Datorteknik TSEA82 + TSEA57
Fö4

Strukturerad programmering med JSP

LINKÖPINGS UNIVERSITET

Datorteknik Fö3 : Agenda

- Lägeskoll
- Strukturerad programmering – JSP
- Kodstil
- Lab 1
- Tid för frågor

Lägeskoll

Var är vi?

V13 **V14** V15 V16 V17 V18 V19 V20 V21

Fö1	Fö4	Fö6	Fö7	Fö8	Fö9	Le		TPVT2
Fö2	La0	La1	La2		La3	La4	La5	LAX!
Fö3	Fö5							

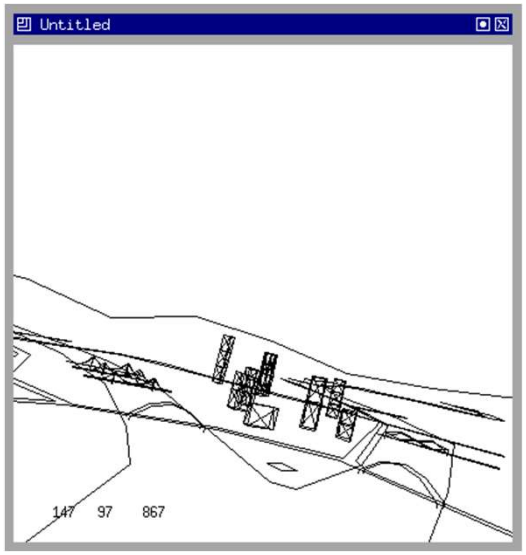
Strukturerad programmering

Algorithms + Data structures = Programs

IOCCC : International Obfuscated C Code Contest

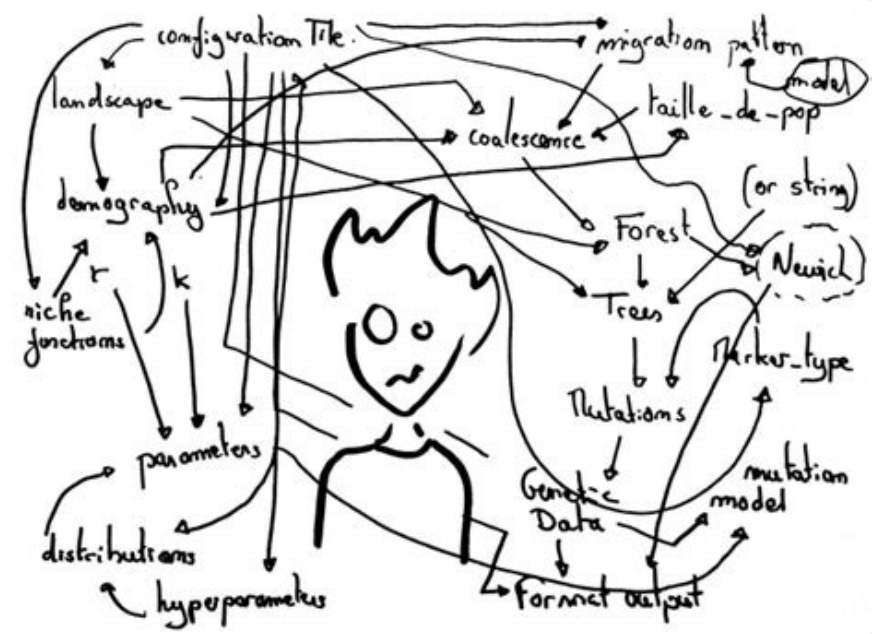
```
#include                                <math.h>
#include                              <sys/time.h>
#include                              <X11/Xlib.h>
#include                             <X11/keysym.h>
                                     double L,o ,P
                                    ,_=dt,T,Z,D=1,d,
                                    S[999],E,h= 8,I,
                                    J,K,w[999],M,m,O
                                    ,n[999],j=33e-3,i=
                                    1E3,r,t, U,W ,N,S=
                                    74.5,l=221,X=7.26,
                                    a,B,A=32.2,c, F,H;
                                    int N,q, C, Y,p,V;
                                   Window z; char f[52]
                                 ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
 ; scanf("%lf%lf%lf",y +n,N+y, y+5)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
 0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
 ; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
 sin(j); a=B*T*D-E*N; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (O+=(I=D*W+E
 *T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
 l+= 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
  *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+N *M;
   XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                  XEvent z; XNextEvent(e ,&z);
                                      ++*((N=XLookupKeysym
                                       (&z.xkey,0))-IT?
                                       N-LT? UP-N?& E:&
                                       J:& u: &h); --*(
                                       DN -N? N-DT ?N==
                                       RT?&u: & W:&h:&J
                                        ); } m=15*F/l;
                                        c+=(I=M/ l,l*H
                                        +I*M+a*X)*_; H
                                        =A*r+v*X-F*l+(
                                        E=.1+X*4.9/l,t
                                        =T*m/32-I*T/24
                                         )/S; K=F*M+(
                                         h* 1e4/l-(T+
                                         E*5*T*E)/3e2
                                         )/S-X*d-B*A;
                                         a=2.63 /l*d;
                                         X+=( d*l-T/S
                                          *(.19*E +a
                                          *.64+J/1e3
                                          )-M* v +A*
                                          7)*_; l +=
                                          K *_; W=d;
                                          sprintf(f,
                                          "%5d  %3d"
                                          "%7d",n =l
                                         /1.7,(C=9E3+
                                  O*57.3)%0550,(int)i); d+=T*(.45-14/l*
                                 X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                                 *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
                                 179*v)/2312; select(p=0,0,0,0,&G); v-=(
                                  W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                                   )/107e2)*_; D=cos(o); E=sin(o); } }
```

"It is an quasi-annual contest to see who can write the most unreadable, unintelligible, unmanageable, but legal C program."



