Denne guide er oprindeligt
udgivet på Eksperten.dk

COMPUTERWORLD

# C++ Historie og Programmering - Del 1

**Denne guide er til dig der altid har undret dig over hvordan du får en computer til at udføre specifikke opgaver. C++ er et sprog, som man bruger til at fortælle computeren hvad den skal gøre.**

Skrevet den **26. Apr 2013** af **mbm2007** I kategorien **Programmering / C/C++** | ★★★☆☆

## Sproget

Jeg vil nu give dig som nybegynder, en god basis viden for programmering, samt give dig det teoretiske indblik i forskellige begreber indenfor C++. Disse grundlæggende begreber indenfor programmering indbefatter principper som løkker, logiske udtryk, variabler mm. Disse principper går igen i de fleste programmeringssprog, men jeg vil i denne artikelserie fokusere på implementeringen af disse begreber i sproget C++. Afslutningsvis vil jeg selvfølgelig fortælle dig hvordan, hvor og hvornår du kan bruge sproget C++, så du kan komme i gang med dit første program.

*Jeg betragter en nybegynder, som en der ikke kender computerens historie, men en der kan finde ud af at bruge computeren til at gå på internettet og eksempelvis sende mails. Samtidig skal nybegynderen også være i stand til at kende forskellen på skærm og computer*

## Historien

Lige siden den elektroniske computers opfindelse i slutningen af 1940, har det været muligt at få en computer til at beregne ting.

En elektronisk computer er bygget op af elektroniske komponenter, som Motherboardet der forbinder alle øvrige komponenter med hinanden, som RAM (Random Access Memory) hvor programmer kan gemme deres midlertidige hukommelse, som CPU'en (Central processing unit) der er selve computerens hjerne, som Harddisken der lagrer ens billeder, videoer og film, som grafik kortet, der sørger for at vise billeder på skærmen og meget mere.

Inde i computerens RAM, er der millioner, billioner af transistorer, som tilsammen danner rigtig mange kredsløb som enten kan være tændt eller slukket. En transistor er ligesom en kontakt på væggen, som enten kan være tændt eller slukket. Computeren benytter sig af rigtig mange af disse kontakter til midlertidigt at huske hvad den skal gøre mv. Hermed kan computeren kun forstå tændt eller slukket. Man siger at tallet 1, indikerer tændt og 0 indikerer slukket.

De første programmører var derfor nødt til at skrive nuller og ét taller til computeren, for at få den til at beregne forskellige regnestykker, hvilket man kalder for binære tal eller nærmere betegnet maskin-kode. Der var altså ved computerens spæde start ikke rigtig noget decideret sprog som man kunne skrive til computeren for at få den til at udføre givne opgaver.

En dag var der nogle kloge hoveder, som kom til at tænke over om de kunne gøre det nemmere for mennesker at få computeren til at gøre forskellige ting. Dertil opfandt de et programmeringssprog (Plankalkül). Denne opfindelse fik dannet en helt ny æra indenfor computer teknologier.

Et programmeringssprog går ud på at et menneske kan skrive menneske læselige tegn, som senere kan laves om til nuller og ét taller som computeren kan forstå. Selve den proces at lave den menneske

læselige kode om til nuller og ét taller kalder man kompilation. Så når man oversætter fra menneske læselig kode til nuller og ét taller, kompilerer man koden.

Imidlertid fik man nu brug for at computeren selv kunne oversætte den menneskelæselige kode, i dit tilfælde C++ til nuller og ét taller. Dette kunne gøres med en såkaldt Compiler (på dansk: en oversætter) som kompilerer din kode til maskinkode for dig.

I 1979 opfandt danskeren Bjarne Stroustrup sproget C++, som du nu kommer til at lære. Dette sprog satte nye standarder for programmeringen i hele verden, og blev derfor også meget populært.

På eksempelvis engelsk er der en bestemt måde hvorpå man skal sammensætte ord, man skal tage højde for hvordan de skal stå i forhold til hinanden og så videre. Han definerede også måden hvorpå C++ skulle skrives. Måden hvorpå koden skal skrives, for at den kan blive oversat til maskinkode(0 og 1 taller), kalder man for sprogets syntaks.

