Föreläsning 6

- Programmeringsexempel: sköldpaddsgrafik
  - Illustrerar
    - Moduler
    - Imperativ programmering
- Programstruktur
    - (Förra gången: uttryck & kommandon, procedurer & funktioner, moduler)
    - Blockstruktur
    - Deklarationer
    - Deklarationers räckvidd (scope)
    - Nästlade block och namnkrockar
    - Rekursion

Sköldpaddsgrafik

Modulen TurtleGraphics

```
DEFINITION MODULE TurtleGraphics;

TYPE PenMode = (Black, None);

PROCEDURE Move(Distance:INTEGER);

PROCEDURE Turn(Degrees:INTEGER);

PROCEDURE PenColor(Pen:PenMode);

PROCEDURE ClearScreen;

PROCEDURE TurtleStop;

END TurtleGraphics.
```

- Se även Appendix F i boken.

Exempel

- Rita en triangel
- Rita en polygon

...

Programstruktur

Deklarationer

Deklarationer namnger saker

- konstanter
- typer
- variabler (platser i minnet)
- procedurer
- parametarar till procedurer

Deklarationer finns på två ställen i program

- Modul = import-lista + *deklarationer* + kommandon
- Procedur = procedurhuvud + *deklarationer* + kommandon

Block

- Block = deklarationer + kommandon

```
Block = {Declaration} [BEGIN StatementSequence] END.
```

  - Modul = import-lista + *block*
  - Procedur = procedurhuvud + *block*

Programstruktur

Block

```
MODULE m;

VAR v: ...;

PROCEDURE p1 (c:...);
  VAR w : ...;
BEGIN
 ...
END p1;

PROCEDURE p2 (d:...);
  VAR y : ...;
BEGIN
 ...
END p1;

BEGIN
  ...
END m.
```

- Deklarationer gäller inom blocket
- Block kan vara nästlade
- Deklarationer i ett omkringliggande block gäller i ett inre block.

Programstruktur

Block

Variabler

- En variabel är (som andra namngivna saker) bara åtkomlig i det block där den deklarerats
- Hur länge behåller en variabel sitt värde?

```
MODULE lifetime;

FROM InOut IMPORT WriteCard;

PROCEDURE p(first:BOOLEAN);
  VAR n:CARDINAL;
BEGIN
  IF first THEN
    n:=0
  ELSE
    INC(n)
  END;
  WriteCard(n,3)
END p;

BEGIN
  p(TRUE);
  p(FALSE);
  p(FALSE)
END lifetime.
```


Programstruktur

Block

Semantik för block

- Vad händer med block när programmet körs?
  1. Reservera plats i minnet för deklarerade variabler.
  2. Utför kommandona.
  3. Återlämna det reserverade minnet.
- Punkt 3 betyder att variablernas värden inte kan bevaras!

Programstruktur

- Exempel på nästlade block: vad skriver programmet ut?

```
MODULE Scope1;
FROM InOut IMPORT WriteInt, WriteLn;

VAR x , y : INTEGER;

PROCEDURE Print(m,n:INTEGER);
BEGIN
  WriteInt(m,5); WriteInt(n,5);
  WriteLn
END Print;

PROCEDURE First;
VAR x : INTEGER;
BEGIN
  x := 7;  y := y + 6;
  Print(x,y)
END First;

PROCEDURE Second;
VAR x , y : INTEGER;
BEGIN
  x := 2;  y := x + 3;
  Print(x ,y )
END First;

BEGIN
  x := 1;  y := 2;
  Print(x ,y );
  First;
  Print(x ,y );
  Second;
  Print(x ,y )
END Scope1.
```


Rekursion

Rekursiva funktioner

- Rekursion fungerar i stort sett likadant i SML och Modula-2

Exempel: fac(n) = n! = 1*2*...*n

SML

```
fun fac n = if n<2
            then 1
            else n * fac(n-1)
```

Modula-2

```
PROCEDURE fac(n:CARDINAL):CARDINAL;
BEGIN
  IF n<2 THEN
    RETURN 1
  ELSE
    RETURN n * fac(n-1)
  END
END fac;
```

Beräkning av rekursiva funktioner

- Hur mycket minne behövs?

...

Svansrekursion och iteration

Svansrekursiv version av fac i SML

```
fun fac n =
  let fun fac' acc i =
              if i<=n
              then fac' (acc*i) (i+1)
              else acc
  in fac' 1 2
```

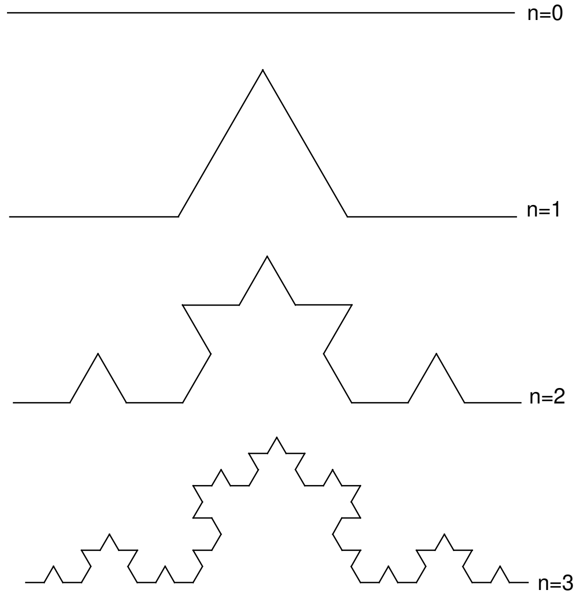
- Samma funktion i Modula-2? ...

