Objekt-Orienteret Programmering

Primitive Typer

Aslak Johansen asjo@mmmi.sdu.dk
Peter Nellemann pmn@mmmi.sdu.dk

September 11, 2024

SDU

Part 0:
Hello, World

Hello, World

"When the first caveman programmer chiseled the first program on the walls of the first cave computer, it was a program to paint the string 'Hello, world' in Antelope pictures. Roman programming textbooks began with the 'Salut, Mundi' program. I don't know what happens to people who break with this tradition, but I think it's safer not to find out."

– The Linux Kernel Module Programming Guide (2007-05-18 ver 2.6.4)

Hello, World

> *"When the first caveman programmer chiseled the first program on the walls of the first cave computer, it was a program to paint the string 'Hello, world' in Antelope pictures. Roman programming textbooks began with the 'Salut, Mundi' program. I don't know what happens to people who break with this tradition, but I think it's safer not to find out."*
> – The Linux Kernel Module Programming Guide (2007-05-18 ver 2.6.4)

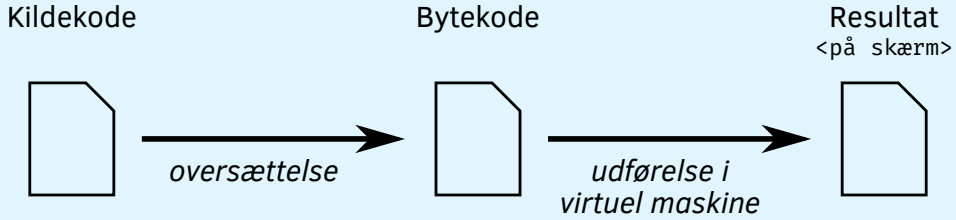
```
Console.WriteLine("Hello, World");
```

Fra Kildekode til Afvikling

Hvad sker der egentligt, når I afvikler jeres C# program?

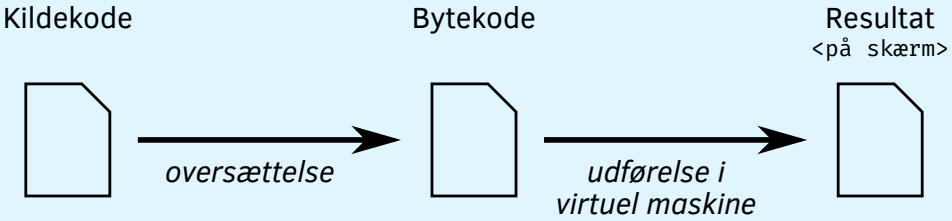
Fra Kildekode til Afvikling

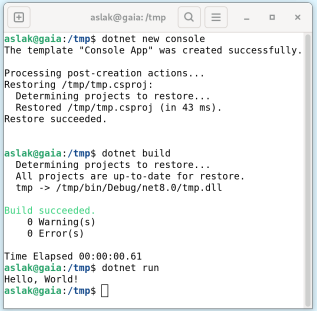
Hvad sker der egentligt, når I afvikler jeres C# program?



Fra Kildekode til Afvikling

Hvad sker der egentligt, når I afvikler jeres C# program?





Part 1:
Datatyper

Primitive Datatyper

Datatyper bruges til at beskrive "typer af data".

Data er ofte noget der bliver produceret (eller ændret) når vi kører et program. Vi bruger typen til at beskrive rummet af gyldige værdier for data.

C# har primitive datatyper indenfor 4 kategorier:

- Heltal
- Kommatal
- Sandhedsværdi
- Tegn

Det er de første typer vi skal beskæftige os med.

Primitive Datatyper ▷ Heltal

Datatype	Fortegn	Mulige værdier:	Plads
`sbyte`	×	-128 til 127	8 bit
`byte`		0 til 255	8 bit
`short`	×	-32768 til 32767	16 bit
`ushort`		0 til 65535	16 bit
`int`	×	-2147483648 til 2147483647	32 bit
`uint`		0 til 4294967295	32 bit
`long`	×	-9223372036854775808 til 9223372036854775807	64 bit
`ulong`		0 til 18446744073709551615	64 bit

Bemærk manglen på konsistens i navngivning: Særudganven af byte er *signed* og særudgaverne for de resterende er *unsigned*.

Der er også en `nint` og en `nuint`. Hold jer fra dem.

