

Tilstander og invarianter

Om å lage gode programmer

Kan ChatGPT hjelpe oss med å programmere?

+ Noen få eksamenstips fra Stein

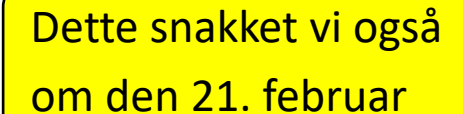
+ litt  og litt 

IN1010 – 16. mai 2023

Stein Gjessing

For å være en god programmerer må du vite

- Hva tilstander i et program er
- Hva tilstandsoverganger i et program er
- Hva betingelser / antagelser er
- Hva for- og bak-betingelser er
- Hva en invariant er
 - Løkkeinvarianter
 - Invarianter på data i objekter ("class invariants") ←
- Pluss tips fra Stein: Litt om hva da skal gjøre under eksamen



Dette snakket vi også
om den 21. februar





Litt historie for informatikere

- Alan Turing
 - Turing-maskin - teoretisk informatikk
- The imitation game -> Turings test
 - Kan maskiner oppføre seg intelligent (1950) ?
2022: ChatGPT -> **JA**
- Von Neumann - maskin



Om å lage gode programmer

1. Tenk og resoner og lag programmet ditt mest mulig riktig med en gang
 - Tenk på tilstander og invarianter
2. Når du tror programmet er riktig (men ikke før): Test
3. Lag små biter program om gangen
 - Lett å resonere om
 - Lett å teste (enhetstesting)
 - Virker hele tiden (selv om det ikke gjør alt det skal)



IKKE

- Prøve og feile
- Håpe at programmet virker
- Legg til ny kode og håpe at programmet virker nå
- Tro at programmet er riktig når du har testet det én gang

DERIMOT MÅ DU

- Resonere og tenke
- Planlegge
- Skjønne hva som skjer i programmet ditt
- Skjønne hva feilene er før du retter dem opp

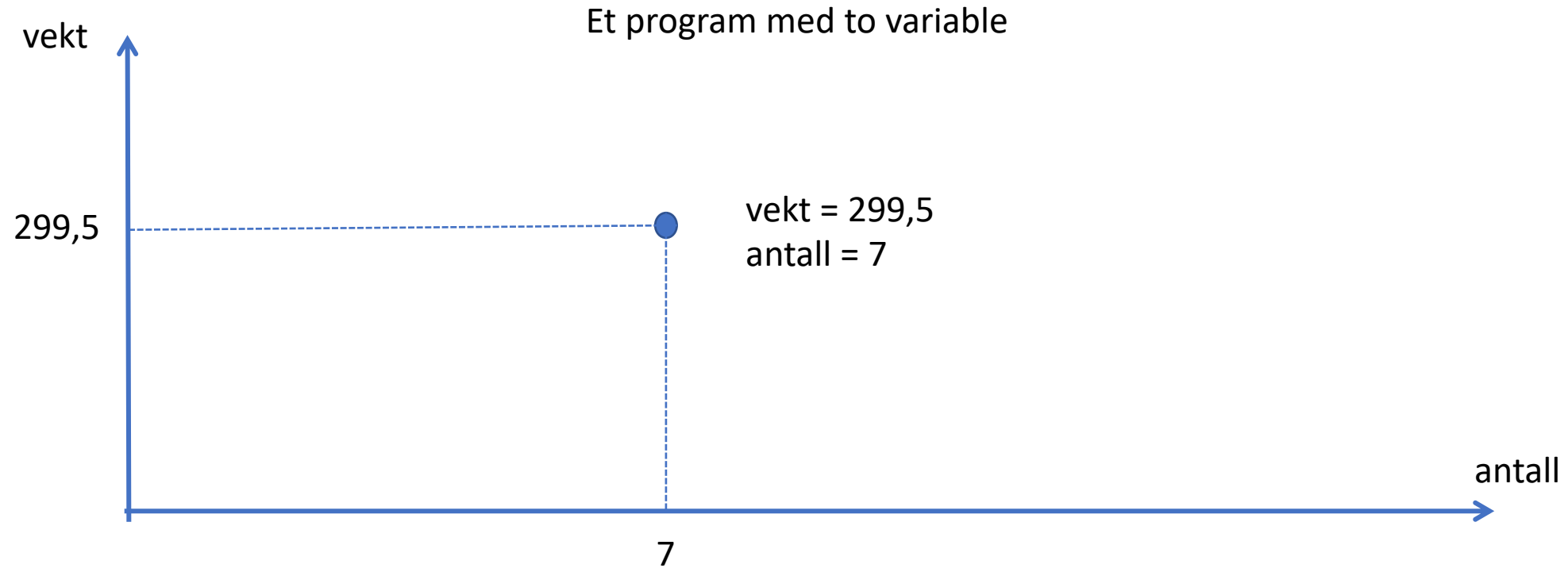


En tilstand i programmet

(IN1150 – Logiske metoder)

