oss innenfor parametertypen Integer:

```
      Adder (tall1, tell2);
      Adder (a, b);
      Adder (x, y);
      Adder (5, 10);   -mulig fordi vi har verdi-parametre her
```

Ved kjøring gir dette programmet utskriften:

```
      Starttall i Hovedprog.: 5  10
      I prosedyren er tallene:  6   11
      Sluttall  i Hovedprog.: 5  10
```

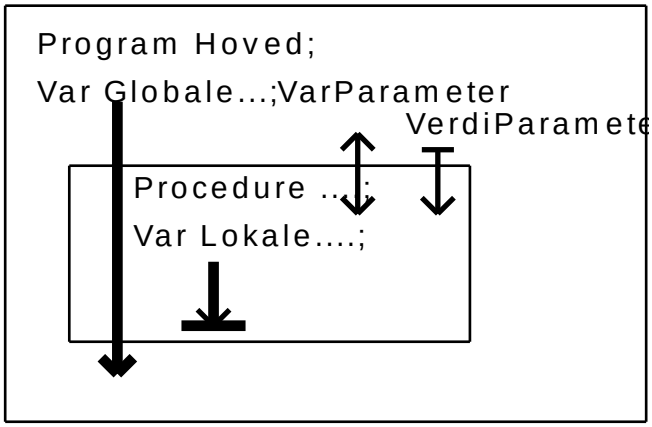
Det skjer ingen endring tilbake til hovedprogram fordi verdi-parametre betyr "enveiskjøring".

Parametrene til prosedyrer må alltid være av same datatype som de er definert i prosedyren. Følgende oppkall vil derfor være feil:

```
      Adder('500', '100');
```

siden "dråpene" ('..') angir at det er tekststrenger.

Skjematisk kan vi sette opp dataflyten i Pascal slik (her har vi også tatt med gyldighetsområdet for globale og lokale variable):



Et naturlig spørsmål vil da være om det ikke vil greie seg med å gjøre alle variable vi trenger globale? Dermed kunne vi klare oss uten noen som helst parameteroverføringer i prosedyrehodene... Men dette er en strategi som garantert fører til rot etter en stund.

Det er en god regel å definere mest mulig **lokale** variable, og minst mulig globale variable. Med dette oppnår vi en bedre oversikt ved at variable er definert i tilknytting til der de benyttes. Dessuten hindrer at globale variable "flyter rundt" i programmet og blir brukt til ulike formål på mange ulike steder. For det tredje tar lokale variable mindre plass under kjøring, fordi lokale variable blir slettet når prosedyren avsluttes. De data vi bruker å definere **globalt**, er slike som så og si alle prosedyrene benytter fordi de representerer den sentrale datastrukturen i hele programmet.

Vi bruker prosedyrer med **parametre** for å sikre oss at dataflyten i de ulike delene av programmet skjer på en kontrollert måte. Uten denne kontrollen blir det fort altfor mye å tenke på samtidig. Og det er som kjent begrenset hva et menneske kan klare å ha i hodet på samme tid...

c. Funksjoner.
Funksjoner i Pascal er egentlig en bestemt type prosedyrer, nemlig slike som gir ut kun ett resultat. Siden de alltid gir ut ett resultat, må de tilhøre en bestemt datatype (Integer, Real osv.). De må også brukes annerledes enn prosedyrer.

Vi har allerede brukt en del innebygde funksjoner i Pascal, slik som Random o.l. Se læreboka.

Funksjoner defineres omtrent som prosedyrer, men det tillates ikke Var-parametere:

```
FUNCTION  FunkNavn (Parameterliste) : FunkDatatype;
Var ....
Begin
....
FunkNavn := uttrykk;
....
End;
```

Parameterlisten inneholder de INN-parametrene funksjonen kan benytte.
Funksjonens datatype kan være Integer, Real, Char, String eller Boolean, som for vanlige variable.
Eksempler finner dere i læreboka i avsnitt 10.3 .

Når funksjonen skal kalles fra f.eks hovedprogrammet, gjøres det på en litt annerledes måte enn for prosedyrer:

```
Variabel := FunkNavn(Parameterliste);
```

og her må "Variabel" være av samme datatype som FunkNavn.

Egentlig er dette temmelig likt det funksjonsbegrepet vi kjenner fra matematikken, der vi er vant med å skrive:

```
y := F(x)
```

men i Pascal er vi ikke begrenset til å regne med tall, og funksjonene kan være funksjoner av flere variable.

Vi gjentar:
- En variabelparameter blir brukt både til å overføre data inn i og ut av prosedyrer/funksjoner.
- Prosedyrer/funksjoner kan ha flere variabelparameter, og de kan ha en blanding av verdi- og variabelparameter.
- Verdiparametre kan på sin side kalles opp med både verdier som tall, variable og beregningsuttrykk.

D. Eksempel: Beregning av trekantareal.
La oss illustrere bruken av prosedyrer og funksjoner, med en velkjent aktivitet fra ungdomsskolen: Beregning av areal av rettvinklede trekanter. Dette er ganske enkelt, men hvordan blir det så på en datamaskin?

