Programmering för naturvetare

Kursmål

- att ge grundläggande kunskap om programmering i moderna programmeringsspråk.
- att ge insikt om grundläggande datalogiska begrepp.

Programmering för naturvetare

- Kursen består av två delar:

Del I

Funktionell programmering i SML.

Del II

Imperativ programmering i Modula-2.

Programmering för naturvetare, II

Föreläsare

Thomas Hallgren

- Rum 2418, Matematiskt Centrum
- Telefon: 772 5422 (hem 43 23 03)
- Datorpost: hallgren@cs.chalmers.se

Övningsledare & labhandledare

Jenny Petersson

- Telefon: 772 1098, rum 2243
- Datorpost: jenny@cs.chalmers.se

Alvaro Tasistro

- Telefon 772 1024, rum 2434
- Datorpost: tato@cs.chalmers.se

Programmering för naturvetare, II

- Imperativ programmering i Modula-2.

Kurslitteratur

- J.M. Adams, P.J. Gabrini & B. L. Kurtz:
  An Introduction to Computer Science with Modula-2
- Häfte med övningsuppgifter
- Lab-PM
- m.m.

All information finns i WWW

- http://www.cs.chalmers.se/kurser/naptv/del2/

Laborationer

Introduktionslab

- Lär dig använda Modula-2-kompilatorn

  (test av automatiskt labrättning)

Två "riktiga" laborationer

- Lärorika programmeringsuppgifter

Föreläsning 1

- Varför imperativ programmering?
- En snabb titt på Modula-2
- Olika programmeringsmiljöer
- Hur man kompilerar och kör Modula-2-program
- Skillnader mellan olika språk
  - Syntax
  - Semantik

De fyra programmeringsparadigmerna

- Funktionell programmering
- Imperativ programmering
- Objektorienterad programmering
- Logikprogrammering

- Programspråk är oftast utformade för att passa *en* av de fyra paradigmerna.

Imperativa språk

Skillnaden mellan funktionellt & imperativt?

Program i funktionella språk

- Funktionsdefinitioner + ett uttryck som ska beräknas

```
fun sum []       = 0
  | sum (x::xs)  = x + sum xs

sum [3,5,6,8]
```

Program i imperativa språk

- Definition av variabler, funktioner & procedurer + en följd av kommandon som ska utföras

```
VAR n, sum : INTEGER;

BEGIN
  sum := 0;
  LOOP
    ReadInt(n);
  IF NOT Done THEN EXIT END;
    sum := sum + n
  END;
  WriteInt(sum, 1)
END
```


Varför imperativ programmering?

Fördelar med funktionella språk

- Programmering på hög abstraktionsnivå
  - Programmen blir kortare
  - Programutvecklingen blir snabbare och säkrare
- Lätt att skriva återanvändbara programdelar
  - Polymorfi (**`'a list`**)
  - Högre ordningens funktioner (**`map`**, **`filter`**)
  - Abstrakta datatyper

men...

- In/utmatning, dialogen med användaren, passar inte in
- Programmen kan bli långsamma och kräva mycket minne
- Funktionella språk är relativt nya och så gott som okända utanför den akademiska världen

Varför imperativ programmering?

För- & nackdelar med imperativa språk

- Imperativa språk ligger närmare datorns sätt att arbeta
  - Maskinspråk är imperativa i von Neumann-datorer

```
      ...
 2  zero     104
 4  load     100
 6  jmpz      24
 8  input    108
10  load     108
      ...
```

- Programmeraren styr mer detaljerat vad datorn ska göra
  - effektivare lagring av datastrukturer
  - programmen kan bli snabbare
  - mer jobb för programmeraren!
- In- & utmatning är naturligt att beskriva som kommandon
- De mest använda språken är imperativa

