

Programmering i C

Lektion 1

8. september 2009

Målgruppe

Motivation

Indhold

Form

Materiale

Kursusintroduktion

- 1 Målgruppe
- 2 Motivation
- 3 Indhold
- 4 Form
- 5 Materiale

- Folk der har styr på programmering, og som har programmeret i C før
- Folk der har styr på programmering
- Folk der aldrig har programmeret før

- Folk der har styr på programmering, og som har programmeret i C før
- Folk der har styr på programmering
- Folk der aldrig har programmeret før

- Nødvendigt i resten af studiet (for nogen)
- Kan anvendes i projektet
- Spændende

5 / 47

- 1 Introduktion og Kontrolstrukturer (i dag)
- 2 Funktioner (Mandag 14/9)
- 3 Datatyper (Tirsdag 15/9)
- 4 Pointers (Fredag 18/9)
- 5 Opsummering (Mandag 28/9)

6 / 47

I dag:

8 : 45 til 10 : 00	Forelæsning (med pauser)
10 : 15 til 12 : 00	Opgave regning

Resten af gangene:

8 : 15 til 10 : 00	Opgave regning
10 : 15 til 12 : 00	Forelæsning (med pauser)

7/47

- **C Language Tutorial**

[http:](http://www.cprogramming.com/tutorial/c/lesson1.html)

[//www.cprogramming.com/tutorial/c/lesson1.html](http://www.cprogramming.com/tutorial/c/lesson1.html)

- **Noter til et tidligere kursus om programmering i C**

<http://www.cs.aau.dk/~normark/c-prog-06/html/notes/theme-index.html>

8/47

Introduktion

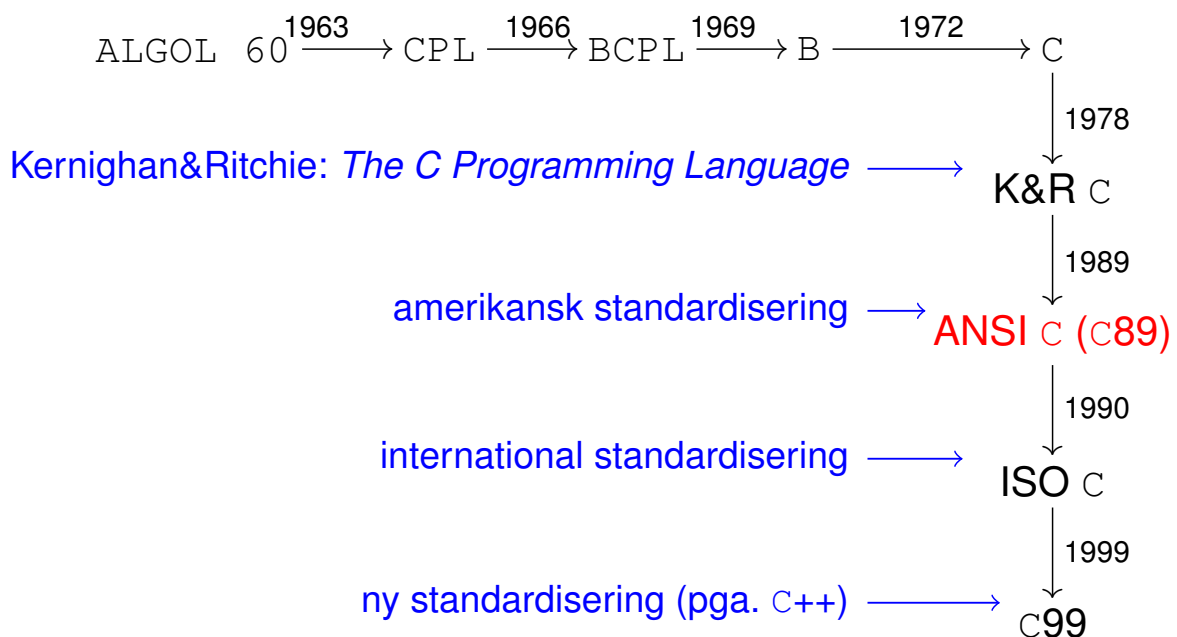
- 6 IDE - Integrated Development Environment
- 7 Historie
- 8 Programmer
- 9 Variable
- 10 Datatyper
- 11 Kontrol strukturer
- 12 Udtryk
- 13 Assignments
- 14 Operatorer
- 15 I/O
- 16 Eksempel
- 17 Processen
- 18 At compilere

