

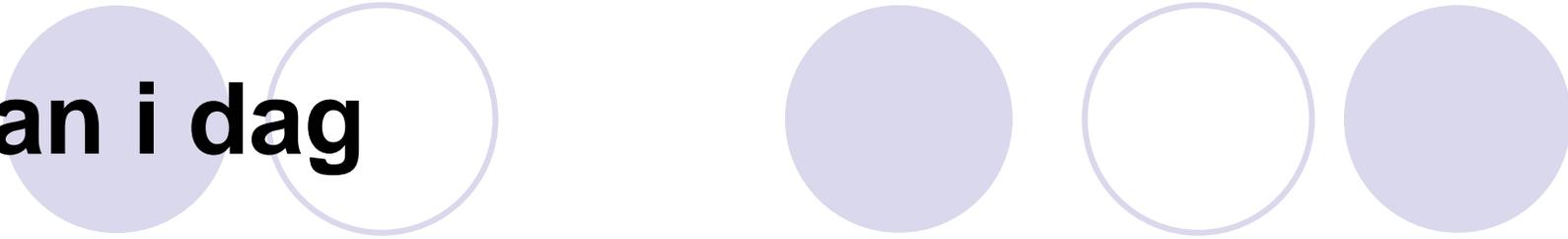
Programmering i C

Videre med C
(2 af 4)

19. marts 2007

Mads Pedersen, OZ6HR
mads@oz6hr.dk

Plan i dag



- Brush-up fra sidst
- Videre med C
 - Kontrolløkker (*while, for, ...*)
 - Conditional Execution (*if, if/else*)
 - Funktioner
- I princippet nok til de fleste programmer
- Slutter med en simpel lommeregner

Brush-up fra sidst

- Introduktion (historie, hvor bruges C)
- Værktøjet Dev-C++ (gratis og bl.a. på dansk)
- Kodeprocessen
(Kode → Kompiler → Link → Eksekver)
- Layout i en C-fil (*#include*, *main*, ...)
- Datatyper (*int*, *short*, *long*, *float*, *double*, *char*)
- Variabler og aritmetik
- Input og output (*scanf* og *printf*)

Kontrolløkker: *Loops* (1)

- Indtil nu sekventiel afvikling
 - Første linje, så næste linje, ...
 - Én efter én
- Uholdbart hvis et program skal køre "x antal mange gange", i flere timer eller "altid"

Kontrolløkker: *Loops* (2)

- Eksempel: Skriv "Hello World" 5 gange.
Nemt nok:

```
#include <stdio.h>
main()
{
    printf("Hello World\n");
    printf("Hello World\n");
    printf("Hello World\n");
    printf("Hello World\n");
    printf("Hello World\n");
}
```

- Hvad hvis vi nu bedes om 100 gange, 1.000 eller 10.000 gange "Hello World"?
 - Ikke længere en smart måde!

Kontrolløkker: *Loops* (3)

- Løkken *while* indføres nu:

```
while (condition)
{
    statements;
}

... eller ...
do
{
    statements;
} while (condition)
```

- F.eks. med logiske expressions:

```
#include <stdio.h>
main()
{
    int i = 0;

    while (i < 20)          /* true så længe i < 20 */
    {
        printf("Hello World\n");
        i = i + 1;         /* eller i++; */
    }
}
```

Man kan bruge:

- < Mindre end
- > Større end
- <= Mindre end eller lig med
- >= Større end eller lig med
- == Lig med
- != Ikke lig med

Kontrolløkker: *Loops* (4)

- Uendelig løkke

```
#include <stdio.h>
main()
{
    while (1 == 1)          /* Altid true */
    {
        printf("Hello World\n");
    }
}
```

Kontrolløkker: *Loops* (5)

- Ofte anvendt konstruktion med en *while*-løkke:

```
counter = start_value;
while (counter <= finish_value)
{
    statements;
    counter++;
}
```

- Dette er en *for*-løkke:

```
for (counter = start_value; counter <= finish_value; counter++ )
{
    statements;
}
```

- Enklere konstruktion

- Inkrementeringsvariablen *counter* indlejres

Kontrolløkker: *Loops* (6)

