Fejlhåndtering

Selv de bedste programmører laver af og til fejl! Dette kommer sikkert som en overraskelse for de fleste, bortset fra de, der har arbejdet med et hvilket som helst større program ☺.

Fejl kan være af mange forskellige slags, men nogle af dem er nemmere at håndtere end andre. Det skyldes ikke mindst at VBA editoren er god til at hjælpe med at undgå en række fejltyper, fx syntaktiske fejl i koden, brug af ikke erklærede variabler mm. Desuden indeholder editoren en række værktøjer, som gør det nemmere at finde fejl.

Der er dog også fejltyper som ingen editor kan hjælpe med. Dette kan være fejl, hvor koden egentlig virker, altså kører uden problemer; den gør bare ikke det, man havde forventet. En anden type fejl er de, som opstår fordi en bruger af koden, ikke gør som der forventes. Den første af de sidstnævnte kan kun findes via test og atter test af koden i alle mulige (og umulige) sammenhænge. Den sidste kan vi ikke teste og ud, og er derfor nødt til at programmere funktioner ind i vores kode, der tager højde for den slags fejl.

Fejlfinding af koden

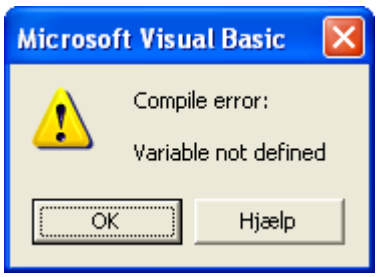
Lad mig starte med et par eksempler på, hvordan man kan finde – og rette fejl i sin kode. For at sikre at alle variabler bliver erklæret, se artiklen om variabler og konstanter, kan man sætte denne linje ind først i sit modul

```
Option Explicit
```

Den fortæller VBA editoren, at der kun må anvendes variabler, der eksplicit er erklæret med en Dim sætning. I flere af eksempler under Makroer, anvender jeg såkaldt implicit variabelerklæring; det vil sige at jeg anvender en variabel uden at erklære den først. Det er i virkeligheden dårlig programmering, da alle variabler så bliver erklæret som datatypen Variant, men det er hurtigere, når det skal gå stærkt. Starter jeg derimod modulet med ovenstående sætning, tvinger systemet mig til at erklære alle mine variable, inden de bruges. Glemmer jeg det alligevel, fx som her:

```
Sub Fejl1()
 txt = ActiveCell.Value
End Sub
```

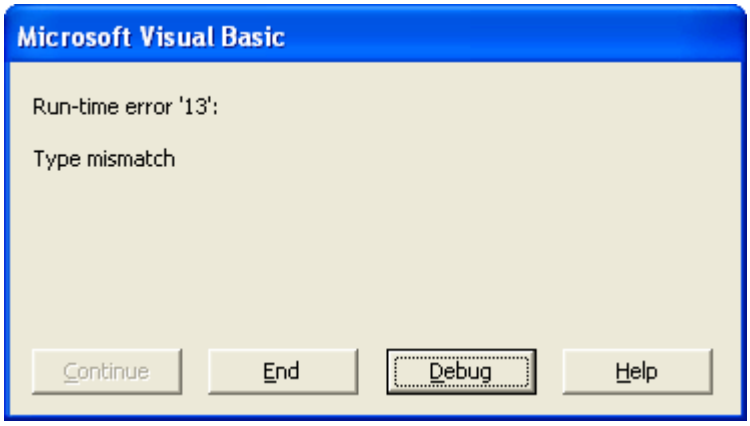
vil jeg blive mindet om det af VBA allerede mens jeg skriver koden, da der er tale om en såkaldt kompileringsfejl:



Dette vil være bedre:

```
Sub Fejl1()
Dim txt As Integer
    txt = ActiveCell.Value
End Sub
```

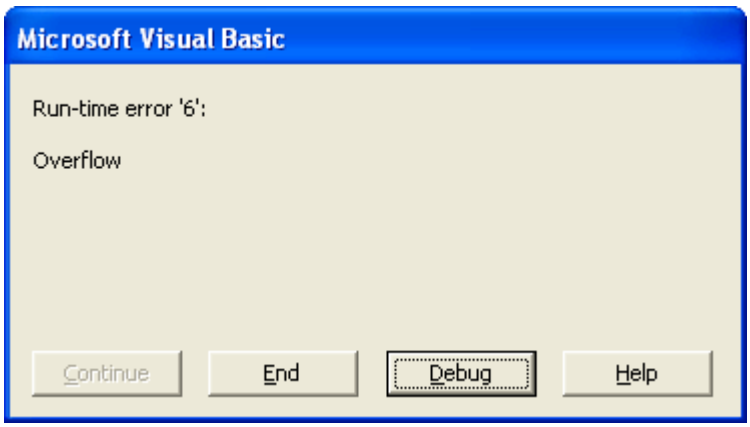
med mindre værdien i cellen selvfølgelig ikke passer til den valge datatype. Så får jeg (eller brugeren) **DET** at vide. Men først når koden afvikles, en såkaldt kørselesfejl eller *Run-time error*.