9/47

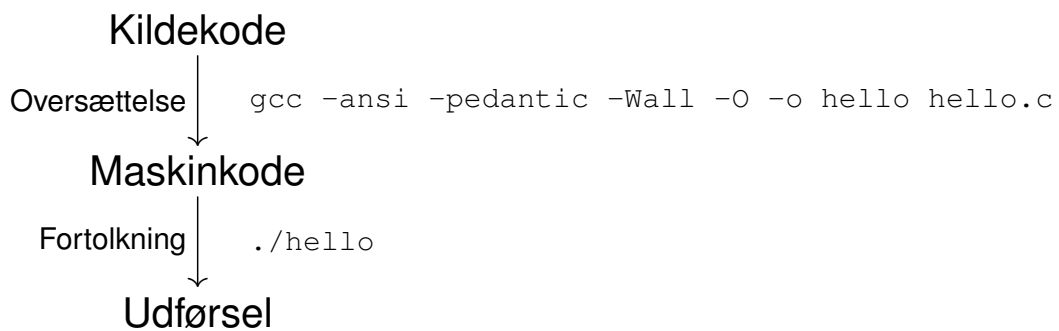
- Dagens vigtigste formål: At I skriver og kompilerer jeres første program
- Dagens første opgave: At installere CodeBlocks (lille demo)
- Næste gang: Øvelser i kontrolstrukturer

- Hvorfor C ?
- Ligner mange andre programmeringssprog
 - et lavniveau imperativt programmeringssprog
 - (imperativ vs. funktionel vs. objektorienteret (vs. ...))
 - tæt knyttet til operativsystemet UNIX
 - udbredt sprog til systemprogrammering

11 / 47



12 / 47



```
#include <stdio.h>
```

```
int main( void ) { /* helloworld.c */
    printf( "Hello , world!\n" );
    return 0;
}
```

13/47

- en **variabel** er en navngiven plads i computerens lager
- en variabel kan indeholde en værdi af en bestemt type
- variables værdier kan ændres ved **assignment**-kommandoer
- variable skal **erklæres** før brug

```
#include <stdio.h>
```

```
int main(void) { /* variable.c */
    int a, b, c;
    a = 5;
    b = 3;
    c = a / b;
    printf( "%d divideret med %d giver %d\n",
           a, b, c );
    printf( "Hov, hvad er nu det?\n" );
    return 0;
}
```

14/47

- en **variabel** er en navngiven plads i computerens lager
- en variabel kan indeholde en værdi af en bestemt type
- variables værdier kan ændres ved **assignment**-kommandoer
- variable skal **erklæres** før brug

```
#include <stdio.h>
```

```
int main(void) { /* variable2.c */
  int a = 5, b = 3, c;
  c = a / b;
  printf("%d divideret med %d giver %d\n",
        a, b, c);
  printf("Hov, hvad er nu det?\n");
  return 0;
}
```

15/47

- en **variabel** er en navngiven plads i computerens lager
- en variabel kan indeholde en værdi af en bestemt type
- variables værdier kan ændres ved **assignment**-kommandoer
- variable skal **erklæres** før brug
- variable skal **altid** tildeles startværdier

```
#include <stdio.h>
```

```
int main(void) { /* variable-noinit.c */
  int a, b, c;
  c = a / b;
  printf("%d divideret med %d giver %d\n",
        a, b, c);
  printf("Hov, hvad er nu det?\n");
  return 0;
}
```

16/47

heltal	reelle tal	tegn	strenge
short	float	char	char *
int	double		
long	long double		

```
#include <stdio.h>
```

```
int main(void) { /* variable-float.c */
  int a = 5, b = 3;
  double c;
  c = (double)a/ b;
  printf("%d divideret med %d giver %f\n",
        a, b, c);
  printf("Det var bedre!\n");
  return 0;
}
```

17/47

```
#include <stdio.h>
```

```
int main(void) { /* elefant.c */
  int a = 1;
  printf("%d elefant kom marcherende, \
hen ad edderkoppens fine spind\n", a);

  while(a <= 10) {
    a = a + 1;
    printf("%d elefanter kom marcherende, \
hen ad edderkoppens fine spind\n", a);
  }

  return 0;
}
```

18/47

Udtryk:

