

## Kapitel 4 Løkker i C#

Løkker en vigtig del af alle programmeringssprog, og C# er ikke andeles. En løkke er en måde at udføre en del af koden gentagne gange. Ideen er at du fortsætter med at udføre en opgave indtil en betingelse er opfyldt. Kun da vil løkken blive brudt. Et eksempel! Antag at du vil lægge alle tallene fra 1 til 10 sammen. Du kan gøre det på følgende måde:

```
int answer;
```

```
answer = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10;
```

Det vil være en måde at gøre det hvis du kun har 10 tal. Men forestil dig du har tusindvis af tal? Du vil i hvert fald ikke skrive dem alle sammen! I stedet for vil du gøre brug af en løkke, der gentagende gange vil lægge tal til.

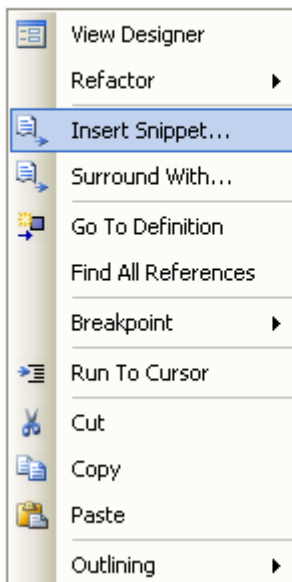
### For løkker i C#

Den første type af løkke vi skal se på er en for løkke. Andre typer af løkke er do løkken og while løkken, som du snart møder. Men for løkken er den mest almindelige type af løkke du har brug for. Lad os bruge den til at lægge tallene fra 1 til 100 sammen.

### Øvelse 21

Start et nyt projekt ved at vælge **File** og **New Project**. Tilføj en knap til din nye formular. Dobbeltklik på knappen for at se koden. For hurtigt at tilføje en stump kode til en løkke, kan du højre klikke et vilkårligt sted mellem de krøllede parenteser i knappens kode. I menuen vælger du **Insert Snippet**:

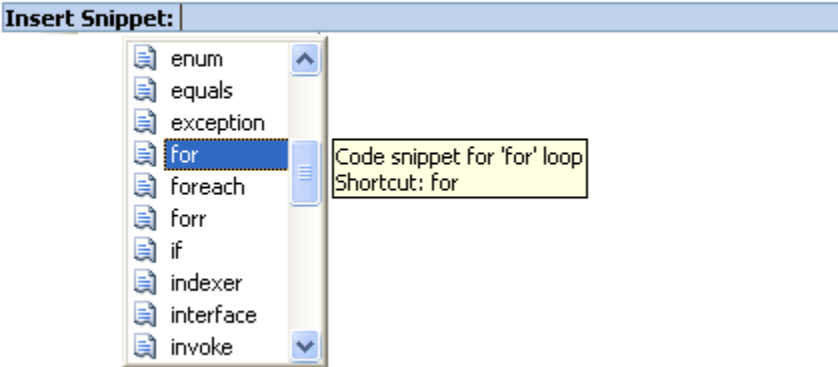
```
private void button1_Click(object sender, EventArgs e)
{
    |
}
```



Når du klikker på Insert Snippet, vil du få følgende liste af muligheder:

```
private void button1_Click(object sender, EventArgs e)
{

```



Rul ned og klik på **for**. Du får nu tilføjet noget kode:

```
private void button1_Click(object sender, EventArgs e)
{
    for (int i = 0; i < length; i++)
    {
    }
}
```

Det ser noget kompliceret ud, så vi gennemgår det lige. Her er en for løkken uden noget imellem de bløde parenteser:

```
for ( )
{
}
```

Du starter med ordet for, efterfulgt af et par bløde parenteser. Det du fortæller C# mellem de bløde parenteser er hvor mange gange du vil have løkken skal gentages. Efter de bløde parenteser, sætter du et par krøllede parenteser. Den kode du vil have gentaget skrives mellem de krøllede parenteser.

Standard koden som C# indsætter til dig er:

```
int i = 0; i < length; i++
```

Der er tre dele i mellem de bløde parenteser:

1. Hvilket tal vil du starte med?
2. Hvor mange gange vil du have løkken gentaget?
3. Hvordan du have opdateret hver gang løkken gentages?