[http://blog.aerojockey.com/post/iocccsim]

Spaghetti code



Spaghetti-kod är typiskt ett resultat av ett ostrukturerat eller ogenomtänkt programflöde.

En vanlig orsak är överanvändning av hopp till avlägsna delar av koden, istället för att samla och gruppera kodstycken som relaterar till varandra.

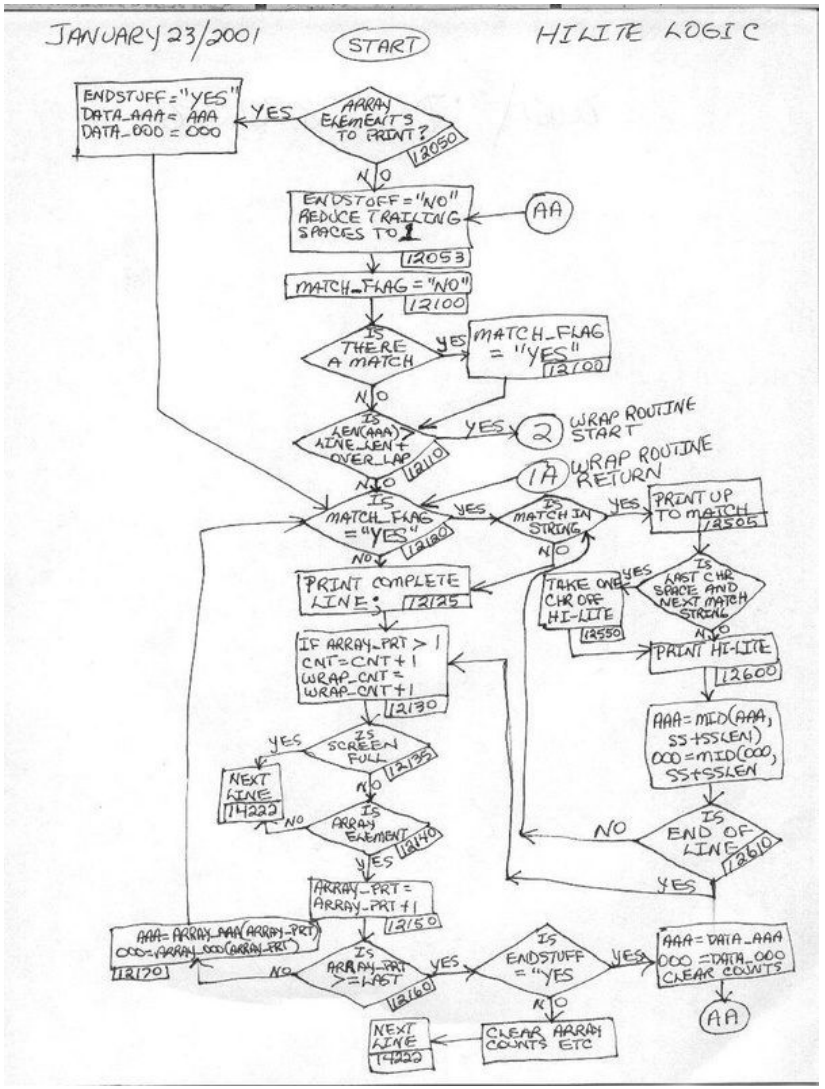
Även datastrukturen spelar en viktig roll.

Beware of the Spaghetti Monster

Spaghetti code

Flödesdiagram är väl bra, eller?
Överblick?
Var börjar det?
Var slutar det?
Vilka är programmets beståndsdelar?
Kan man återanvända kod som upprepar sig?
Går det ens att se var koden upprepar sig?

Det finns ingen STRUKTUR!!



Strukturerad programmering

Att kunna ett programspråk
är **INTE** detsamma
som att kunna programmera!

[Jfr : Trafikregler vs Bilkörning]