- 7
- x, a, b
- a + b, a - b rest ved (heltals)division
↓
- a * b, a / b, a % b
- a < b, a <= b, a == b etc. (boolske udtryk)

assignment/tildeling

c = a / b

expression/udtryk

Prioritering: * beregnes før + etc.:

$$3 + 5 * 7 = 3 + (5 * 7)$$

Associering: Operationer med samme prioritet foretages fra venstre til højre:

$$10 - 5 - 2 = (10 - 5) - 2 \neq 10 - (5 - 2)$$

19/47

- **a = i + 5:** udtrykket i + 5 beregnes, og a tildeles den beregnede værdi
- dvs. + har højere prioritet end =
- men i C er **a = i + 5** også et **udtryk!** Udtrykkets værdi er ligeledes i + 5

⇒ misbrug:

#include <stdio.h>

```
int main(void) { /* misbrug.c */
    int a, b, c;
    a = b = c = 7;
    printf("a: %d, b: %d, c: %d\n", a, b, c);
    a = 1 + (b = 2*(c = 3));
    printf("a: %d, b: %d, c: %d\n", a, b, c);
    return 0;
}
```

20/47

- increment-operator: skriv $i++$ eller $++i$ i stedet for $i = i + 1$
- decrement-operator: skriv $i--$ eller $--i$ i stedet for $i = i - 1$
- **men** det er også et udtryk ... :
 - $i = 7; a = ++i \Rightarrow i=8, a=8$
 - $i = 7; a = i++ \Rightarrow i=8, a=7 !$ Hvorfor?
- også **akkumulerende assignment-operatorer**:

$a += 5$	$a = a + 5$
$a -= 7$	$a = a - 7$
$a *= 4$	$a = a * 4$
$a /= 3$	$a = a / 3$ etc.

21 / 47

Udskrivning med printf :

- `printf(kontrolstreng, parametre)`
- kontrolstreng: almindelige tegn udskrives uændret, **konverteringstegn** erstattes med parametre, som er formateret i h.t. konverteringsspecifikationen
- `printf` returnerer antallet af udskrevne tegn
- se `printf-eks.c`

22 / 47

Udskrivning med printf :

- printf(*kontrolstreng*, *parametre*)
- kontrolstreng: almindelige tegn udskrives uændret, **konverteringstegn** erstattes med parametre, som er formateret i h.t. konverteringsspecifikationen
- printf returnerer antallet af udskrevne tegn
- se `printf-eks.c`

Indlæsning med scanf:

- scanf(*kontrolstreng*, *parametre*)
- kontrolstreng (næsten) analog til printf, men parametrene skal være **adresser** på variable (**pointere**): `&a`
- scanf returnerer antallet af gennemførte indlæsninger
- se `scanf-eks.c`

23/47

Et større eksempel:

```
#include <stdio.h>
```

```
#define PI 3.141592653589793
```

```
int main( void ) { /* circle.c */
    double radius;
```

```
    printf( "\n%s\n\n%s",
            "This program computes the area of a circle.",
            "Input the radius: ");
```

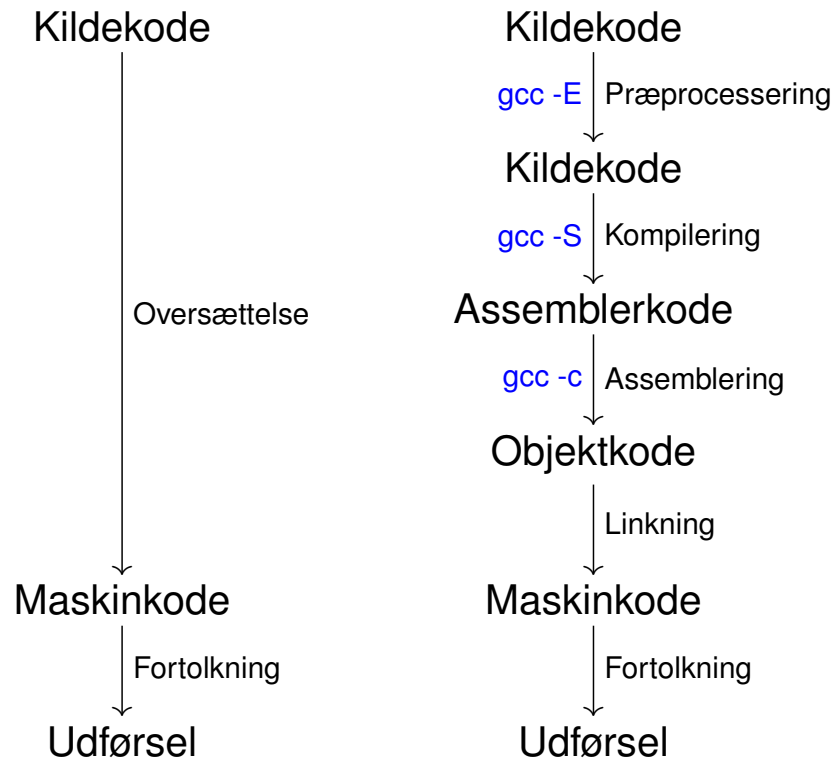
```
    scanf( "%lf ", &radius );
```

```
    printf( "\n%s\n%s%.2f%s%.2f%s%.2f\n%s%.5f\n\n",
            "Area = PI * radius * radius",
            "      = ", PI, " * ", radius, " * ", radius,
            "      = ", PI * radius * radius );
```

```
    return 0;
```