Resultat:

0	-17.8
20	-6.7
40	4.4
60	15.6
80	26.7
100	37.8
120	48.9
140	60.0
160	71.1
180	82.2
200	93.3
220	104.4
240	115.6
260	126.7
280	137.8
300	148.9

- Eksempel på for-løkke:

$$^{\circ}C = \frac{5}{9} \cdot (^{\circ}F - 32)$$

```
#include <stdio.h>

main()
{
    int fahr;

    for (fahr = 0; fahr <= 300; fahr = fahr + 20)
        printf("%4d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
}
```

```
#include <stdio.h>

main()
{
    float fahr, celsius;

    fahr = 0;

    while ( fahr <= 300 )
    {
        celsius = (5.0/9.0) * (fahr-32.0);
        printf("%4.0f %6.1f\n", fahr, celsius);
        fahr = fahr + 20;
    }
}
```

Conditional Execution: *if*, *else* (1)

- Vi har brug for at kunne indføre en mekanisme, der kun udføres, hvis bestemte forhold er tilstede.
- Dette hedder i programmeringssprog *if*:

```
if (condition)
{
    statement; /* Udføres kun hvis condition er opfyldt => true */
}
```

- F.eks.:

```
if (total > 0)
{
    printf("OK");
}
```

Conditional Execution: *if*, *else* (2)

- Eksempel med både *if* og *while*:

```
#include <stdio.h>

main()
{
    int a, b;

    while (a < 999)
    {
        printf("\nEnter first number: ");
        scanf("%d", &a);

        printf("\nEnter second number: ");
        scanf("%d", &b);

        if (a<b)
            printf("\nFirst number is less than second\n\n");
        if (b<a)
            printf("\nSecond number is less than first\n\n");
    }
}
```

Conditional Execution: *if, else* (3)

- Udfør hvis *condition* opfyldt, ellers gør noget andet: *If/else*

```
if (condition)
{
    statement1;
}
else
{
    statement2;
}
```

```
if (condition1)
{
    statement1;
}
else if (condition2)
{
    statement2;
}
else
{
    statement3;
}
```

Conditional Execution: *if*, *else* (4)

- Logical Expressions (igen)

Man kan bruge:

- < Mindre end
- > Større end
- <= Mindre end eller lig med
- >= Større end eller lig med
- == Lig med
- != Ikke lig med

Conditional Execution: *if*, *else* (5)

- Eksempel med både *if* / *else*:

```
#include <stdio.h>

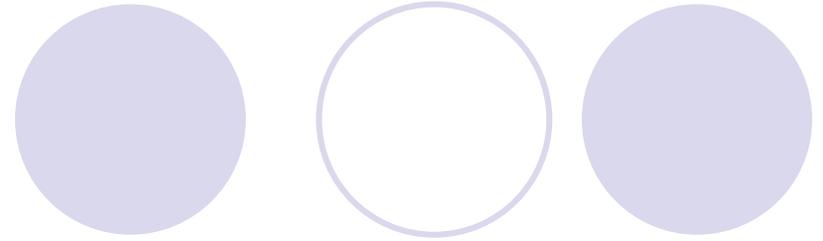
main ()
{
    int num1, num2;

    printf("\nEnter first number ");
    scanf("%d", &num1);

    printf("\nEnter second number ");
    scanf("%d", &num2);

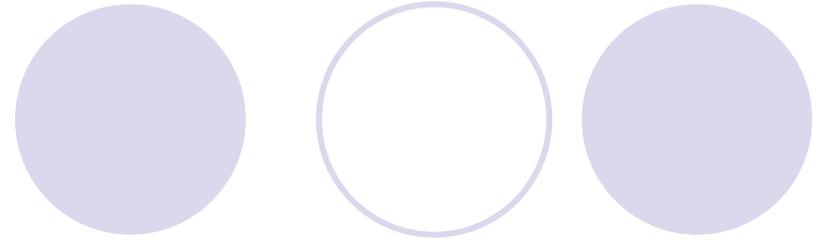
    if (num2 == 0)
        printf("\n\nCannot divide by zero\n\n");
    else
        printf("\n\nAnswer is %d\n\n", num1 / num2);
}
```