# Kode sætninger

Når man vil have en computer til at gøre ting, kan man kun få den til at gøre én ting ad gangen (hvis man ser bort fra at de fleste computere har flere CPU-kerner(hjerner) i dag, både fysiske og/eller virtuelle). Hvad den skal gøre skriver man til den i sætninger som den læser og udfører, én for en. Disse sætninger kalder man for: kode-sætninger.

Fuldstændigt ligesom man på dansk skriver sætninger bare med en anderledes syntaks, skriver man nu bare kode-sætninger som computeren forstår. Hvis du nu ville lave en simpel lommeregner på computeren, som kun er i stand til at addere, kunne du skrive følgende kode til Compileren:

```
Spørg brugeren om et tal;
Spørg brugeren om et andet tal;
Regn summen af de to tal ud;
Skriv summen på skærmen;
```

Læg mærke til semikolonet (;), som står som afslutning bag hver kodesætning. På dansk og mange andre sprog afslutter man sin sætning med et punktum. I C++ afslutter man en sætning med semikolon (;). Læg ydermere mærke til at ovenstående kode ikke er C++, da den ikke følger syntaksen for rigtig kode. Den skal bare vise princippet bag hvad computeren skal gøre, hvorfor ovenstående kode kaldes pseudo kode.

Compileren vil kigge på ovenstående kode og derefter udføre(eksekvere) hver kode-sætning i den rækkefølge de står. Men hvad nu hvis man har en række sætninger som man gerne vil gruppere, fordi formålet med de givne sætninger er én specifik ting?

Kodesætninger indeni tuborgklammer ({}), hører og skal eksekveres sammen. Det vil altså sige at vi kan lægge ovenstående pseudo-kode eksempel ind i tuborgklammer, og hermed vil compileren betragte disse "omklamrede" sætninger som én del:

```
{
Spørg brugeren om et tal;
Spørg brugeren om et andet tal;
Regn summen af de to tal ud;
Skriv summen på skærmen;
}
```

Når man sætter sætninger ind imellem tuborgklammer, kalder man for en blok. En samling af kode-sætninger man lægger ind i en blok, kaldes for en kode-blok.

# Variablen

En anden ting man kunne få computeren til, kunne være at gemme brugerens navn, redigere i det så det står bagvendt og derefter skrive det ud på skærmen. Dette kan man gøre med variabler.

Hvordan ville du gribe sådan et problem an?
1. Du kunne først spørge brugeren om hans/hendes navn
2. For at kunne redigere i hans/hendes navn skal du på en eller anden måde kunne huske det navn som brugeren indtaster, så det gør du ved hjælp af en variabel.
3. Derefter vender du hans/hendes navn om.
4. Til sidst skriver du det omvendte navn ud på skærmen.

Når brugeren så lukker programmet igen vil navnet forsvinde fra hans/hendes skærm igen. Navnet har været midlertidigt husket i en såkaldt variabel, men når programmet lukker forsvinder denne også.

Det er sådan en variabel fungerer. Den kan bruges som midlertidig lagringsplads så længe programmet kører, men slettes/frigives ligeså snart programmet slutter/lukker. Forestil dig at en variabel er en lille kasse. I den kasse kan man lægge en ting ned i(en værdi). Man kan sætte et lille mærkat på kassen, så man ved hvad der er nede i kassen(et variabelnavn). Samtidigt kan man være rigtig velorganiseret og definere hvilken type af ting man lægger ned i kassen(variabeltype).

Grunden til at det hedder en variabel er, at man kan tage en ting ud af kassen og man kan lægge en anden ting derned. Det vil sige, at har man først lagt én ting derned, kan man sagtens tage den op igen og lægge en anden ting derned (det er variabelt hvad der kan være i en variabel). Dog kun så længe den nye ting er af samme type som den oprindelige ting man lagde ned, man vil jo gerne være velorganiseret.

Syntaksen er heldigvis defineret, dvs. den måde man laver en sådan kasse/variabel på:

Type Mærkat = Værdi;

Læg mærke til "ligheds" tegnet. Dette tegn fungerer ligesom du husker det fra matematiktimerne, hvor man eksempelvis sagde (x=5). Dette tegn definerer altså at alt på højre side af lighedstegnet skal lægges ned i kassen(variablen). I dette tilfælde lægges tallet 5 ned i kassen med mærkatet "x". Husk dog også på, at da det er en kode-sætning til compileren, skal den afsluttes med et semikolon, og ikke med punktum. Ovenstående er selvfølgelig pseudo-kode.

