Datorteknik

Tomas Nordström

Föreläsning 9

Programmering

För utveckling av verksamhet, produkter och livskvalitet.

HÖGSKOLAN HALMSTAD

Föreläsning 9

Programmering del 2

- Programmering och systemutveckling

- Kravanalys,
- Planering,
- Design & Implementering,
- Testning

Programmering

Programmering handlar om kreativ problemlösning.

- Datastrukturer – Hur ser den information ut som vi arbetar med? Input? Output? Mellanliggande informationstillstånd?
- Algoritm – ”metoden”, ”sättet”, ”principen” att lösa ett problem. Hur löser man problemet ur en mer övergripande abstrakt synvinkel?
- Implementation – att konkret skriva ett program i ett visst språk.

- **programmering** - att tänka ut, planera, framställa och testa datorprogram.
- Att "skriva kod" är alltså inte samma sak som att programmera, det är bara en del av jobbet.
- När programmering görs med ingenjörsmässiga metoder talar man om programutveckling eller systemutveckling.

Traditionell Arbetsgång

Projektfas, allmänt	Software Engineering-fas	produkt/dokument
1. förstå problemet	kravanalys	kravspecifikation
2: planlägg lösningen	planering (tid, resurser)	projektplan
3: genomför planen	design (konstruktion) implementation (kodning)	designspec., ev på flera nivåer kod
4: utvärdera resultatet	testning (validering, verifikation)	ny kod, uppdaterade dokument

Unified Modeling Language (UML)

- Formal System Design Process
- Allows us to use a formal process & tools to facilitate and automate design steps:
  - Requirements
  - Specification
  - System architecture
  - Coding/chip design
  - Testing

- Developed by Grady Booch et al.
  - Version 1.0 in 1997 (current version 2.4.1)
  - Maintained by Object Management Group (OMG): www.omg.org
  - Resources (tutorials, tools): www.uml.org

[http://www.eng.auburn.edu/~nelson/courses/elec5260_6260/]

Kravspecifikation

- En **Kravspecifikation,** eller **kravspec** som det ofta förkortas, är inom mjukvaruutveckling resultatet av en rad processer vars syfte är att samla in och sammanställa information som beskriver hur ett system eller en applikation skall bete sig. Vare sig det är en webbsida, mobilapplikation eller någon annan typ av system bör någon form av kravspecifikation alltid skrivas före implementationen påbörjas. Detta är i synnerhet viktigt om användaren och utvecklaren består av två olika parter.

[http://www.kravspecifikation.se/]

Mall för kravspecifikation

- Bakgrund
  - Produkten
  - Organisation
- Funktionella krav
- Icke-funktionella krav
  - Användarvänlighet
  - Kapacitet
  - Underhållbarhet
  - Tillgänglighet
- Dokumentation
- Leveransvillkor

[http://www.kravspecifikation.se/]

Planering (tid, resurser)
Projektplan

- Måste göras vid lite större projekt, men inte idag!

Design (konstruktion)

Modularisering

- Modularisering gör större programmeringsprojekt mer hanterbara
- Skapa delar som kan utvecklas och testas oberoende av varandra

- Lite beroende på programmeringsparadigm så tänker man lite olika vid sin detaljerade modularisering. Är det subsystem, funktioner/ metoder eller kanske datastrukturer/objekt som ska avgöra avgöra modulariseringen?

Programmeringsparadigm

- Ett **programmeringsparadigm** är en övergripande "teori" om hur program bör organiseras och struktureras.
- Programmeringsparadigmer är språkoberoende i bemärkelsen att paradigmet inte uttryckligen talar om språksyntax eller semantik, utan om övergripande begrepp och synsätt på program och programutveckling.

De vanligaste programmeringsparadigmerna är:

- Strukturerad programmering
- Procedurell programmering
- Objektorienterad programmering
- Logikprogrammering
- Funktionell programmering

[Wikipedia]

Java
Objekt-orientering

Attribut

- Ett attribut är en variabel eller konstant som beskriver en egenskap hos ett objekt. Ett attribut ges ett namn så att det kan anropas/ användas i programmet. Beroende på programmeringsspråk så kan man också behöva ange typ för attributet (heltal, flyttal, en pekare eller en textsträng till exempel) och det är ofta också möjligt att ange olika sekretessnivåer för attribut. Detta för att hindra eller öppna upp för åtkomst av andra delar av programmet som vill veta vad attributet har för värde eller namn.