Primitive Datatyper ▷ Kommatal

Datatype	Mulige værdier:	Plads
`float`	7 betydende cifre†	32 bit
`double`	15 betydende cifre†	64 bit
`decimal`	28 betydende cifre†	128 bit

I kommer primært til at bruge `double`.

Primitive Datatyper ▷ Sandhedsværdi og Tegn

Datatype	**Mulige værdier:**	**Plads**
`char`	Code unit: A, i, ?, 2, k, _, . osv. *Et unicode tegn**	16 bit
`bool`	true / false	8 bit eller 32 bit

Primitive Datatyper ▷ Sandhedsværdi og Tegn

Datatype	**Mulige værdier:**	**Plads**
`char`	Code unit: A, i, ?, 2, k, _, . osv. *Et unicode tegn**	16 bit
`bool`	true / false	8 bit eller 32 bit

En `char` repræsenterer en unicode *code unit*. Dette er oftest nok til at repræsentere et hvilkårligt tegn, men ikke altid. Undlad derfor at bruge denne type med mindre I virkelig forstår hvad I laver. Det bedre alternativ `string` vil blive introduceret senere.

Primitive Datatyper ▷ Sandhedsværdi og Tegn

Datatype	Mulige værdier:	Plads
`char`	Code unit: A, i, ?, 2, k, _, . osv. *Et unicode tegn**	16 bit
`bool`	true / false	8 bit eller 32 bit

En `char` repræsenterer en unicode *code unit*. Dette er oftest nok til at repræsentere et hvilkårligt tegn, men ikke altid. Undlad derfor at bruge denne type med mindre I virkelig forstår hvad I laver. Det bedre alternativ `string` vil blive introduceret senere.

Afhængigt af om man erklærer en enkeltstående `bool` eller placerer en række af dem i et array (som vil blive introduceret senere) så bruges der enten 32 bit eller 8 bit.

Datatype Øvelser

Opsummering:

- Heltal (10 stk)
- Kommatal (3 stk)
- Tegn (1 stk)
- Boolean (1 stk)

Datatype Øvelser

Opsummering:

- Heltal (10 stk)
- Kommatal (3 stk)
- Tegn (1 stk)
- Boolean (1 stk)

Hvad passer til at beskrive:

- Højden af jeres forelæser?

Datatype Øvelser

Opsummering:

- Heltal (10 stk)
- Kommatal (3 stk)
- Tegn (1 stk)
- Boolean (1 stk)

Hvad passer til at beskrive:

- Højden af jeres forelæser?
- Antallet af studerende i denne klasse?

Datatype Øvelser

Opsummering:

- Heltal (10 stk)
- Kommatal (3 stk)
- Tegn (1 stk)
- Boolean (1 stk)

Hvad passer til at beskrive:

- Højden af jeres forelæser?
- Antallet af studerende i denne klasse?
- Om man er kommet til tiden?

Datatype Øvelser

Opsummering:

- Heltal (10 stk)
- Kommatal (3 stk)
- Tegn (1 stk)
- Boolean (1 stk)

Hvad passer til at beskrive:

- Højden af jeres forelæser?
- Antallet af studerende i denne klasse?
- Om man er kommet til tiden?
- Ens navn?

Datatype Øvelser

Opsummering:

- Heltal (10 stk)
- Kommatal (3 stk)
- Tegn (1 stk)
- Boolean (1 stk)

Hvad passer til at beskrive:

- Højden af jeres forelæser?
- Antallet af studerende i denne klasse?
- Om man er kommet til tiden?
- Ens navn?
- Hvor mange jordbær der er per deltager til en børnefødselsdag?

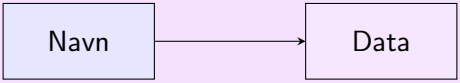
Part 2:
Variable

Variable ▷ Definition

En variabel er den konstruktion man bruger til at *navngive* et stykke data.

Værdien af de data som variabel peger på kan ændres* ... via navnet.

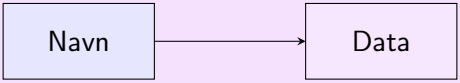
Variable ▷ Regler



Syntaksregler for navnet:

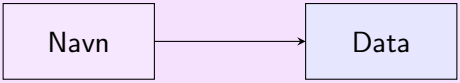
- Begynder med et bogstav eller en underscore*.
- Kan bestå af tal, bogstaver og/eller underscores.