Nu har du fået en forståelse for syntaksen bag opstilling af en variabel, men nu vil der nok melde sig en række spørgsmål:
Hvilke typer kan jeg benytte mig af? Hvilket mærkat skal jeg sætte på min kasse? Og hvad skal jeg lægge derned?

# Variablen - Typerne

I C++ findes de såkaldte primitive typer. Grunden til at de bliver kaldt primitive typer er ganske enkelt bare at de er bygget ind i selve sproget, at de ikke er lavet af selve programmøren.

Her er en oversigt over nogle af de primitive typer i C++:

| Type | Hele ordet | Dansk = Hvad kan det være? |
|---|---|---|
| int | Integer | Heltal = 1,2,3,4,5,6,7..... |

float | Float | Flydende = 1.0,1.1,1.2.....
double | Double | Dobbelt = 1.0,1.1,1.2.....
char | Character | Bogstav = A,B,C,D.....
string | String | Streng = "Al slags tekst"
bool | Bool | Boolsk = true eller false
---

Disse primitive typer kan bruges til at lave variabler i C++. Hvis du eksempelvis vil lave en variabel hvor du lægger kommatallet 12,4 ned, kunne du skrive:
float minvariabel = 12.4;

Læg her mærke til at komma i programmeringssprog er et punktum. Det er fordi man på engelsk bruger komma og punktum omvendt i forhold til hvad vi gør i Danmark. Komma i England er tusindtals separator, og punktum er decimalseparator. Læg ydermere mærke til at vi både kan bruge typerne float og double til at udtrykke et kommatal. Forskellen på disse er præcisionen hvorved de kan indeholde kommatal. Float er mindst "præcis", hvilket betyder at den ikke kan indeholde ligeså mange decimaler efter "kommaet" som double kan.

Som det fremgår af tabellen kan du se en kolonne med navnet Type. Det er de forkortelser som man har givet typerne, og det er dem man benytter sig af når man skriver koden. Dvs. at når du skal lave en variabel med en type, skriver du ikke integer, men int i stedet for. Det gør det både lettere og hurtigere at skrive:
integer minvariabel = 10; -> FORKERT!
int minvariabel = 10; -> KORREKT!

Kolonnen "Hele ordet" er der hvor du kan se hvad forkortelsen står for. F.eks. står forkortelsen int for Integer, som på dansk betyder heltal.
Kolonnen "Dansk = Hvad kan det være?" Specificerer to ting. Det på venstre side af lighedstegnet er Det danske ord for forkortelsen, mens den højre side er de tal man kan lægge ned i variablen, adskilt med komma.

Du tænker måske hvad man skal bruge heltal til når man bare kan bruge double til at tælle alle tal også komma tal:
At repræsentere et tal som kommatal på en computer og at regne med disse tal er langsommere end at regne med heltal. Derfor skal man overveje om man kun har brug for at arbejde med heltal.

Dertil er det en god huskeregel: "Hvis du undre dig om hvor meget? kan du bruge double", "Hvis du undre dig om hvor mange? kan du bruge int". Hvis du eksempelvis skal tælle hvor mange gange nået er sket, undrer du dig om "hvor mange?", derfor giver det ikke mening at have kommatal. Hvis du derimod skal måle på hvor stort noget er, undrer du dig om "hvor meget?", hvorfor det giver mening at benytte sig af float eller double.

Jeg vil ikke komme ind på hvordan disse typer fungerer i forhold til hinanden, hvordan de lagres i computerens hukommelse, samt hvorfor det f.eks. hedder en double.

# Variablen - Mærkatet

Nu da du har fået en forståelse for hvilke typer din variabel kan bestå af, er det på tide at snakke om hvilket mærkat/variabelnavn du kan sætte på kassen.

I C++ kan dit mærkat være alle bogstaver fra A-Z, dvs. intet æ-ø-å:
MånensTyngdeKraft = Forkert!!!
MaanensTyngdeKraft = Rigtigt!!!

Det kan ikke starte med et tal, men ellers kan du selv bestemme hvor tallet skal være:
1MaanensTyngdeKraft = Forkert!!!
MaanensTyngdeKraft1 = Rigtigt!!!