Metoder

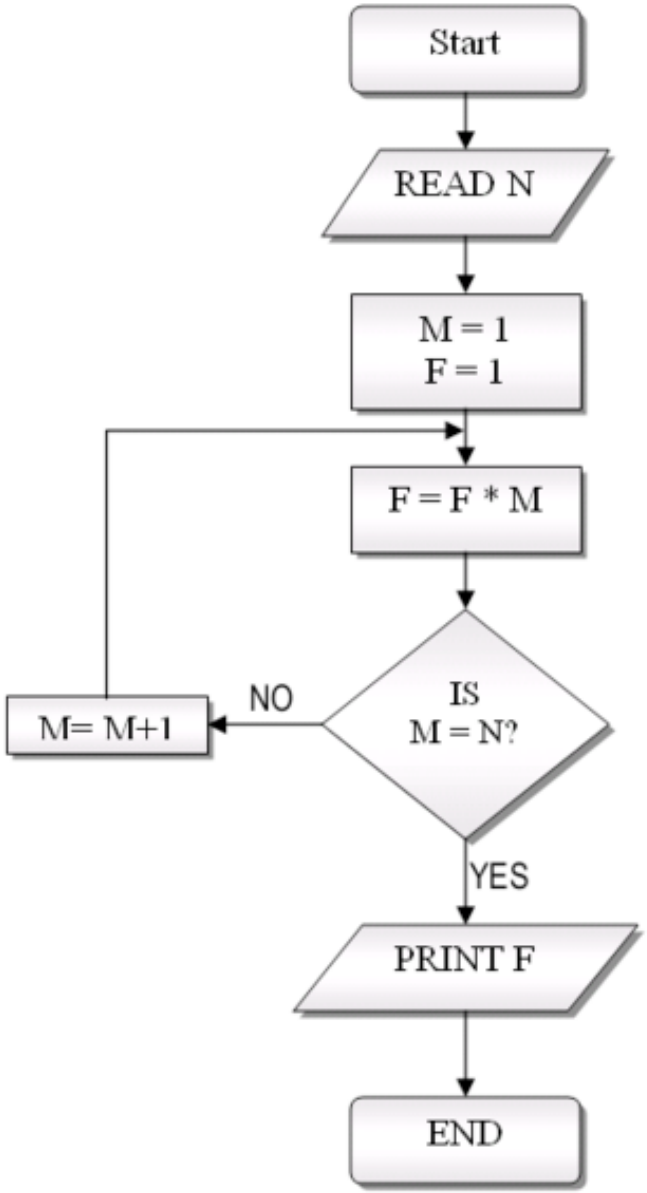
- En metod är en funktion som kan hämta information från ett objekt eller manipulera objektets attribut. En metod som hämtar information benämns ofta getter (efter engelskans get, hämta) och en metod som ändrar ett attribut kallas setter (efter engelskans set, ställa in).

[Wikipedia]

Flow-chart

- Start and end symbols
- Arrows = Showing "flow of control”
- Input/Output
- Generic processing steps
- Conditional or decision

A flowchart for computing the factorial of N



[Wikipedia]

Implementation

- Identifiera Subrutiner/Procedurer/Funktioner
- Beskriv varje subrutin
  - Namnge
  - Kort beskrivning av vad som sker
  - Definiera gränssnitt (API - Application Programming Interface), dvs in och utparametrar och hur de skickas till och ifrån subrutinen.
  - Eventuella sidoeffekter (register som förstörs eller påverkan på yttre enheter via I/O)
  - Kodexempel hur rutinen anropas
  - Resultatexempel ifrån några typiska/kritiska inparametrar
- Koda – Beskriva Algoritmer
- Testa (först varje subrutin var för sig)

Algorithmic Problem Solving

- An **algorithm** is a well-defined computational procedure consisting of *a set of instructions*, that takes some value or set of values, as *input*, and produces some value or set of values, as *output*.