Programmet skal først lese inn de tre sidene i trekanten. Dette skal gjøres i en egen prosedyre. Deretter testes om at alle sider er positive. Så skal en prosedyre sette disse i slik rekkefølge at det største tallet står sist.
En boolsk funksjon skal brukes til å teste om trekanten er rettvinklet. Hvis trekanten er rettvinklet, skal arealet utregnes. Hvis ikke skal bare en beskjed om at trekanten ikke er rettvinklet komme på skjermen.

Programmet skal spørre om det ønskes å regne for flere trekanter, og hvis dette ønskes, skal programmet regne for neste trekant, osv.:

```
Program ArealAvRetteTrekanter;
       (* Programmet leser inn lengden av de tre sidene i trekanten,
          tester om trekanten er rettvinklet, og regner ut arealet
          av trekanten hvis så er tilfelle. Hvis ikke kommer kun en
          beskjed om at trekanten ikke er rettvinklet. Programmet
          regner for flere trekanter så lenge brukeren ønsker dette.
          Programmet er laget av Ole Mikalsen. *)
uses Wincrt;
var
   side1, side2, side3  : Real;
   Valg   : Char;

Procedure LesInnSidene (VAR Side1, Side2, Side3 : Real);
begin
   repeat  (* hvis ikke alle tall positive *)
      write ('Oppgi side 1 : ');
      readln (Side1);
      write ('Oppgi side 2 : ');
      readln (Side2);
      write ('Oppgi side 3 : ');
      readln (Side3);
      writeln;
   until (Side1 > 0) AND (Side2 > 0) AND (Side3 > 0);
end;

Procedure Bytt (Var Nr1,Nr2: Real);
          (* Bytter de to variable. Nyttes av prosedyren SettStoersteSist *)
var
      HjelpeTall : Real;
begin
      HjelpeTall := Nr1;
      Nr1 := Nr2;
      Nr2 := HjelpeTall;
end;   (* prosedyre BYTT *)

Procedure SettStoersteSist (VAR Side1, Side2, Side3 : Real);
       (* Setter det største tallet sist slik at det kan beregnes om
          trekanten er rettvinklet. Benytter prosedyre Bytt *)
Begin  (* prosedyre SettStoersteSist *)
```

```
   IF Side1 > Side2 then BYTT(Side1,Side2);
   IF Side2 > Side3 then BYTT(Side2,Side3);
End;

Function Rettvinklet (Side1, Side2, Side3 : Real) : Boolean;
begin
   IF Sqr(Side1) + Sqr(Side2) = Sqr(Side3) then
      Rettvinklet := True   (*Denne testen kan være uheldig for real-var..*)
   else
      Rettvinklet := False;
end;


BEGIN { Hovedprogram }
   repeat (* så lenge bruker vil *)
      LesInnSidene (Side1, Side2, Side3);
      SettStoersteSist (Side1, Side2, Side3);
      IF Rettvinklet (Side1, Side2, Side3) then
         writeln ('Arealet av trekanten er : ',Side1 * Side2 /2 :4:2)
      else
         Writeln ('Trekanten er ikke rettvinklet !!!!!');
      writeln;
      write ('Ønsker du å regne for flere trekanter ? ( J / N )');
      Readln (Valg);
      writeln;
   until Valg in (.'n','N'.);
END.
```

E. SAMMENDRAG. HOVEDBEGREPER I PASCAL.
****

Operasjon	**Eksempel**
Variabeltilordning:	`Tall := 55;` `Navn := 'Tor';`
Lesing (input):	`Read (Tall);`
Skriving (output):	`Write ('Verdien er ', Tall:5);`

Valg (test):	`If (Tall>10) then Write('Du fikk rett')`
	`else Write ('Jeg vant ');`
	
	`Case Tall of`
	`1..10 : Writeln ('Jeg vant');`
	`11..100: Writeln ('Du fikk rett');`
	`Else Writeln ('Den gikk ikke');`
	`end;`

Repetisjon (løkke):	`For Teller:= 1 to Tall do`
	`Begin`
	`Writeln ('Teller er ',Teller);`
	`End;`
	
	`While (Tall*Tall < 1000) do`
	`Begin`
	`Writeln ('Kvdratet er :',Tall*Tall);`
	`Tall := Tall + 1;`
	`End;`
	
	`Repeat`
	`Writeln ('Kvadratet er :',Tall*Tall);`
	`Tall := Tall +1;`
	`until (Tall*Tall > 1000);`

Nye Prosedyrer:

```
Procedure Ny (Tall:Integer; Var Svar: Integer);
      Var
            Begin
            ---------
            End;
```

Prosedyrekall:

```
Ny (Tall,Svar);
Ny (55,A);
```

Som vi forstår så kjører datamaskinen kun regelbaserte programmer. Det er formelt bevist (C.Bøhm og G.Jacopini, 1966), at alle problemer som lar seg løse ved hjelp av datamaskin, kan løses ved bruk av tre grunnbegreper:

- -sekvens (begin..end, tilordning, input, output)
- -valg (if..then..else..)
- -repetisjon (while..do..)

Dette er grunnfunksjonene i Pascal og alle strukturerte, 3.generasjons programmeringsspråk. Programmering dreier seg altså dypest sett om fornuftig bruk og kombinasjon av disse tre begreper!