Funktioner (1)



- I andre sprog kan de hedde subrutiner eller procedurer
- Funktioner er geniale!
- Basalt: En mængde kode, der er grupperet og givet et navn. Kan bruges ved at kalde navnet på funktionen.
- I gamle dage: *goto*

Funktioner (2)



- Eksempel: Funktion der kan:

```
printf("Hello");  
total = total + 1;
```

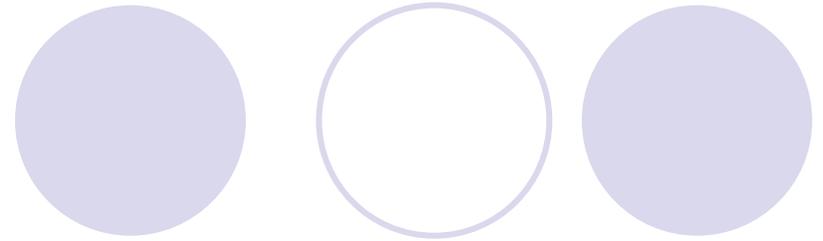
- Funktioner:

```
demo()  
{  
    printf("Hello");  
    total = total + 1;  
}
```

- Kaldes således:

```
main()  
{  
    demo();  
}
```

Funktioner (3)

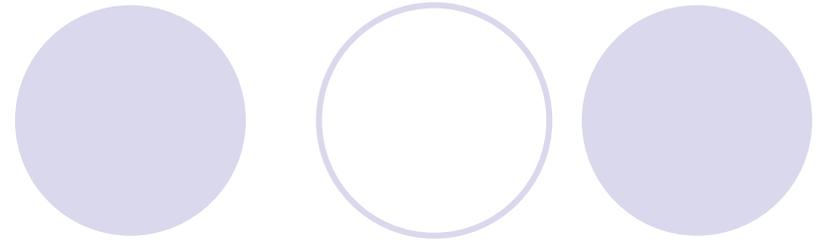


- Fordele ved funktioner:
 - Samme funktion kan kaldes mange gange og fra forskellige steder
 - Et stort program kan med fordel splittes op i mindre dele (=funktioner) med logiske navne

Citat:

"Functions are the building blocks of programs."

Funktioner (4)

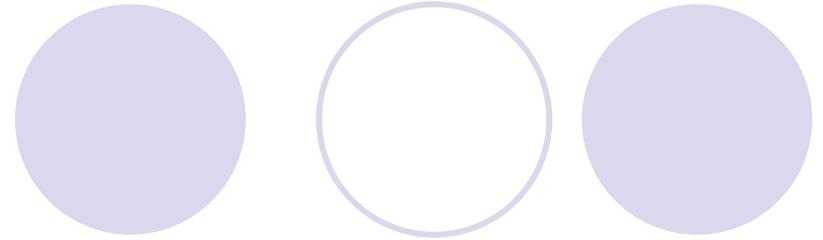


- For at lave brugbare funktioner skal der være en mulighed for at give data til en funktion:

```
sum( int a, int b)
{
    int result;
    result = a + b;
}
```

- ... men *result* er *destroyed*, når funktionen er færdig.
- Altså: Funktioner har deres eget *scope*!

Funktioner (5)



- Returnering af værdier:

```
int sum(int a, int b)
{
    int result;
    result = a + b;
    return result;
}
```

- Kan kaldes på følgende måder:

```
#include <stdio.h>

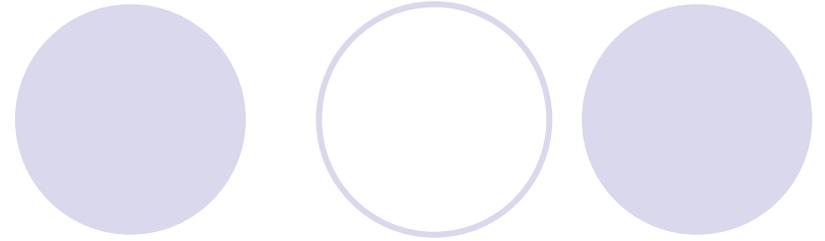
main()
{
    int r;

    r = sum(1, 2);
    printf("Resultatet er %d: \n", r);    /* Giver 3 */

    r = sum(1, 2) * 3;
    printf("Resultatet er %d: \n", r);    /* Giver 9 */

    r = (3 + sum(1, 2)) / 3;
    printf("Resultatet er %d: \n", r);    /* Giver 2 */
}
```

