

# Programmering i C

## Lektion 3

18. november 2008

Kontrolstrukturer

Udvælgelse

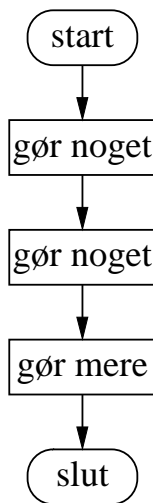
Gentagelse

Eksempler

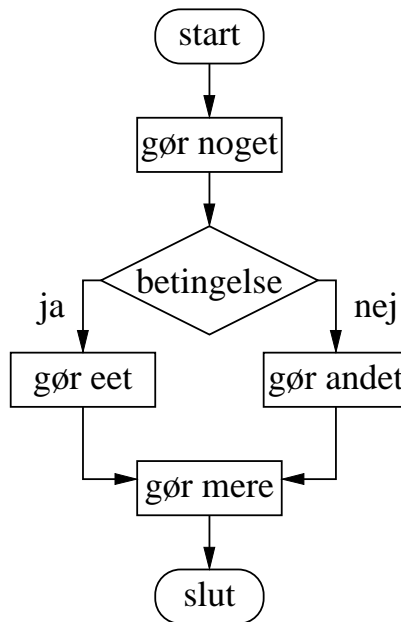
## Fra sidst

- 1 Kontrolstrukturer
- 2 Udvælgelse
- 3 Gentagelse
- 4 Eksempler

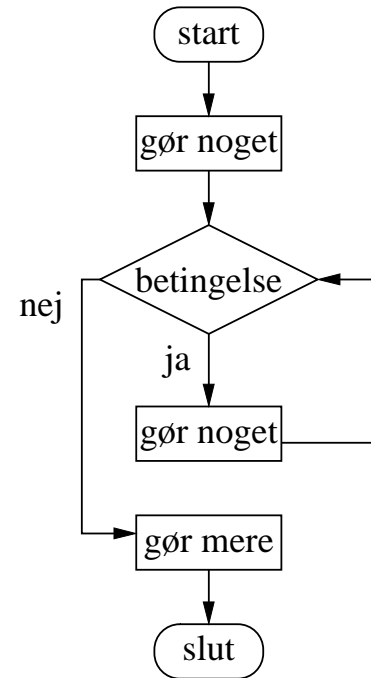
## Sekventiel kontrol



## Udvælgelse



## Gentagelse



3/23

- med **if**

```
if ( udtryk ) kommando1; else kommando2;
```

- med **switch**

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

- med **den betingede operator ?:**

```
udtryk ? udtryk1 : udtryk2
```

```
f.x. min=( a < b ? a : b );
```

(smart, men undgå!)

4/23

- med **while**  
`while( udtryk ) kommando;`
- med **for**  
`for( start; forts; update ) kommando;`
- med **do**  
`do kommando; while( udtryk )`  
f.x.  
`do scanf( "%c" , &ans );`  
`while( ans!= 'n' && ans!= 'y' );`

5/23

- Løsninger på Opgave 3 fra sidste gang.
  - med **while**: [gaet.c](#)
  - med **for** (måske lidt søgt ...): [gaet2.c](#)

6/23

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

int main( void) { /* gaet.c */
    int hental;
    int gaet= 0;
    int forsoeg= 0;

    printf( "\nWe generate a random number between 1 and 1000\n\
and let you guess it, at each step telling you\n\
the relation between your guess and our number.\n");

    /* initialise random number generator */
    srand(( unsigned int) time( 0));
    /* generate random number between 1 and 1000 */
    hental= rand()% 1000+ 1;

    while( gaet!= hental) {
        forsoeg++;
        printf( "\nEnter your guess: ");
        scanf( "%d", &gaet);

        if( gaet!= hental)
            printf( "Your guess is too %s.\n", gaet< hental? "small": "big");
        else
            printf( "\nSuccess!\nYou needed %d tries.\n", forsoeg);
    }
}
```

7/23

## Funktioner

- 5 Funktioner
- 6 Eksempel
- 7 Parametre
- 8 Rekursive funktioner
- 9 Parametre til main()

- at opdele et større program i mindre enheder  $\Rightarrow$  funktioner
- abstraktion!
- top-down-programmering

```
type navn( parametre ) {  
    deklARATIONER;  
    kommandoer;  
}
```

9/23

Et program der indlæser et tal; hvis tallet er primtal udskrives "PRIMA," ellers udskrives næste primtal:

```
#include <stdio.h>  
  
int main( void ) { /* prim.c */  
    int tal;  
  
    tal= indlaes(); /* et funktionskald */  
    if( prim( tal) ) /* et funktionskald */  
        printf( "PRIMA\n" );  
    else {  
        tal= nextPrime( tal); /* endnu et */  
        printf( "Next prime is %d\n", tal );  
    }  
  
    return 0;  
}
```

10/23

At indlæse et heltal:

```
/* en funktionsdefinition */
int indlaes( void) {
    int tal;

    printf( "\nEnter a number: ");
    scanf( "%d", &tal);

    return tal;
}
```

11/23

Find ud af om et heltal er et primtal (*Er det den bedste måde at gøre det på?*):

