Velkommen til kurset

INF3110/4110

Programmeringsspråk

Først det praktiske

Foreleser: Roger Antonsen
E-post: rantonse@ifi.uio.no
Kontor: rom 3403
Treffetid: etter avtale

- Undervisning
  Forelesninger: onsdager kl. 14:15 - 16:00
  Gruppetimer: én dobbeltime per uke
  . . . og mye *selvstudium*!
  10 studiepoeng $\approx$ 12 timer i uken
- Obligatorisk oppmøte i dag!
- Hjemmeside: http://www.ifi.uio.no/inf3110/h03/
- Nyhetsgruppe: ifi.in211

Kursets innhold

«Metoder for språkbeskrivelse og semantikk, inklusive statiske og dynamiske aspekter, typing, implementasjon og kjøresystemer. Ulike typer programmeringsspråk, inklusive funksjonelle språk og PROLOG.»

Anbefalte forkunnskaper

«kunnskaper tilsvarende INF1020 - Algoritmer og datastrukturer/INF 110»

- rekursjon, lister, trær, . . .
- datastrukturer, algoritmer, . . .
- lese enkle C-programmer
- programmere Java

Pensum

- Ghezzi & Jazayeri: Programming language concepts (tredje utgave), John Wiley & sons, 1998.
- Krogdahl & Ore: Om syntaks og syntaksanalyse, kompendium 47. Kan kjøpes i bokhandelen.
- Kristoffersen: Funksjonell programmering i standard ML, kompendium 61. Kan leses og skrives ut fra kursets hjemmeside.

I tillegg: Forelesninger, obligatoriske oppgaver og ukeoppgaver.

Se hjemmesiden til kurset for detaljene!

Gruppeundervisningen

- Arne Martin Güettler (arneh@ifi.uio.no)
  Ellef Gjelstad (ellefg@ifi.uio.no)
- begynner i uke 36 (3.-5. september)!
- Ukeoppgaver (over det som ble gjennomgått uken før på forelesning).

Vurdering

- Avsluttende skriftlig eksamen på 3 timer.
- Tre obligatoriske innleveringer, hvor to av disse blir vurdert.
- Vekting: sannsynligvis 20%/20%/60%.

Det var det praktiske!

Programmeringsspråk

FORTRAN, ALGOL 60, COBOL, APL, LISP, SNOBOL4, PL/I, BASIC, SIMULA 67, Algol 68 Pascal, PROLOG, C, Mesa, Setl, Concurrent Pascal, Scheme, CLU, Euclid, Modula-2 Ada, Smalltalk, OPS5, ICON, C++, CLIPS, ML, Oberon-2, Eiffel, CLOS, Modula-3 Tcl/Tk, Perl, Python, Java (hentet fra boka s.15)

- hvorfor er det så mange programmeringsspråk?
- hva er et programmeringsspråk?
- hvordan beskriver man et programmeringsspråk?
- hva er forskjellene mellom ulike programmeringsspråk?

Det er gode grunner til at det er så mange språk:

- Oppgavene man ønsker å løse er forskjellige mph. størrelse, kompleksitet, innhold, . . .
- Det er forskjellige krav til hastighet, plass, sikkerhet, . . .
- Programmerere er ulike!
- Informatikken er ennå en ung vitenskap.

En beskrivelse av et (programmerings-) språk består av to hovedkomponenter:

Syntaktiske regler forteller hva slags *form* et lovlig program skal ha.

Semantiske regler sier noe om hva de ulike setningene i språket *betyr*.

Noen ulike paradigmer

Objekt-orientert programmering
klasser og objekter (velkjent stoff)

Imperativ programmering
variable, tilordninger, sekvenser, tilstander

Funksjonell (applikativ) programmering
"alt" er funksjoner,verdier og typer (ikke tilordning)

Logisk (deklarativ) programmering
"*hva* istedenfor *hvordan*"

I dette kurset

- forstå hvordan man beskriver et språk *helt presist*; syntaksanalyse, grammatikker, BNF, språkklasser, automater,. . .
- forstå de underliggende mekanismene bedre / hva som skjer når et program blir eksekvert; kjøretidssystemer, statiske og dynamiske aspekter,typemekanismer,. . .
- programmeringserfaring; spesielt funksjonell programmering i ML, også noe PROLOG

Vi skal se på hvilke *relevante* egenskaper ved språk som gjør de like eller ulike. Hva er essensen ved et programmeringsspråk? Hvordan kan vi skille språkene fra hverandre?

Vi begynner med å se på:

binding og variable

Binding

Programmer består av **enheter** (entiteter) med tilhørende **egenskaper** (attributter).

Eksempler:

- En variabel har et navn og en type.
- En rutine har et navn og formelle parametre (med typer).
- En setning har assosierte aksjoner.

Det å sette verdien til en slik egenskap kalles **binding**. Dette er en av de essensielle egenskapene som skiller programmeringsspråk fra hverandre. La oss se noen eksempler på **når** binding kan skje.

Binding kan skje

i språkdefinisjonen

I Java representerer int et *heltall* som er et kjent matematisk begrep. Dermed vet vi hvordan +, * etc. skal oppføre seg.

i implementasjonen

I C er det implementasjonen som bestemmer hvor mange byte som brukes for en int.

Binding kan skje

under kompileringen

I Pascal er typen Integer predefinert, men kan omdefineres i et program. Dvs. ved språkimplementasjonen er bindingen til Integer gitt, men denne endres ved kompilering.

under kjøringen

I de fleste programmeringsspråk er variable bundet til en verdi under kjøringen, og denne bindingen kan endres mange ganger underveis.