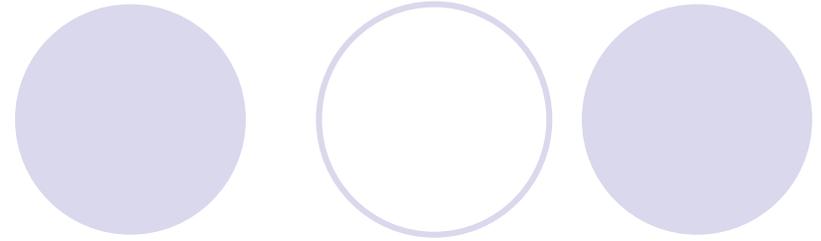
Funktioner (6)



- Generelt er formen:

```
Return-type FunctionName(type declared parameter list)
{
    statements that make up the function;
}
```

Funktioner (7)



● Standard-funktioner i C (uddrag)

stdio.h: I/O functions:

getchar() returns the next character typed on the keyboard.
putchar() outputs a single character to the screen.
printf() as previously described
scanf() as previously described

string.h: String functions

strcat() concatenates a copy of str2 to str1
strcmp() compares two strings
strcpy() copies contents of str2 to str1

ctype.h: Character functions

isdigit() returns non-0 if arg is digit 0 to 9
isalpha() returns non-0 if arg is a letter of the alphabet
isalnum() returns non-0 if arg is a letter or digit
islower() returns non-0 if arg is lowercase letter
isupper() returns non-0 if arg is uppercase letter

math.h: Mathematics functions

acos() returns arc cosine of arg
asin() returns arc sine of arg
atan() returns arc tangent of arg
cos() returns cosine of arg
exp() returns natural logarithm e
fabs() returns absolute value of num
sqrt() returns square root of num

time.h: Time and Date functions

time() returns current calendar time of system
difftime() returns difference in secs between two times
clock() returns number of system clock cycles since program execution

stdlib.h: Miscellaneous functions

malloc() provides dynamic memory allocation
rand() random number
srand() used to set the starting point for rand()

Afsluttende eksempel:

Simpel lommeregner

```
#include <stdio.h>

main()
{
    int mode;
    int tal1, tal2, resultat;

    /* Mode */
    printf("\nMads' simple lommeregner\n\n");
    printf(" 0 = Afslut programmet\n");
    printf(" 1 = Addition (+)\n");
    printf(" 2 = Subtraktion (-)\n");
    printf(" 3 = Multiplikation (*)\n");
    printf(" 4 = Division (/)\n");

    printf("\nSkriv mode: ");
    scanf("%d", &mode);

    if (mode == 0)
    {
        return; /* Afslutter programmet */
    }

    if (mode < 0 || mode > 4)
    {
        printf("\nUgyldig mode...\n\n");
    }

    ... Fortsættes ↗
```