Algorithm

- A simple definition: *A set of instructions for solving a problem.*

- Knuth (1968, 1973) has given a list of five properties that are widely accepted as requirements for an algorithm:
  - ***Finiteness****: "An algorithm must always terminate after a finite number of steps"*
  - ***Definiteness****: "Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case"*
  - ***Input****: "...quantities which are given to it initially before the algorithm begins. These inputs are taken from specified sets of objects"*
  - ***Output****: "...quantities which have a specified relation to the inputs"*
  - ***Effectiveness****: "... all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time by a man using paper and pencil"*

- Can first be presented in *pseudo-code* or *flowchart*, and finally in actual code

[http://www.scriptol.com/programming/algorithm-definition.php]

Find minimum, maximum and average (1/3)

- Version 1

First, you initialise *sum* to zero, *min* to a very big number, and *max* to a very small number.

Then, you enter the numbers, one by one.

For each number that you have entered, assign it to *num* and add it to the *sum*.

At the same time, you compare *num* with *min*, if *num* is smaller than *min*, let *min* be *num* instead.

Similarly, you compare *num* with *max*, if *num* is larger than *max*, let *max* be *num* instead.

After all the numbers have been entered, you divide *sum* by the numbers of items entered, and let *ave* be this result.

End of algorithm.

Find minimum, maximum and average (2/3)

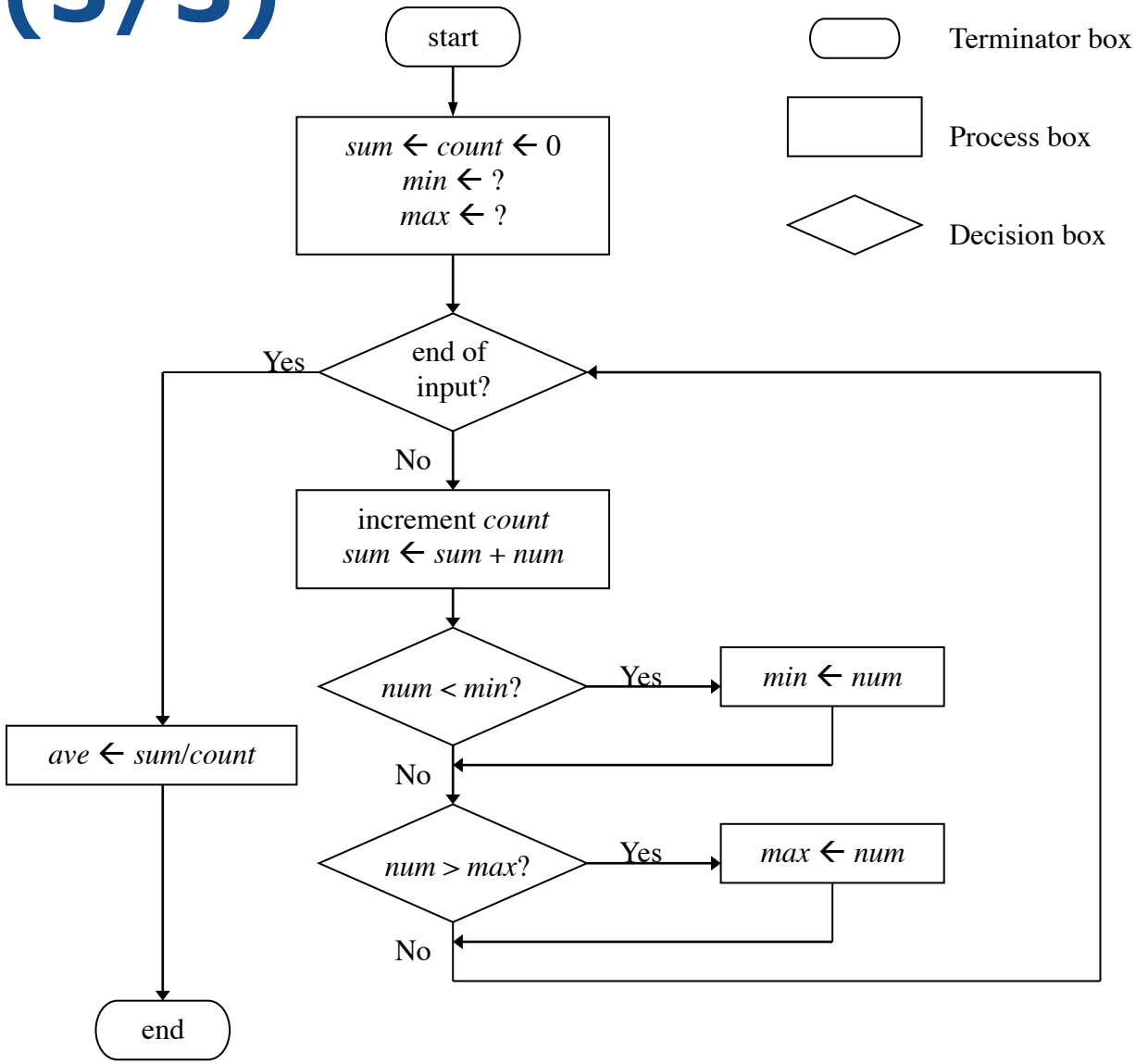
- Version 2

```
sum ← count ← 0      // sum = sum of numbers;
                     // count = how many numbers are entered?
min ← ?              // min to hold the smallest value eventually
max ← ?              // max to hold the largest value eventually }
for each num entered,
        increment count
        sum ← sum + num
        if num < min then min ← num
        if num > max then max ← num

ave ← sum / count
```