- En fullstendig beskrivelse av hele tilstanden til programmet ditt er en beskrivelse av alle verdiene på alle variablene i programmet + hvor programutførelsen er i øyeblikket (PC, Program Counter).
- Matematikk: Det N -dimesjonale rommet (N er antall variabler i programmet)
 - Endelig og tellbart

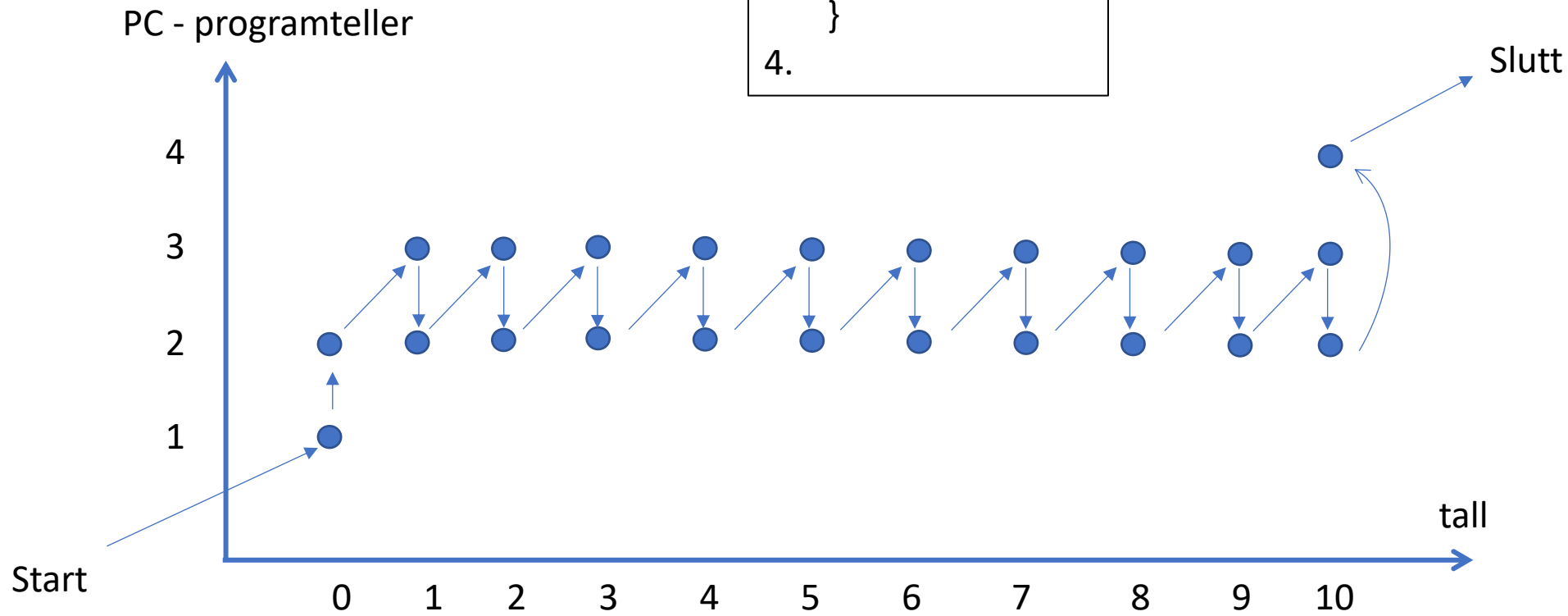
Tilstander og tilstandsrom



To variabler: 2-dimensjonalt rom
Hundre variabler: 100-dimensjonalt rom

Utvikling av tilstandsrommet

```
1. int tall = 0;  
2. while (tall < 10) {  
3.     tall++;  
4. }
```



Tilstandsdiagram med tilstandsoverganger

Utsagn/betingelser og invarianter

- Alle kombinasjoner av verdier av de N variablene vil ikke oppstå i et program
 - Dvs. alle punkter i tilstandsrommet er ikke mulig i et program
 - Programmet vårt styrer tilstander og tilstandsoverganger
- Å programmere vil si å ha kontroll på utvikling av programmets tilstand
 - Utvikling av verdien av alle variablene i forhold til hverandre.
- Som informatiker må vi derfor beskrive hvordan programmet vårt avgrenser tilstandsrommet
 - Spaghetti-kode (goto) hopper gjennom tilstandsrommet og er umulig å forstå
- **En betingelse** er en antagelse/utsagn om programmets tilstand
 - avgrenser tilstandsrommet
- **En invariant** er en betingelse som gjelder gjennom (nesten) hele programmet eller gjentatte ganger i programmet