Variable ▷ Regler



Syntaksregler for navnet:

- Begynder med et bogstav eller en underscore*.
- Kan bestå af tal, bogstaver og/eller underscores.



Dataene:

- Har en datatype.

I C# knyttes navnet til en bestemt datatype.

Variable ▷ Erklæring

Hver gang vi skriver og afslutter en "linje" kode i C# har vi lavet et *statement**.

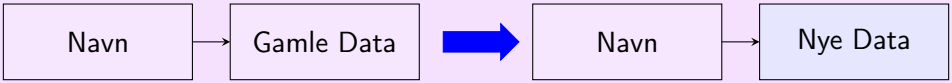
Syntaks:
⟨*type*⟩⟨*variable*⟩;

Eksempel:

```
double value;
```

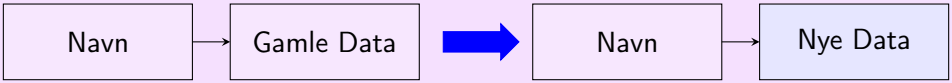
Variable ▷ Tildeling

Ved tildeling overskrives det datarum som variablen peger på med en (ny) værdi:



Variable ▷ Tildeling

Ved tildeling overskrives det datarum som variablen peger på med en (ny) værdi:



Syntaks:
⟨*variable*⟩ = ⟨*expression*⟩;

Eksempel:

```
value = 42;
```

Bemærk: Dette kræver at variablen allerede er erklæret.

Variable ▷ Erklæring og Tildeling

Erklæring og tildeling kan udtrykkes igennem et enkelt *statement*.

Variable ▷ Erklæring og Tildeling

Erklæring og tildeling kan udtrykkes igennem et enkelt *statement*.

Syntaks:
⟨*type*⟩⟨*variable*⟩ = ⟨*expression*⟩;

Eksempel:

```
int value = 42;
```

Variable ▷ Eksempler

```
double price       = 0.0;
int    pageCount   = 217;
int    year        = 2019;
int    authorCount = 1;
```

Part 3:
Statements og Expressions

Et *statement* er enheden af et trin der skal udføres.

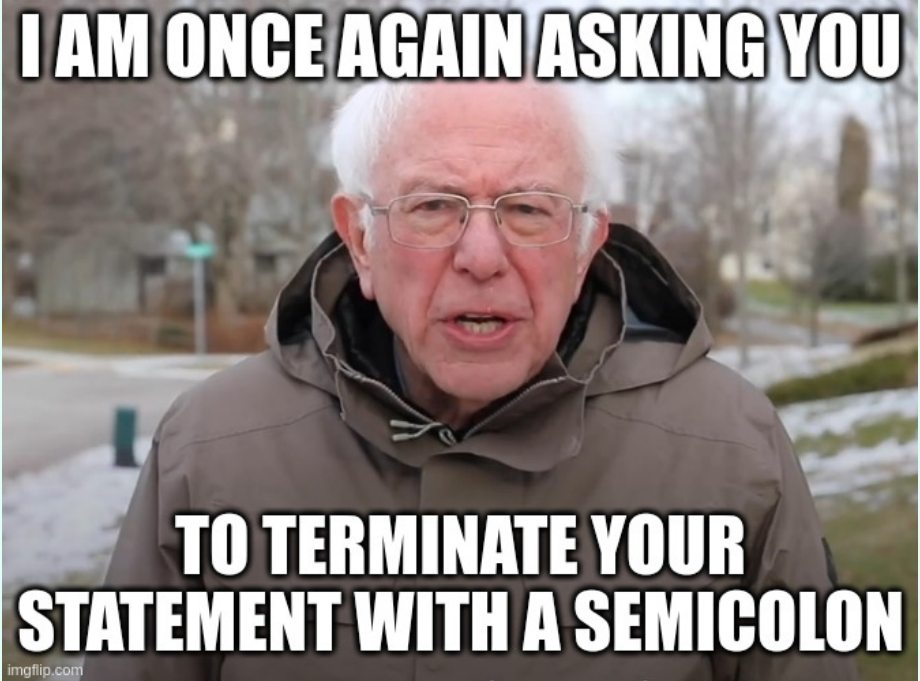
Statements skrives typisk på en linje for sig selv afsluttet af et semikolon.

Statements udføres én af gangen i rækkefølge.