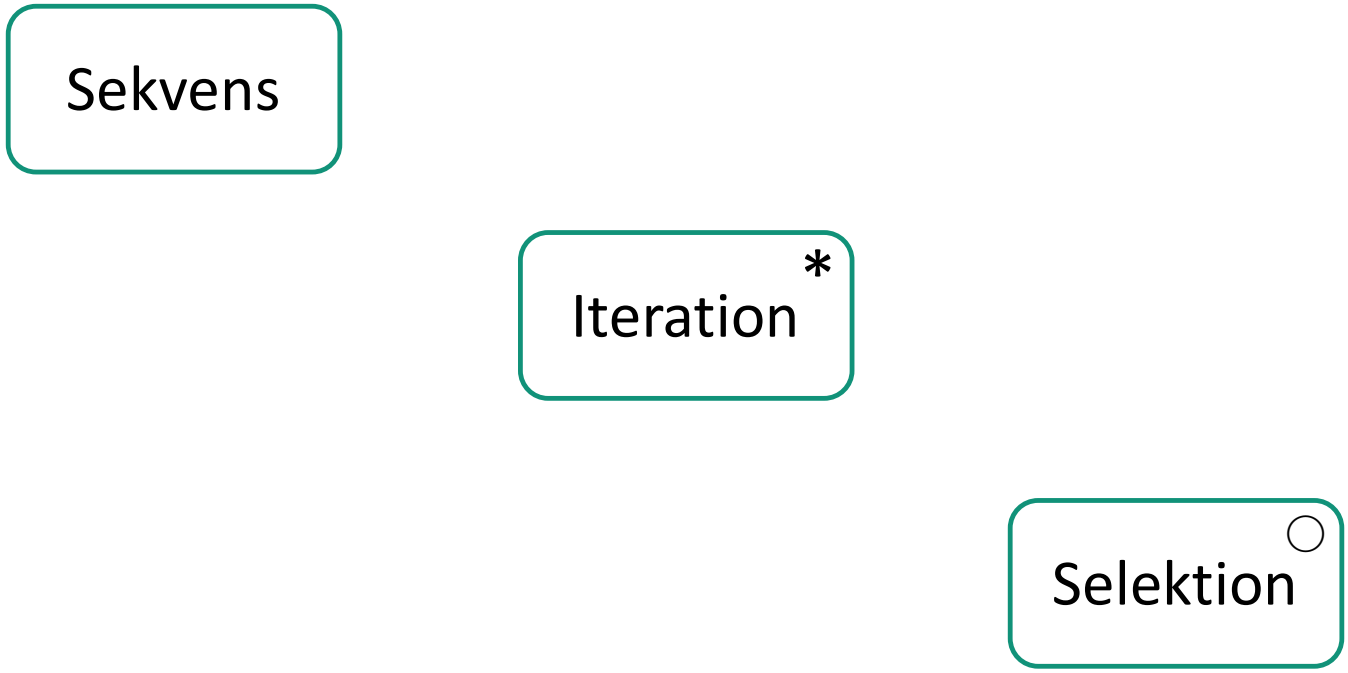
JSP – Jackson Structured Programming

JSP-princip

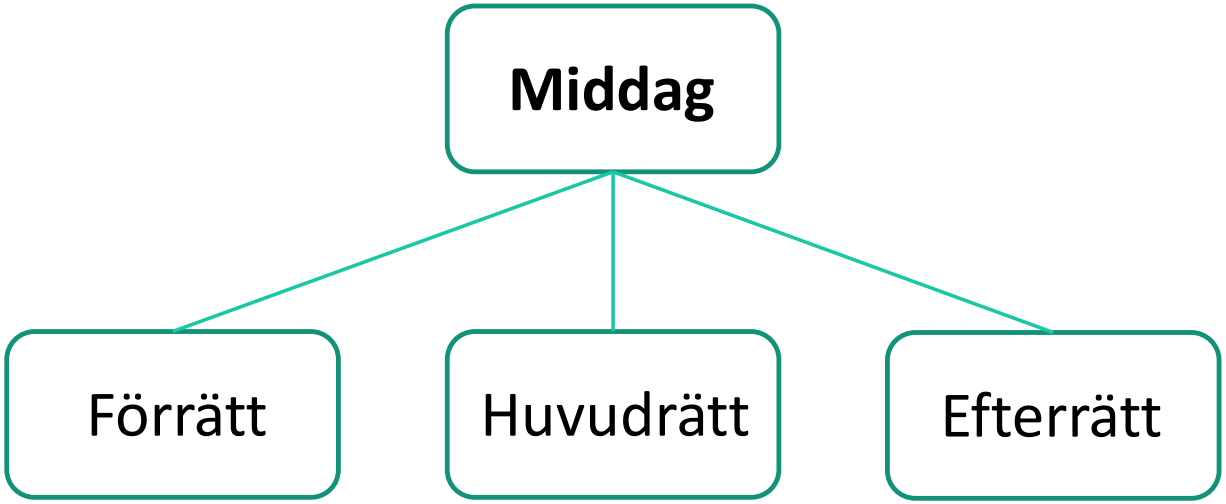
- Data kan beskrivas i strukturdiagram
- Programmet ska följa datat.