Der må ikke være specielle tegn men der må godt være underscore (_) :
-Maanens/Tyngde+Kraft = FOKERT!!!
_MaanensTyngdeKraft = Rigtigt!!!

Nu kender du typerne du kan gøre brug af og du ved hvordan du skal lave dine mærkater. Nu skal du bare vide hvad du skal lægge ned i dine variabler(kasser), hvilket er yderst simpelt.

Det hele afhænger af typen. Hvis typen af din variabel er int, dvs. et heltal, kan værdien kun være heltal. Dvs. 1,2,3,4,5..... mv. Hvis du bliver i tvivl, kan du blot benytte dig af ovenstående tabel for at se hvad du kan lægge i de forskellige typer af variabler.

Lad os prøve at lave en variabel. Som du nok kan huske laver man en variabel sådan her:
Type Mærkat = Værdi;

Dette er igen pseudo-kode hvilket vi nu vil erstatte med kode som compileren kan forstå og oversætte til maskinkode. Vi vil gerne have en kasse hvori vi kan gemme min alder. Og vi vil gerne lave et mærkat som hedder Magnus_Alder. Jeg er 15 år gammel, og alder måles ikke med kommatal, derfor vælger jeg typen int (heltal) til min variabel. Værdien bliver blot min alder:

```
Type Mærkat      = Værdi; // Pseudo
int  Magnus_Alder =    15; // Den egentlige syntaks
```

Herover har vi nu lavet en kasse med min alder. Hvis vi senere i programmet skal se hvad der er nede i kassen skriver vi blot variabelnavnet:
Magnus_Alder

Programmet vil så sørge for at erstatte Magnus_Alder med den egentlige værdi som ligger i kassen:
15

Det er princippet bag variabler: at man kan lægge en værdi ned i dem, og senere igen og igen hive værdien op og bruge den, eller ændre den.

Husk at se bort fra de to skråstreger og teksten til højre for dem. De har intet med koden at gøre men forklarer bare hvad der står på linjen.

# Kommentarer - Gør livet lidt lettere

Netop dette leder til det næste emne: Kommentarer.
Nu er vi jo trods alt mennesker, og selvom programmeringskode i dag er meget menneske læseligt, er der stadig problemer selv for professionelle programmører med at gennemskue store mængder kode.

Hvis man i sin kode skriver to skråstreger, så er al tekst bag disse ignoreret af compileren. Dvs. at compileren ikke prøver at oversætte dine kommentarer til maskinkode (0 | 1 taller).

Der er to forskellige måder at skrive sine kommentarer på. Der er flere linjers kommentarer, og der er én linjes kommentarer.

To skråstreger efterfulgt af noget tekst er én linjes kommentarer. Dvs. at du ikke kan gøre sådan her:
// Her er min kommentar

den er meget flot

Her vil compileren tro at den skal oversætte "den er meget flot" til maskinkode.

Derfor er det også defineret at man skal kunne lave flerlinjes kommentarer, så man kan skrive hele romaner hvis man har lyst. Dette gør man lidt anderledes:

```
/*
Her er min kommentar
den er meget flot
og kan spænde over så mange linjer jeg vil
så længe jeg slutter med:
*/
```

Læg mærke til at skråstreg efterfulgt af stjerne siger at her starter kommentaren, mens stjerne efterfulgt af skråstreg siger at her slutter kommentaren.

Som sagt er kommentarer meget nyttige. Hvis du har lavet en kode for et halvt år siden kan du sikkert ikke huske hvad det var de forskellige kodesætninger gjorde. Derfor er det en god ide at skrive kommentarer, så når du og andre skal gennemgå koden igen, er det meget lettere.

# Logiske udtryk - Er jeg større end dig?

I computerprogrammering får man i høj grad brug for at tjekke bruger input. "Bruger input"? Bruger input er simpelt nok, det som brugeren giver af oplysninger til programmet. Og nogle gange får man brug for at reagere forskelligt alt efter hvad brugeren giver af oplysninger. Den egentlige implementering, hvor man kan få fat i brugerens input, lader jeg være op til en senere artikel.

Du kunne f.eks. spørge brugeren: Hvad er 5 + 5?
Og derefter vil du tjekke om deres svar er rigtigt. Dette gør man med de såkaldte logiske udtryk. Hvordan man gør det i C++ vil jeg nu fortælle.