1

24/47



gcc -ansi -pedantic -Wall -O -o ting ting.c

↑ Vi følger ANSI-standarden

↑ Vi vil have alle advarsler som ANSI kræver

↑ Vi vil også have en masse andre advarsler

↑ Vi optimerer for at fange flere mulige fejl

↑ Output skal hedde ting

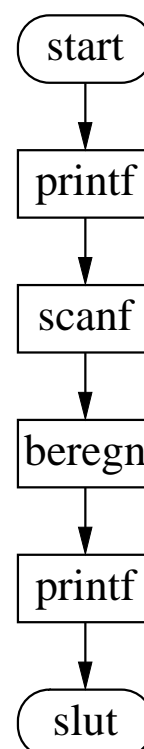
Kontrolstrukturer

- 19 Sekventiel kontrol
- 20 Logiske udtryk
- 21 Short circuit evaluering
- 22 Udvælgelse af kommandoer

27 / 47

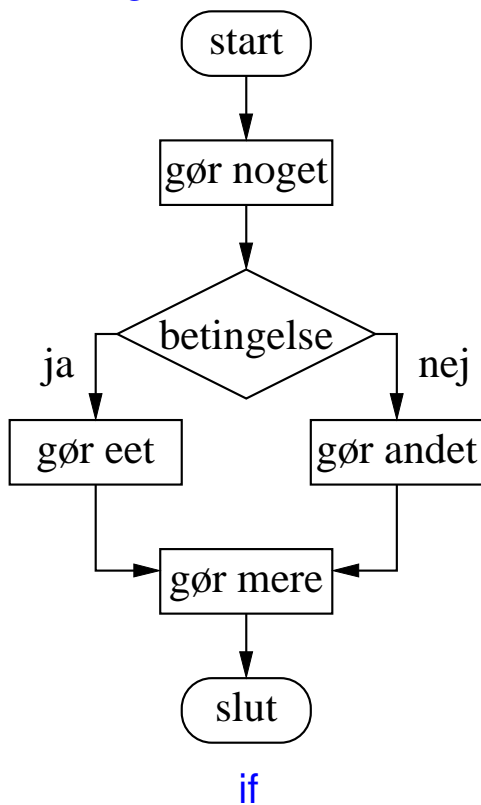
```
#include <stdio.h>
```

```
int main( void ) { /* seconds.c */  
    long int input, temp, h, m, s;  
    printf( "Giv mig et heltal!\n" );  
    scanf( "%ld", &input );  
    h= input/ 3600;  
    temp= input- h* 3600;  
    m= temp/ 60;  
    s= temp% 60;  
    printf( "\n%ld sekunder svarer til \  
%ld timer, %ld minutter og %ld sekunder\n",  
           input, h, m, s );  
    return 0;  
}
```