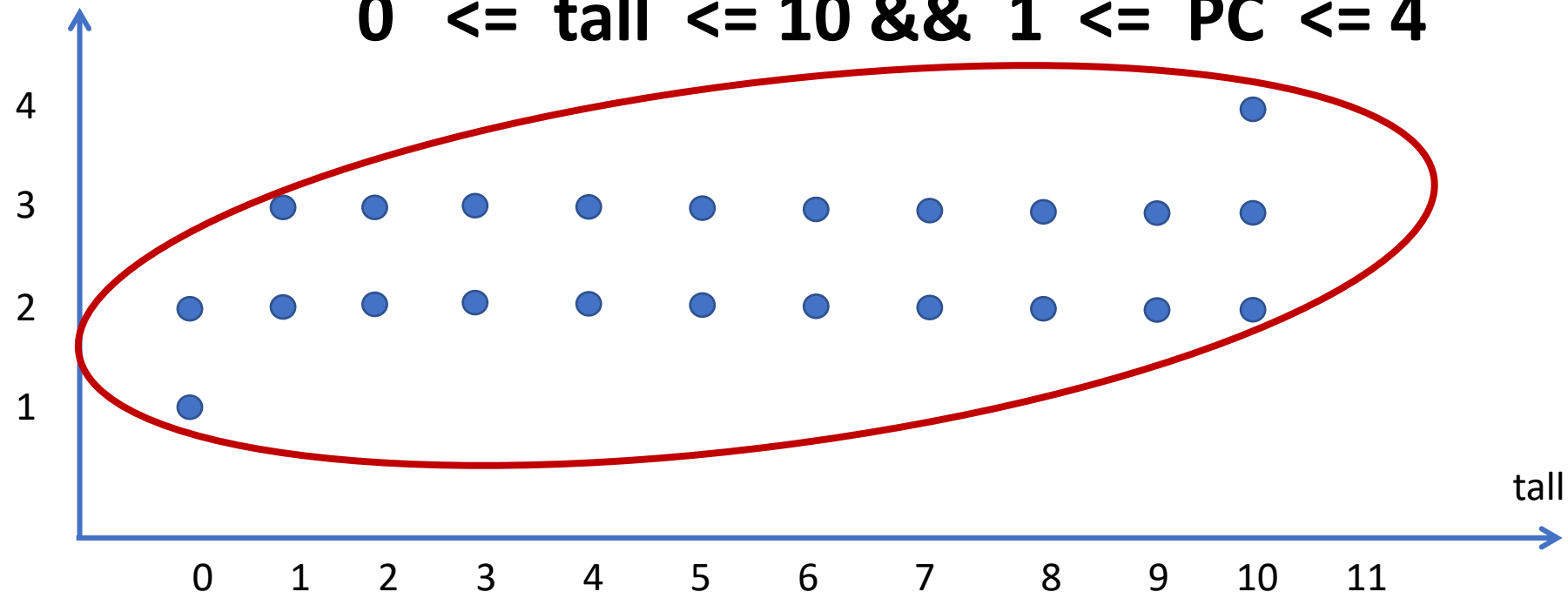
Mer om betingelser

- En betingelse er en beskrivelse av en (gruppe av) tilstand(er)
 - En antagelse / utsagn / assertion / a condition
 - F.eks.: $0 \leq \text{tall} \leq 10$
- En **invariant** er en betingelse som gjelder (nesten) hele tiden i (en del av) programmet vårt

En betingelse begrenser tilstandsrommet til programmet

PC - programteller

$0 \leq \text{tall} \leq 10 \ \&\& \ 1 \leq \text{PC} \leq 4$



```
1. int tall = 0;  
2. while (tall < 10) {  
3.     tall++;  
4. }
```



Eksempel på løkke som etablerer betingelse

Oblig 5: I Monitor2 sin metode hentUtTo():

```
while ( !A && ! B ) {
```

```
}
```

```
// Nå er alle hashMap-er ferdig behandlet og det er bare én igjen (A)
```

```
// eller
```