Bemærk at hver af de tre dele er adskilt af et semikolon. Her er den første del:

```
int i = 0; i < length; i++
```

Og her er den anden del:

```
int i = 0; i < length; i++
```

Og her er den tredje del:

```
int i = 0; i < length; i++
```

Den første del på listen ovenfor (Hvilket tal vil du starte fra?) er dette:

```
int i = 0;
```

Det standard koden gør er at den sætter en integer variable i op (et populært navn til for løkker.) Den tildeler da værdien 0 til variabelen i. Den vil bruge den værdi for i som start værdien for løkken. Du kan også angive start variabelen uden for koden, hvis du foretrækker det. Det ser sådan ud:

```
int i  
for (i = 0; i < length; i++)  
{  
}
```

Den anden del på listen (hvor mange gange skal løkken køre?) er dette:

```
i < length;
```

Dette siger, hvis du kan huske den betinget logik fra tidligere, at "i er mindre end length". Length er ikke et nøgleord. Du har derfor brug for at definere en variabel der hedder length, eller erstatte ordet length med et tal. Så enten dette:

```
for (int i = 0; i < 101; i++)
```

Eller dette:

```
int length = 101;  
for (int i = 0; i < length; i++)  
{  
  
}
```

i det første eksempel har vi bare skrevet i < 101. I det andet eksempel har vi defineret en variabel, der hedder length, og gemt værdien 101 i den. Vi sammenligner derefter de to variabler, og undersøger om i er mindre end length:

```
i < length;
```

Hvis `i` er mindre end `length`, vil betingelsen ikke være opfyldt og C# vil fortsætte løkken. Med andre ord "Fortsætte med at køre i ring så længe `i` er mindre end `length`."

Du behøver ikke at kalde variabelen for `length`. Det er bare et variabelnavn, så du kan selv finde på noget. For eksempel:

```
int endNumber = 101;

for (int i = 0; i < endNumber; i++)
{

}
```

her kalder vi variabelen for `endNumber` i stedet for `length`. Den anden del siger "Fortsæt løkken så længe `i` er mindre end `endNumber`".

Den tredje del på listen (hvordan vil du opdatere hver gang du gentager løkken?) så sådan ud:

**`i++`**

Den sidste del i en for løkke kaldes for Update Expression. I de to første dele, angiver du en start værdi og en slut værdi til løkken. Men C# ved ikke hvordan den skal komme fra det ene tal til det andet. Du er nødt til at fortælle hvordan den kommer dertil. Ved at skrive `i++`, lægger du 1 til værdien af `i` hver gang løkken gentages. Dette:

**`variable_name++`**

Er en kort måde at sige:

**`variable_name = variable_name + 1`**

Det eneste du gør, er at du lægger 1 til hvad der nu er i forvejen i variabelen. Da du er i en løkke, vil C# fortsætte med at lægge 1 til værdien af `i` hver gang løkken gentages. Den stopper for med at lægge 1 til når den sidste betingelsen er nået (når `i` ikke længere er mindre end `length`).

For lige at gentage, så skal du bruge en startværdi til løkken, hvor mange gange den skal gentages og hvordan den kommer fra det ene tal til det andet.

De tre dele er:

for (**Start\_Value; End\_Value; Update\_Expression**)

### *Øvelse 22*

Ok, nu skal vi prøve vores teori af i praksis. Indtast følgende kode til din knap:

```
private void button1_Click(object sender, EventArgs e)
{
    int answer = 0;

    for (int i = 1; i < 101; i++)
    {
        answer = answer + i;
    }

    MessageBox.Show(answer.ToString());
}
```

Den egentlige kode til løkken er den der står mellem de krøllede parenteser:

**answer = answer + i;**

Det er sandsynligvis den del der driller meste i en for løkke – det at vide hvad du skal skrive af kode! Husk på hvad det er du vil: tving C# til at udføre en del af koden et antal gange. Vi vil have lagt tallene fra 1 til 100 sammen, og vi bruger en variable med navnet **answer** til at opbevare resultatet på vores addition. Da værdien i **i** forøges hver gang med en når løkken gentages, kan vi bruge den værdi i vores sammenlægning. Her er værdien første gange løkken køres:

**answer = answer + i;**

0 =      0 +    1

Den anden gang løkken køres:

**answer = answer + i;**

1 =      1 +    2

Den tredje gang løkken køres:

**answer = answer + i;**

3 =      3 +    3

Og den fjerde:

**answer = answer + i;**

6 =      6 +    4

Bemærk hvordan værdien af **i** forøges hver gang løkken køres. Hvis du først udfører additionen efter lighedstegnet giver det mere mening (Som en ekstra øvelse kan du svare på hvad den femte omgang af løkke giver?)

Kør dit program og klik på knappen. Meddelelsesboksen skulle gerne give svaret 5050.