De tre første bindingene kalles **statiske bindinger**, den siste kalles **dynamisk binding**.

Variable

Sett fra en programmerers synspunkt, har en variabel følgende egenskaper:

- navn - en streng av tegn for å kunne referere til variabelen
- skop - de deler av programmet hvor variabelen er kjent
- levetid
- type
- adresse i lageret - («*l-value*»)
- verdi - («*r-value*»)

Navn

En variabel får navnet i en deklarasjon. Noen språk (som Perl, PHP, Fortran, Scheme) krever ingen eksplisitt deklarasjon. Variablen blir deklarert implisitt første gang den påtreffes.

Skop

Enhver variabel (og deklarasjon) har sitt **skop**, dvs. den delen av programmet hvor den er synlig. Enhver forekomst er kun i skopet til én deklarasjon med samme navn.

```
1  #include <stdio.h>
2  int x = 1;
3
4  void f()
5  {
6    printf("x = %d\n", x);
7  }
8
9  int main()
10 {
11   int x = 2;
12   f();
13 }
```

Skopet begynner vanligvis der hvor variabelen deklareres og strekker seg til et punkt som bestemmes av det aktuelle språket.

Eksempel i C:
Skrives det ut 1 eller 2?

Problem med skop a la C

Hvordan kan to funksjoner kalle hverandre når de først er kjent etter deklarasjonen?

Løsningen er å tillate separat deklarasjon av signaturen.

```
1  float f2(int a, int b);
2
3  int f1(float x)
4  {
5    ... f2(..., ...) ...
6  }
7
8  float f2(int a, int b)
9  {
10   ... f1(...) ...
11 }
```

Språkets skopregler forteller hva skopet er. De vanligste er:

C Skopet starter ved deklarasjonsstedet.

Simula Skopet starter ved begin i blokken.

Java Som C for lokale variable og som Simula for klasseatributter.

Perl Som C for variable og som Simula for funksjoner.

Alle disse språkene (unntatt Perl) har **statisk skop**.

Alle disse skopene varer til blokkens avslutning (med end eller }). For alle språkene gjelder at lokale redeklarasjoner kan gi «hull» i skopet.

Skop i hele blokken (SIMULA)

```
1  begin
2   integer x = 1;
3
4   begin
5     procedure P1;
6     begin
7       OutText("indre x = ");  OutInt(x, 0);  OutImage;
8     end;
9
10    integer x = 2;
11
12    P1;
13  end;
14
15  OutText("ytre x = ");  OutInt(x, 0);  OutImage;
16 end
```

```
|   x=1
|   x=1
|   x=1
    | x=2
    | x=2
    | x=2
    | x=2
    | x=2
    | x=2
    | x=2
    | x=2
    | x=2
    | x=2
|   x=1
|   x=1
|   x=1
```

Resultatet av kjøringen:

indre x =

ytre x =

Statisk skop (også kalt leksikalsk skop)

Skopet kan avgjøres ved å se på den syntaktiske strukturen til programmet. For hver variabelforekomst i programmet kan man finne en unik deklarasjon som hører til forekomsten. Kan gjøres *uten* å kjøre programmet; det er det vi mener med statisk.

Dynamisk skop

Noen språk har **dynamisk skop**, med regelen:
En forekomst binder til den sist evaluerte deklarasjon som ikke er avsluttet. Skopet er dermed bestemt av *kjøringen*, og kan ikke avgjøres på forhånd. (LISP)

Levetid

En variabels levetid er den tiden den opptar plass i lageret. Dette er normalt den tiden programmet er i variabelens skop, men det er unntak:

- En variabel lever selv om den midlertid er skjult av en annen variabel.
- Noen språk har nøkkelord som gir utvidet levetid (som static i Java og C).
- Ved å bruke dynamisk allokering (new i Java, malloc i C) får brukeren full kontroll over levetiden.

Type

En variabels type angir et sett av verdier og mulige operasjoner på disse.

- De fleste språk har statisk typing - typen bindes ved kompilering
- Noen språk har dynamisk typing - typen kan forandres under kjøring

Det blir snart mer om dette.

```
1 x = y;
```

Adresse i lageret («*l-value*»)

Variable som forekommer på venstre side av tilordningsoperatoren "=" står vanligvis for adressen i lageret. Hvor en variabel plasseres i lageret er oftest dynamisk bestemt.

Verdien («*r-value*»)

Variable som forekommer på høyre side av tilordningsoperatoren "=" står vanligvis for innholdet eller verdien til variabelen, dvs. det som lageret inneholder.

```
1 x = x + 2;
```

En variabels verdi er det mest dynamiske vi har, men selv der finnes unntak:

- I noen språk kan man definere konstanter som er kjent under kompileringen (som #define i C).
- Noen språk kan «fryse» en variabel slik at verdien siden ikke endres (som final i Java).

Initialverdi

Hvilken verdi har en variabel når den opprettes?

Noen muligheter:

- en tilfeldig verdi (gitt av hva som lå der før)
- en fast verdi (0 i Simula)
- en initialverdi må alltid oppgis (Modula-3)
- det er umulig å bruke en variabel før den har fått en verdi (Java)

Oppsummering

- Dette var (grovt sett) 2.2 og 2.3 i læreboken.
- Kapittel 1 er selvstudium.

27/8
Typer. Les kapittel 3 frem til 3.3.1.
ML. Begynn på ML-kompendiet.

3/9
Fortsetter med ML.

10/9
Ingen forelesning.

17/9
Fortsetter med ML.