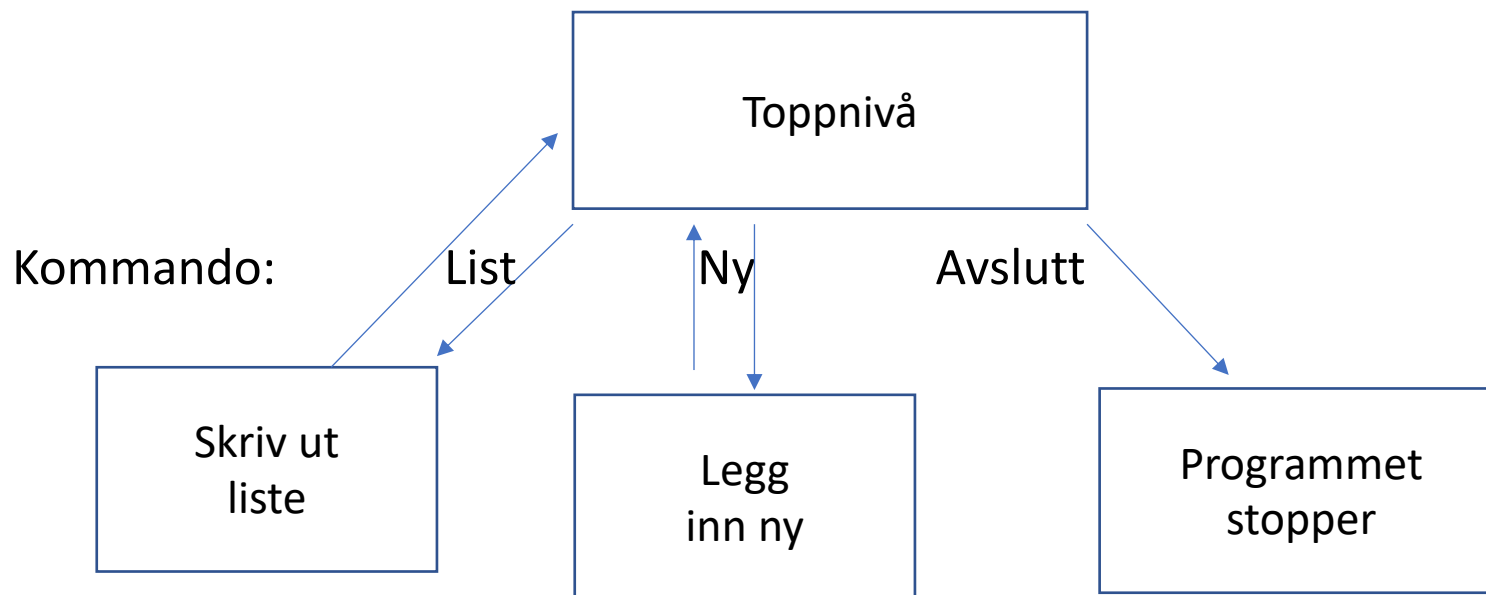
```
// det finnes to hashMap-er i Subsekvensregisteret (B)
```

```
// A || B
```

Litt mer om tilstander og tilstandsoverganger

- Grupper av tilstander
- For eksempel en kommandoløkke:

En meget grov
inndeling av
tilstandene til
programmet



De blå pilene er
→
tilstandsoverganger

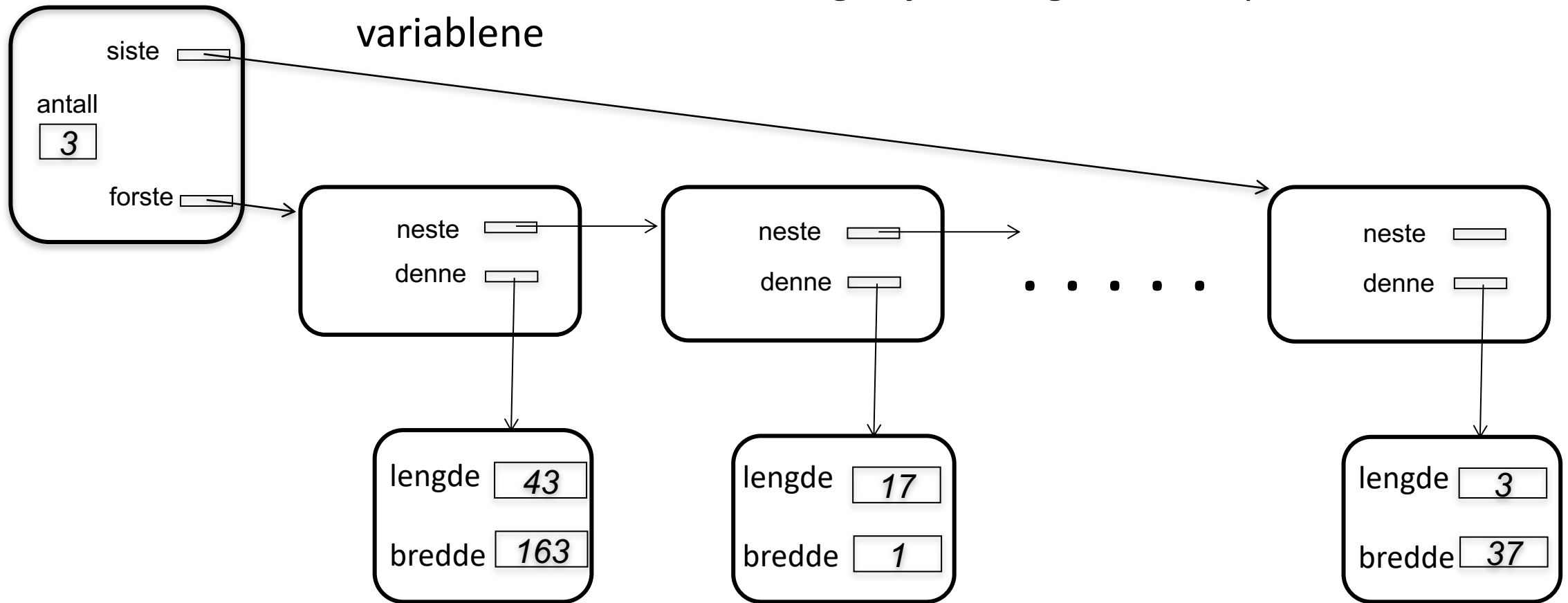
Tilstandsdiagram.