Algorithms + Datastructures = Programs

JSP : Strukturdiagram Byggblock



JSP : Sekvens : först det ena, sen det andra och sen …

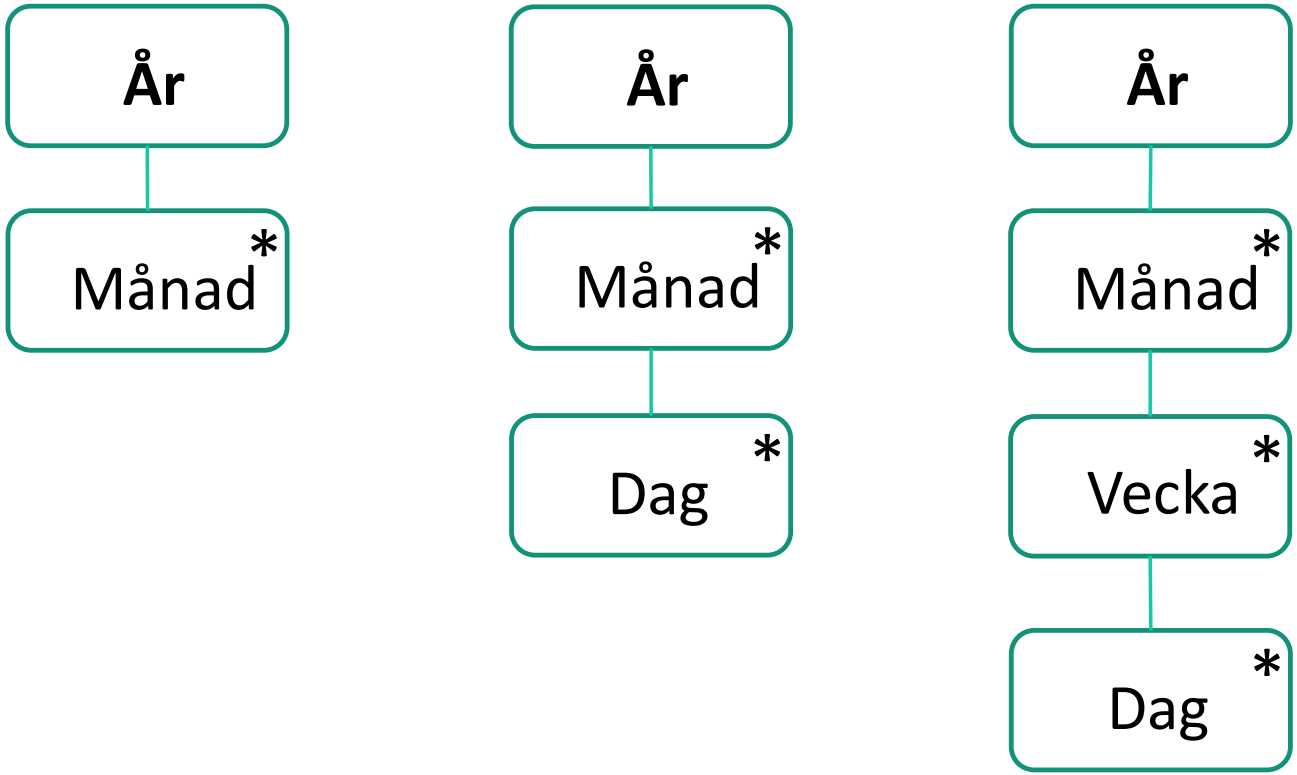


-Definiera konstanter
-Initiera hårdvara
-...

-Utför databehandling
-Leverera delresultat
-...

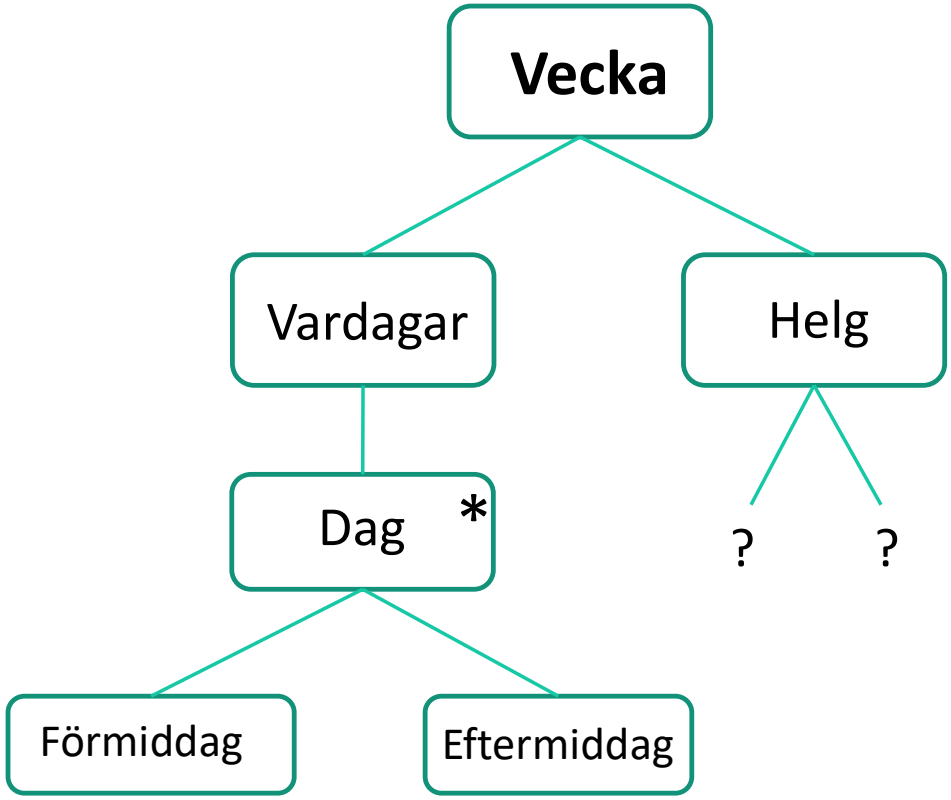
-Slutresultat
-Avsluta HW
-...

JSP : Iteration : om och om igen och igen …



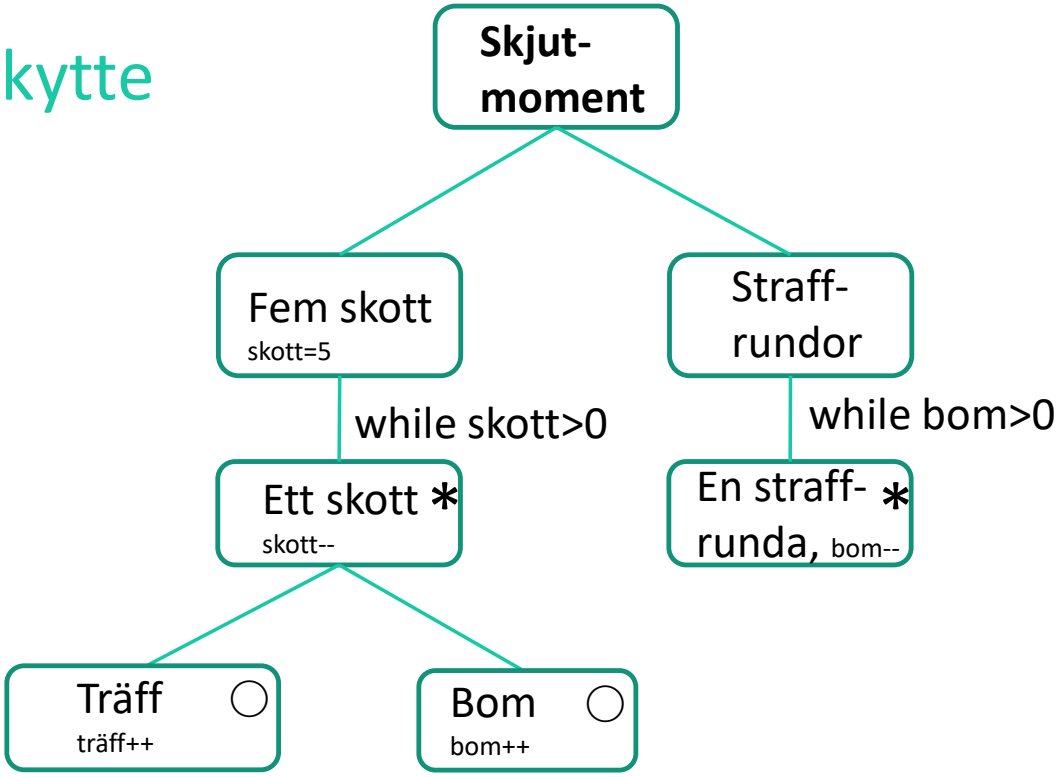
Iterationsdelen kan
Utföras **NOLL** gånger!