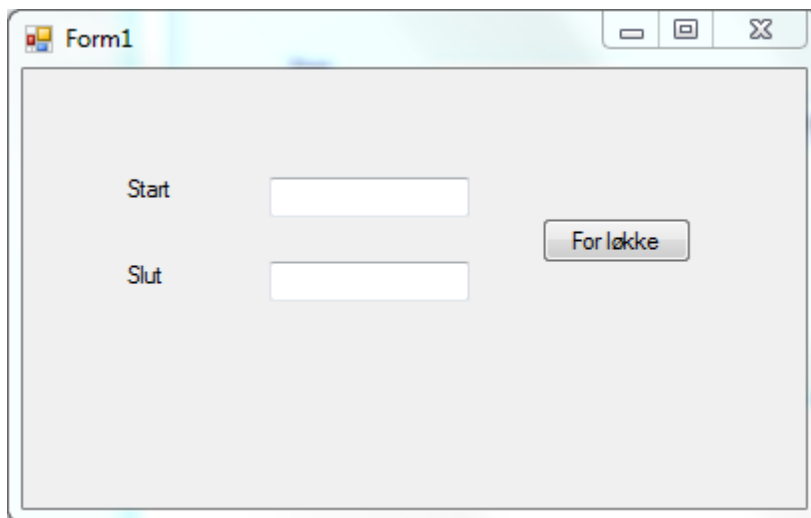
I den næste del skal vi se nærmere på løkkens startværdi og slutværdien i løkken.

### For løkkens start og slut værdier

I koden fra det forrige eksempel skrev vi start og slut værdien til for løkken. Du kan dog også hente disse værdier fra tekstbokse.

#### Øvelse 23

Tilføj to tekstbokse til din formular. Tilføj endvidere et par labels. I den første label skriver du **Start**. I den anden label skriver du **Slut**. Din formular vil nu ligne denne vist nedenfor:



Vi vil nu tage start og slut værdierne fra tekstboksene. Derefter vil bruge værdierne i vores for-løkke.

Dobbeltklik på din knap for at se koden (eller tast F7). Definer to variabler, der indeholder værdierne fra tekstboksen:

```
int loopStart;  
int loopEnd;
```

Gem nu tallene fra tekstboksene i de to nye variabler:

```
loopStart = int.Parse(textBox1.Text);  
loopEnd = int.Parse(textBox2.Text);
```

Nu da vi har værdierne fra tekstboksene kan vi bruge dem i for løkken. Ændre din for løkke til dette:

```
for (int i = loopStart; i < loopEnd; i++)  
{  
    answer = answer + i;  
}
```

Det eneste vi har ændret er den del mellem de bløde parenteser. Den første del er nu ændret fra:

```
int i = 1
```

til dette:

```
int i = loopStart
```

I stedet for at gemme værdien 1 i variabelen der hedder *i*, kan vi opbevare hvad vi har lyst til i variabelen der hedder *loopStart*. Det du taster i den første tekstboks bruges nu som for løkkens startværdi.

I den anden har vi ændret:

```
i < 101
```

til dette:

```
i < loopEnd
```

Vi bruger værdien der er gemt i **loopEnd**. Vi fortæller C# at den skal fortsætte løkken hvis værdien i variabelen *i* er mindre end **loopEnd**. (Husk at da vores Update Expression er **i++**, vil C# fortsætte med at lægge 1 til værdien af *i* hver gang for løkken kører. Når *i* ikke længere er mindre end **loopEnd**, vil C# stoppe.)

Kør dit program og tast 1 i den første tekstboks og 10 i den anden tekstboks. Klik på knappen. Din meddelelsesboks skulle gerne vise resultatet 45.

Kan du se problemet? Hvis du lægger tallene fra 1 til 10 sammen er svaret forkert! Det skulle gerne være 55 og ikke 45. Kan du se hvorfor 45 vises i meddelelsesboksen og ikke 55? Hvis du ikke kan så stop op et øjeblik og tænk over hvorfor.

(Der er slevfølgelig et andet problem. Hvis du ikke taster noget i tekstboksen vil dit program crashe! Det gør det fordi C# ikke kan konvertere tallet fra tekstboksen og gemme det i variabelen. Man kan trodsalt ikke konvertere noget der ikke findes! Vi ser på hvordan vi løser dette problem i slutningen af kapitlet.)

I det næste afsnit vil vi bruge vores formular til at lave et gangetabels program i C#.