Find minimum, maximum and average (3/3)

- Flowchart



Beskriv varje subrutin

- Namnge subrutinen
- Kort beskrivning av vad som sker
- Definiera gränssnitt (API - Application Programming Interface), dvs in och utparametrar och hur de skickas till och ifrån subrutinen.
- Eventuella sidoeffekter (register som förstörs eller påverkan på yttre enheter via I/O)
- Kodexempel hur rutinen anropas
- Resultatexempel ifrån några typiska/kritiska inparametrar

Exempel på subrutinbeskrivning BinHex

```
; ========================
; Binhex subroutine
;
; Purpose: Binhex subroutine converts a 16 bit value to an ASCII string
;
; Initial Condition
;   First parameter on the stack is the value
;   Second parameter is the address of the string
; Final Condition
;   the HEX string occupies 4 bytes beginning with
;   the address passed as the second parameter
; Registers changed: No registers are affected
; Example Call:
;     LDR   R0, Number  ;Load number to register
;     LDR   R1, =String  ;load address of string
;     PUSH {R1}         ;and store it
;     PUSH {R0}         ;and store number to stack
;     BL    Binhex      ;branch/link
; Sample case:
;   Initial condition:   [SP] (top of stack) : 4CD0
;                        [SP+4] : Address for subr. output string
;   Final condition:     The string '4''C''D''0' in ASCII in memory
;
```

BinHex Implementation

```
Binhex
        MOV     R8, R7              ;save stack pointer for later
        STMFD   R7, {R0-R6,R14}     ;push contents of R0 to R6, and LR onto the stack
        MOV     R1, #4              ;init counter
        ADD     R7, R7, #4          ;adjust pointer
        LDR     R2, [R7], #4        ;get the number
        LDR     R4, [R7]            ;get the address of the string
        ADD     R4, R4, #4          ;move past the end of where the string is to be stored

Loop
        MOV     R0, R2              ;copy the number
        AND     R0, R0, #Mask       ;get the low nibble
        BL      Hexdigit            ;convert to ASCII
        STRB    R0, [R4], #-1       ;store it
        MOV     R2, R2, LSR #4      ;shift to next nibble
        SUBS    R1, R1, #1          ;decr counter
        BNE     Loop                ;loop while still elements left

        LDMFD   R8, {R0-R6,R14}     ;restore the registers
        MOV     PC, LR              ;return from subroutine
```

Exempel på subrutinbeskrivning HexDigit

```
; ========================
; Hexdigit subroutine
;
; Purpose: Hexdigit subroutine converts a Hex digit to an ASCII character
; Initial Condition:
;     R0 contains a value in the range 00 ... 0F
; Final Condition
;     R0 contains ASCII character in the range '0' ... '9' or 'A' ... 'F'
; Registers changed: R0 only
;
; Call example:
;    MOV   R0, #6              ;convert 6 into '6' (0x36)
;    BL    Hexdigit            ;convert to ASCII
; Sample case
;    Initial condition         R0 = 6
;    Final condition           R0 = 36 ('6')
```

