

Objekt-Orienteret Programmering

Control Flow

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

September 13, 2024

Part 0: Boolean Typen

Boolean Typen ▷ Definition

Type der anvendes til at beskrive udsagn som værende sande eller falske.

Boolean værdier: `{true, false}`

Alternativt: `{1, 0}` eller `{åben, lukket}` eller `{aktiv, inaktiv}` eller ...

Der findes en primitiv type i C# til at beskrive booleans: `bool`.

Mathematik

C#

$$a = b$$

a == b

$$a \neq b$$

a != b

$$a > b$$

a > b

$$a < b$$

a < b

$$a \geq b$$

a >= b

$$a \leq b$$

a <= b

Boolean Typen ▷ Boolske Expressions

Boolske expressions er expressions der evaluerer til enten `true` eller `false`.

Eksempler:

```
bool lastLevel      = (level == levelCount);  
bool modernTime     = (year > 0);  
bool newBestScore   = score >= bestScore;
```

Boolean Typen ▷ Boolske Operatorer

Navn	Eksempel	Sandt når
NOT	<code>!value</code>	et udtryk ikke er sandt
AND	<code>value1 && value2</code>	begge udtryk er sande
OR	<code>value1 value2</code>	mindst ét udtryk er sandt
XOR	<code>value1 ^ value2</code>	netop ét udtryk er sandt

“modsat af værdien”

a	!a
0	1
1	0

Boolean Typen ▷ Boolske Operatorer ▷ Sandhedstabel for AND Operatoren

“sand hvis og kun hvis begge sider er sande”

a	b	a && b
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Typen ▷ Boolske Operatorer ▷ Sandhedstabel for OR Operatoren

“sand hvis mindst et af udtrykkene er sande”

a	b	a b
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Typen ▷ Boolske Operatorer ▷ Sandhedstabel for XOR Operatoren

“sand hvis netop ét udtryk er sandt (exclusive or)”

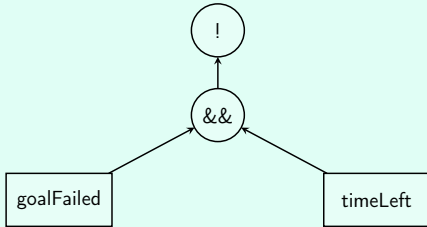
a	b	a && b
0	0	0
0	1	1
1	0	1
1	1	0

Boolean Typen ▷ Eksempel

```
bool goalFailed = false;  
bool timeLeft   = false;  
bool success    = !goalFailed && timeLeft;
```

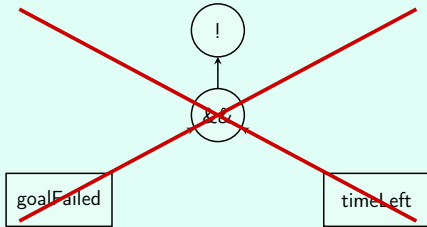
Boolean Typen ▷ Eksempel

```
bool goalFailed = false;  
bool timeLeft   = false;  
bool success    = !goalFailed && timeLeft;
```



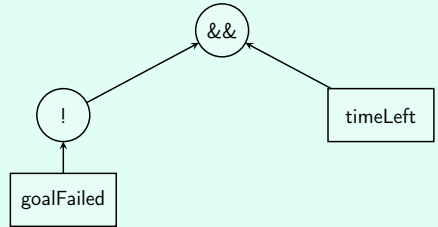
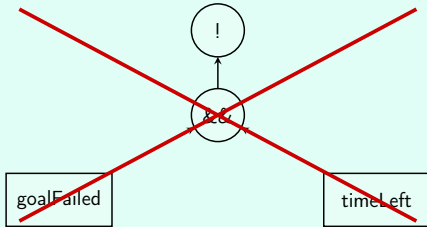
Boolean Typen ▷ Eksempel

```
bool goalFailed = false;  
bool timeLeft   = false;  
bool success    = !goalFailed && timeLeft;
```



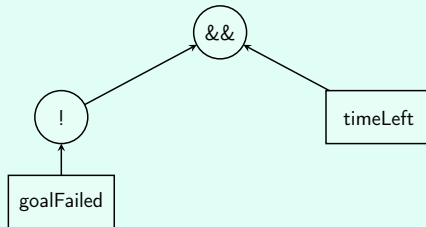
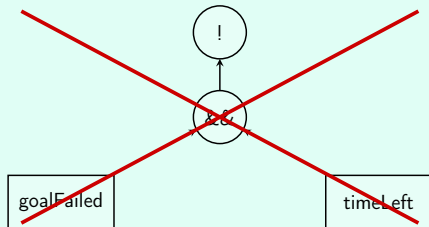
Boolean Typen ▷ Eksempel

```
bool goalFailed = false;  
bool timeLeft   = false;  
bool success    = !goalFailed && timeLeft;
```