## Et gangetabels program i C#

Vi kan nu skrive et lille gangetabels program. Vi vil bruge tekstboksene til at bede brugeren om at indtaste start- og slutværdier. Vi vil bruge disse værdier til at vise en 10 tabel. Hvis brugeren taster 1 i den første tekstboks og 5 i den anden tekstboks, vil vi få vist:

```
1 gange 10 = 10
```

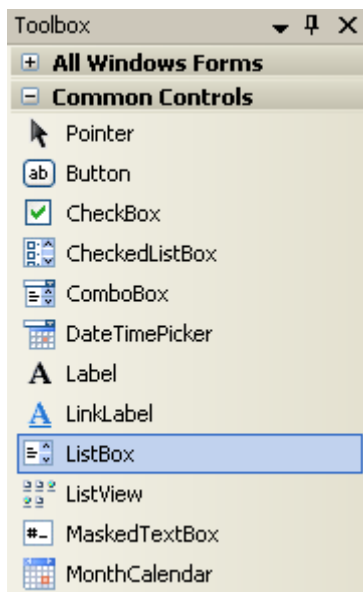
```
2 gange 10 = 20
```

```
3 gange 10 = 30
```

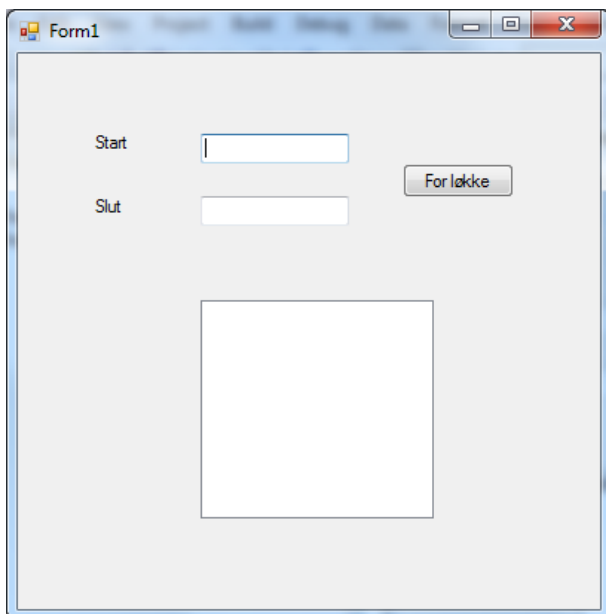
```
4 gange 10 = 40
```

```
5 gange 10 = 50
```

I stedet for at bruge en meddelelsesboks til at vise resultatet vil vi bruge en listeboks. En listeboks bruges til, ikke overraskende, at vise en liste af elementer. Men det er nemmere at vise hvad det er vi skal, i stedet for at forklare det. Brug Toolbox til at tilføje en listeboks til din formular:



Ændre størrelsen på din listeboks og din formular så det ligner den formular vist nedfor:



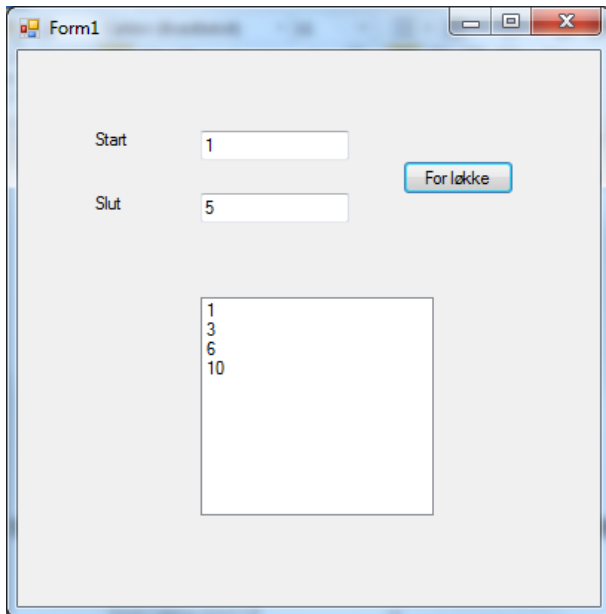
Dobbeltklik på knappen for at se koden. Tilføj denne linje til din forløkke (linjen er vist med fed nedenfor);

```
for (int i = loopStart; i <= loopEnd; i++)  
{  
    answer = answer + i;  
    listBox1.Items.Add( answer.ToString() );  
}
```



Du starter med at skrive navnet på din listeboks (her **listBox1**). Efter punktummet ser du IntelliSense dukke op. Vælg **Items** på liste. **Items** er en betingelse for listeboksen. Det refererer til de elementer der er på listen. Efter ordet **Items**, vil metoden **Add** tilføje elementer til din listeboks. Mellem de bløde parenteser taster du det du vil have tilføjet til liste. I vores tilfælde er det bare **answer**, konverteret til en string.