Vi tager udgangspunkt i eksemplet ovenfor. Forestil dig for nu at du kan få fat i det brugeren skriver. Det brugeren har indtastet som svar lægger du så nu ned i variablen du kalder Bruger_Svar. Derefter vil du tjekke om dette er rigtigt:

```
Bruger_Svar == 10
```

Dette er et logisk udtryk, da det enten kan være Rigtigt eller Forkert, (True eller False). Resultatet af ovenstående udtryk er altså af typen bool, da det kun kan være true eller false.
Det skarpe øje vil nok undre sig over om det er en fejl, at jeg har sat to lighedstegn, hvilket det ikke er. Lad os igen kigge på syntaksen for at lave en variabel:

```
Type Mærkat = Værdi;
```

Her kan vi se at ét lighedstegn betyder at tildele variablen en værdi (det er en tildelingsoperator). Tildeling betyder at give noget en værdi, og det er jo hvad man gør når man giver an variabel en værdi. Det tildelingsoperatoren gør er at operere(at gøre noget) for en, deraf navnet operator.

Og da ét ligheds tegn betyder tildelingsoperator for compileren, måtte man jo finde på noget andet som lighedsoperator, som tjekker om noget er ens. Derfor besluttede man at == tjekker om noget er lig med noget andet.

Der er også andre operatorer såsom større end (>), mindre end (<), ikke lig med (!=).

Her er et eksempel: Du vil gerne tjekke om min alder er større end din, men samtidigt vil du også tjekke om min alder er mindre end din fars alder, og hvis alt dette er opfyldt vil du skrive "du er ældre end mig, men ikke min far".

Hvordan vil du gøre det med den viden du lige har fået? Det kan du ikke fordi du ikke kender de logiske operatorer som er:
&& det betyder OG.
|| det betyder ELLER.

Så nu kan du sammensætte et udtryk som passer på eksemplet ovenfor:

(Min_Alder > Din_Alder && Min_Alder < Din_Fars_Alder)

<, && og > kalder man for operatorerne og alt ovenfor kalder man for et udtryk. Da det enten kan udtrykke rigtigt eller forkert. Hvad vil ovenstående udtryk blive hvis min alder er 15, og din fars 65? true eller false?

# Hvad nu hvis...

Nu har du lært hvordan man kan sammenligne værdier, hvordan man tjekker om en værdi er større end en anden osv. Men du har ikke lært syntaksen for hvordan man gør dette i C++.

I C++ er der noget der hedder en "if-sætning". Det er et sted hvori man kan tjekke om et udtryk er sandt eller falsk.

Syntaksen er således:

```
if(Udtryk)
{
   Hvis Udtryk er sandt så gør det inde i denne blok;
}
else
{
   Ellers gør det inde i denne blok.
}
```

Læg mærke til at vi lægger udtrykket ind imellem parenteser, så compileren ved hvornår udtrykker starter og slutter.

Hvis vi kigger på ovenstående kode kan vi sige: if (Hvis), udtrykket er korrekt, så gør det der står i første blok, else(Ellers) gør det i anden blok.

Denne "If sætning" er nok den vigtigste del af programmeringssproget C++ at kunne. Så vi tager lige et konkret eksempel:

```
if(5 > 6)
{
   Så skriv: Det er sgu da mærkeligt;
}
else
{
   Ellers skriv: Det regnede jeg også med;
```

}

Ovenfor kan du se at vi tjekker om 5 er større end 6, og hvis det er sandt bliver vi mærkelige i hovedet, ellers bliver "glade" for det var jo hvad vi regnede med.

# Løkken - Igen og igen....

En dag vil man også gerne have computeren til at gøre en ting flere gange, eller bare uendeligt. Dette kan man gøre med en "løkke".

En løkke er noget der bliver udført igen og igen. Som sagt kan man tage sine kodesætninger og lægge dem ind i tuborgklammer. Når man gjorde dette hed det en blok. I C++ definerer man først hvor mange gange computeren skal gøre en ting. Derefter skriver man den blok der skal udføres de specificerede gange.

Der er tre måder hvorpå du i C++ kan lave en løkke. Jeg har valgt at fortælle de to. Det er "while" løkken og "for" løkken.