Jeg kan så klikke på knappen Debug, og få fremhævet den linje i koden, der har fejlet:

```
Sub Fejl1()
Dim txt As Integer
    txt = ActiveCell.Value
End Sub
```

I dette tilfælde står der en tekst i cellen og den kan ikke rummes i en variabel med datatypen Integer, der jo er beregnet til tal. Jeg må nu afgøre om variablen skal erklæres om så den kan rumme tekst i stedet, eller om den (som Variant) skal kunne rumme både tekst og tal. Skal den kun kunne rumme tal, som oprindeligt angivet, skal jeg nok af kodet en fejlrutine (se nedenfor), så brugeren ikke pludselig ender i VBA editoren med ovenstående billede vist, hvis han på trods af alle gode intentioner, alligevel har tastet noget "forkert" i cellen. Når forkert her sættes i anførselstegn er det fordi, det kun er for koden, det er forkert. Det kan være helt rigtigt set med brugerens øjne. I mit tilfælde vælger jeg, at lade variablen forblive som den er. Fejlrutinen koder jeg først senere ☺. I stedet instruerer jeg i første omgang brugeren om, at der altså skal stå et tal i den aktive celle, når koden afspilles. Så to minutter efter har brugeren i telefonen igen: Han står igen i editoren, dog nu med denne fejlmeddelelse:



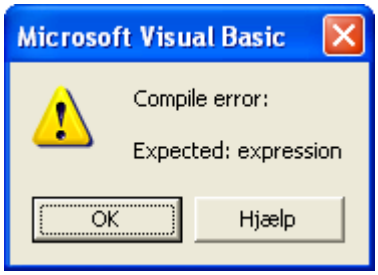
"Overflow" betyder her, at tallet i cellen ikke kan rummes i datatypen. En Integer rummer kun tal mellem -32.786 og 32.767, så hvis tallet i cellen er større eller mindre end disse værdier opstår ovenstående fejl. Også denne må jeg tage højde for i min fejlhåndteringsrutine. Jeg kan selvfølgelig hjælpe brugeren til at taste det rigtige, ved at anvende regnearkets datavalideringsfunktion på cellerne, men der er grænserne for mulighederne her, så uden programmeret datavalidering og/eller fejlhåndtering er det dømt til at gå galt. Begge dele vender jeg tilbage til nedenfor.

Men lad mig starte med begyndelsen (eller det jeg mener er begyndelsen): fejl som VBA editoren opdager mens jeg koder.

Jeg er ved at skrive min kode og er nået til = efter txt, altså

```
Txt =
```

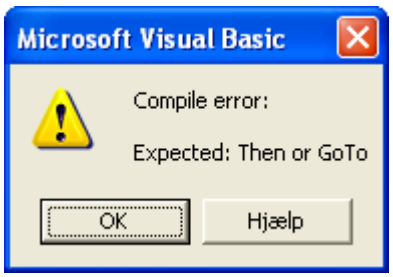
og så får jeg trykket linjeskift inden jeg har skrevet, hvilken værdi variablen skal tildeles. Også dette udløser en kompileringsfejl



En anden klassisk kompileringsfejl er fx

```
Sub Fejl1()
     Dim num1 As Byte, num2 As Byte
If num1 > num2
```

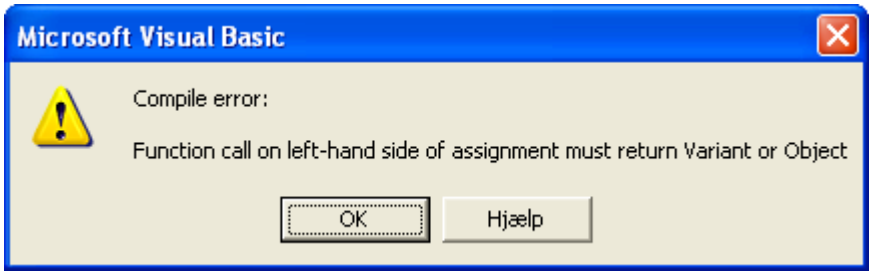
som giver



Hvis der trykkes lineskift inden der skrives Then eller GoTo. Skrives derimod

```
Sub Fejl1()
    Dim num1 As Byte, num2 As Byte
    num1 = 5
    num2 = 3
    If num1 > num2 Then
        MsgBox = "Olsen"
End Sub
```

opstår ingen kompileringsfejl undervejs i skrivningen, men først når koden afprøves:



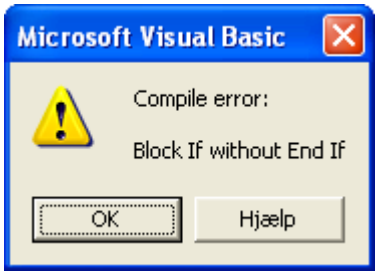
Denne meddelelse skyldes at jeg har skrevet

```
    MsgBox = "Olsen"
```

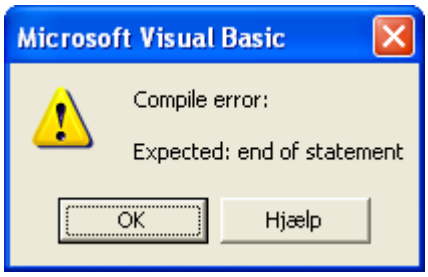
hvilket ikke er "legalt" i VBA, da den tror jeg vil tildele en værdi til meddelelsesboksen og det er ikke muligt. Det hedder bare

```
    MsgBox = "Olsen"
```

så det retter jeg, hvorefter jeg i stedet får denne fejl, når koden køres

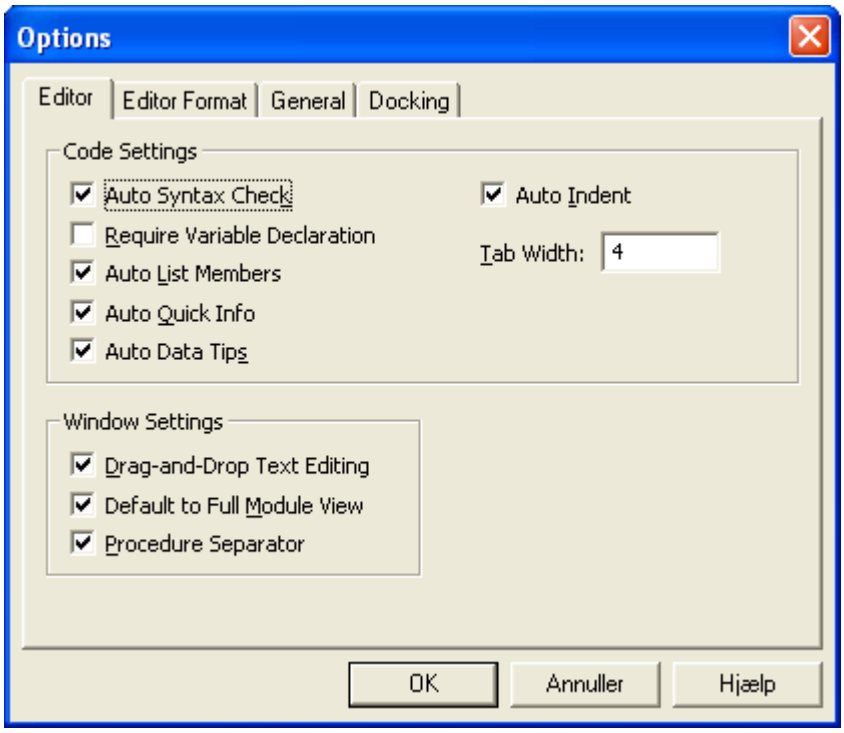


Glemmer jeg anførselstegn omkring "Olsen", tror det at der er tale om en variabel, og så får jeg den første fejlmeddelelse, der blev vist i denne artikel. Glemmer jeg kun det første anførselstegn, men husker det sidste, vises



Når jeg trykker linjeskift. Husker jeg det første, men glemmer det sidste anførselstegn, er VBA editoren flink, og sætter det selv for mig.

Jeg bliver altså hjulpet ganske fint undervejs af den indbyggede syntakskontrol, som kan slås til og fra under Tools – Options. Som standard er den slået til:



Her kan man også indstille en række andre forhold omkring editoren. Sætte man flueben i *Require Variable Declaration* skriver den selv Option Explicit øverst i alle nye moduler, der oprettes. De øvrige elementer vil jeg ikke gennemgå i denne forbindelse.