Du kan slette linjen med din meddelelsesboks, hvis du vil, da du ikke skal bruge den mere. Kør dit program og tast 1 i den første tekstboks og 5 i den anden tekstboks. Klik på knappen og din formular skulle nu gerne se sådan ud:



Det er meningen at programmet skal lægge tallene fra 1 til 5 sammen, eller de tal der nu er tastet ind i tekstboksene. Listeboksen viser dig et svar, for hver gang løkken kører. Men det viser dog kun 4 tal. Hvis du løste problemet med hvorfor du fik svaret 45 og ikke 55, vil du allerede vide hvorfor du kun har fire tal på din liste i listeboksen. Hvis ikke skal du undersøge den første linje i for løkken:

**for (int i = loopStart; i < loopEnd; i++)**

Problemet er den anden del i løkkens kode:

**i < loopEnd**

Vi fortæller C# at køre i ring mens værdien for **i** er mindre en **loopEnd**. C# stopper løkken når værdierne er de samme. Værdien i **loopEnd** er 5 i vores lille program. Vi fortæller C#, "Forsæt løkken så længe værdien for **i** er mindre en 5. Stop for løkken hvis det er 5 eller mere."

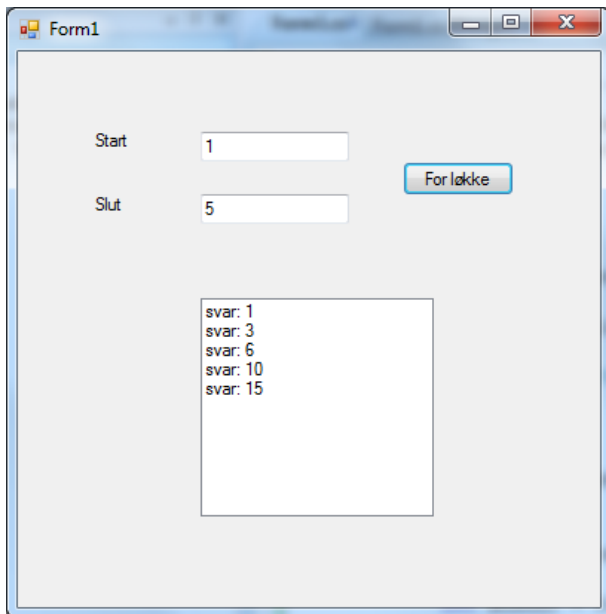
Det er klart at vi brugt den forkerte betingede operator. I stedet for at bruge mindre end, skal vi bruge... ja hvilken skal vi bruge? Erstat symbolet **<** med det rigtige.

### Øvelse 24

Hvis du vil have lidt ekstra point, kan du da finde en anden løsning på problemet? En hvor du kan bevare mindre end symbolet?

Hvis du vil tilføje noget mere information til din listeboks, kan du skifte kode til følgende:

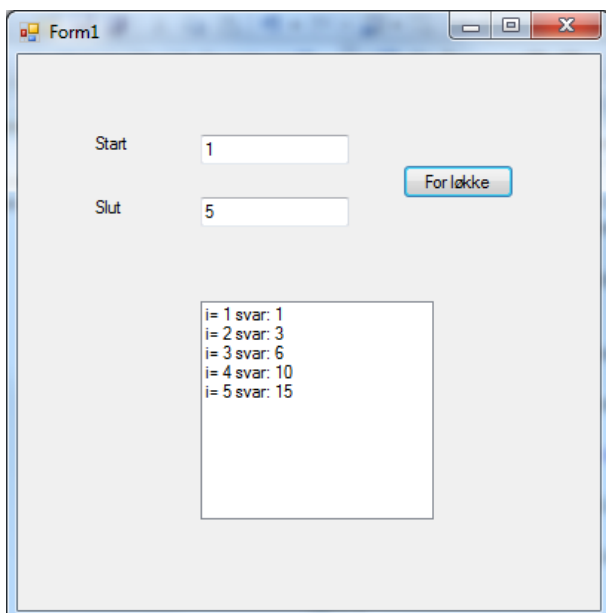
```
listBox1.Items.Add( "svar = " + answer.ToString( ) );
```



For at gøre det endnu mere klart, kan du tilføje noget mere tekst til din listeboks. Prøv dette:

```
listBox1.Items.Add( "i = " + i + " svar = " + answer.ToString( ) );
```

Kør dit program og din listeboks ser nu sådan ud:



Vi har nu tilføjet værdien af variabelen *i*. Det gør det mere klart hvordan værdien af *i* ændre sig hver gang løkken kører.

## Gangetabel programmet

Vi har nu alle ingredienserne der skal til for at vi kan skrive vores gangetabelprogram.