Boolean Typen ▷ Eksempel

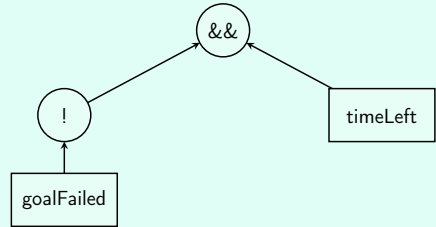
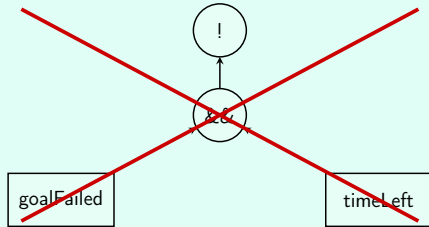
```
bool goalFailed = false;  
bool timeLeft   = false;  
bool success    = !goalFailed && timeLeft;
```



Hvorfor?

Boolean Typen ▷ Eksempel

```
bool goalFailed = false;  
bool timeLeft   = false;  
bool success    = !goalFailed && timeLeft;
```



Hvorfor?

Operator precedens!

Part 1:

Valg i Expressions

Valg i Expressions ▷ Ternær (eng: ternary) Operator

En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

Valg i Expressions ▷ Ternær (eng: ternary) Operator

En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

Syntaks:

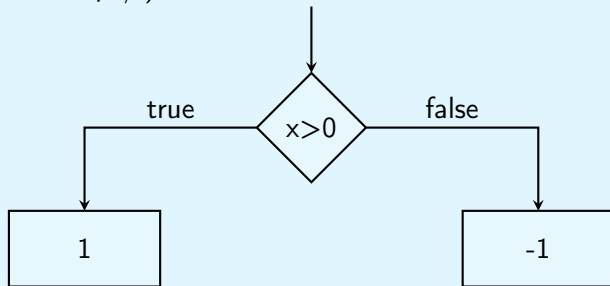
$(\langle condition \rangle ? \langle true - expr \rangle : \langle false - expr \rangle)$

Valg i Expressions ▷ Ternær (eng: ternary) Operator

En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

Syntaks:

($\langle condition \rangle$? $\langle true - expr \rangle$: $\langle false - expr \rangle$)



Valg i Expressions ▷ Ternær (eng: ternary) Operator

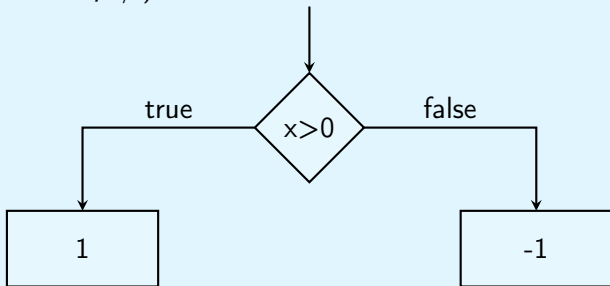
En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

Syntaks:

($\langle condition \rangle$? $\langle true - expr \rangle$: $\langle false - expr \rangle$)

Eksempel:

```
int y = (x>0 ? 1 : -1);
```



Valg i Expressions ▷ Ternær (eng: ternary) Operator

En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

Syntaks:

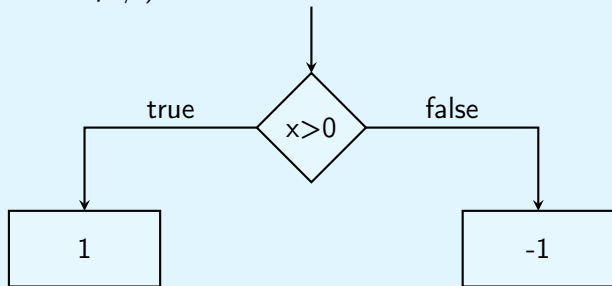
$(\langle condition \rangle ? \langle true - expr \rangle : \langle false - expr \rangle)$

Eksempel:

```
int y = (x>0 ? 1 : -1);
```

Hvad er værdien af y når

1. $x \mapsto 12$?



Valg i Expressions ▷ Ternær (eng: ternary) Operator

En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

Syntaks:

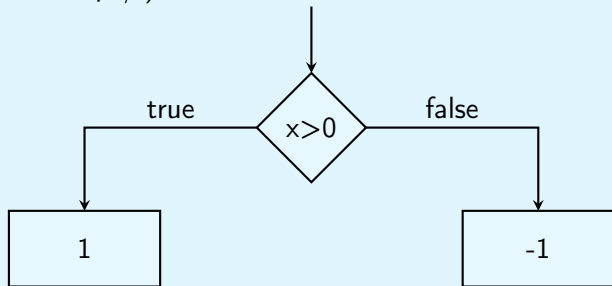
$(\langle condition \rangle ? \langle true - expr \rangle : \langle false - expr \rangle)$

Eksempel:

```
int y = (x>0 ? 1 : -1);
```

Hvad er værdien af y når

1. $x \mapsto 12?$
2. $x \mapsto 0?$



Valg i Expressions ▷ Ternær (eng: ternary) Operator

En operator der afhængigt af en "condition" evaluerer til resultatet af ét af to expressions.