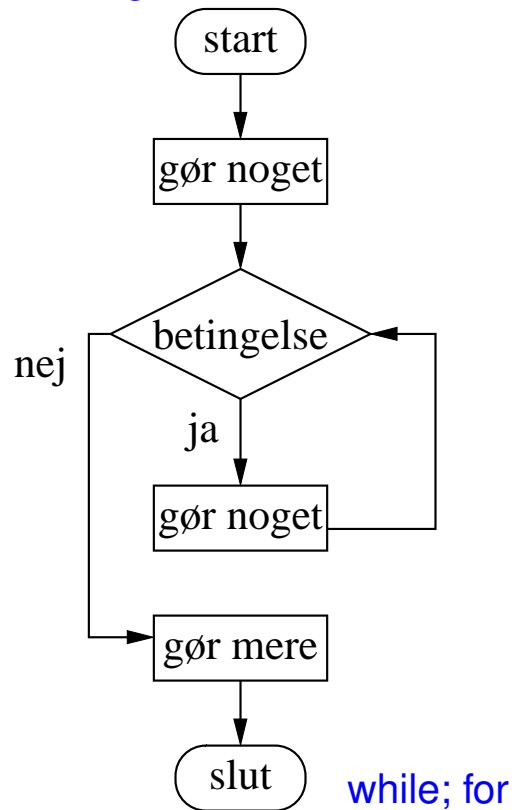


28 / 47

Udvælgelse af kommandoer:



Gentagelse af kommandoer:



29 / 47

Udvælgelse: **if**(*logisk udtryk*)Gentagelse: **while**(*logisk udtryk*)

Logiske udtryk:

- $x < y$, $x \leq y$, $x \geq y$, $x > y$, $x \neq y$, $x == y$
- $!A$, $A \& \& B$, $A \parallel B$, hvor A og B selv er logiske udtryk
- har værdien *falsk* (0) eller *sandt* (1, i de fleste(!) compilere)
- $\& \&$ har højere prioritet end \parallel

⇒ brug parenteser!

(Hvad er værdien af $3 == 5 \parallel 1 == 1 \& \& 1 == 2 ? \dots$)[\[oper.c\]](#)

```
# include <stdio.h>

int main( void) { /* lighed.c */
    int a, b, lig;

    printf( "Vi sammenligner to tal.\n\
Output 0 betyder at de er forskellige.\n\n\
Må jeg bede om to heltal?\n");
    scanf( "%d %d", &a, &b);

    lig = (a == b);

    printf( "\nOutput: %d\n", lig);
    return 0;
}
```

31 / 47

Observation:

- Hvis **A** er falsk, da er **A&&B** også falsk
 - Hvis **A** er sandt, da er **A||B** også sandt
- ⇒ i udtrykket **A&&B** beregnes **B** kun hvis **A** er sandt
– og i udtrykket **A||B** beregnes **B** kun hvis **A** er falsk
- Smart, men kilde til fejl

32 / 47


```
# include <stdio.h>
```

```
int main(void) { /* lighed2.c */  
    int a, b;  
    char lig;  
  
    printf("Vi sammenligner to tal.\n\n\  
Må jeg bede om to heltal?\n");  
    scanf( "%d %d", &a, &b);  
  
    (a == b) && (lig = ' ');  
    (a != b) && (lig = 'u');  
  
    printf("%d er %clig %d\n", a, lig , b);  
    return 0;  
}
```

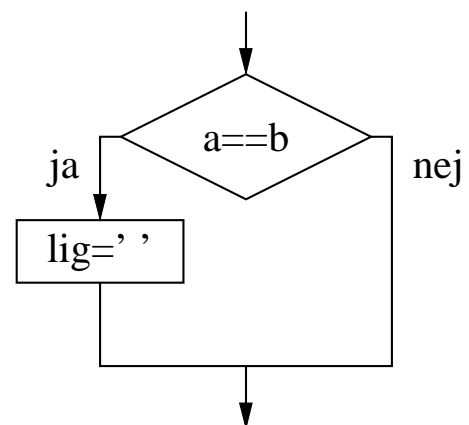
33/47

Udvælgelse med &&:

- `(a == b) && (lig = ' ');`
- kryptisk...

Udvælgelse med if:

- `if (a== b) lig = ' ';`
- det var bedre!



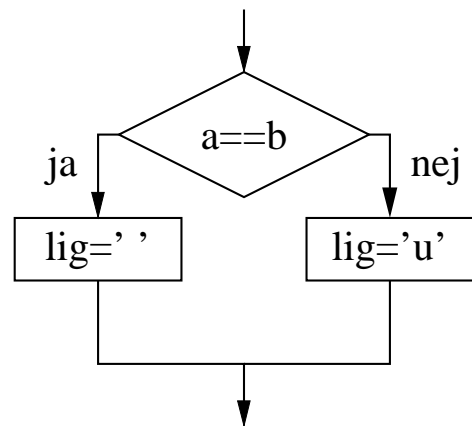
34/47

Udvælgelse med &&:

- `(a == b) && (lig = ' ');`
- kryptisk...