Vend tilbage til dit kodevindue. Det vi skal bruge nu er en for løkke der skal udregne og vise 10 tabellen. Vi har brug for endnu en variable, der skal opbevare tallet 10. Tilføj følgende variabel:

```
int multiplyBy = 10;
```

Det eneste vi skal nu er at vi skal ændre på koden mellem de krøllede parenteser i for løkken. Lige nu har vi:

```
answer = answer + i;  
listBox1.Items.Add("i = " + i + " svar = " + answer.ToString());
```

Slet disse linjer og erstat dem med disse to:

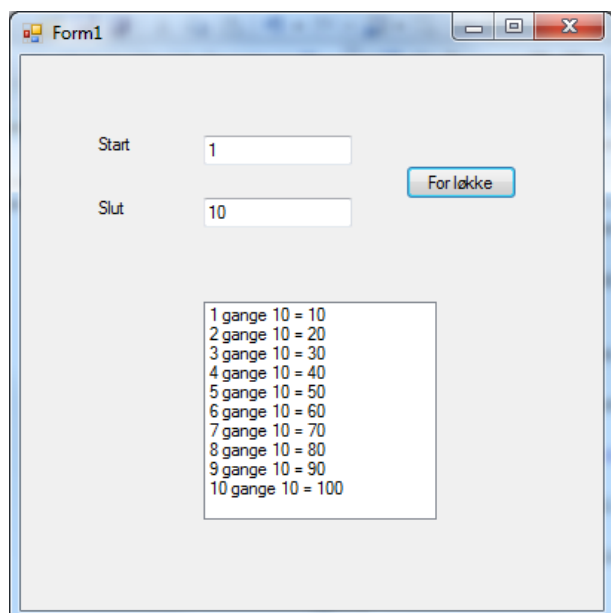
```
answer = multiplyBy * i;  
listBox1.Items.Add(i + " gange " + multiplyBy + " = " + answer.ToString());
```

Listeboksen ser lidt rodet ud, men prøv at undersøge delen mellem de bløde parenteser:

```
i + " gange " + multiplyBy + " = " + answer.ToString()
```

Det er bare en kombination af variabelnavne og direkte tekst. Den første linje med kode er en simple multiplikation. Vi ganger det der i variabelen multiplyBy (som her er 10) med det der er i variabelen i. C# lægger 1 til værdien af i hver gang løkken kører, så answer bliver opdateret og vises i listeboksen.

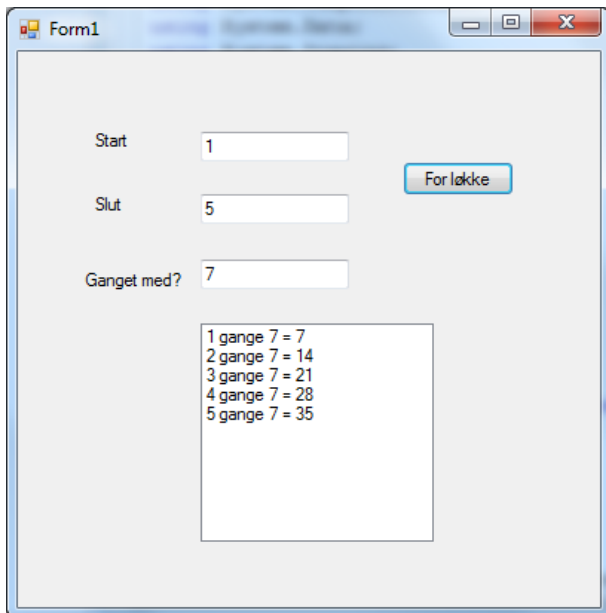
Kør dit program. Tast 1 i den første tekstboks og 10 i den anden tekstboks. Klik på knappen for at se din gangetabel:



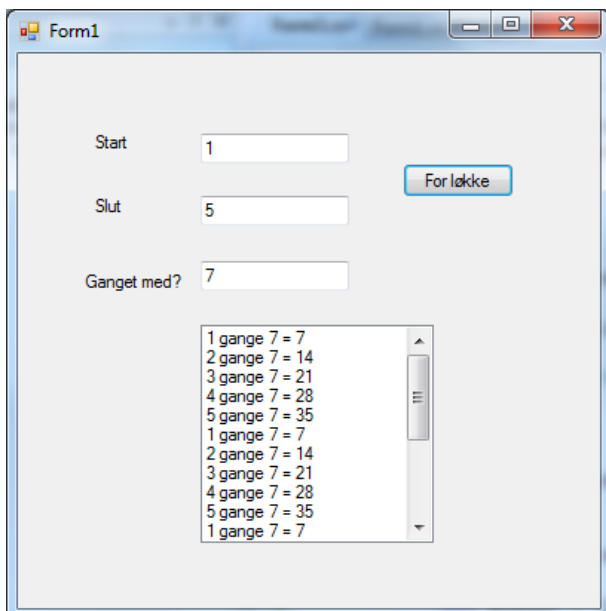
Med nogle få linjers kode og med hjælp fra en for løkke, har vi lavet et program der faktisk laver en hel del. Forestil dig hvor besværligt det ville have været hvis ikke vi havde brugt vores løkke.