  FORTRAN, COBOL,

  C, Pascal, Modula-2, Ada,

  C++
- Det blir lättare att få jobb

En snabb till på Modula-2

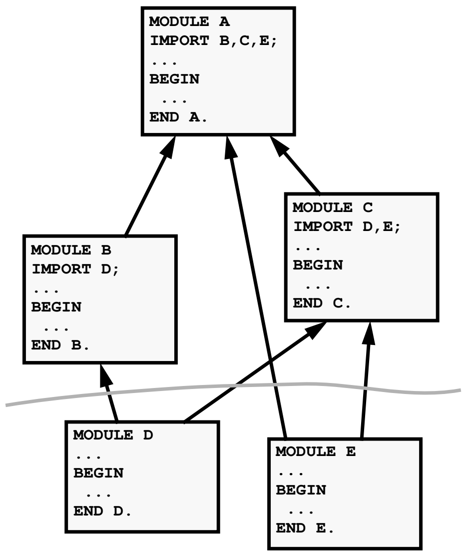
Ett exempelprogram

- Läs in n, beräkna och skriv ut $\sum_{i=0}^{n} i$

```
MODULE Sum;
  (** program that computes the sum of the           **)
  (** numbers 0+1+2+...+n where n is given as input **)

FROM     InOut
 IMPORT  ReadCard, WriteCard,
         WriteString, WriteLn;

VAR      n, number, sum : CARDINAL;

BEGIN
   WriteString("Enter the last number of the series: ");
   ReadCard(n);
   sum:=0; number:=0;
   LOOP
         (* sum = 0+1+...+number *)
         IF number=n
            THEN EXIT;
         END;
         number:=number+1;
         sum:=sum+number
   END;
   WriteString("The sum of the series is ");
   WriteCard(sum,6);
   WriteLn;
END Sum.
```


Program i Modula-2

Ett program består av ett antal moduler



- Modul ≈ abstrakt datatyp

Moduler

Modulen för huvudprogrammet

MODULE *namn*;

import-lista

deklarationer

BEGIN
  satser
END *namn*.

Andra moduler

- Två delar

  DEFINITION MODULE ...

  IMPLEMENTATION MODULE ...

Biblioteksmoduler

- In- och utmatning
- Stränghantering
- Matematiska funktioner
- ...

Import-listan och deklarationer

Import-listan

- För att använda saker som definieras i andra moduler

```
IMPORT InOut;
```

eller

```
FROM InOut IMPORT WriteString, ReadCard;
```

Deklarationer

- För att definiera nya saker

```
CONST pi = 3.1415926;

TYPE Color = (Red, Green, Blue);

VAR n : CARDINAL;

PROCEDURE sum(n:CARDINAL):CARDINAL;
BEGIN
  ...
END sum;
```


Fördefinierade typer i Modula-2

Namn	Beskrivning	Exempel	Namn i SML
`INTEGER`	Heltal	`-123 0 345`	`int`
`CARDINAL`	0 och positiva heltal	`98 0`	–
`REAL`	Reella tal	`12.34 -3.14`	`real`
`BOOLEAN`	Sanningsvärden	`TRUE FALSE`	`bool`
`CHAR`	Enstaka tecken	`"a" "7" "?"`	–
`String`	Följder av tecken	`"Hej hopp"`	`string`

Satser

Tilldelning

- *variabel := uttryck*

```
sum := sum + 1
```

Procedur-anrop

```
WriteCard(sum, 6)
```

Sekvens

sats1 ;
sats2 ;
...

Val

```
IF number = n THEN EXIT END
```

Upprepning

LOOP
satser
END

Uttryck

Enkla uttryck

```
7
3.14
"hej hopp"
sum
```

Sammansatta uttryck

```
sum + 1
2*k
NOT b
f(3*n+1,0)
(age>12) AND (age<20)
```

Operatorer

```
= # < <= > >=

+ - OR

* / DIV MOD AND
```

Vad som inte finns i Modula-2

- Polymorfi
- Högre ordningens funktioner (...)
- Algebraiska datatyper & mönstermatchning
- Egna operatorer

