Föreläsning 4

- Typer, fortsättning
  - (se bilder från föreläsning 3)
  - Fördefinierade typer
  - Uppräkningstyper
  - Skalära typer
- Sammansatta typer
  - (Poster)
  - ARRAY-typer
  - Flerdimensionella ARRAY-typer
  - Strängar
  - Öppna ARRAY-parametrar
- Mer om funktioner och procedurer
  - Kommandon och uttryck
  - Referentiell transparens
  - Funktionsprocedurer
  - Sidoeffekter

Typer

Sammansatta (strukturerade) typer

Sammansatta typer i SML

- Tupler
  - (3,4) : int * int
  - (3,true,"hej") : int * bool * string
- Listor
  - [1,2,3] : int list
  - [(true,5), (false,3)] : (bool * int) list

Sammansatta typer i Modula-2

- Istället för tupler: poster

```
TYPE Date = RECORD
              year:  Year;
              month: Month;
              day:   Day
            END;
```

- Istället för listor: array-typer (fält, vektorer)

```
TYPE Histogram = ARRAY [1..6] OF CARDINAL;
```


ARRAY-typer

```
CONST N=6;
TYPE Vector=ARRAY [1..N] OF INTEGER;
```

Indextyp *Elementtyp*

- Längden är förbestämd (till skillnad från listor i SML)
- Elementen väljs ut med indexering.
- Elementtypen kan vara vilken typ som helst.
- Indextypen kan vara någon skalär typ (utom **REAL**)
- Variabler av ARRAY-typ har plats för ett värde av elementtypen för varje värde av indextypen

```
VAR v:Vector;
```

v=

7	3	2	8	0	1
1	2	3	4	5	6

v=

Index	Värde
1	7
2	3
3	2
4	8
5	0
6	1

ARRAY-typer

Indexering

Syntax

```
a[i]
```

Att komma åt värdet på elementen

- Antag: v=

7	3	2	8	0	1
1	2	3	4	5	6

i=__

Uttryck	Värde
v[4]	8
v[4-1]	2
v[i+1]	8
v[2*i]	1

Att ändra värdet på elementen

```
v[1]:=v[2];
v[i]:=0;
INC(v[4]);
```

v=

3	3	0	9	0	1
1	2	3	4	5	6

ARRAY-typer

Exempel

Skalärprodukt av två vektorer

- $\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$

```
CONST N=10;

TYPE Vector = ARRAY [1..N] OF REAL;

PROCEDURE innerProduct(a,b:Vector):REAL;
VAR i:[1..N];
    p:REAL;
BEGIN
  p:=0.0;
  FOR i:=1 TO N DO
    p:=p+a[i]*b[i]
  END;
  RETURN p
END innerProduct;
```

- Fungerar endast på vektorer av dimension 10.
- Lätt att ändra dimension!

```
    CONST N=42;
```

- Tips: namnge alla konstanter som man kanske skulle vilja ändra någon gång i framtiden.

ARRAY-typer

Programmeringsexempel

- Läs in en följd av tal och skriv ut den baklänges. Programmet ska fungera för alla talföljder av längd upp till 100.

Problem?

- Talföljdens längd är inte känd i förväg, men ARRAYer har en fast storlek.
- Lösning: använd en ARRAY som har plats för den längsta tillåtna följden + ett sätt att hålla reda på hur stor del av ARRAYen som verkligen används.
- Två sätt att hålla reda på hur mycket som används:

  1. Använd en extra variabel som innehåller antalet använda element.

  2. Lagra ett speciellt värde efter det sista använda elementet.

ARRAY-typer

Flerdimensionella fält

- Elementtypen kan vara vilken typ som helst, t ex en ny array-typ:

```
TYPE M = ARRAY [1..3] OF ARRAY [1..5] OF REAL;

VAR m:M;
```

m=

1	0	2	0	3	2	7	4	7	6	3	5	6	5	9

m[1]=

1	0	2	0	3

m[1][3]= 2

Förkortad notation

```
TYPE M = ARRAY [1..3], [1..5] OF REAL;

m[2,4] = m[2][4] = 7
```

m=

1	0	2	0	3
2	7	4	7	6
3	5	6	5	9

ARRAY-typer

Strängar

- Typen **STRING** är ej fördefinierad i vårt Modula-2-system
- Använd t ex:

```
VAR s: ARRAY [0..79] OF CHAR;
```

- Strängar av varierande längd?

Exempel

```
MODULE hej;

FROM InOut IMPORT
  WriteString,ReadString,WriteLn;

TYPE String80=ARRAY [0..79] OF CHAR;

VAR name:String80;

BEGIN
  WriteString("Hej, vad heter du? ");
  ReadString(name);
  WriteString("Trevligt att träffas, ");
  WriteString(name);
  WriteString("!");
  WriteLn
END hej.
```

ARRAY-typer

Strängar

- Strängars längd är oftast inte känd i förväg!
- Ett speciellt tecken lagras efter det sista om strängen är kortare än fältet

```
TYPE String16=ARRAY [0..15] OF CHAR;

VAR s:String16;

s:="sextonteckensord"
```

s=

"s"	"e"	"x"	"t"	"o"	"n"	"t"	"e"	"c"	"k"	"e"	"n"	"s"	"o"	"r"	"d"

```
s:="kortare"
```

s=

"k"	"o"	"r"	"t"	"a"	"r"	"e"	0C								

- Anm:

```
0C = CHR(0)

"0" = 60C = CHR(48)
```

Se tabellen med teckenkoder i Appendix A.