JSP : Kombinerad sekvens och iteration



JSP : Selektion : det ena eller det andra eller …

Skidskytte



Iterationsdelen kan Utföras **NOLL** gånger!

JSP : Regler

En komponent får endast ha delar av samma typ:
- En sekvens ska endast bestå av sekvensdelar
- En selektion ske endast bestå av selektionsdelar
- En iteration ska endast bestå av en (1) iterationsdel

Delarna kan dock utgöras av komponenter av annan typ.

En iteration kan ske noll gånger:
- Tänk **`while()`** inte **`if()`**

En selektion måste ha minst två delar:
- Ett val måste ha mer än ett alternativ
- Ofta ett defaultalternativ också

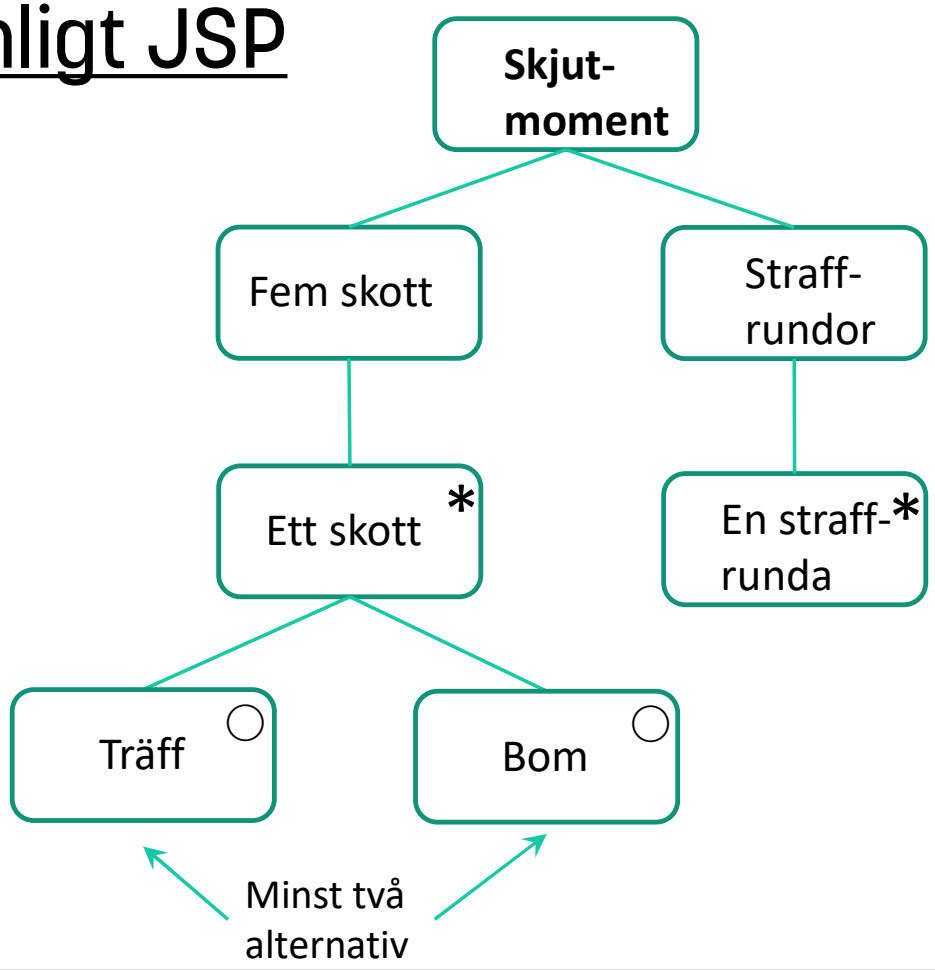
Korrekt enligt JSP

Sekvens

Sekvensdel
Iteration

Iterationsdel
Selektion

Selektionsdelar



Delar av samma typ

Sekvensdel
Iteration

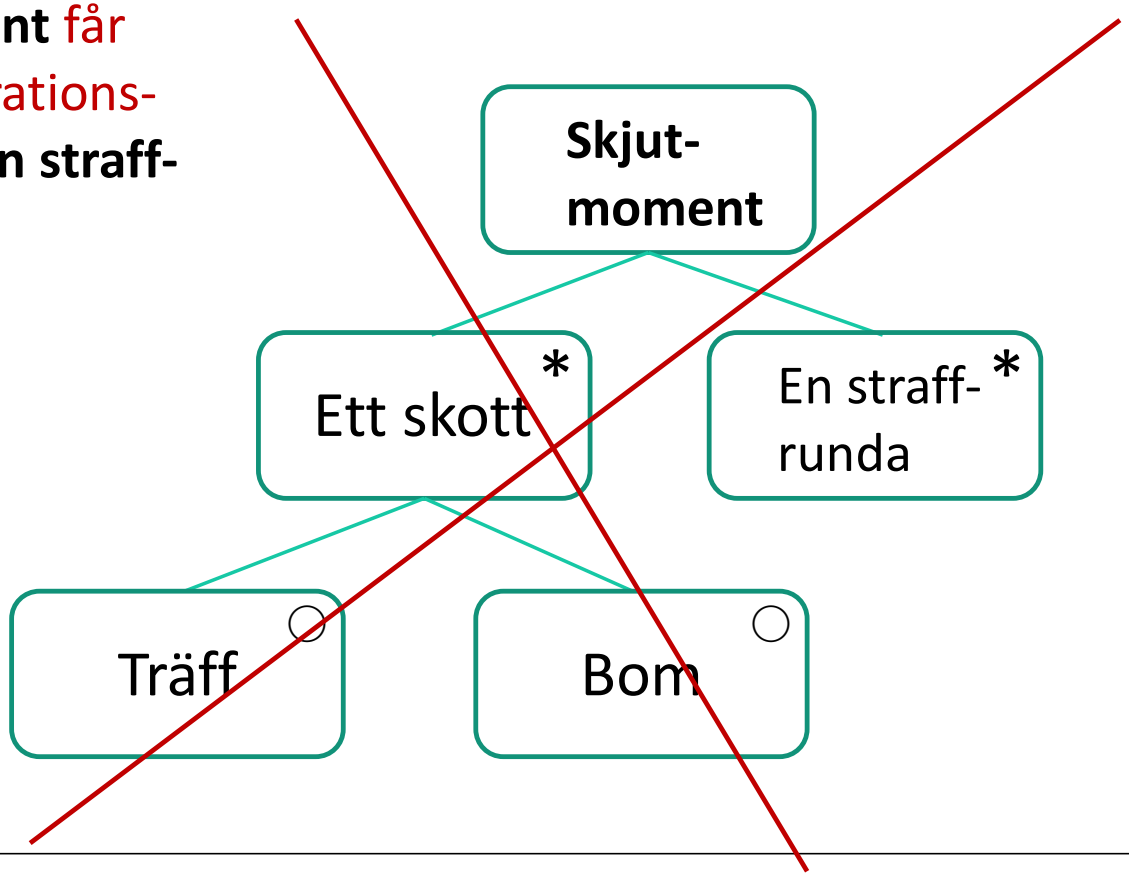
Eventuellt noll gånger

Iterationsdel

En upprepad del

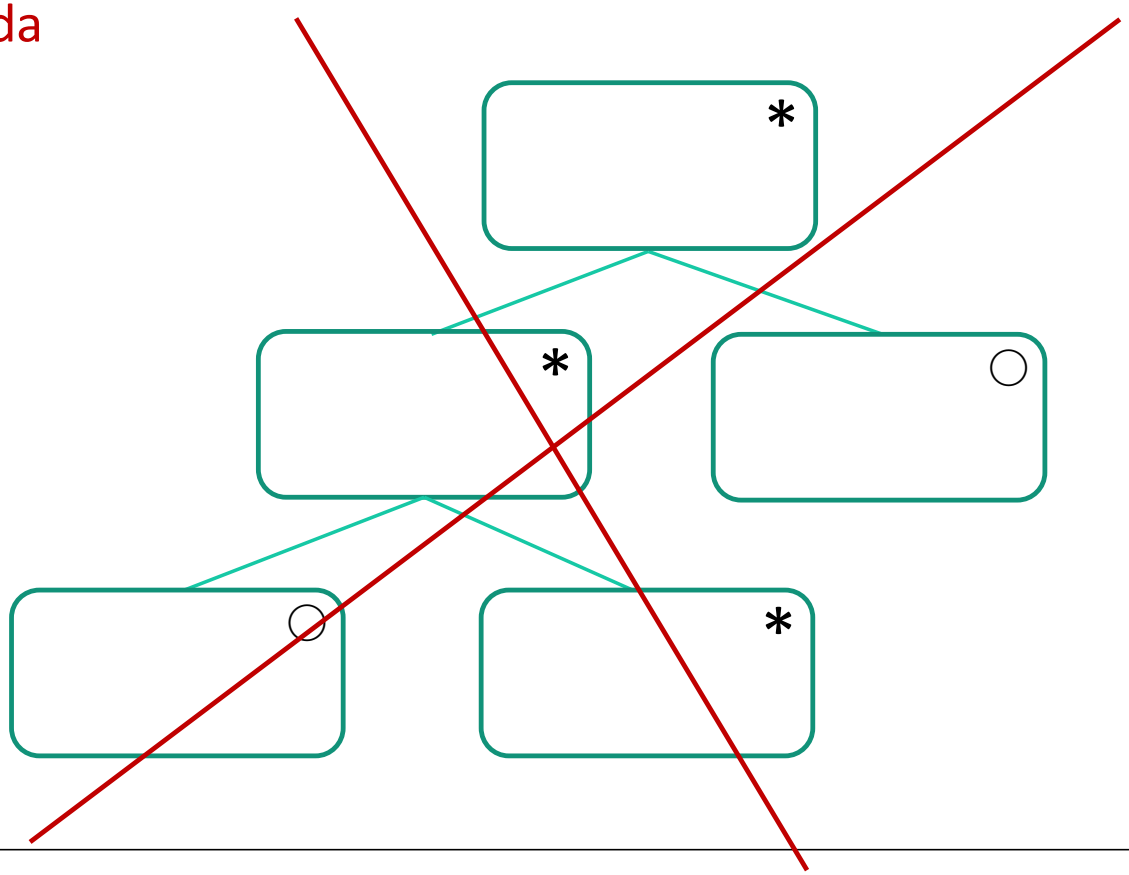
Inte korrekt enligt JSP

Sekvensen **Skjutmoment** får t ex inte utgöras av iterations-delarna **Ett skott** och **En straff-runda**!!