While-løkken bruger man hvis man ikke ved det præcise antal gange noget skal udføres. Man bruger også while løkken når noget skal køre uendeligt.

For-løkken bruger man når man kender det antal gange noget skal udføres. Og når man skal gennemgå en stor mængde defineret data.

Her kommer der et While løkke eksempel:

```
while(Min_Alder < 18)
{
  Så skriv: DU er ikke myndig endnu!;
}
```

Denne ovenstående kode kan beskrives: så længe(while) min alder er mindre end 18. Så skriv at jeg ikke er myndig endnu. Læg mærke til at det er samme syntaks som da vi gennemgik "if sætningen", det eneste der er anderledes er at vi har byttet if ordet ud med while.

Men en "For løkke" har ikke samme syntaks som en while løkke. "For løkkens" syntaks er som følgende:

```
for(Initialisering; Betingelse; In-dekremmentering)
{
    Blokken der bliver udført det bestemte antal gange;
}
```

Her er der tre fancy ord.

Initialisering = Det betyder at man giver en variabel en værdi før løkken starter.

Betingelse = Det er at man skriver den betingelse der skal tjekkes for rigtigt eller forkert. Lige så længe betingelsen er rigtig, vil løkken køre.

In-dekrementering = Det betyder her at man enten kan lægge noget oveni (inkrementering) eller trække noget fra "dekrementering". Det er her man in eller dekrementerer en tæller-variabel, hver gang løkken har kørt, simpelthen for at holde styr på hvor mange gange løkken har været udført.

Initialisering, Udtryk og In-dekremmentering er alle indeni en parentes og de er adskilt af semikolon.

Et konkret eksempel ville være:

```
for(int Min_Alder = 0; Min_Alder < 18; Min_Alder++)
{
   Jeg vokser!!!;
}
```

I inkrementeringsdelen laver vi en variabel(kasse) hvori vi lægger værdien 0.

I udtryksdelen tjekker vi om variablens værdi er mindre end 18. Dvs. at løkken vil køre så længe Min_Alder er mindre end 18.

I inkrementeringsdelen lægger vi hele tiden 1 oveni Min_Alder for hver gang løkken har kørt.

Nu ved du at variabler er der man gemmer ting i, og du ved hvordan deres syntaks er.

Du ved at operatorer er dem som sammenligner og gør ting for dig.

Du ved at ved at lave kommentarer i kode, gør du det nemmere for andre og dig selv at læse det på et senere tidspunkt.

Nu sidder du nok tilbage med en flad fornemmelse af, at du for det første ikke ved hvor du skal bruge alt denne viden fra. For det andet ved du ikke hvad du skal bruge det til og for det tredje mangler du et konkret eksempel du bare kan copy paste og se det virke.

Men du må vente lidt endnu. Hold ud!
Fordi nu kender du basis syntaksen for C++. Og du kender forskellige begreber som Løkker, variabler osv. Men du mangler lige det sidste.

# Hvad skal jeg gøre?

Når du skriver din kode, og skal lave store projekter engang. Får du brug for at dele din kode op i flere filer. Men compileren som skal oversætte din kode til nuller og et taller, bliver nødt til at kende disse filer før at den kan sætte dem sammen til ét helt program.

Derfor har C++ compileren, en preprocessor indbygget. En preprocessor er et lille program som erstatter dette:

```
#include <iostream>
```

Med den aktuelle kode som compileren kan forstå. Så inden at compileren begynder at oversætte til maskinkode køre "pre" processoren, og erstatter preprocessor sætninger med den reelle kode.

For at preprocessoren ved at sætningen er tilrettelagt til lige netop den, skriver man en firkant foran sætningen. Husk at preprocessoren ikke skal have et semikolon bagved sætningen.

Vi tager udgangspunkt i denne sætning:

```
#include <iostream>
```

(#) definerer at det er preprocessoren og ikke compileren der skal kigge på sætningen.

(include) fortæller preprocessoren at den skal inkludere en anden fil i denne fil.

Alt mellem < og > er den fil som preprocessoren skal lede efter. Altså den fil som den skal inkludere.