### Øvelse 25

Lige nu ganger vi med 10 og viser derfor 10 tabellen. Tilføj endnu en tekstboks til din formular. Tilføj et label der spørger hvilken gangetabel de ønsker. Brug denne værdi i din kode til at vise denne gangetabel. Hvis brugeren taster 7 i den nye tekstboks, vil vi få vist 7 tabellen når der klikke på knappen. Sådan skal dit program se ud når du er færdig:



Når du afslutter denne øvelse, kan du prøve at klikke knappen nogle få gange. Du vil bemærke (så længe du har tal i tekstboksene) at listeboksen ikke rydder sig selv. Du vil se dette i din listeboks:



Den nye information bliver bare tilføjet til slutningen. For at løse dette kan du tilføje følgende linje hvor som helst for din løkke, men indenfor knappens kode.

```
listBox1.Items.Clear( );
```

i stedet for at bruge metoden Add(), bruger du metoden Clear(). Dette vil rydde listen for elementer. Kan du se hvorfor denne linje med kode ikke vil være en god ide at skrive inden i løkken? Eller efter løkken?

Nok om **for** løkker. Vi vil nu kort se på to andre typer af løkker: do løkken og while løkken.

## Do og While løkker i C#

På samme måde som en **for** løkke kan gentage en kode for dig, kan **Do** og **While** løkke også gentage en kode. Vi starter med Do løkken.

### Do løkker i C#

Uanset hvilken løkke du vælger er ideen den samme: kør i ring og udfør den samme kode indtil en given betingelse er opfyldt. Forskellen på Do og While løkken er strukturen. Her ser du opbygningen af Do løkken:

```
do
{

} while (true);
```

Bemærk semikolonnet i koden ovenfor. Det kommer i slutningen, efter de krøllede parenteser. Men du starter med ordet **do**, efterfulgt af et par krøllede parenteser. Efter de krøllede parenteser skriver du ordet **while**. Efter while og i mellem et par bløde parenteser skriver du betingelsen. C# vil køre løkken igennem indtil betingelsen mellem de bløde parenteser er opfyldt. Først da vil den bryde ud af løkken. Her er et eksempel, der bruger vores gangetabel:

```
do
{
answer = multiplyBy * i;
listBox1.Items.Add(answer.ToString());
i++;
} while (i <= loopEnd);
```

Denne gang har vi altså brugt Do løkken istedet for For løkken. Løkken vil fortsætte indtil værdien i variablen i er mindre end eller lig med værdien i variablen loopEnd. Den anden ting man skal bemærke er at vi selv skal forøge (lægge en til) værdien af i (i++). Vi gør det hver gang løkken kører. Hvis vi ikke forøgede værdien i i ville den altid være mindre end loopEnd. Vi ville da have en uendelig løkke, og programmet ville crashe. Men det vi rent faktisk siger:

**”Fortsæt med afvikle koden mellem de krøllede parenteser så længe i er mindre end eller lig med loopEnd.”**

### While løkker i C#

While løkker ligner meget Do løkker i strukturen. Her er en While løkke:

```
while (true)
{

}
```

Og her er koden til gangetabellen igen:

```
while (i <= loopEnd)
{
    answer = multiplyBy * i;
    listBox1.Items.Add(answer.ToString());
    i++;
}
```

While løkker er nemmere at bruge end Do løkker. Hvis du ser på koden ovenfor kan du se at While delen står i starten, i stedet for i slutningen som en Do løkke. Koden du skal afvikle står stadig mellem de krøllede parenteser. Og du skal stadig kunne stoppe løkken (i++).