Olika implementeringar av programspråk

Interaktiva system

- SML-systemet

```
muppet52% sml
Standard ML of New Jersey, Version 0.93, February 15, 1993
with the extended environment from ÅW's book
val it = () : unit
- fun square x : int = x*x;
val square = fn : int -> int
- map square [5,7,9];
val it = [25,49,81] : int list
-
```

- Man kan enkelt mata in och testa varje funktion för sig

Batch-system, kompilatorer

- Modula-2-systemet

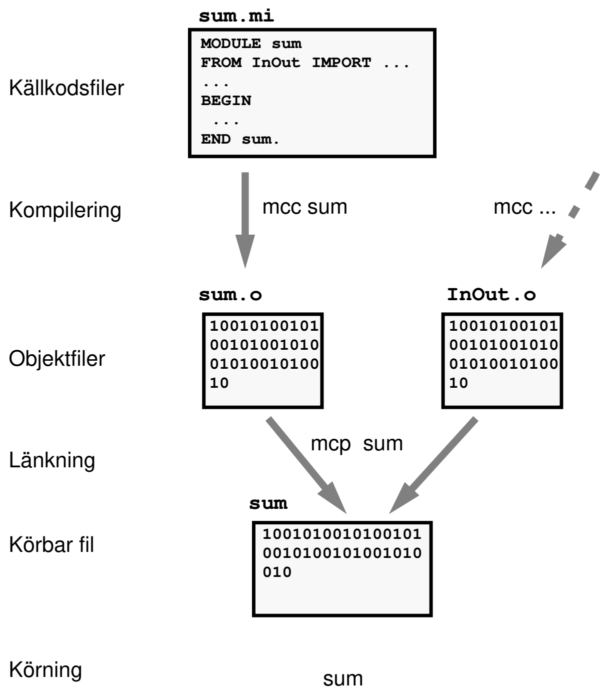
```
muppet52% emacs sum.mi
muppet52% mcc sum
muppet52% sum
Please enter the last number of the series: 100
The sum of the series is   5050
muppet52%
```

- För att kunna testköra måste man ha ett komplett program

Samband mellan språk och implementeringar

- Exempelspråk: BASIC, Haskell, C, Ada

Att kompilera och köra Modula-2-program



Utveckling av Modula-2-program

Typisk arbetsgång (små program)

- Förstå problemet
- Lös problemet, välj/uppfinn en algoritm
- Skriv algoritmen i Modula-2
- Mata in/redigera programmet i en fil
- Kompilera programmet
- Kör programmet

Olika typer av fel

Fel som upptäcks av kompilatorn

- Syntaxfel
- Typfel

Fel som upptäcks av datorn vid körning

- division med noll, etc

Fel som inte kan upptäckas av datorn

- Programmet är ett korrekt Modula-2-program, men löser inte problemet.

Språkhistoria

Släktträd

Skillnader mellan olika språk

Syntax (hur program ser ut)

Lexikala enheter

- Hur namn (identifierare) får se ut
- Reserverade ord
- Stora/små bokstäver
- Radbrytningar & blanktecken

Grammatik

- Operatorprioriteter
- Hur man markerar hur saker hör ihop
- Ordning mellan deklarationer

Typsystem

- Ingen typkontroll
- Typkontroll under körningen (dynamisk)
- Typkontroll vid kompileringen (statisk)
- Starkt typade språk
- Monomorfa/polymorfa typsystem
- Automatisk typhärledning/manuell typgivning

Skillnader mellan olika språk

Semantik (vad program betyder)

- Inom de olika programmeringsparadigmerna finns fundamentalt olika sätt att beskriva vad program betyder. Dessa beskrivningar kallas...

Beräkningsmodeller

- Vad man tänker sig att ett program är och hur man föreställer sig att program körs

(implementeringen kan vara annorlunda)

Funktionella språk

- Ett uttryck som ska reduceras till kanonisk form

Imperativa språk

- En följd av kommandon som ska utföras

Logikspråk

- Ett påstående som ska bevisas

Objektorienterade språk

- En samling objekt som reagerar på meddelanden...