Engelsk: state diagram

Moduler i programmet

Datastrukturtegninger

Når vi har tegnet en datastruktur er dette en beskrivelse av (noe av) programmets tilstand på et bestemt sted i programutførelsen.
Datastruktur: Hovedsakelig objekter og verdiene på de sentrale variablene



Og bare for å ha nevnt det: Tilstander i parallelle programmer

- Tilstandene og tilstandsovergangene i ett sekvensielt program kan vi kanskje ha nogenlunde kontroll på
- Tilstandene i et parallelt program er mye vanskeligere å holde orden på fordi tilstandsrommet er delt i K deler, der K er antall tråder.
- Hvis disse trådene opererer helt uavhengig, er tilstandsovergangene som programmet til sammen utfører svært mye mer utfordrende å holde orden på (men det er kanskje ikke nødvendig)
 - Splitt og hersk

Hvis mulig, bruk uforandelige (immutable) data

- Tilstanden til uforandelige data er, ja nettopp, uforandelig
 - ÉN TILSTAND
- Dette gjør slike data lette å resonere om
 - F.eks. geografiske data
- Perfekt for tråder
- Hvis du må forandre disse data, kast alt og bruk / lag en helt ny kopi / versjon.



To viktige typer av invarianter

- Data i et objekt er alltid styrt av en (eller flere) beskrivelser av sammenhengene mellom dataene i objektet (invarianter (konsistensregler)) - **objektinvarianter** / klasseinvarianter
 - 21. februar (og repetisjon på neste bilder)
- Når du lager en løkke er det alltid en beskrivelse av en tilstand (en invariant) som sier hvor langt arbeidet i løkka er kommet - **løkkeinvariant**
 - Kommer om fire bilder

En invariant er en betingelse som gjelder gjennom (nesten) hele programmet eller gjentatte ganger i programmet

Invarianter på data i objekter – class invariants

```
public void settInn (Kanin en)
```

Pre-condition: Invarianten gjelder

```
if (antall < MAXANT) {  
    alle [antall] = en;  
    antall ++;  
}
```

Post-condition: Invarianten gjelder

```
public Kanin taUt()
```

Pre-condition: Invarianten gjelder

```
if (antall > 0) {  
    antall --;  
    return alle [antall];  
} else return null;
```

Post-condition: Invarianten gjelder

Navn: antall

3

Type: int

Kanin [] alle

0

1

2

3

MAXANT - 1

Invariant:

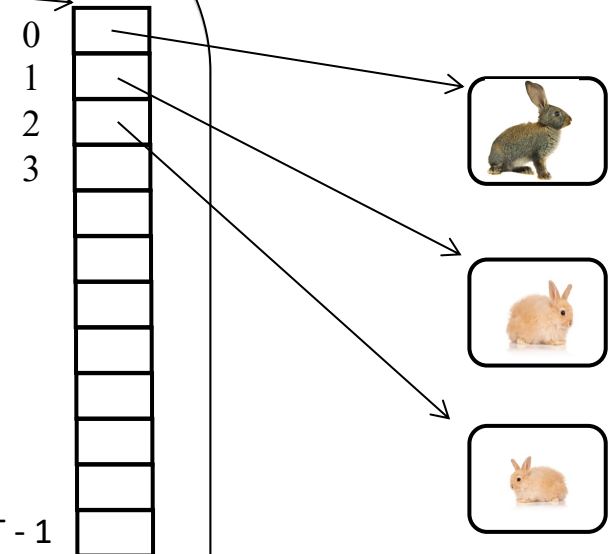
$0 \leq \text{antall} \leq \text{MAXANT}$ og
alle kaninene er lagret fom.
alle[0] tom. alle[antall-1]

```
public Kaningard ()
```

Pre-condition: true

```
antall = 0;
```

Post-condition: Invarianten gjelder



Invarianten holder i det objektet opprettes **OG** alle metodene bevarer invarianten -> Da holder invarianten alltid

```
public void settInn (Kanin en)
```

Pre-condition: Invarianten gjelder

```
if (antall < MAXANT) {  
    alle [antall] = en;  
    antall ++;  
}
```

Post-condition: Invarianten gjelder

```
public Kanin taUt()
```

Pre-condition: Invarianten gjelder

```
if (antall > 0) {  
    antall --;  
    return alle [antall];  
} else return null;
```