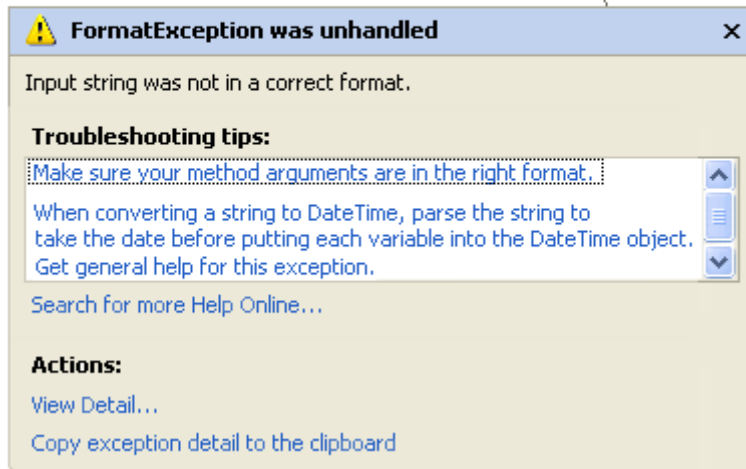
Forskellen på de løkker er at koden i Do løkken bliver udført mindst en gang, fordi while delen er i slutningen. Med while delen i begyndelsen, vil din slut-betingelse i de bløde parenteser have mulighed for allerede at være opfyldt (true) i er måske større end loopEnd. I det tilfælde vil C# afbryde med det samme, og koden mellem de krøllede parenteser vil ikke blive udført.

Det kan være meget svært at afgøre hvilken løkke du skal bruge. Men du skal ikke bekymret hvis du ikke helt har forstået det endnu. Du vil få en masse øvelse efterhånden som du kommer igennem disse noter. Men husk at løkker er svære at blive fortrolig med, og du skal ikke tabe humøret hvis du ikke helt mestre dem endnu. Vi vil forlade dette komplekse emne nu og slutte kapitlet af med et problem og en løsning.

### Undersøg blanke tekstbokse i C#

Der er et problem med tekstboksen i vores tabelprogram. Hvis du ikke taster noget i tekstboksen, vil programmet crashe. Prøv det. Start dit program og lad en af tekstboksene være tom. Klik på knappen. Du skulle nu gerne få en underlig og ikke særlig hjælpsom fejlmeddelelse:

```
int answer = 0;
int loopStart = 0;
int loopEnd;
int multiplyBy = int.Parse(textBox3.Text);
```



C# fremhæver den skyldige linje med gult. Det gør dette fordi det ikke kan konvertere tal fra en tekstboks og gemme dem i en variabel. Du kan trods alt ikke forvente at den kan konvertere noget der ikke findes. For at kurere dette kan du bruge en metode der hedder **TryParse**.

For at konvertere et tal fra en tekstboks til et heltal, har du tidligere brugt:

```
loopStart = int.Parse(textBox1.Text);
```

Du har altså analyseret tallet i tekstboksen, og ændrer det til et heltal int. Men det fungerer ikke med tomme tekstbokse, og det vil heller ikke blive undersøgt om der er indtastet noget overhovedet, som f.eks. ordet tre i stedet for tallet 3. Det vi skal gøre er at vi skal prøve (try) at analysere (parse) tallet i tekstboksen. Du spørger derfor C# om den kan konvertere indholdet til et tal. Hvis det ikke kan lade sig gøre kan du vise en fejlmeddelelse til brugeren. Her er noget kode der prøver at konvertere noget data fra den første tekstboks. Det er lidt kompliceret, så vi gennemgår det lige:

```
int outputValue = 0;
bool isNumber = false;

isNumber = int.TryParse(textBox1.Text, out outputValue);
if (!isNumber)
{
    MessageBox.Show("Tast et tal i tekstboksen!");
}
else
{
    //Resten af koden kommer her
}
```

De første to linjer definerer et par variabler, et heltal og en Boolesk. Variablen **outputValue** skal bruges af **TryParse**. Du prøver at udtrække en Boolesk værdi (true eller false) OG en string bestående af tekst:

```
isNumber = int.TryParse(textBox1.Text, out outputValue);
```

Variablen `isNumber` vil enten være `true` eller `false`, afhængigt af om C# kan konvertere teksten i tekstboksen til et heltal (`int`). Hvis du prøvede at konvertere en `double` variable ville din kode se sådan ud:

```
double outputValue = 0;
bool isNumber = false;

isNumber = double.TryParse(textBox1.Text, out outputValue);
```

Output værdien du skal bruge nu er en `double` (Du undersøger om C# kan konvertere til en `double` værdi). Værdien i `isNumber` vil stadig være enten `true` eller `false` (kan der konverteres eller ej).

Efter du har brugt `TryParse`, skal du undersøge om værdien er `true` eller `false`:

```
if (!isNumber)
{
    MessageBox.Show("Indtast et tal!");
}
else
{
    //Resten af koden kommer her
}
```

Hvis du kan huske afsnittet om betinget operatorer vil du vide at linjen:

**if (!isNumber)**

læses:

**if NOT true**

hvis du foretrækker det kan du også skrive denne linje som:

**if (isNumber == false)**

Linjen læses nu som:

**"Hvis isNumber har værdien false"**