Syntaks:

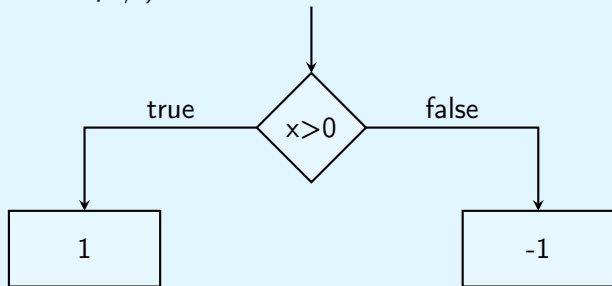
$(\langle condition \rangle ? \langle true - expr \rangle : \langle false - expr \rangle)$

Eksempel:

```
int y = (x>0 ? 1 : -1);
```

Hvad er værdien af y når

1. $x \mapsto 12?$
2. $x \mapsto 0?$
3. $x \mapsto \text{true}?$



Part 2: Valg i Statements

Valg i Statements ▷ If og Else

Ud fra evalueringen af et boolsk expression kan et program – på kørelstidspunktet – bestemme hvorvidt et statement skal udføres.

Valg i Statements ▷ If og Else

Ud fra evalueringen af et boolsk expression kan et program – på kørselstidspunktet – bestemme hvorvidt et statement skal udføres.

Syntaks:

```
if ( ⟨condition⟩ )  
    ⟨statement⟩
```

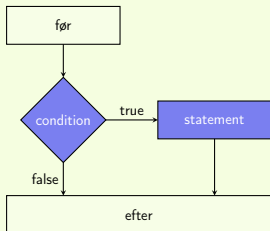
```
if ( ⟨condition⟩ )  
    ⟨true – statement⟩  
else  
    ⟨false – statement⟩
```

Valg i Statements ► If og Else

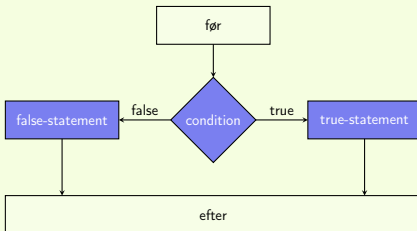
Ud fra evalueringen af et boolsk expression kan et program – på kørelstidspunktet – bestemme hvorvidt et statement skal udføres.

Syntaks:

```
if ( <condition> )  
    <statement>
```



```
if ( <condition> )  
    <true – statement>  
else  
    <false – statement>
```



Valg i Statements ▷ Branching Eksempel

Eksempel:

```
Console.WriteLine("Input is "+i);  
if (i%2==1)  
    i++;  
Console.WriteLine("Output is "+i);
```

Valg i Statements ▷ Branching Eksempel

Eksempel:

```
Console.WriteLine("Input is "+i);  
if (i%2==1)  
    i++;  
Console.WriteLine("Output is "+i);
```

Tolkes: Læg én til i hvis i modulo 2 er 1 (altså, hvis i er et ulige tal).

Valg i Statements ► Branching Eksempel

Eksempel:

```
Console.WriteLine("Input is "+i);  
if (i%2==1)  
    i++;  
Console.WriteLine("Output is "+i);
```

Tolkes: Læg én til i hvis i modulo 2 er 1 (altså, hvis i er et ulige tal).

Bemærk: Branches (her ved hjælp af `if`) indfører valg i udførelsen af et program.

Part 3:

Gruppering af Statements

Gruppering af Statements ► Sekvenser med Blocks

En block er et statement der selv kan indeholde en sekvens af flere statements.

Gruppering af Statements ► Sekvenser med Blocks

En block er et statement der selv kan indeholde en sekvens af flere statements.

Syntaks:

```
{  
    ⟨statement — sekvens⟩  
}
```

Gruppering af Statements ► Sekvenser med Blocks

En block er et statement der selv kan indeholde en sekvens af flere statements.

Syntaks:

```
{  
    ⟨statement — sekvens⟩  
}
```

Husk: Statements adskildt af semi-kolon'er er en sekvens af statements.

Gruppering af Statements ▷ Sekvenser med Blocks

En block er et statement der selv kan indeholde en sekvens af flere statements.

Syntaks:

```
{  
    ⟨statement — sekvens⟩  
}
```

Husk: Statements adskilt af semi-kolon'er er en sekvens af statements.