(iostream) Står for "In and Out stream". Det er en standard fil, hvori der er defineret metoder til at få tekst fra brugeren (In), skrive tekst på skærmen (Out), og dette sker i en bestemt rækkefølge (stream)

Et spørgsmål der nu kommer op er hvor starter mit program, og hvordan ved compileren hvor det starter?

I næsten alle programmeringssprog er der en bestemt kodeblok der køres når programmet startes, denne blok kaldes for main og skrives typisk sådan her:

```
int main()
{
  // Al kode herinde vil blive kørt når programmet starter
}
```

Den måde at isolere kode-stykker på i funktioner, vil jeg komme ind på i en senere artikel.

# Så er vi der næsten...

Det du lige nu har brug for er en compiler, den som oversætter din kode til maskin kode. Du kan downloade den jeg bruger her:
http://www.codeblocks.org/downloads

Når du har downloadet setup-filen kan du prøve at kigge på denne YouTube video som vil vise dig hvordan du installerer og opretter et nyt projekt i CodeBlocks programmet:
http://www.youtube.com/watch?v=ec-yYMTUV3c

Indsæt denne kode og kompiler på F9 knappen og du er "Good to go":

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Hello world!" << endl;
   return 0;
}
```

Nu kan du selv eksperimentere med koden og ændre Hello World teksten til noget andet. DU kan prøve at lave løkker så den skriver det flere gange:

```
#include <iostream>
using namespace std;
int main()
{
   for(int i = 0;i<5;i++)
   {
```

```
        cout << "Hello world!" << endl;
    }
    return 0;
}
```

Du kan lave variabler og udskrive dem:

```
#include <iostream>
using namespace std;
int main()
{
    string hej = "Hej med dig";
    cout << hej << endl;
    return 0;
}
```

Ovenstående kode-eksempler vil blive forklaret i en senere artikel.

Prøv dig frem! Held og lykke.

**Kommentar af thegenuine d. 04. Aug 2010 | 1**

Godt stykke arbejde. Jeg har kun lige skimtet det igennem, men det er lige sådan noget, jeg har ledt efter. Programmeringssprog for dummies ;) - jeg ser frem til, det næste afsnit kommer.

**Kommentar af Jackount d. 07. Aug 2010 | 2**

Hvor fanden blev min kommentar af???java script: void(0);

**Kommentar af hrc d. 15. Sep 2010 | 3**

Hvis man vil starte med at programmere, så er der alternativer til C++ som tillader lidt færre unoder. Et godt programmeringssprog skal være opdragende og det er C++ ikke!

**Kommentar af hrc d. 15. Sep 2010 | 4**

C++ is like teenage sex

* It's on everyone's mind all the time.
* Everyone talks about it all the time.
* Everyone thinks everyone else is doing it.
* Almost no one is really doing it.
* The few who are doing it are:
    * doing it poorly.
    * sure it will be better next time.
    * not practicing it safely.
* Everyone's bragging about their successes all the time, although
  very few have actually had any.

**Kommentar af Mr6Pack d. 11. Nov 2011 | 5**

Super indlæg. Har hjulpet utrolig meget og er allerede igang med at lave andre små projekter

Tak

**Kommentar af olebole d. 03. Apr 2012 | 6**

<ole>

@**hrc**: Glæd dig, til dine voksenår! Selvom du tydeligvis er frustreret nu, kan jeg trøste dig med, det bliver meget lettere med alderen  *o)

/mvh
</bole>

**Kommentar af olebole d. 26. Apr 2013 | 7**

**@mbm2007:** Lidt korrekturlæsning ville gøre underværker. Og så tror jeg, du mener 'Betingelse', når du skriver 'Udtryk'  =)

Hils i øvrigt Bjarne, når du ser ham næste gang. Det er en ære at læse en guide, skrevet af en af hans gamle kammerater  *D

**Kommentar af mbm2007 d. 26. Apr 2013 | 8**

Tak for responsen @olebole :)
Det er taget til efterretning.

**Kommentar af olebole d. 27. Apr 2013 | 9**

Læsere ville måske have mere gavn af en rettelse  *o)

**Kommentar af mbm2007 d. 24. Jul 2013 | 10**

#9 - Når jeg benytter ordet "efterretning" i forbindelse med en bøjning af "at tage", kan det sagtens betyde, at jeg gør noget ved det:
"modtage en besked og rette sig efter den" - sproget.dk