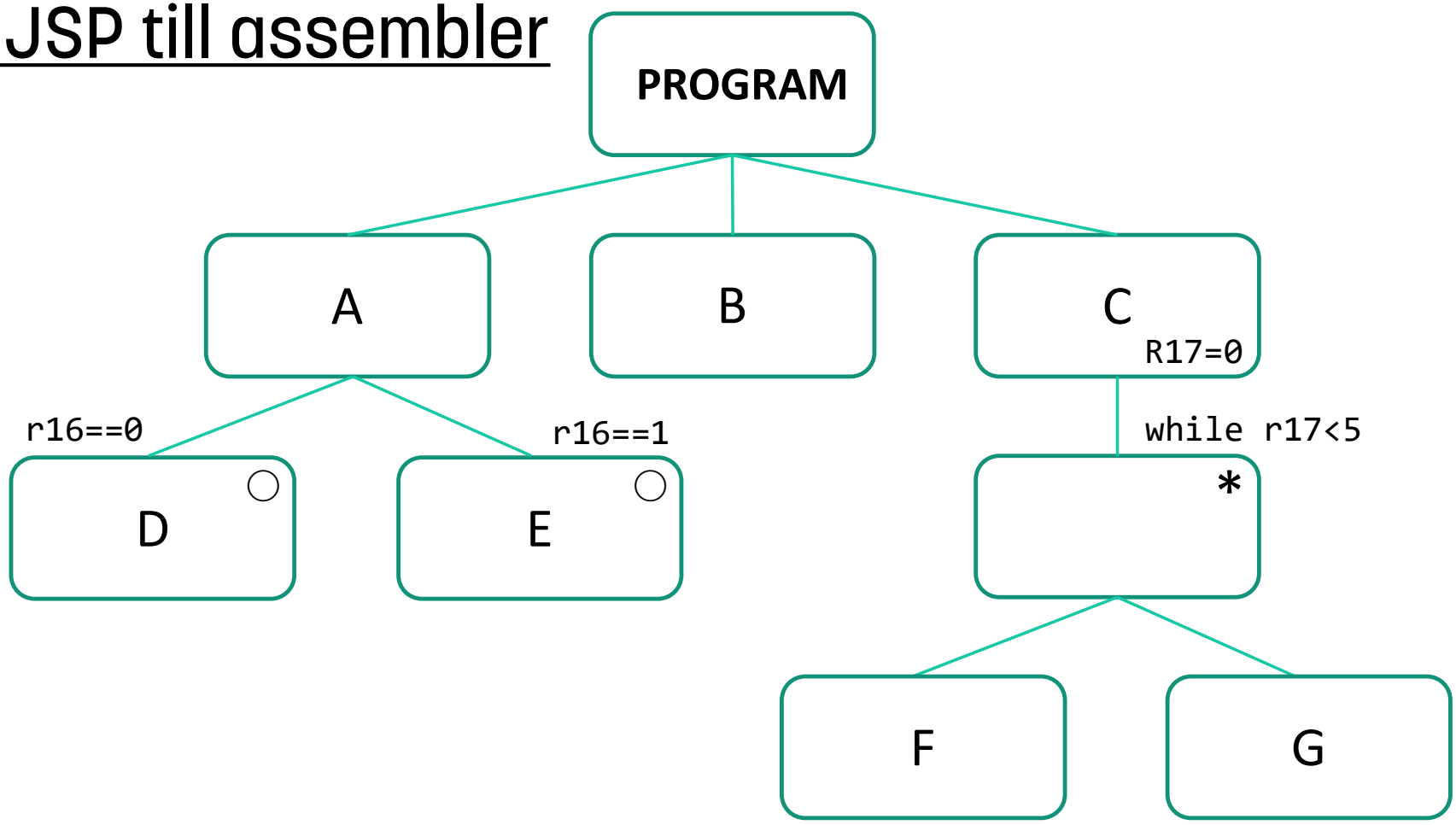


Inte korrekt enligt JSP

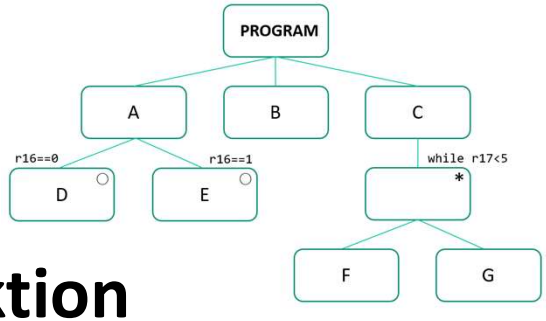
Man kan ej heller blanda
delar av olika typ!!



Från JSP till assembler



Från JSP till assembler



Sekvens

```
PROGRAM:
    call A
    call B
    call C
LOOP:
    ...
    jmp  LOOP


B:
    ...
    ret
```

Selektion

```
A:
    cpi  r16,0
    brne A_1
    call D
    jmp  A_EXIT
A_1:
    cpi  r16,1
    brne A_DEFAULT
    call E
    jmp  A_EXIT
A_DEFAULT:
    ...
A_EXIT:
    ret
```

Iteration

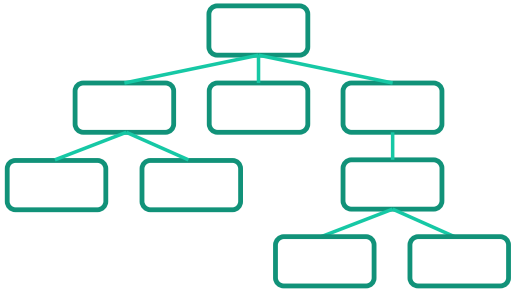
```
C:
    clr  r17
C_LOOP:
    cpi  r17,5
    breq C_EXIT
    call F
    call G
    inc  r17
    jmp  C_LOOP
C_EXIT:
    ret
```

Kodstil

Från idé till program

Idé

Strukturerad lösning
t ex via JSP



Programkod
.asm

```
     .org 0
     jmp  MAIN

     .dseg $0100
ARR:
     .byte $0C

     .cseg
MAIN:
     ldi  r16,HIGH
     out  SPH,r16
     ldi  r16,LOW
     out  SPL,r16
     ...
```

Maskinkod
.hex

```
:1001000049726F6E
:100110006D616964
:10012000656E7275
:100130006C657A20
:
:
:
:
:
:
```

Övergripande princip

Definiera först, använd sen

Vektortabell : I huvudsak, hopp till huvudprogram men avbrottsvektorer

Minnesanvändning : I huvudsak, deklaration av var minnesanvändningen börjar, men även deklaration av variabler i minnet.