Det er meget anvendeligt, for nu kan vi erstatte vores enkelt-linje statements med en block, og dermed flere statements!

Gruppering af Statements ▷ Sekvenser med Blocks

En block er et statement der selv kan indeholde en sekvens af flere statements.

Syntaks:

```
{  
    ⟨statement — sekvens⟩  
}
```

Husk: Statements adskilt af semi-kolon'er er en sekvens af statements.

Det er meget anvendeligt, for nu kan vi erstatte vores enkelt-linje statements med en block, og dermed flere statements!

Note: Blocks implementerer konceptet om “sekvenser” i vores programmer.

Gruppering af Statements ▷ Branching med Blocks (1/2)

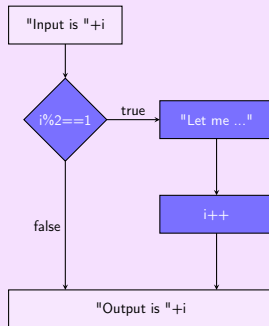
Eksempel:

```
Console.WriteLine("Input is "+i);  
if (i%2==1) {  
    Console.WriteLine("Let me adjust your value ...");  
    i++;  
}  
Console.WriteLine("Output is "+i);
```

Gruppering af Statements ▷ Branching med Blocks (1/2)

Eksempel:

```
Console.WriteLine("Input is "+i);  
if (i%2==1) {  
    Console.WriteLine("Let me adjust your value ...");  
    i++;  
}  
Console.WriteLine("Output is "+i);
```



Gruppering af Statements ▷ Branching med Blocks (2/2)

Eksempel:

```
double radius = 12.0;
if (radius < 0.0) {
    Console.WriteLine("Incorrect input");
} else {
    double area = 3.14159 * radius * radius;
    Console.WriteLine("Area is " + area);
}
```


Gruppering af Statements ▷ “Dangling Else” Problemet (1/2)

```
int i = 1, j = 2, k = 3;  
if (i < j)  
    if (i > k)  
        Console.WriteLine("A");  
else  
    Console.WriteLine("B");
```

Hvad bliver der printet?

Gruppering af Statements ▷ “Dangling Else” Problemet (1/2)

```
int i = 1, j = 2, k = 3;  
if (i < j)  
    if (i > k)  
        Console.WriteLine("A");  
else  
    Console.WriteLine("B");
```

Hvad bliver der printet?

Der bliver printet “B” da ...

Gruppering af Statements ▷ “Dangling Else” Problemet (2/2)

C# tolker koden sådan her:

```
int i = 1, j = 2, k = 3;  
if (i < j)  
    if (i > k)  
        Console.WriteLine("A");  
else  
    Console.WriteLine("B");
```

Gruppering af Statements ▷ “Dangling Else” Problemet (2/2)

C# tolker koden sådan her:

```
int i = 1, j = 2, k = 3;  
if (i < j)  
    if (i > k)  
        Console.WriteLine("A");  
else  
    Console.WriteLine("B");
```

Ud over at muliggøre sekvenser, løser blocks også andre problemer; herunder “dangling else”-problemet.

Gruppering af Statements ▷ “Dangling Else” Problemet (2/2)

C# tolker koden sådan her:

```
int i = 1, j = 2, k = 3;  
if (i < j)  
    if (i > k)  
        Console.WriteLine("A");  
else  
    Console.WriteLine("B");
```

Ud over at muliggøre sekvenser, løser blocks også andre problemer; herunder “dangling else”-problemet.

Derfor: **Brug altid blocks, også ved enkelt-linje statements!**

Part 4:

Multiple Valgmuligheder i Statements

Multiple Valgmuligheder i Statements ► Else-If Chained Branches

```
char direction = 'n';
if (direction=='N') {
    Console.WriteLine("Going north ...");
} else if (direction=='S') {
    Console.WriteLine("Going south ...");
} else if (direction=='E') {
    Console.WriteLine("Going east ...");
} else if (direction=='W') {
    Console.WriteLine("Going west ...");
} else {
    Console.WriteLine("I don't understand !?!");
}
```

Multiple Valgmuligheder i Statements ► Else-If Chained Branches

```
char direction = 'N';  
if (direction=='N') {  
    Console.WriteLine("Going north ...");  
} else if (direction=='S') {  
    Console.WriteLine("Going south ...");  
} else if (direction=='E') {  
    Console.WriteLine("Going east ...");  
} else if (direction=='W') {  
    Console.WriteLine("Going west ...");  
} else {  
    Console.WriteLine("I don't understand !?!");  
}
```