Eksempel på en sådan sekvens:

```
double height = 1.84;
int
count;
count = 1;
```

Hvert statement afsluttes med et semikolon.


I AM ONCE AGAIN ASKING YOU
TO TERMINATE YOUR STATEMENT WITH A SEMICOLON
imgflip.com

Statements og Expressions ▷ Expressions

Et *expression* er en stump af kode der evaluerer til en værdi.

Vi kan bruge matematiske operatorer til at opbygge expressions:

Navn	**Betydning**	**Eksempel**	**Resultat**
+	Addition	`34+1`	35
-	Subtraktion	`34.0-0.1`	33.9
*	Multiplikation	`300*30`	9000
/	Division	`1.0/2.0`	0.5
%	Rest	`20%3`	2

Vi kalder "=" for en type af tildelingsoperator (eng: assignment operator): Det er en operator, der kan give en variabel en ny værdi.

Der findes andre typer af tildelingsoperatorer (*mathematically augmented assignments*) der fungerer som *shorthands* for hyppige anvendelser:

Operator	**Navn**	**Eksempel**	**Ækvivalent**
`+=`	Addition tildeling	`i += 8`	`i = i + 8`
`-=`	Subtraktion tildeling	`i -= 8`	`i = i - 8`
`*=`	Multiplikation tildeling	`i *= 8`	`i = i * 8`
`/=`	Division tildeling	`i /= 8`	`i = i / 8`
`%=`	Rest tildeling	`i %= 8`	`i = i % 8`

Statements og Expressions ▷ Evaluering af en Tildeling

Hvad sker der her:

```
x = x + 1;
```

Statements og Expressions ▷ Evaluering af en Tildeling

Hvad sker der her:

```
x = x + 1;
```

Det bliver evalueret sådan her:

1. Oversætteren finder strukturen i koden. I dette tilfælde er der ét statement, som er en tildeling.

Statements og Expressions ▷ Evaluering af en Tildeling

Hvad sker der her:

```
x = x + 1;
```

Det bliver evalueret sådan her:

1. Oversætteren finder strukturen i koden. I dette tilfælde er der ét statement, som er en tildeling.
2. Først evalueres det expression der er på højre side af tildelingsoperatoren. I dette tilfælde slås værdien af `x` op og derefter lægges der én til den.

Statements og Expressions ▷ Evaluering af en Tildeling

Hvad sker der her:

```
x = x + 1;
```

Det bliver evalueret sådan her:

1. Oversætteren finder strukturen i koden. I dette tilfælde er der ét statement, som er en tildeling.
2. Først evalueres det expression der er på højre side af tildelingsoperatoren. I dette tilfælde slås værdien af `x` op og derefter lægges der én til den.
3. Dernæst gemmes resultatet af denne evaluering i den variabel der står på venstre side af tildelingsoperatoren. I dette tilfælde gemmes resultatet af forrige trin tilbare i en en variabel der indgik i evalueringen.

Statements og Expressions ▷ Evaluering af en Tildeling

Hvad sker der her:

```
x = x + 1;
```

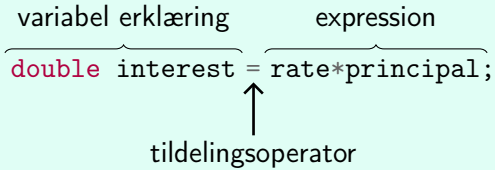
Det bliver evalueret sådan her:

1. Oversætteren finder strukturen i koden. I dette tilfælde er der ét statement, som er en tildeling.
2. Først evalueres det expression der er på højre side af tildelingsoperatoren. I dette tilfælde slås værdien af `x` op og derefter lægges der én til den.
3. Dernæst gemmes resultatet af denne evaluering i den variabel der står på venstre side af tildelingsoperatoren. I dette tilfælde gemmes resultatet af forrige trin tilbare i en en variabel der indgik i evalueringen.

"=" er altså ikke en lighedsoperator, men en tildelingsoperator: Den repræsenterer en handling!

Statements og Expressions ▷ Kombinationer

Vi kan kombinere en variabel erklæring, en tildelingsoperator og et expression til et statement:

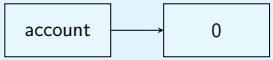


Hermed opnår vi:

- Erklæring af en variabel.
- Tildeling af den værdi som er resultatet af evalueringen af et expression.
  - Værdien af dette expression afhænger af værdierne af to andre variable.