Subrutiner : Definition av subrutiner, gärna grupperade i samhörande moduler.
Även definition av avbrottsrutiner.

Huvudprogram : Initiering av variabler/data, följt av huvudloop med anrop av subrutiner och villkorsstyrda förgreningar.

Vektortabell

Definition av **minnesanvändning**

Definition av **subrutiner**/ underfunktioner

Huvudprogram med huvudloop

```
     .org 0
     jmp  MAIN

     .dseg $0100
ARR:
     .byte $0C

     .cseg
SUB1:
     push r16
     ...
     pop  r16
     ret

     ...

MAIN:
     ldi  r16,HIGH
     out  SPH,r16
     ldi  r16,LOW
     out  SPL,r16
MAIN_LOOP:
     ...
     call SUB1
     ...
     jmp  MAIN_LOOP
```

Vektortabell

I programminnets början finns en vektortabell dit exekveringen automatiskt hoppar vid reset/start av processorn (adress `0x000`), eller vid avbrott (någon annan adress beroende på vilket avbrott som inträffat).

I vektortabellen ska det bara finnas hoppinstruktioner som hoppar vidare till rätt plats i programmet beroende på vad som hänt.

```
.org 0x000
jmp   MAIN
.org 0x001
jmp   INT0_vect
.org 0x015
jmp   ADC_vect
...
```

Minnesanvändning

Deklaration av var minnesanvändningen börjar inleds med
`.dseg` *`adress`*
vilket talar om var datasegmentet börjar.

Inne i datasegmentet kan man deklarera namn på variabler och hur mycket minne respektive variabel ska ha

```
variabel1:      .byte     minnesmängd
variabel2:      .byte     minnesmängd
```

Avslutningsvis talar man om var datasegmentet slutar och kodsegmentet börjar med
`.cseg`

```
    .dseg  0x0100

VAR1:
    .byte  0x0C
VAR2:
    .byte  0x02

    .cseg
```

```
VAR1: [$0100..$010B]
VAR2: [$010C..$010D]
```

Subrutiner

Subrutiner är ”underprogram” som anropas från andra delar av programmet, och där man återvänder (gör retur) till efter anropet när subrutinen är färdig.

Inne i en subrutin förekommer ofta behovet av en lokal användning av register, som kanske används på andra platser i programmet. Innehållet i de register som används lokalt sparas lämpligen undan (`push`) på processorns stack i början på subrutinen, och återhämtas (`pop`) i slutet. Detta kallas för att spara och återhämta kontext.

Alla symboliska adresser (lablar) i en subrutin ges lämpligen samma stam-namn (med något tillägg) som subrutinens namn.

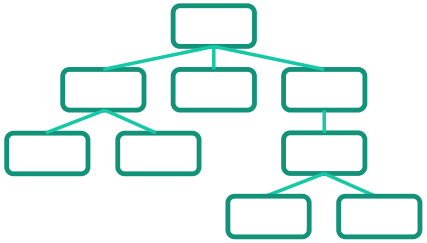
```
; SUB ADC_INIT
; Initialize A/D
ADC_INIT:
    push   r16
    ...
    ldi    r16,4
    out    ADMUX,r16
    ...
ADC_INIT_EXIT:
    pop    r16
    ret
```

Huvudprogram

Huvudprogrammet ligger lämpligen sist i programkoden, då det typiskt kommer att använda och anropa de tidigare definitionerna av subrutiner. *Definiera först, använd sen.*

Huvudprogrammet inleds typiskt med någon form av initering följt av en huvudloop där programmet sedan för evigt kör.

Det är egentligen i huvudprogramet som själva strukturen av det tidigare JSP-diagrammet bor. Huvudprogrammet byggs upp av villkorsstyrda programförgreningar och anrop av subrutiner.



```
; ### Main program
MAIN:
    ; set stack
    ldi    r16,HIGH
    out    SPH,r16
    ...
    call   ADC_INIT
    ...
MAIN_LOOP:
    ...
MAIN_GET_KEY:
    call   READ_KBD
    cpi    r16,4
    brne   MAIN_GET_KEY
    ...
    jmp    MAIN_LOOP
```

Prioritetsordning

1. **Funktion**: Programmets funktion är överordnat, men nästan lika viktigt är nästa punkt, struktur.
2. **Struktur**: För att ett program ska fungera bra, vara smidigt att utveckla och vidareutveckla, vara läsbart och senare kunna optimeras är dess struktur avgörande.
3. **Optimering**: När funktion och struktur finns på plats blir optimering ofta enkelt. Det handlar då typiskt om att stegvis reducera ”randvillkor” mellan olika subrutiner eller moduler, om nu detta är nödvändigt. En skicklig och erfaren programmerare kan säkert uppnå extremt optimerad kod genom gå direkt på optimeringssteget och hoppa över strukturingen, men sådan kod blir ofta obskyr och mer eller mindre omöjlig att vidareutveckla.

Labb 1

...

Tid för Frågor

Anders Nilsson

www.liu.se

LINKÖPINGS UNIVERSITET