Multiple Valgmuligheder i Statements ► Switch Statements

```
switch ( ⟨expr⟩ ) {  
  case ⟨expression⟩ :  
    ⟨statement – sekvens⟩ ;  
    break;  
  case ⟨expression⟩ :  
    ⟨statement – sekvens⟩ ;  
    break;  
  case ⟨expression⟩ :  
    ⟨statement – sekvens⟩ ;  
    break;  
  default :  
    ⟨statement – sekvens⟩ ;  
    break;  
}
```

Multiple Valgmuligheder i Statements ► Switch Statements

```
switch ( <expr> ) {  
  case <expression> :  
    <statement – sekvens> ;  
    break;  
  case <expression> :  
    <statement – sekvens> ;  
    break;  
  case <expression> :  
    <statement – sekvens> ;  
    break;  
  default :  
    <statement – sekvens> ;  
    break;  
}
```

Notes:

- Hvis ikke **break** anvendes, fortsættes til næste case.
- Afviklingen må ikke ramme den } der afslutter **switch** statementet. Indtil videre kan vi kun opnå dette med et **break** statement til sidst.
- **default** er valgfri og “fanger” alt der ikke rammer en case.

Multiple Valgmuligheder i Statements ► Switch Eksempel

```
switch (direction) {  
    case 'N':  
        Console.WriteLine("Going north ...");  
        break;  
    case 'S':  
        Console.WriteLine("Going south ...");  
        break;  
    case 'E':  
        Console.WriteLine("Going east ...");  
        break;  
    case 'W':  
        Console.WriteLine("Going west ...");  
        break;  
    default:  
        Console.WriteLine("I don't understand !?!");  
        break;  
}
```

Part 5:
Gentagelse af Statements

Gentagelse af Statements ▷ Loop Konstruktioner

Print "Hello, World":

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

Gentagelse af Statements ▷ Loop Konstruktioner

Print "Hello, World" 2 gange:

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

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

Gentagelse af Statements ▷ Loop Konstruktioner

Print "Hello, World" 3 gange:

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

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

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

Gentagelse af Statements ▷ Loop Konstruktioner

Print "Hello, World" 3 gange:

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

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

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

Hvad hvis vi vil printe det n gange?

Gentagelse af Statements ▷ Loop Konstruktioner

Print "Hello, World" 3 gange:

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

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

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

Hvad hvis vi vil printe det n gange?

Så bruger vi et *loop*!

Gentagelse af Statements ▷ Loop Konstruktioner

Print "Hello, World" 3 gange:

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

Hvad hvis vi vil printe det n gange?

Så bruger vi et *loop*!

I dag skal vi lære om tre typer loops:

1. while
2. do-while
3. for

Part 5.1:

Loops med “while” Statements

Loops med “while” Statements ▷ Introduktion

Det mest basale type af programloop i C#.

Afvikler et *statement* så længe en *condition* er sand.

Typisk anvendt henover en *block*, hvor en sekvens afvikles så længe en *condition* er sand.

Loops med “while” Statements ▷ Syntaks

Syntaks:

```
while ( ⟨condition⟩ ) ⟨statement⟩
```

Loops med “while” Statements ▷ Syntaks

Syntaks:

while (*⟨condition⟩*) *⟨statement⟩*

Loopets “*continuation condition*” testes i starten af hvert gennemløb.

Loops med “while” Statements ▷ Syntaks

Syntaks:

while (*⟨condition⟩*) *⟨statement⟩*

Loopets “*continuation condition*” testes i starten af hvert gennemløb.

Hvis denne *condition* er sand, afvikles *statement*; ellers ikke.

Loops med “while” Statements ▷ Syntaks

Syntaks:

while (*⟨condition⟩*) *⟨statement⟩*

Loopets “*continuation condition*” testes i starten af hvert gennemløb.

Hvis denne *condition* er sand, afvikles *statement*; ellers ikke.

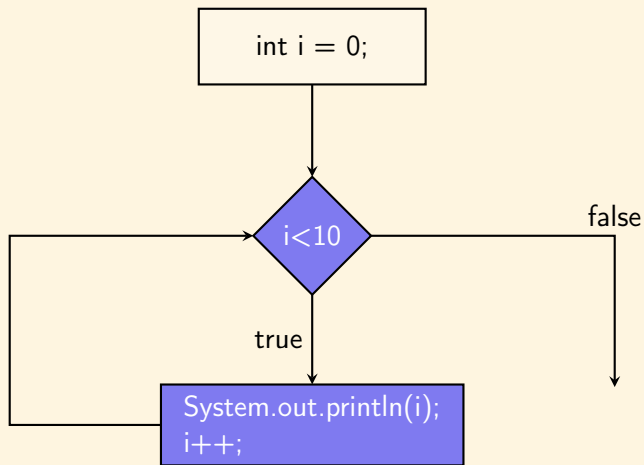
Statementet afvikles altså $0 - \infty$ gange.