Part 4:
Eksempler

Eksempel ▷ Konto



Kontekst: Variablen `account` bruges til at representere en bankkonto.

Variablens livscyklus:

1. **Oprettelse af konto**.
2. Indsættelse af løn: +10.000,00 kroner.
3. Køb af kursusbog: -500,00 kroner.
4. MobilePay overførsel for vens kursusbog: +500,00 kroner.

```
double account = 0; ←
account += 10000;
account -= 500;
account += 500;
```

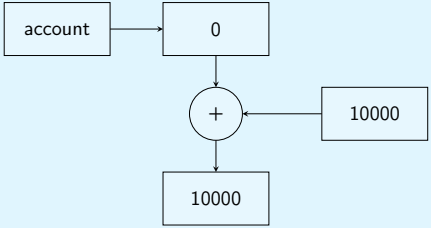
Eksempel ▷ Konto

Kontekst: Variablen `account` bruges til at representere en bankkonto.

Variablens livscyklus:

1. Oprettelse af konto.
2. **Indsættelse af løn: +10.000,00 kroner**.
3. Køb af kursusbog: -500,00 kroner.
4. MobilePay overførsel for vens kursusbog: +500,00 kroner.

```
double account = 0;
account += 10000; ←
account -= 500;
account += 500;
```

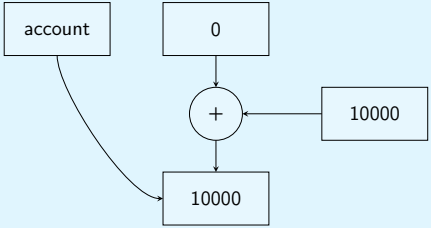


Eksempel ▷ Konto

Kontekst: Variablen `account` bruges til at representere en bankkonto.

Variablens livscyklus:

1. Oprettelse af konto.
2. **Indsættelse af løn: +10.000,00 kroner**.
3. Køb af kursusbog: -500,00 kroner.
4. MobilePay overførsel for vens kursusbog: +500,00 kroner.



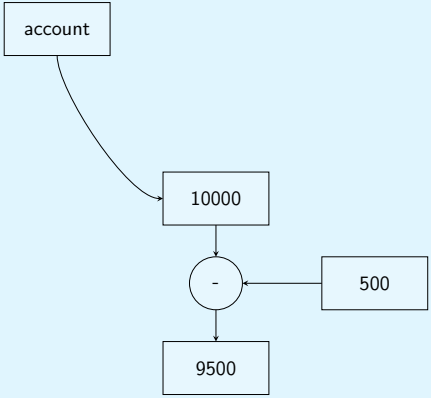

```
double account = 0;
account += 10000; ←
account -= 500;
account += 500;
```

Eksempel ▷ Konto

Kontekst: Variablen `account` bruges til at representere en bankkonto.

Variablens livscyklus:

1. Oprettelse af konto.
2. Indsættelse af løn: +10.000,00 kroner.
3. **Køb af kursusbog: -500,00 kroner**.
4. MobilePay overførsel for vens kursusbog: +500,00 kroner.



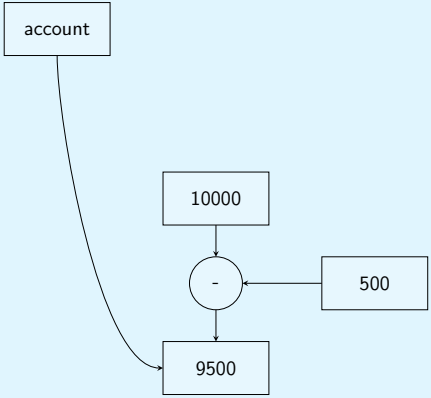

```
double account = 0;
account += 10000;
account -= 500; ←
account += 500;
```

Eksempel ▷ Konto

Kontekst: Variablen `account` bruges til at representere en bankkonto.