Programmeringsexempel

```
PROCEDURE WriteString80(s:String80); ...
```


ARRAY-typer

ARRAYer i tilldelningar och som argument

variabel **:=** *uttryck*

- Vid tilldelning måste variabeln ha samma typ som uttrycket (undantag: olika delintervall av samma grundtyp kan blandas)
- Motsvarande krav på argumenten i proceduranrop.
- ARRAYer måste ha samma indextyp och samma elementtyp.

```
VAR s80a,s80b: String80;
    s16: String16;

s80a:=s80b;
WriteString80(s80a);

s80a:=s16;           (* FEL! *)
WriteString80(s16);  (* FEL! *)
```

Strängkonstanter

- Strängkonstanter anpassas automatiskt till den typ som krävs i sammanhanget.

```
s80a:="strängkonstant";

s16 :="strängkonstant";

WriteString80("strängkonstant");
```


ARRAY-typer

Öppna ARRAY-parametrar

- Tillåter procedurer att ta argument av ARRAY-typ med en bestämd elementtyp, men en godtycklig indextyp!

Indextypen lämnas "öppen"!

Exempel

```
PROCEDURE WriteString(s:ARRAY OF CHAR);
...


WriteString(s16);
WriteString(s80);
```

Indexering?

- Oavsett argumentets indextyp får den öppna array-parametern index **[0..**n**−1]**, där n är antal element i argument-arrayen.
- **HIGH(s)** ger n-1, om **s** är en öppen array-parameter.

...

Modulen Strings

```
DEFINITION MODULE Strings;

   TYPE String = ARRAY [0..255] OF CHAR;

   PROCEDURE EmptyString (VAR str: ARRAY OF CHAR);
      (* str := "" *)

   PROCEDURE StrEq (VAR x, y: ARRAY OF CHAR): BOOLEAN;
      (* x = y , only significant characters. *)

   PROCEDURE Length (VAR str : ARRAY OF CHAR) : CARDINAL;
      (* returns the number of significant characters. *)

   PROCEDURE pos (substr: ARRAY OF CHAR; str: ARRAY OF CHAR):
     CARDINAL;
      (* ... *)

   PROCEDURE Concat
      (s1, s2: ARRAY OF CHAR; VAR result: ARRAY OF CHAR);
      (* ... *)

   PROCEDURE compare (s1, s2: ARRAY OF CHAR): INTEGER;
      (* Compares 's1' with 's2' and returns -1 if s1 < s2, *)
      (* 0 if s1 = s2, or 1 if s1 > s2 *)

   PROCEDURE CAPS (VAR str: ARRAY OF CHAR);
      (* CAP for the entire 'str' *)

END Strings.
```

Funktioner och procedurer

Kommandon och uttryck

Uttryck

- Ett uttryck *beräknas* för att få fram dess *värde*:

  2+3*4 ==> 2+7 ==> 9

- Beräkningen i sig har inga effekter på omgivningen:

  Det spelar därför ingen roll om samma beräkning görs en eller flera gånger (förutom att det kan ta olika lång tid).

  Det spelar heller ingen roll i vilken ordning deluttryck beräknas.

  Man kan använda lagar bekanta från matematiken:

  a+b = b+a

  2*a = a+a

- Dessa egenskaper i ett språk kallas *referentiell transparens.*

  Viktig del i funktionella språk.

Funktioner och procedurer

Kommandon och uttryck

Kommandon

- Kommandon *utförs* för att de har någon (permanent) *effekt* på omgivningen:

```
i:=3;
```

  Effekt: variabeln **i** har värdet 3 i fortsättningen

```
WriteString("Hej");
```

  Effekt: strängen "Hej" skrivs ut på bildskärmen.

- Ett kommandos effekter påverkar efterföljande kommandon och uttryck:

```
i:=3;
WriteInt(i*i,5);  (* 9 skrivs ut *)
i:=8;
WriteInt(i*i,5);  (* 64 skrivs ut *)
```

- Det spelar roll i vilken ordning kommandon utförs
- Det spelar roll om ett kommando utförs en eller flera gånger.

Funktioner och procedurer

Funktioner

Funktioner används i uttryck

- för att slippa skriva nästan samma uttryck många gånger.
- för att göra annars krångliga uttryck mer överskådliga.

- Ett funktionsanrop är ett uttryck.

  Exempel: ...

Procedurer

Procedurer används i kommandon

- för att slipa skriva nästan samma följd av kommandon många gånger.
- för att göra krångliga följder av kommandon mer överskådliga.

- Ett proceduranrop är ett kommando.

  Exempel: ...

Funktioner och procedurer

Funktionsprocedurer

- Funktionsdefinitioner innehåller en följd av kommandon som avslutas med **RETURN**-kommandot.
  - När en funktion anropas utförs kommandona.
  - Kommandon kan ha effekter.
  - Funktionsanrop kan därmed ha effekter.
  - Beräkningen av ett uttryck kan därmed ha effekter!
  - Exempel: ...

    Sådana effekter kallas *sidoeffekter.*
  - Sidoeffekter förstör referentiell transparens!

    Exempel:

    2*f(3) behöver inte bli samma som **f(3)+f(3)**

    f(3)+f(4) behöver inte bli samma som **f(4)+f(3)**

Programmeringsmetodik

Sidoeffekter gör program svårare att förstå och förändra.

- Undvik sidoeffekter!

  (Undantag: felsökning med hjälp av extra utskrifter i programmet)