Loops med “while” Statements ▷ Eksempel

```
int i = 0;  
while (i<10) {  
    Console.WriteLine(i);  
    i++;  
}
```

Loops med "while" Statements ▷ Eksempel

```
int i = 0;  
while (i<10) {  
    Console.WriteLine(i);  
    i++;  
}
```



Loops med “while” Statements ▷ Øvelse

```
int i = 0;
while (i<4) {
    i++;
    Console.WriteLine("i is now " + i);
}
```

Loops med “while” Statements ▷ Øvelse

```
int i = 0;
while (i<4) {
    i++;
    Console.WriteLine("i is now " + i);
}
```

Efter afvikling af denne kode har i værdien 4.

Loops med “while” Statements ▷ Øvelse

```
int i = 0;
while (i<4) {
    i++;
    Console.WriteLine("i is now " + i);
}
```

Efter afvikling af denne kode har i værdien 4.

Spørgsmål: Hvad står der i den sidste linje som programmet udskriver?

Part 5.2:

Loops med “do-while” Statements

Loops med “do-while” Statements ▷ Introduktion

Modsat et “while”-loop, men evaluerer continuation condition **efter** statement.

- ▶ Konsekvens: Statementet afvikles altid én gang indledningsvist.

Loops med “do-while” Statements ▷ Introduktion

Modsat et “while”-loop, men evaluerer continuation condition **efter** statement.

- ▶ Konsekvens: Statementet afvikles altid én gang indledningsvist.

Anvendes som “while” oftest til at modificere en variable i et statement så længe en continuation condition er sand; dog mindst én gang.

Loops med “do-while” Statements ▷ Syntaks

Syntaks:

do $\langle statement \rangle$ *while* ($\langle condition \rangle$)

Loops med “do-while” Statements ▷ Syntaks

Syntaks:

do $\langle statement \rangle$ *while* ($\langle condition \rangle$)

Continuation condition testes første gang efter én gennemkørsel.

Loops med “do-while” Statements ▷ Syntaks

Syntaks:

do $\langle statement \rangle$ *while* ($\langle condition \rangle$)

Continuation condition testes første gang efter én gennemkørsel.

Statement afvikles herefter indtil continuation conditionen er falsk.

Loops med “do-while” Statements ▷ Syntaks

Syntaks:

do $\langle statement \rangle$ *while* ($\langle condition \rangle$)

Continuation condition testes første gang efter én gennemkørsel.

Statement afvikles herefter indtil continuation conditionen er falsk.

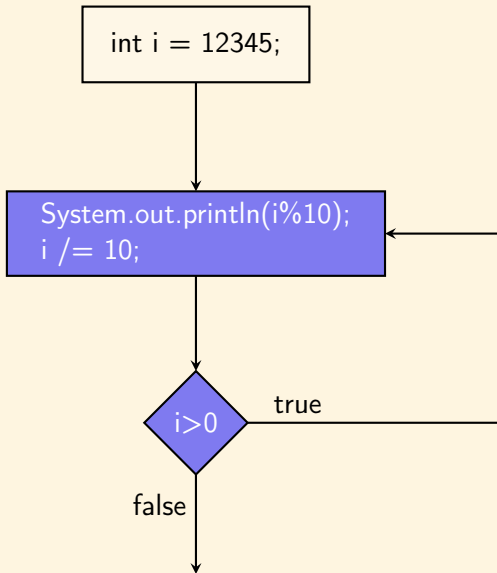
Statementet afvikles altså $1 - \infty$ gange.

Loops med “do-while” Statements ▷ Eksempel

```
int i = 12345;  
do {  
    Console.WriteLine(i%10);  
    i /= 10;  
} while (i>0);
```

Loops med "do-while" Statements ▷ Eksempel

```
int i = 12345;  
do {  
    Console.WriteLine(i%10);  
    i /= 10;  
} while (i>0);
```



Gentagelse af Statements ▷ While som Erstatning for Do-While

```
do {  
    <statement>  
} while ( <condition> );
```

Gentagelse af Statements ▷ While som Erstatning for Do-While

```
do {  
    <statement>  
} while ( <condition> );
```

```
<statement>  
while ( <condition> ) {  
    <statement>  
}
```


Gentagelse af Statements ▷ While som Erstatning for Do-While

```
do {  
    <statement>  
} while ( <condition> );
```

```
<statement>  
while ( <condition> ) {  
    <statement>  
}
```

Spørgsmål: Hvilken variant er mest elegant, og hvorfor?