Något som alla språk borde ha

En abstraktionsmekanism

- Det måste finnas ett sätt att dela upp program i delar och ge delarna namn.
  - Stora problem löses genom att dela in dem i delproblem. Detta måste kunna avspeglas i programstrukturen.
  - Återanvändning av programdelar blir helt opraktiskt utan namngivning.

(undantag: tidig FORTRAN, BASIC)

En parameteriseringsmekanism

- Namngivna delar måste kunna ha parametrar, dvs hål som fylls i då man refererar till den namngivna delen.

(undantag: BASIC, Forth, kombinatorspråk)

Beräkningsmodellen i funktionella språk

- Ett program är ett uttryck som ska reduceras till kanonisk form.
- *Uttryck* är den viktigaste beståndsdelen i funktionella program.
- *Funktioner* kan ses som ett sätt att *namnge uttryck* med hål i. Argumenten till en funktion anger vad som ska fyllas i hålen.

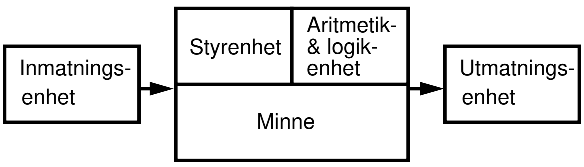
Exempel på uttrycksreduktion

```
fun square x : real = x*x
fun belopp (x,y) = sqrt (square x + square y)
```

```
belopp (3.0,4.0)
⇓
sqrt (square 3.0 + square 4.0)
⇓
sqrt (3.0 * 3.0 + square 4.0)
⇓
sqrt (9.0 + square 4.0)
⇓
sqrt (9.0 + 4.0 * 4.0)
⇓
sqrt (9.0 + 16.0)
⇓
sqrt 25.0
⇓
5.0
```


Beräkningsmodellen i imperativa språk

- Ett program är en följd av kommandon som ska utföras.
- *Kommandon*, *satser*, är den viktigaste beståndsdelen i imperativa program.
- De grundläggande kommandona utförs genom att styrenheten skickar lämpliga styrsignaler till de övriga delarna i datorn



Resultatet kan bli att

- ett värde hämtas från en inmatningsenhet och lagras på någon plats i minnet,
- ett värde kopieras från minnet till en utmatningsenhet,
- ett värde beräknas och lagras i minnet.

- En namngiven följd av satser (med hål i) kallas en *procedur*.
- I högnivåspråk används *variabler* för att hantera minnet.

Exempel på körning av imperativa program

```
❶ReadCard(n);
❷sum:=0; ❸number:=0;
LOOP
 (* sum = 0+1+...+number *)
 ❹IF number=n THEN ❺EXIT END;
 ❻number:=number+1;
 ❼sum:=sum+number
END;❽
```

Pos	n	sum	number	Kommentar
❶	?	?	?	antag att 2 matas in
❷	2	?	?	sum:=0
❸	2	0	?	number:=0
❹	2	0	0	number=n är FALSE
❻	2	0	0	number:=number+1
❼	2	0	1	sum:=sum+number
❹	2	1	1	number=n är FALSE
❻	2	1	1	number:=number+1
❼	2	1	2	sum:=sum+number
❹	2	3	2	number=n är TRUE
❺	2	3	2	EXIT
❽	2	3	2	

Blandning av imperativt och funktionellt

SML

- In- & utmatning görs imperativt.

```
- val fil = open_out "/tmp/blaha";
val fil = - : outstream
- output (fil,"Hej hopp");
val it = () : unit
```

- Det finns variabler (kallas referenser).

```
- val n = ref 5;
val n = ref 5 : int ref
- n:= !n + 1;
val it = () : unit
- n;
val it = ref 6 : int ref
```

Modula-2

- Det finns uttryck och funktioner.

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

- Alla imperativa språk har uttryck (utom maskinspråk).