EDA011/EDA017 – Programmeringsteknik för F, E, I, π, N, och BME

Introduktion

Christian.Soderberg@cs.lth.se

22 januari 2013

EDA011 eller EDA017?

- Två kurser:
  - EDA011: N och BME – grundkurs i programmering, objektorientering och Java (7.5 hp).
  - EDA017: F, E, I och π – samma som EDA011, men dessutom beräkningsprogrammering (7.5 + 1.5 = 9 hp).
- Krav för godkänt – EDA011:
  - Godkända inlämningsuppgifter (6 stycken).
  - Godkänd tentamen.
- Krav för godkänt – EDA017:
  - Samma som för EDA011.
  - Dessutom kunskapskontroll i beräkningsprogrammering.

EDA011 eller EDA017?

- Två kurser:
  - EDA011: N och BME – grundkurs i programmering, objektorientering och Java (7.5 hp).
  - EDA017: F, E, I och π – samma som EDA011, men dessutom beräkningsprogrammering (7.5 + 1.5 = 9 hp).
- Krav för godkänt – EDA011:
  - Godkända inlämningsuppgifter (6 stycken).
  - Godkänd tentamen.
- Krav för godkänt – EDA017:
  - Samma som för EDA011.
  - Dessutom kunskapskontroll i beräkningsprogrammering.

Datorns uppbyggnad

- Processor: datorns hjärna – utför alla beräkningar.
- Minne: kan lagra värden
  - RAM-minne
  - Hårddisk
- Inmatningsenheter – exempelvis tangentbord.
- Utmatningsenheter – exempelvis bildskärm.

ANT8-datorn

- ANT8-datorn är en enkel påhittad dator.
- Precis samma principer som verkliga datorer.
- Processor med 16 register, $r_0, r_1, \ldots r_{15}$, och ett register som kallas *PC* (*Program Counter*).
- RAM-minne med 256 minnesadresser – adresserna är 0 . . . 255.
- Tangentbord för inmatning.
- Bildskärm för utmatning.

Instruktioner

- ANT8 kan utföra 16 olika instruktioner.
- En instruktion kodas med ett antal heltal som lagras i RAM-minnet (von Neumanns idé).
- Processorns PC-register håller reda på var i RAM-minnet nästa instruktion som skall utföras ligger (von Neumanns idé).
- Efter varje instruktion räknas PC-registret upp, så att nästa instruktion 'pekas ut'.
- När ett program startar innehåller PC-registret alltid adressen 0 (där alltså den första instruktionen finns).

ANT8-instruktioner

- Lagra värdet v i register r_k:

$$1, k, v$$

- Öka värdet i register r_k med v:

$$2, k, v$$

- Läs in värde från tangentbordet, och lagra i register r_k:

$$14, k$$

- Skriv ut värdet i register r_k:

$$15, k$$

- Stoppa processorn:

$$0$$

ANT8-instruktioner

- Addera register r_i och r_j, lagra summan i r_k:

$$8, k, i, j$$

- Gör så att vi utför instruktionen i adress a i nästa steg:

$$3, a$$

Problem med maskinkod

- Instruktionskoderna är olika för olika slags processorer.
- Det är besvärligt att hålla reda på koderna för de olika instruktionerna (även om vi bara behöver göra det för en processor).
- Det är jobbigt att hålla reda på var (i vilka register och adresser i RAM-minnet) olika värden är lagrade.
- Det är väldigt jobbigt att hålla reda på adresserna för olika hopp, och lätt att räkna fel.
- Även enkla saker kan kräva många instruktioner.

Bättre alternativ

- *Assemblerspråk*: använd namn på instruktioner, och skriv in dem som text. Denna text översätts sedan av ett program till maskinkod.
- *Variabler* för att lagra värden.
- Kraftfullare sätt att uttrycka vad programmeraren vill – exempelvis konstruktioner för att göra beräkningar, göra olika saker beroende på olika villkor, och upprepa saker.
- *Högnivåspråk*, som ett särskilt program (en *kompilator*) kan översätta till maskinkod.
- I kursen skall vi lära oss högnivåspråket *Java*.

Variabler

- I Java lagrar vi värden som vi vill använda i *variabler*.
- En variabel kan ses som en behållare som vi kan lagra *ett* värde i – namnet 'variabel' visar att vi kan ändra värdet i behållaren.
- Varje variabel har ett *namn* och en *typ*.
- Det finns många olika typer av värden, denna vecka kommer vi att använda heltal (`int`) och reella tal (`double`).

Tilldelningar

- Vi lagrar värden i våra variabler med *tilldelningssatsen* (=).
- Till vänster om likhetstecknet måste vi ha ett variabelnamn, till höger ett *uttryck*.
- *a* ← *b* skrivs i Java som `a = b;`
- Våra uttryck kan vara aritmetiska uttryck, med exempelvis +, -, * och / – prioritetsreglerna är de som vi är vana vid (vi kan även använda parenteser).
- Varje variabel har bara ett värde åt gången, när en variabel får ett nytt värde så försvinner det gamla värdet.