Udvælgelse med if:

- `if (a== b) lig = ' ';`
- det var bedre!
- `if (a==b) lig = ' ';`
`else lig = 'u';`



if (udtryk) kommando1; else kommando2;

- først beregnes **udtryk**
- hvis **udtryk** er sandt, udføres **kommando1**
- hvis **udtryk** er falsk, udføres **kommando2**

35/47

include <stdio.h>

```

int main(void) { /* lighed3.c */
    int a, b;
    char lig;

    printf("Vi sammenligner to tal.\n\n\
Må jeg bede om to heltal?\n");
    scanf( "%d %d", &a, &b);

    if (a== b) {
        lig = ' ';
    } else {
        lig= 'u';
    }
    printf("%d er %clig %d\n", a, lig , b);
    return 0;
}
  
```

36/47

Kontrolstrukturer, 2.

- 23 Kommandoblokke; scope
- 24 Udvælgelse med if, 2.
- 25 Udvælgelse med switch
- 26 Gentagelse med while
- 27 Gentagelse med for
- 28 Opsummering

37 / 47

Problem: Vil gerne udvælge mellem to blokke af kommandoer

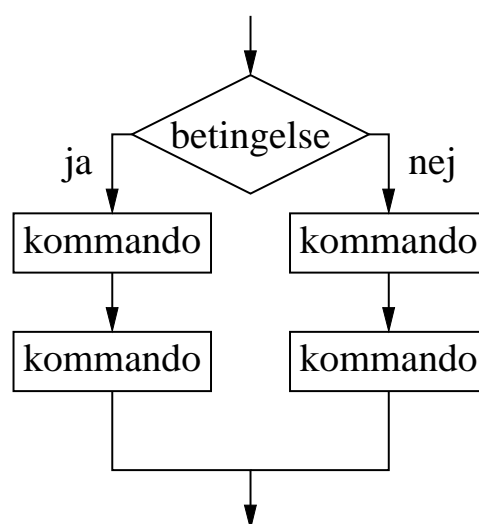
Løsning: **Sammensætning af kommandoer:**

```

if ( a== b)
{
    c= 1;
    d= 2;
}
else
{
    c= 7;
    d= 5;
}
    
```

} blok

} blok



38 / 47

- blok = antal kommandoer omkranset af { og }
- en blok behandles som én kommando
- blokke kan indlejres i hinanden

39 / 47

- blok = antal kommandoer omkranset af { og }
- en blok behandles som én kommando
- blokke kan indlejres i hinanden
- i starten af en blok kan variabelerklæringer forekomme
- !! disse variable er lokale for blokken (deres scope er blokken)

40 / 47

- blok = antal kommandoer omkranset af { og }
- en blok behandles som én kommando
- blokke kan indlejres i hinanden
- i starten af en blok kan variablerklæringer forekomme
- !! disse variable er lokale for blokken (deres scope er blokken)

```
#include <stdio.h>
int main(void){ /* blok.c */
    int a=5;
    printf("Før: a==%d\n",a);

    { /* en blok */
        int a=7; /* deklARATION */
        printf("I: a==%d\n",a);
    }

    printf("Efter: a==%d\n",a);

    return 0;
}
```

```
#include <stdio.h>
int main(void){ /* blok2.c */
    int a=5;
    printf("Før: a==%d\n",a);

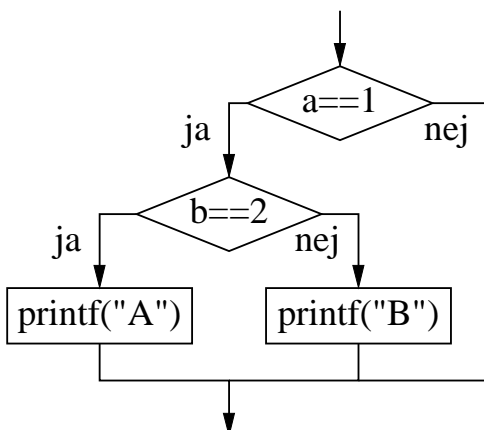
    { /* en blok */
        a=7; /* assignment! */
        printf("I: a==%d\n",a);
    }

    printf("Efter: a==%d\n",a);

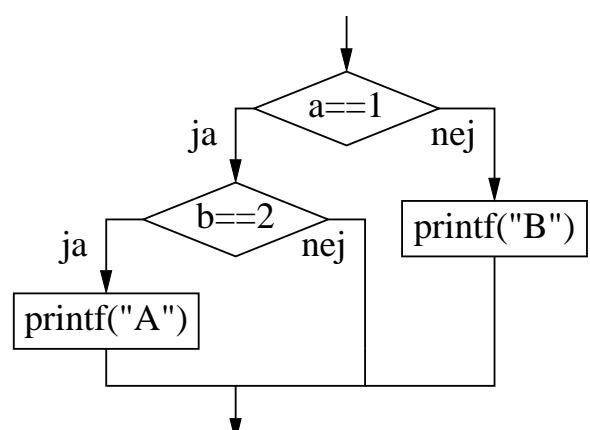
    return 0;
}
```