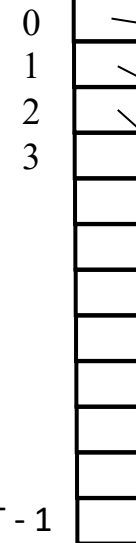
Post-condition: Invarianten gjelder

Navn: antall

3

Type: int

Kanin [] alle



Invariant:
 $0 \leq \text{antall} \leq \text{MAXANT}$ og
alle kaninene er lagret fom.
alle[0] tom. alle[antall-1]

```
public Kaningard ()
```

Pre-condition: true

```
antall = 0;
```

Post-condition: Invarianten gjelder



Java har et eget nøkkelord: assert

1. Aktiveres under kjøring med opsjonen -ea
2. Vil stoppe og avbryte programmet hvis regnes ut til false
3. null-pekere og andre feil bør heller sjekkes med Exceptions (unchecked), f.eks. IllegalArgumentException og NullPointerException
4. Assert kan brukes i kode du vet ((tror)) aldri blir utført.

Invarianter på data i løkker

Eksempel: **Finne minste verdi i en tabell**

```
// Vi vet ingenting annet enn at tabell [0] til og med  
// tabell[999] inneholder tall. Vi skal finne det minste
```

```
int minstTilNaa = tabell[0];
```

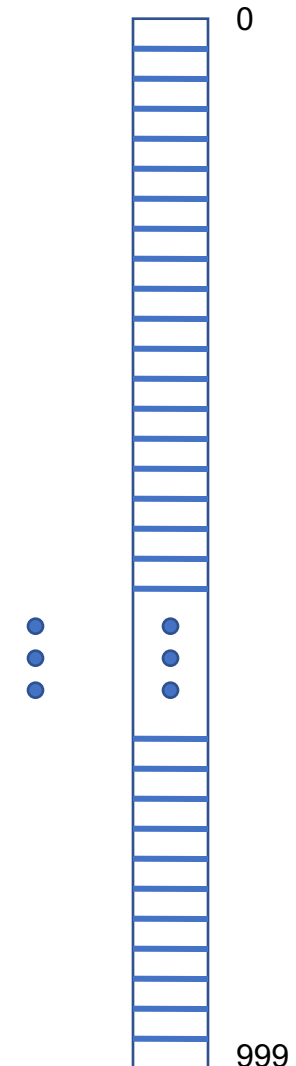
```
for (indeks = 1; indeks < 1000; indeks ++ ) {
```

```
    if (minstTilNaa > tabell [indeks] ) minstTilNaa = tabell[indeks];
```

```
}
```

```
// minstTilNaa inneholder minste verdi i området  
// fra og med tabell[0] til og med tabell[999]
```


minstTilNaa



Vi vet ingenting annet enn at tabell [0] til og med tabell[999] inneholder tall. Vi skal finne det minste

minstTilNaa inneholder minste verdi i området fra og med tabell [0] til og med tabell[indeks]

= Inv(indeks):

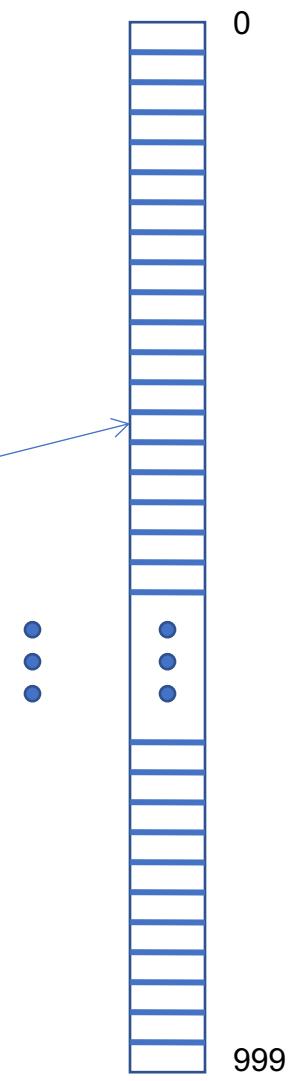
løkke



Induksjons-basis: indeks = 0
Induksjons-skritt: indeks = indeks + 1

minstTilNaa

indeks



Til slutt: indeks == 999:

minstTilNaa inneholder minste verdi i området fra og med tabell [0] til og med tabell[999]

Invarianter på data i løkker

Eksempel: Finne minste verdi i tabell

```
// Vi vet ingenting annet enn at tabell [0] til og med  
// tabell[999] inneholder tall. Vi skal finne det minste
```

```
int minstTilNaa = tabell[0];
```

```
// minstTilNaa inneholder minste verdi i området  
// fra og med tabell [0] til og med tabell[0]
```

```
for (indeks = 1; indeks < 1000; indeks ++ ) {
```

```
// minstTilNaa inneholder minste verdi i området  
// fra og med tabell [0] til og med tabell[indeks-1]
```

```
if (minstTilNaa > tabell [indeks] ) minstTilNaa = tabell[indeks]
```

```
// minst TilNaa inneholder minste verdi i området  
// fra og med tabell [0] til og med tabell[indeks]
```

```
}
```

```
// Nå er indeks == 1000
```

```
// minstTilNaa inneholder minste verdi i området  
// fra og med tabell [0] til og med tabell[indeks-1]
```

```
// Da følger:
```

```
// minstTilNaa inneholder minste verdi i området  
// fra og med tabell[0] til og med tabell[999] !!!!!
```


minstTilNaa

Inv(0)