Gentagelse af Statements ► Do-While som Erstatning for While

```
while ( <condition> ) {  
    <statement>  
}
```

Gentagelse af Statements ► Do-While som Erstatning for While

```
while ( <condition> ) {  
    <statement>  
}
```

```
if ( <condition> ) {  
    do {  
        <statement>  
    } while ( <condition> );  
}
```

Gentagelse af Statements ► Do-While som Erstatning for While

```
while ( <condition> ) {  
    <statement>  
}
```

```
if ( <condition> ) {  
    do {  
        <statement>  
    } while ( <condition> );  
}
```

Spørgsmål: Hvilken variant er mest elegant, og hvorfor?

Gentagelse af Statements ▷ Do-While vs While



Part 5.3:

Loops med “for” Statements

Loops med “for” Statements ▷ Introduktion

Et for-loop er en formalisering af et mønster – “en opskrift” – som ofte ses i programmer.

Loops med “for” Statements ▷ Introduktion

Et for-loop er en formalisering af et mønster – “en opskrift” – som ofte ses i programmer.

Mønsteret består af:

1. En loop **body** som indeholder de statements der skal loopes over.

Loops med “for” Statements ▷ Introduktion

Et for-loop er en formalisering af et mønster – “en opskrift” – som ofte ses i programmer.

Mønsteret består af:

1. En loop **body** som indeholder de statements der skal loopes over.
2. Et **initialization statement** der afvikles én gang i begyndelsen.

Loops med “for” Statements ▷ Introduktion

Et for-loop er en formalisering af et mønster – “en opskrift” – som ofte ses i programmer.

Mønsteret består af:

1. En loop **body** som indeholder de statements der skal loopes over.
2. Et **initialization statement** der afvikles én gang i begyndelsen.
3. En **continuation condition** der bestemmer hvorvidt et loop skal fortsættes.

Loops med “for” Statements ▷ Introduktion

Et for-loop er en formalisering af et mønster – “en opskrift” – som ofte ses i programmer.

Mønsteret består af:

1. En loop **body** som indeholder de statements der skal loopes over.
2. Et **initialization statement** der afvikles én gang i begyndelsen.
3. En **continuation condition** der bestemmer hvorvidt et loop skal fortsættes.
4. Et **update statement** der afvikles i slutningen af hvert gennemløb af loopet.

Loops med “for” Statements ▷ Syntaks

Syntaks:

for (*⟨init – statement⟩* ; *⟨condition⟩* ; *⟨update – statement⟩*) *⟨body – statement⟩*

Loops med “for” Statements ▷ Syntaks

Syntaks:

for ($\langle init - statement \rangle$; $\langle condition \rangle$; $\langle update - statement \rangle$) $\langle body - statement \rangle$

Continuation condition testes i starten af hvert gennemløb. Kroppen afvikles $0 - \infty$ gange.

Loops med “for” Statements ▷ Syntaks

Syntaks:

for ($\langle \text{init} - \text{statement} \rangle$; $\langle \text{condition} \rangle$; $\langle \text{update} - \text{statement} \rangle$) $\langle \text{body} - \text{statement} \rangle$

Continuation condition testes i starten af hvert gennemløb. Kroppen afvikles $0 - \infty$ gange.

“Syntaktisk sukker” for:

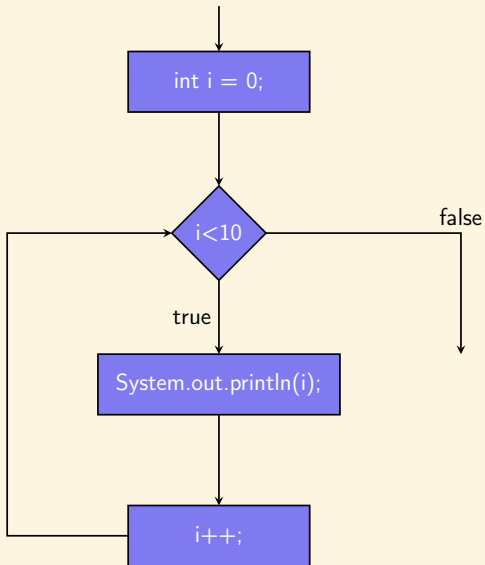
```
<init-statement>  
while (<condition>) {  
    <body-statement>  
    <update-statement>  
}
```

Loops med “for” Statements ▷ Eksempel

```
for (int i=0 ; i<10 ; i++) {  
    Console.WriteLine(i);  
}
```

Loops med "for" Statements ▷ Eksempel

```
for (int i=0 ; i<10 ; i++) {  
    Console.WriteLine(i);  
}
```



Loops med “for” Statements ▷ For vs While

```
for (int years=0 ; years<5 ; years++) {  
    interest = principal * rate;  
    principal += interest;  
    Console.WriteLine(principal);  
}
```

Loops med "for" Statements ▷ For vs While

```
for (int years=0 ; years<5 ; years++) {  
    interest = principal * rate;  
    principal += interest;  
    Console.WriteLine(principal);  
}
```

```
int years = 0; // init  
while (years<5) { // condition  
    interest = principal * rate;  
    principal += interest;  
    Console.WriteLine(principal);  
  
    years++; // update  
}
```

Loops med “for” Statements ▷ Flere Veje til Rom ...