Ett enkelt Java-program

- Alla våra *variabler* måste *deklareras* – vi talar då om vilken typ av värden de kan ha (vi återkommer till detta).
- För att läsa in ett heltal i Java kan vi skriva:
  - `Keyboard.nextInt()`
  - `Keyboard.nextInt(”`*ledtext*`”)`
- För att skriva ut värden i Java kan vi använda:
  - `System.out.print(`*uttryck*`)`
  - `System.out.println(`*uttryck*`)`
- Våra programrader måste stå inne i ett särskilt så kallat *huvudprogram*.

Huvudprogrammet

- Alla Java-program har en speciell form:

```
class ProgramName {

    public static void main(String[] args) {
        ... programrader ...
    }
}
```

- Vi kommer att göra ett litet tillägg.

Våra huvudprogram

- Java-program i denna kurs:

```
class ProgramName {

    public static void main(String[] args) {
        new ProgramName().run();
    }

    void run() {
        ... vårt huvudprogram ...
    }
}
```

- Poängen med detta kommer vi att se senare i kursen.

Att kompilera och köra

- Vi *kompilerar* ett Javaprogram genom att i ett kommandofönster skriva exempelvis:

```
alfa-1: javac Examples.java
```

  Här är ”`Examples.java`” namnet på *källkodsfilen*. Kompilatorn ger en `.class`-fil för varje program i källkodsfilen.
- Vi *kör* ett Javaprogram (en `.class`-fil) genom att skriva exempelvis

```
alfa-1: java Sum
```

Något om övningarna och Linux

- Under övningarna kommer ni att köra på Linux-datorerna i E-husets källare.
- Ni kommer att använda fem program:
  - `ptlogin`, för att ställa er i kö för hjälp.
  - Ett *kommandofönster*, som är ett fönster som ni kan skriva in olika kommandon i.
  - En *texteditor*, som ni kan mata in era program med.
  - En web-läsare, som ni kan läsa kurshemsidan med.
  - `pt-quiz`, som ni kan köra för att testa att ni verkligen har förstått de begrepp som behandlats under passet.

Något om övningarna och Linux

- Ni får själva välja vilken texteditor ni skall använda, jag använder själv en som heter `emacs`.
- I kommandofönstret behöver ni använda fem olika kommandon:
  - **`ls`**: listar filerna i den aktuella katalogen (*list files*).
  - **`cd`**: hoppar till en given katalog (*change directory*).
  - **`mkdir`**: skapar en ny katalog (*make directory*).
  - **`javac`**: kompilerar ett Java-program (*java compiler*).
  - **`java`**: kör ett Java-program.
- De brukar ta någon övning innan man behärskar dessa kommandon.

Grundläggande begrepp i Java

- *Variabler* används för att lagra värden.
- *Deklarationer* beskriver våra variabler för kompilatorn, och avgör vad vi kan göra med dem – kompilatorn upptäcker ofta om vi tänker fel.
- *Satser* är kommandon som översätts till maskinkod av kompilatorn.
- *Tilldelningar* är satser som ger våra variabler värden.
- *Uttryck* används för att beräkna värden.

Olika typer av tal

- Vi skiljer i Java mellan heltal och reella tal:
  - Heltal är exakta (0, 42, −128).
  - Reella tal är avrundade (3.141592653589793 23846, 1.000000000000).
- Den vanligaste heltalstypen heter `int` (efter *integer*).
- Den vanligaste typen för reella tal heter `double` (efter *double precision floating point number*).
- När vi dividerar två heltal med varandra trunkeras resultatet till ett heltal – var observanta på detta under övningen (vi kommer att prata mer om det i vecka tre).

Startvärden

- Oftast ger man variabler värden redan när vi deklarerar dem:

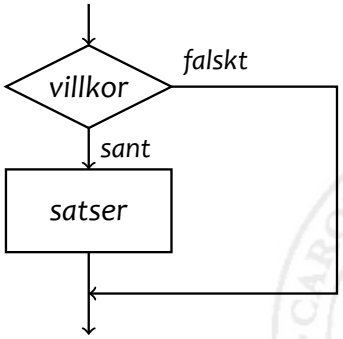
```
int nbrOfStudents = 64;
double height = 6.3;
```

- Om vi inte ger en variabel något värde så sägs den vara oinitierad – vi får inte använda den förrän vi givit den ett värde.

`if`-satsen

Den enklaste `if`-satsen har bara en gren:

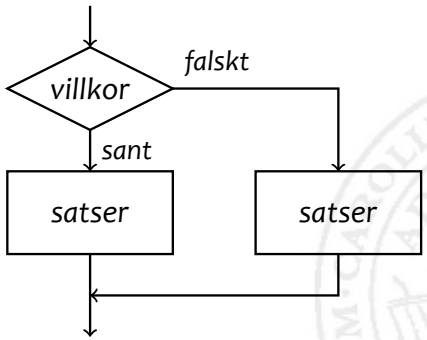
```
if (... villkor ...) {
    ... satser ...
}
```



`if`-satsen

Det finns även en `if`-sats med två grenar:

```
if (... villkor ...) {
    ... satser ...
} else {
    ... satser ...
}
```



`if`-satserna

- Den enklaste `if`-satsen har bara en gren:

```
if (... villkor ...) {
    ... satser ...
}
```

- Det finns även en variant med två grenar:

```
if (... villkor ...) {
    ... satser ...
} else {
    ... satser ...
}
```

- Satserna inuti grenarna kan själva vara `if`-satser.