Induksjons-basis

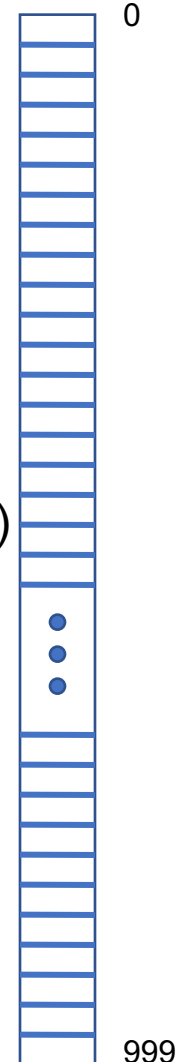
Inv(indeks-1)

Inv(indeks)

Hvis Inv(indeks) er sant og
vi utfører: indeks ++ så er
Inv(indeks-1) sant etterpå!

Induksjons-skritt

resultat



Invarianter på data i løkker

Eksempel: Finne minste verdi i tabell

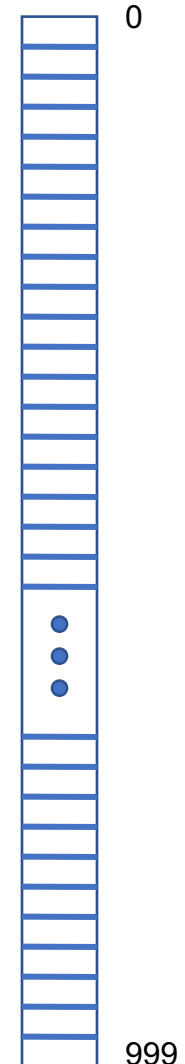

minstTilNaa

// Vi vet ingenting annet enn at tabell [0] til og med
// tabell[999] inneholder tall. Vi skal finne det minste

For-betingelse. Engelsk: Pre-condition

**Bak-betingelse.
Engelsk: Post-condition (Post-assertion)**

// minstTilNaa inneholder minste verdi i området
// fra og med tabell[0] til og med tabell[999]



Pre-condition

Post-condition (post assertion)

- Det er helt vanlig at parameterene til en metode må ha visse verdier
 - Hvis ikke vil ikke metoden gjøre jobben sin riktig
 - Dette kalles metodens «pre-condition»
 - «Forbetingelse» på norsk
- Når en metode er kalt med riktig forbetingelse vil tilstanden når metoden terminerer kunne beskrives av metodens «bak-betingelse»
 - Post assertion (post-condition)

Spesifikasjon av objekters oppførsel (semantikk) ved hjelp av en sekvens av metodekall:

Eksempel: Hvis et stakk-objektet har hatt følgende
sekvens av kall:



settInn(7), settInn(4), taUt(), settInn(8), settInn(2), settInn(9), taUt()

Hva returnerer nå neste kall på taUt() ?

Hva returnerer neste kall på taUt() om dette var en FIFO-kø ?



OK, ikke akkurat eksamensrelevant



Hvilke oppgaver får vi som programmerere og systemutviklere når ChatGPT kan programmere?

Vanskelig å vite men kanskje (Steins synspunkter mai 2023):

- ChatGPT kan gi oss ideer og forslag
 - Men du er ansvarlig og må sjekke
 - Du kan ikke bare godta det som ChatGPT foreslår – ChatGPT lyver
- Derfor: Du må mer enn noen gang være en god programmerer
 - Du blir bare en god programmerer ved å trene og trene
- ChatGPT/AI kan få “oversikt” over store dokumentmengder og spesifikasjoner
 - ChatGPT kan “snakke” med kundene og deg
 - Hjelp til konsistenssjekk.
 - Hjelp til å verifisere koden din? Hjelp til testing ?
- Og har du en god idé til en ny algoritme slår du ganske sikkert ChatGPT
- Industrielle revolusjon / innføring av data i banker: VI BLIR ARBEIDSLEDIGE 😞
 - NEI: Vi får mer ansvarlige og interessante jobber



Hvis tid: eksempler fra ChatGPT

chat.openai.com



I IN1010 ønsker vi å lære å lage gode og robuste programmer !!