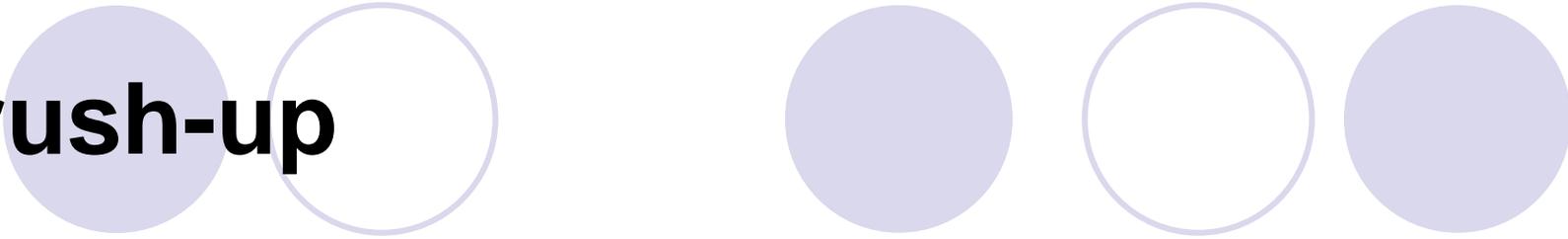
```
else
{
    /* Skriv de 2 tal */
    printf("\nSkriv tal 1: ");
    scanf("%d", &tal1);

    printf("\nSkriv tal 2: ");
    scanf("%d", &tal2);

    /* Foretag udregningen */
    if (mode == 1)
    {
        resultat = tal1 + tal2;
    }
    else if (mode == 2)
    {
        resultat = tal1 - tal2;
    }
    else if (mode == 3)
    {
        resultat = tal1 * tal2;
    }
    else if (mode == 4)
    {
        resultat = tal1 / tal2;
    }

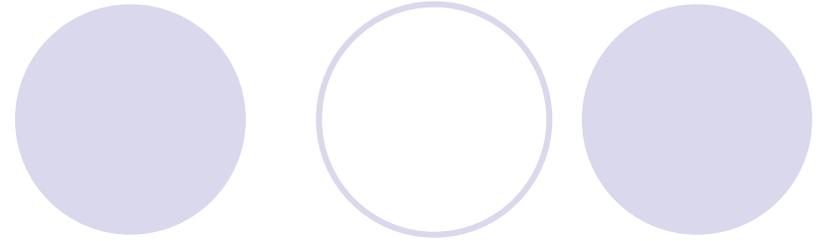
    printf("\nResultatet er: %d\n\n", resultat);
}
}
```

Brush-up



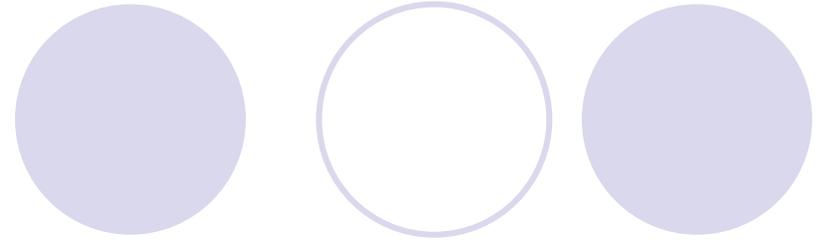
- Videre med C
 - Kontrolløkker (*while, for, ...*)
 - Conditional Execution (*if, if/else*)
 - Funktioner
- I princippet nok til de fleste programmer

Hvad mangler vi?



- Alt det forudgående er faktisk nok for at kunne lave de fleste programmer.
- For at have været hele paletten igennem, mangler vi:
 - *switch* som alternativ til *if/else*
 - *break, continue*
 - Data typer
 - Globale variabler
 - *const* variabler
 - Arrays
 - Pointers
 - Streng
 - Strukturer (*struct*)
 - Filhåndtering

Næste gang



- Praksis

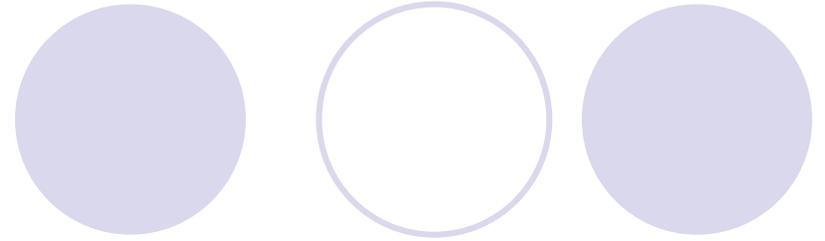
- Find selv på en opgave

- F.eks. de avancerede emner på foregående side

eller

- Jeg finder på en opgave 😊

Mangler I noget?



- Sig til, hvis I mangler info om et specifikt emne.
Så kan vi tage det med næste gang.