Variablens livscyklus:

1. Oprettelse af konto.
2. Indsættelse af løn: +10.000,00 kroner.
3. **Køb af kursusbog: -500,00 kroner**.
4. MobilePay overførsel for vens kursusbog: +500,00 kroner.




```
double account = 0;
account += 10000;
account -= 500; ←
account += 500;
```

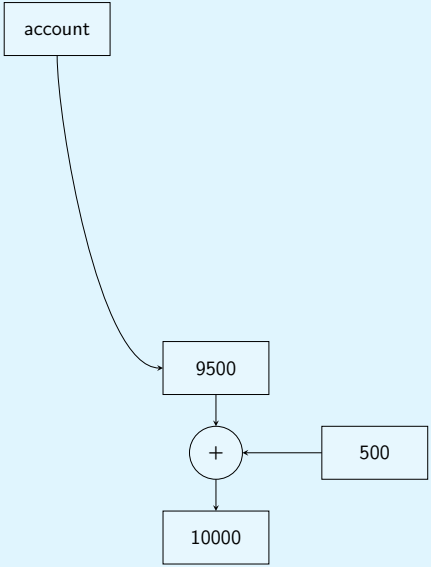
Eksempel ▷ Konto

Kontekst: Variablen `account` bruges til at representere en bankkonto.

Variablens livscyklus:

1. Oprettelse af konto.
2. Indsættelse af løn: +10.000,00 kroner.
3. Køb af kursusbog: -500,00 kroner.
4. **MobilePay overførsel for vens kursusbog: +500,00 kroner**.

```
double account = 0;
account += 10000;
account -= 500;
account += 500; ←
```



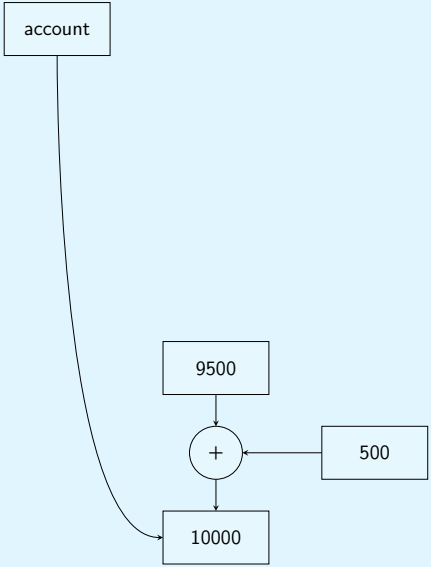
Eksempel ▷ Konto

Kontekst: Variablen `account` bruges til at representere en bankkonto.

Variablens livscyklus:

1. Oprettelse af konto.
2. Indsættelse af løn: +10.000,00 kroner.
3. Køb af kursusbog: -500,00 kroner.
4. **MobilePay overførsel for vens kursusbog: +500,00 kroner**.

```
double account = 0;
account += 10000;
account -= 500;
account += 500; ←
```



Eksempel ▷ Arealberegning

```
double radius; // declare radius
double area;   // declare area

// assign a radius
radius = 20; // radius is now 20

// compute area
area = 3.14159 * radius * radius;

// display results
Console.WriteLine("The area for the circle of radius " +
                  radius + " is " + area);
```

Eksempel ▷ Kommentarer

```
/*
 * this is called a block comment
 */

/* comments do not affect the execution of code */

// this comment ends with the line
```

Eksempel ▷ Kommentarer

```
/*
 * this is called a block comment
 */

/* comments do not affect the execution of code */

// this comment ends with the line
```

(Skriv på engelsk i koden: Man skriver kun kode på engelsk)

Part 5:
Type Casting

Automatisk Typecasting

Gyldigt cast:

```
int    i = 2;
double d = i;
```

Ugyldigt cast:

```
double d = 2.0;
int    i = d;
```

Automatisk Typecasting

Gyldigt cast:

```
int    i = 2;
double d = i;
```

Ugyldigt cast:

```
double d = 2.0;
int    i = d;
```

Hvorfor?

Automatisk Typecasting

Gyldigt cast:

```
int    i = 2;
double d = i;
```

Ugyldigt cast:

```
double d = 2.0;
int    i = d;
```

Hvorfor?

Oversætteren checker om den kan garantere at operationen kan udføres uden tab af information, og dén arbejder udelukkende på typer.