fac med iteration i Modula-2

```
PROCEDURE fac(n:CARDINAL):CARDINAL;
  VAR i,acc:CARDINAL;
BEGIN
  acc:=1;
  i:=2;
  WHILE i<=n DO
    acc:=acc*i;
    i:=i+1
  END;
  RETURN acc
END fac;
```

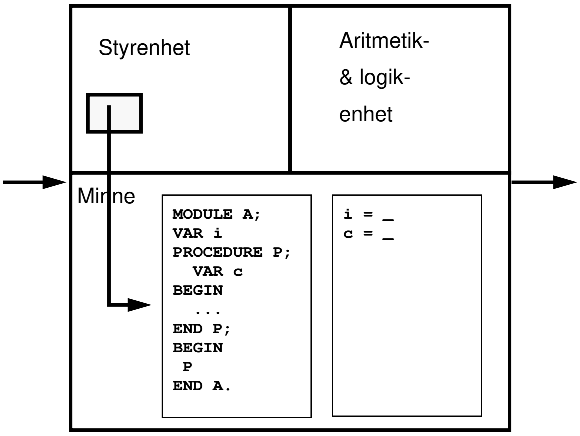
- Notera likheten mellan dessa två program!
- Ofta använder man loopar istället för rekursion i Modula-2.

Rekursiva procedurer



- n=4, 5, 6, ...

von Neumann-datorn



- Minnet är organiserat som en stack som innehåller
  - Variablers värden
  - Återhoppsadresser (var körningen fortsätter vid återhopp från en procedur eller funktion)
- Stacken växer/krymper vid anrop till/återhopp från procedurer.

- Hur fungerar egentligen det här programmet?

```
MODULE reverse2;

FROM InOut IMPORT
  ReadInt,WriteInt,WriteString,WriteLn,Done;

PROCEDURE rev;
VAR num:INTEGER;
BEGIN
  ReadInt(num);
  IF Done() THEN
    rev;
    WriteInt(num,1); WriteLn
  ELSE
    WriteString("Talföljden baklänges:");
    WriteLn
  END
END rev;

BEGIN
  WriteString("Mata in en talföljd:");
  WriteLn;
  rev
END reverse2.
```

Programexemplen

Sköldpaddsgrafik

```
PROCEDURE Triangle(d:CARDINAL);
BEGIN
  Move(d);
  Turn(120);
  Move(d);
  Turn(120);
  Move(d);
  Turn(120);
END Triangle;

PROCEDURE Polygon(n,d:CARDINAL);
  VAR i:CARDINAL;
BEGIN
  FOR i:=1 TO n DO
    Move(d);
    Turn(360 DIV n)
  END
END Polygon;
```


Programexemplen

Fraktaler

```
MODULE Fraktal;

FROM TurtleGraphics IMPORT
  Turn, Move, TurtleStop, PenMode, PenColor;

PROCEDURE Koch(len:INTEGER; n:CARDINAL);
VAR del:INTEGER;
BEGIN
  IF n=0 THEN
    Move(len)
  ELSE
    del:=len DIV 3;
    Koch(del,n-1);
    Turn(60);
    Koch(del,n-1);
    Turn(-120);
    Koch(del,n-1);
    Turn(60);
    Koch(del,n-1)
  END
END Koch;

VAR n:CARDINAL;

BEGIN
  PenColor(None);
  Move(-245);
  PenColor(Black);
  Koch(486,5);
  TurtleStop;
END Fraktal.
```


Programexemplen

Modulen TurtleGraphics

```
IMPLEMENTATION MODULE TurtleGraphics;

FROM MathLib IMPORT real,entier,sin,cos;

FROM SimpleGraphics IMPORT DrawLine;

CONST Pi=3.1415926536;

CONST MaxXSize = 250;
      MaxYSize = 250;

VAR x,y,
    angle:REAL; (* current turtle state, angle in radians *)
    pen:PenMode;
    dx,dy:REAL; (* cos(angle), -sin(angle) *)

PROCEDURE PenColor(Pen:PenMode);
BEGIN
  pen:=Pen;
END PenColor;

PROCEDURE lineTo(x1,y1:REAL);
BEGIN
  IF pen#None THEN
    DrawLine(entier(x),entier(y),entier(x1),entier(y1));
  END;
  x:=x1; y:=y1
END lineTo;

PROCEDURE Move(Distance:INTEGER);
BEGIN
  lineTo(x+dx*real(Distance),y+dy*real(Distance))
END Move;
```


Programexemplen

Modulen TurtleGraphics (forts)

```
PROCEDURE deg2rad(a:INTEGER):REAL;
BEGIN
  RETURN real(a)*(Pi/180.0)
END deg2rad;

PROCEDURE setAngle(rad:REAL);
BEGIN
  angle:=rad;
  dx:=cos(angle);
  dy:=-sin(angle)
END setAngle;

PROCEDURE Turn(Degrees:INTEGER);
BEGIN
  setAngle(angle+deg2rad(Degrees))
END Turn;

PROCEDURE ClearScreen;
BEGIN
  (* ... *)
END ClearScreen;

PROCEDURE TurtleStop;
BEGIN
  (* ... *)
END TurtleStop;

BEGIN
  x:=real(MaxXSize);
  y:=real(MaxYSize);
  setAngle(0.0);
  PenColor(Black)
END TurtleGraphics.
```