Spørgsmål:

Hvad er den bedste måde til at udskrive sekvensen 2, 4, 6, 8, 10, 12, 14, 16, 18, 20?

Loops med “for” Statements ► Flere Veje til Rom ...

Spørgsmål:

Hvad er den bedste måde til at udskrive sekvensen 2, 4, 6, 8, 10, 12, 14, 16, 18, 20?

```
for (int i=1 ; i<=10 ; i++) {  
    Console.WriteLine(2*i);  
}
```

Loops med "for" Statements ▷ Flere Veje til Rom ...

Spørgsmål:

Hvad er den bedste måde til at udskrive sekvensen 2, 4, 6, 8, 10, 12, 14, 16, 18, 20?

```
for (int i=1 ; i<=10 ; i++) {  
    Console.WriteLine(2*i);  
}
```

```
for (int i=2 ; i<=20 ; i = i+2) {  
    Console.WriteLine(i);  
}
```

Loops med "for" Statements ▷ Flere Veje til Rom ...

Spørgsmål:

Hvad er den bedste måde til at udskrive sekvensen 2, 4, 6, 8, 10, 12, 14, 16, 18, 20?

```
for (int i=1 ; i<=10 ; i++) {  
    Console.WriteLine(2*i);  
}
```

```
for (int i=2 ; i<=20 ; i = i+2) {  
    Console.WriteLine(i);  
}
```

```
for (int i=2 ; i<=20 ; i++) {  
    if (i%2==0) // is i even?  
    {  
        Console.WriteLine(i);  
    }  
}
```

Loops med "for" Statements ▷ Flere Veje til Rom ...

Spørgsmål:

Hvad er den bedste måde til at udskrive sekvensen 2, 4, 6, 8, 10, 12, 14, 16, 18, 20?

```
for (int i=1 ; i<=10 ; i++) {  
    Console.WriteLine(2*i);  
}
```

```
for (int i=2 ; i<=20 ; i = i+2) {  
    Console.WriteLine(i);  
}
```

```
for (int i=1 ; i<=1 ; i++) {  
    Console.WriteLine("2 4 6 8 10 12 14 16 18 20");  
}
```

```
for (int i=2 ; i<=20 ; i++) {  
    if (i%2==0) // is i even?  
    {  
        Console.WriteLine(i);  
    }  
}
```

Part 6: Indlejrede Loops

Indlejrede Loops ▷ Eksempel

Da loops selv er statements, kan vi også *indlejre* (eng: neste) dem.

```
for (int y=1 ; y<=10 ; y++) {  
    for (int x=1 ; x<=y ; x++)  
    {  
        Console.Write(string.Format("{0,4}", x*y));  
    }  
    Console.WriteLine("");  
}
```

Indlejrede Loops ► Resultat

Da loops selv er statements, kan vi også *indlejre* (eng: neste) dem.

1										
2	4									
3	6	9								
4	8	12	16							
5	10	15	20	25						
6	12	18	24	30	36					
7	14	21	28	35	42	49				
8	16	24	32	40	48	56	64			
9	18	27	36	45	54	63	72	81		
10	20	30	40	50	60	70	80	90	100	

Part 7: Flow Control

Flow Control ▷ Break

`break` afslutter loopet og hopper til det første statement umiddelbart efter loopet.

Flow Control ▷ Break

`break` afslutter loopet og hopper til det første statement umiddelbart efter loopet.

```
int i;  
for (i=0 ; i<10 ; i++) {  
    if (i==5)  
    {  
        break;  
    }  
}  
Console.WriteLine(i);
```

Flow Control ▷ Break

`break` afslutter loopet og hopper til det første statement umiddelbart efter loopet.

```
int i;  
for (i=0 ; i<10 ; i++) {  
    if (i==5)  
    {  
        break;  
    }  
}  
Console.WriteLine(i);
```

Resultat: 5

Flow Control ► Continue

`continue` springer over den resterende del af body for dette gennemløb.

Flow Control ▷ Continue

`continue` springer over den resterende del af body for dette gennemløb.

```
int i;  
int sum = 0;  
for (i=0 ; i<4 ; i++) {  
    if (i==2) {  
        continue;  
    }  
    sum += i;  
}  
Console.WriteLine(sum);
```


Flow Control ▷ Continue

`continue` springer over den resterende del af body for dette gennemløb.

```
int i;  
int sum = 0;  
for (i=0 ; i<4 ; i++) {  
    if (i==2) {  
        continue;  
    }  
    sum += i;  
}  
Console.WriteLine(sum);
```

Resultat: 4

Spørgsmål?