Eksplicit Typecasting

Løsning:

```
double d = 2.0;
int    i = (int) d;
```

Eksplicit Typecasting

Løsning:

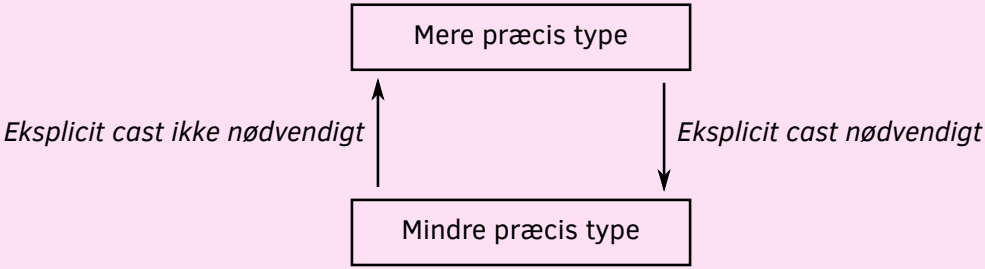
```
double d = 2.0;
int    i = (int) d;
```

Bemærk: Eksplicit typecasting kan koste præcision.

- Ikke nødvendigvis som man intuitivt tænker: $7.96 \rightarrow 7$

Ved at foretage et eksplicit cast fortæller man oversætteren at man påtager sig ansvaret for sådanne tab af præcision.

Typecasting Reglen



Part 6:
Identifiers

Identifiers

Navne på “ting” i programmer kaldes identifiers.

Ting kan fx være klasser og variable.

De må ikke starte med cifre.

Identifiers

Visse navne er reserveret i programmeringssprog.

- Eksempler fra C#: `if`, `while`, `for`, `public`, `static`, `void`, `else` ...

Ofte lagres data i maskinens hukommelse på lokationer med numeriske adresser.

- Identifiers bruges til at tilgå disse data symbolsk i stedet for manuelt at skulle holde styr på de specifikke lokationer.

Eksempler på identifiers: `ComputeArea`, `area`, `radius`, `print`, `_test`

CaSe SeNsItIvItY

CaSe SeNsItIvItY

CaSe SeNsItIvItY

Brug IKKE retard case!

Case Sensitivity

Der er forskel på STORE og små bogstaver i visse sprog ... som fx C#. Vi siger at disse sprog er *case sensitive*.

Der er "god skik" for hvordan ting navngives:

- Variable starter med småt.

Case Sensitivity

Der er forskel på STORE og små bogstaver i visse sprog ... som fx C#. Vi siger at disse sprog er *case sensitive*.

Der er "god skik" for hvordan ting navngives:

- Variable starter med småt.

Tilsvarende er mange filsystemer også case sensitive. Hvis du sidder på et filsystem der ikke er case sensitivt, så betyder dét at noget virker på din maskine ikke nødvendigvis, at det virker på andres. Sørg derfor for at copy-paste filnavne og kommandoer.

Part 7:
De Sidste Detaljer

(Post|Pre) (Increment|Decrement) Operators

Increment: noget forøges
Decrement: noget formindskes
Pre: Ændringen foretages **før** værdien udtrækkes
Post: Ændringen foretages **efter** værdien udtrækkes

Eksempel:

```
int counter = 5;
counter = counter + 1;
counter++;
counter--;
```

Hvad printer følgende kode (hvis vi tilføjede den enkelte linje til ovenstående)?

(Post|Pre) (Increment|Decrement) Operators

Increment: noget forøges
Decrement: noget formindskes
Pre: Ændringen foretages **før** værdien udtrækkes
Post: Ændringen foretages **efter** værdien udtrækkes

Eksempel:

```
int counter = 5;
counter = counter + 1;
counter++;
counter--;
```

Hvad printer følgende kode (hvis vi tilføjede den enkelte linje til ovenstående)?

1. `Console.WriteLine(counter);`

(Post|Pre) (Increment|Decrement) Operators

Increment: noget forøges
Decrement: noget formindskes
Pre: Ændringen foretages **før** værdien udtrækkes
Post: Ændringen foretages **efter** værdien udtrækkes

Eksempel:

```
int counter = 5;
counter = counter + 1;
counter++;
counter--;
```

Hvad printer følgende kode (hvis vi tilføjede den enkelte linje til ovenstående)?

1. `Console.WriteLine(counter);`
2. `Console.WriteLine(++counter);`

(Post|Pre) (Increment|Decrement) Operators

Increment: noget forøges
Decrement: noget formindskes
Pre: Ændringen foretages **før** værdien udtrækkes
Post: Ændringen foretages **efter** værdien udtrækkes

Eksempel:

```
int counter = 5;
counter = counter + 1;
counter++;
counter--;
```

Hvad printer følgende kode (hvis vi tilføjede den enkelte linje til ovenstående)?

