

Programmering i C

Lektion 1

30 oktober 2006

Kursusintroduktion

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



- Folk der har styr på programmering, og som har programmeret i \mathbb{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

- Introduktion
- Kontrolstrukturer
- Funktioner
- Tegn
- Datatyper

x til $x + 35\text{min}$	Forelæsning
$x + 45\text{min}$ til $x + 80\text{min}$	Forelæsning
$x + 90\text{min}$ til $x + 225\text{min}$	Opgaver

- **C Language Tutorial**

`http://einstein.drexel.edu/courses/CompPhys/General/C_basics/`

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

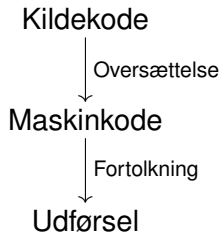
`http://www.cs.aau.dk/~normark/c-prog-06/html/notes/theme-index.html`

– også til selv læsning for dem der ikke følger forelæsningsne!

Introduktion

- 5 Historie; Generaliteter
- 6 Programmer
- 7 Variable
- 8 Datatyper
- 9 Elefanter
- 10 Udtryk
- 11 Assignments
- 12 Operatorer
- 13 I/O
- 14 Eksempel

- ALGOL 60 $\xrightarrow{1963}$ CPL $\xrightarrow{1966}$ BCPL $\xrightarrow{1969}$ B $\xrightarrow{1972}$ C
- Dennis Ritchie, Brian Kernighan
- et lavniveau imperativt programmeringssprog
- (imperativ vs. funktionel vs. objektorienteret (vs. ...))
- tæt knyttet til operativsystemet UNIX
- udbredt sprog til systemprogrammering



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

```
int main( void) {  
    printf( "Hello , world !\n" );  
    return 0;  
}
```

- 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) {  
    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;  
}
```

- 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) {  
    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;  
}
```

- 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) {  
    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;  
}
```

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

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

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

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

```
int main( void) {  
    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;  
}
```

Udtryk:

- 7
- x, a, b
- $a + b$, $a - b$
- $a * b$, a / b , $a \% b$
- $a < b$, $a \leq b$, $a == b$ etc. (boolske udtryk)

assignment
 $c = a / b$
 udtryk

rest ved (heltals)division
 ↓

Prioritering: * beregnes før + etc.:

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

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

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

- $a = i + 5$: 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) {  
    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;  
}
```

- 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 ... :
 - $i = 7; a = ++i \Rightarrow i=8, a=8$
 - $i = 7; a = i++ \Rightarrow i=8, a=7!$
- 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.

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`

Et større eksempel:

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

```
#define PI 3.141592653589793
```

```
int main( void ) {
```

```
    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;
```

```
}
```