Men, allikevel (fordi det å lage robuste programmer kan ta mye tid):

Eksamenstips: Lag programmet akkurat så robust som det bes om

Eksamenstips: Ikke legg inn mer robusthet / funksjonalitet enn det bes om

Eksamenstips (Alle mors -): Les og forstå oppgaven og **SVAR PÅ OPPGAVEN !**



OG SVAR BARE PÅ OPPGAVEN !!

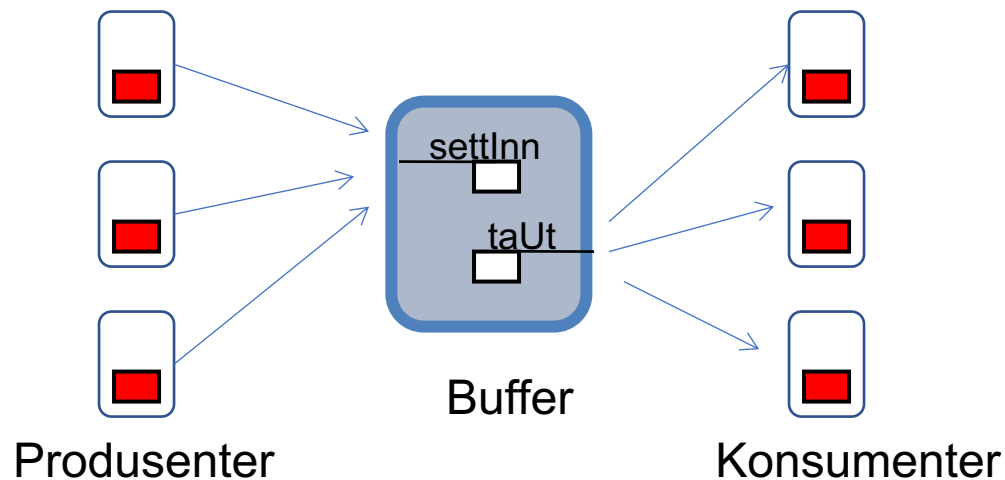
Steins viktigste eksamenstips:

Tegn opp datastrukturen / objektene

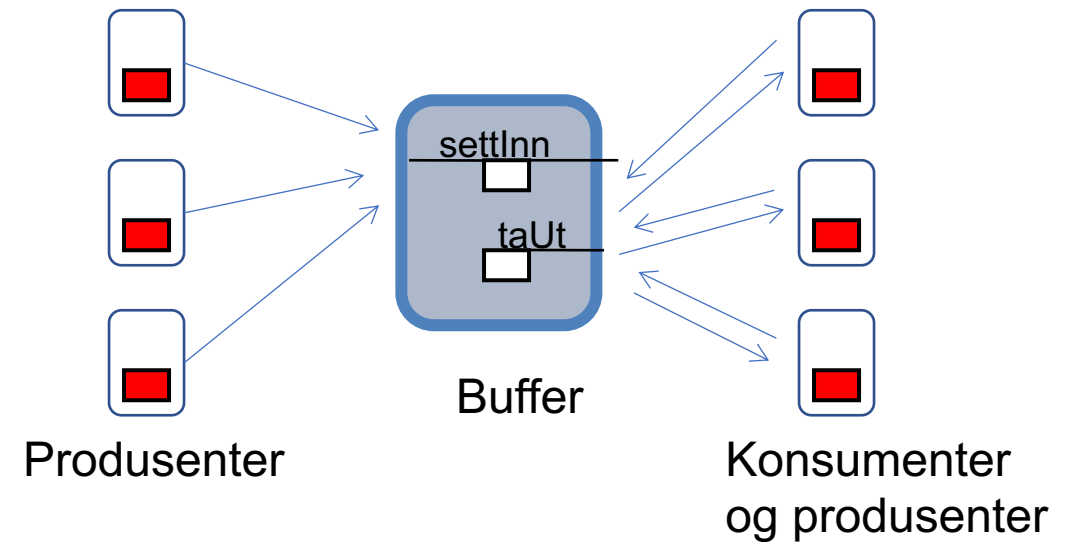
Skjønne hva programmet skal gjøre

f.eks.

Forelesning:



Oblig 5:





I dag har du lært

- En god programmerer tenker i betingelser
 - for-betingelser og bak-betingelser (pre-conditions og post-conditions)
- En god programmerer må tenke i invarianter
 - Data i objekter
 - Løkker
 - Java har en egen konstruksjon: assert (som ikke brukes)
 - Datastrukturtegninger er også en beskrivelse av tilstanden til programmet (på et gitt sted i programutførelsen)
- ChatGPT kan hjelpe deg som programmerer, men du må skjønne hva den foreslår
 - Eneste måte å virkelig skjønne programmer er å KUNNE PROGRAMMERE SELV
 - Ikke tro at det er enkelt å lære seg å programmere – TRENING – TRENING - TRENING
- Til eksamen må du **tenke og resonere** og lage riktige programmer med en gang
 - Du får ikke testet programmene dine