1. `Console.WriteLine(counter);`
2. `Console.WriteLine(++counter);`
3. `Console.WriteLine(counter++);`

Literals

En *literal* er – i sig selv – en konstant værdi, der noteres direkte i programkoden.

```
bool   b  = true;
float  f1 = 3.14F;
double d1 = 3.14D;
double d2 = 3.14; // double er default for kommatal
string s  = "dette er en \"string\"";
```

Der bruges punktum som kommatalsseparator og ingen tusindtal-separator.

Literaler er typede!

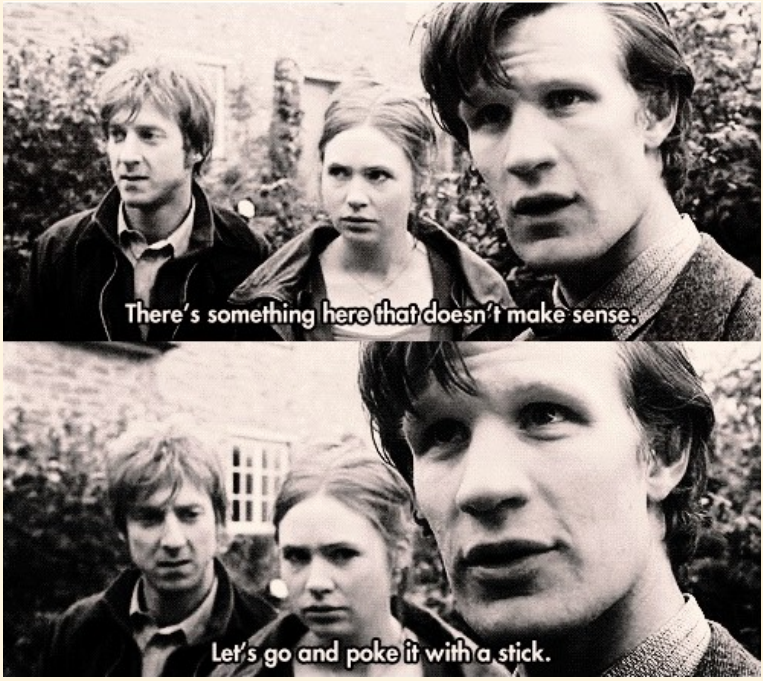
I string literaler bruger man backslash til at bypasse (nydansk: escape) den normale betydning af det efterfølgende tegn.

- Hvordan laver man så en string literal der indeholder en backslash?

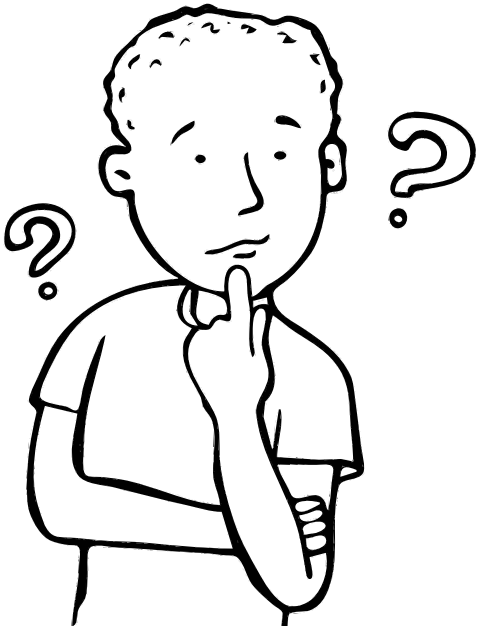
Opsummering

```
/**
 * This program that will compute the amount of interest that is earned on $17,000 invested at
 * an interest rate of 0.027 for one year.
 */

/* Declare the variables. */
double principal; // The value of the investment.
double rate; // The annual interest rate.
double interest; // Interest earned in one year.

/* Do the computations. */
principal = 17000;
rate = 0.027;
interest = principal * rate; // Compute the interest.
principal += interest;

// Compute value of investment after one year, with interest.
// (Note: The new value replaces the old value of principal.)
/* Output the results. */
Console.WriteLine("The interest earned is $"+interest);
Console.WriteLine("The value of the investment after one year is $"+principal);
```

Afledt fra bogen "Introduction to Programming Using Java" af David J. Eck.


There's something here that doesn't make sense.
Let's go and poke it with a stick.

Spørgsmål?



https://openclipart.org/detail/236687/boy-thinking-of-question