“Dangling else”-problemet:

```
if (a == 1)
    if (b == 2)
        printf("A");
else
    printf("B");
```



```
if (a == 1) {
    if (b == 2) {
        printf("A");
    }
} else {
    printf("B");
}
```



- en else knytter sig altid til den inderste if
- brug kommandoblokke for at undgå tvivl!

Hvad hvis der er flere end to valgmuligheder? Brug **switch** !

```
#include <stdio.h>
```

```
int main(void) { /* switch.c */
    int a;
    char * dyr;
    printf("Giv mig et heltal!\n");
    scanf("%d", &a);

    switch (a) {
    case 1: dyr= "hest"; break;
    case 2: dyr= "gris"; break;
    case 3: dyr= "abe"; break;
    default: dyr= "ko"; break;
    }

    printf("\nDu er en %s!\n", dyr);
    return 0;
}
```

43/47

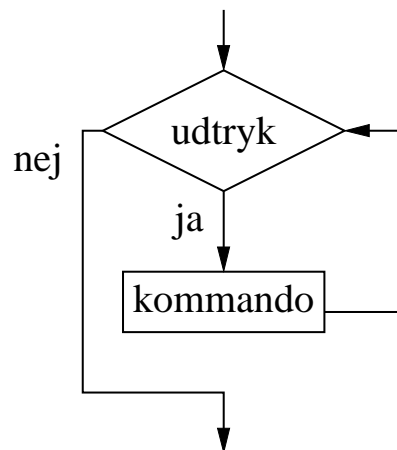
```
switch (udtryk) {
case const1: command1;
case const2: command1;
    ...
case constN: commandN;
default: command;
}
```

- først beregnes **udtryk**. Resultatet skal være et heltal eller noget der ligner (f.x. en `char`)
- **udtryk** == **const_i** ⇒ **command_i** udføres. Herefter udføres **command_{i+1}** osv.
- **udtryk** != **const_i** for alle *i* ⇒ default-kommandoen udføres, og herefter de efterfølgende! Hvis der ingen **default** er, gøres ingenting.
- man ønsker næsten altid at afslutte et **case** med en **break**-kommando; så springes de efterfølgende kommandoer over.

44/47

while (udtryk) kommando;

- først beregnes **udtryk**
- hvis **udtryk** er sandt, udføres **kommando**, og løkken startes forfra
- hvis **udtryk** er falsk, afsluttes løkken



```

#include <stdio.h>
int main(void) { /* while.c */
    int h = 0;
    while (h != 1234) {
        printf("Indtast det hemmelige heltal: ");
        scanf("%d", &h);
    }
    printf("\nHurra!\n");
    return 0;
}
  
```

45/47

for (init; condition; update) kommando;

(den mest generelle løkkekonstruktion i C)

- 1 først udføres **init**
- 2 så beregnes **condition**, og hvis den er falsk, afbrydes
- 3 **kommando** udføres
- 4 **update** udføres, og vi springer tilbage til trin 2.

```

#include <stdio.h>
int main(void) { /* for.c */
    int i = 1;

    printf("%d elefant\n", i);
    for(i= 2; i<=10; i++) {
        printf("%d elefanter\n", i);
    }
    return 0;
}
  
```

46/47

- Øvelser i dag.
 - Installere Code Blocks
 - Skrive "Hello World" program
 - Andre simple programmer
- Øvelser næste mandag
 - Kontrolstrukturer

- Udvalgelse
 - If
 - Switch
- Gentagelse
 - While
 - For
- Struktur
 - Kommando blokke