HexDigit Implementation

```
Hexdigit
          CMP       R0, #0xA                    ;is the number 0 ... 9?
          BLE       Addz                        ;if so, branch
          ADD       R0, R0, #"A" - "0" - 0xA    ;adjust for A ... F
Addz
          ADD       R0, R0, #"0"                ;change to ASCII
          MOV       PC, LR                      ;return from subroutine
```

Exempel på huvudrutin

```
StackStart EQU       0x20004000          ;declare where top of stack will be

Main
           LDR       R7, =StackStart     ;Top of stack = 0x20004000
           LDR       R0, Number          ;Load number to register
           LDR       R1, =String         ;load address of string
           STR       R1, [R7], #-4       ;and store it
           STR       R0, [R7], #-4       ;and store number to stack
           BL        Binhex              ;branch/link

Done       B Done                        ;all done


Number     DCD       &4CD0               ;number to convert
String     DCB       4, 0                ;counted string for result
```

Uppgift: Äggklocka

- Skapa en ”äggklocka” som kan visa minuter och sekunder kvar innan ett larm ljuder

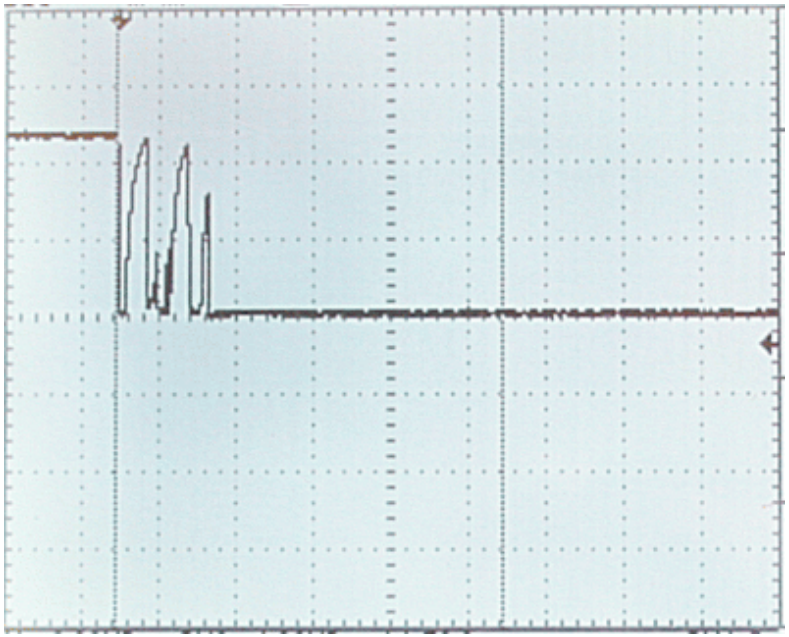
- Basfunktioner: Visa tid, ljuda alarm
- I/O-funktioner: två knappar, två lysdioder, en klockdisplay (7-segments display med 4 siffror)
- Inställningar: ??

Modularisering

- Vilka vore lämpliga moduler?
- Vilka är gränssnitten?
- Hur ska vi testa modulerna?

Knapp ”debouncing”

- Debouncing görs typiskt på ett av två sätt:
- Med timer och avbrott
- Med pulling och test av knappens tillstånd



Debouncing the switch using the timer and interrupt.

- The idea is to use the **SysClk** configured to generate an **interrupt** every **1mS** and use this interrupt for implementing the Input debounce.

Debouncing Flow diagram

- The Input is recognize at 1 or 0 if it stay at 1 or 0 for a time > of REFdebounce.
- In another words, the 1 or the 0 must be 1 or 0 for all the REFdebounce time.
- If during the samples the input change from 0 to 1 to 0, every change reset the In1_0 and In1_1.
- In this way when In1_0 or In1_1 exceeds or is equal to REFdebounce means that it were at 1 or 0 for at least of the REFdebounce time.