```
int prim( int tal) {
    int isprime= 1;
    int i;

    for( i= 2; i<= tal- 1; i++) {
        if( tal% i== 0) {
            isprime= 0;
            break;
        }
    }

    return isprime;
}
```

**break:** Springer ud af en [switch](#), [while](#), [do](#) eller [for](#)

12/23

Returner næste primtal:

```
int nextPrime( int tal) {
    tal++;
    while( !prim( tal)) tal++;

    return tal;
}
```

Bemærk genbrug af [prim](#)-funktionen.

13/23

Funktioner skal erklæres før de bliver brugt:

```
#include <stdio.h>

int indlaes( void);
int prim( int tal);
int nextPrime( int tal);

int main( void) { /* prim.c */
    int tal;

    tal= indlaes(); /* et funktionskald */
    if( prim( tal)) /* et funktionskald */
        printf( "PRIMA\n");
    else {
        tal= nextPrime( tal); /* endnu et */
        printf( "Next prime is %d\n", tal);
    }

    return 0;
}
```

14/23

Hele programmet: [prim.c](#)

15/23

```
type navn( parametre ) {  
    deklARATIONER ;  
    kommandoer ;  
}
```

- En parameter i en funktions*definition* kaldes en **formel parameter**. En formel parameter er et variabelnavn.
- En parameter i et funktions*kald* kaldes en **aktuel parameter**. En aktuel parameter er et udtryk der beregnes ved funktionskaldet.

16/23



```
type navn( parametre ) {  
    deklARATIONER ;  
    kommandoer ;  
}
```

- Antallet og typer af aktuelle parametre i kaldet skal modsvare antallet og typer af formelle parametre i definitionen.

**definition:** `int days_per_month( int m, int y ) {`

**kald:** `dmax= days_per_month( m, y );`

17/23

```
type navn( parametre ) {  
    deklARATIONER ;  
    kommandoer ;  
}
```

- Antallet og typer af aktuelle parametre i kaldet skal modsvare antallet og typer af formelle parametre i definitionen.
- I C overføres funktionsparametre som **værdiparametre**. Dvs.
  - værdien af parametren *kopieres* til brug i funktionen,
  - ændringer af værdien har ingen indvirkning på programmet udenfor funktionen,
  - når funktionskaldet ender, ophører værdien med at eksistere.

18/23

```
type navn( parametre ) {
    deklARATIONER ;
    kommandoer ;
}
```

- Antallet og typer af aktuelle parametre i kaldet skal modsvare antallet og typer af formelle parametre i definitionen.
- I C overføres funktionsparametre som **værdiparametre**. Dvs.
  - værdien af parametren *kopieres* til brug i funktionen,
  - ændringer af værdien har ingen indvirkning på programmet udenfor funktionen,
  - når funktionskaldet ender, ophører værdien med at eksistere.
  - Dette kan “omgås” ved brug af pointers

19/23

**rekursiv funktion** = funktion der *kalder sig selv*

Eksempel: fakultetsfunktionen:  $n! = 1 \cdot 2 \cdot 3 \cdots n = n \cdot (n - 1)!$

```
unsigned long fakultet( unsigned long n ) {
    if( n== 1)
        return 1;
    else
        return n* fakultet( n-1);
}
```

[fak.c]

– smart og kompakt måde at kode på (men nogle gange ikke særlig hurtig afvikling)

20/23

Eksempel: Fibonaccital:

$$f_1 = 1 \quad f_2 = 1 \quad f_n = f_{n-1} + f_{n-2}$$

$$1, 1, 2, 3, 5, 8, 13, 21, 34, \dots$$

```
unsigned long fibo( int n) {
    switch( n) {
    case 1: case 2:
        return 1; break;
    default:
        return fibo( n- 1)+ fibo( n- 2);
    }
}
```

[\[fibonacci.c\]](#)

21 / 23

```
int main( void) { – en funktion!
```

Generel form: `int main( int argc , char** argv ) {`

Parametrene tages fra **kommandolinien**.

- `argc` er antallet af argumenter
- `argv` er et *array af strenge* med alle argumenter; `argv[0]` er programnavnet

Eksempel: `./argtest 15 hest`

[\[argtest.c\]](#)

```
⇒ argc == 3
   argv[0] == "argtest"
   argv[1] == "15"
   argv[2] == "hest"
```

Eksempel: Et faktultetsprogram der tager tallet som input på kommandolinien:

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

unsigned long fakultet( unsigned long n);

int main( int argc, char** argv) { /* fak2.c */
    char * myself= argv[0];
    unsigned long tal;
    char * endptr; /* needed for strtol */

    if( argc== 1)
        printf( "Error: %s needs one argument\n", myself);
    else { /* convert argv[1] to int */
        tal= strtol( argv[1], &endptr, 10);
        printf( "\nThe factorial of %lu is %lu\n", \
                tal, fakultet( tal));
    }
    return 0;
}
```