Kompileringsfejl vises altså typisk undervejs (med få undtagelser), mens kørselsfejl først vises, når koden afspilles. Det er i forbindelse med disse, samt de "logiske" fejl (de fejl hvor koden virker, men gør noget forkert) af afprøvning kommer ind.

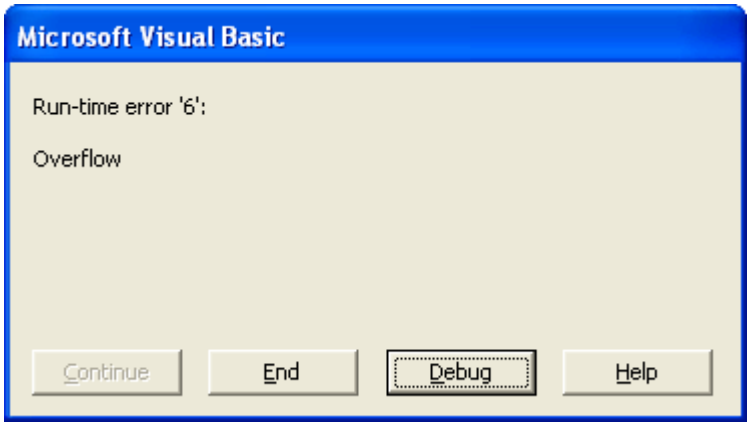
Kørselsfejl, der opdages mens man programmerer (på grund af programmeringsfejl) er ikke nødvendigvis nemme at rette, men de er nemme at finde. Det samme er ikke nødvendigvis gældende for de logiske fejl. Her kræver det typisk, at man udarbejder en form for testprogram med nogle kendte udfald af forskellige situationer, og så afprøver om koden giver de forventede udfald. Ideelt set skal man afprøve samtlige mulige situationer, som koden kan bringes i, man ved store programmer er dette simpelthen ikke muligt, med mindre man har råd til at ansætte en hær af testere. Et stort program rummer adskillige millioner linjer kode, og at få afprøvet alle krinkelkroge af disse, er næppe muligt. Når vi taler VBA er kodemængderne dog i almindelighed langt mindre, og deler man sine Subs op i passende små enheder, burde det være muligt at få aftestet alle muligheder. Til brug for fejlfinding af denne type har VBA editoren nogle hjælpemidler, men man kan også selv indsætte en form for fejlfiningsrutiner i sin kode. Hvad man foretrækker, er nok mest et spørgsmål om smag, men der kan være en fordel ved at indsætte i hvert fald nogen egenudvikling, da det kan hjælpe i en situation, hvor der fx skal gives telefonsupport.

Fejlfindingsværktøjer

VBA editoren har som nævnt nogle indbyggede værktøjer, der kan gøre fejlfinding nemmere. Man kan fx indsætte pausepunkter i sin kode, man kan køre koden linje for linje mm. Dem vil jeg ikke komme nærmere ind på i denne artikel, men vil i stedet koncentrere mig om, hvordan man kan håndtere de fejl, der uvægerligt vil opstå, fx fordi brugeren gør noget andet, end programmøren havde forventet mm.

Fejlhåndtering

Hver gang der opstår en kørselsfejl i en VNA procedure, udløses en fejlkode. Som fx her, hvor fejlen har nummer 6:



Denne fejlkode kan man fange i sit program, og så lade programmet reagere på fejlkoden. Man kan lave sådanne fejlhåndteringsfunktioner i de enkelte procedurer, men man kan også lave en generel fejlhåndteringsrutine, som gælder for alle procedurer. Jeg vil her se på begge muligheder.

Her har vi en kode, som på overfladen ser OK ud:

```
Sub Fejl2()
    Dim Ans As Integer
    Ans = InputBox("Indtast et tale mellem 1 og 20000")
    …
End Sub
```

Den viser en inputbox, hvor brugeren bliver bedt om, at taste et tal mellem 1 og 20.000. Så langt så godt. Prikkerne indikerer bare, at mere kode følger. Vi har erklæret variablen, som skal indeholde tallet som Integer (heltal), hvilket jo betyder, at den kan "rumme" tal op til over 32.000, så det burde være godt nok. Men hvis brugeren alligevel taster noget andet, fx et for stort tal eller en tekst, går det galt. Et for stort tal giver fejlkode 6 (Overflow) og en tekst vil give fejlkode 13 (Type mismatch). Da begge muligheder er indlysende bør vi tage højde for dem i koden. Det er her fejlhåndteringen kommer ind i billedet.

Fejlkoderne håndteres i et objekt som kaldes Err-Objektet. Det har nogle egenskaber, fx Number og Description. Mest interessant er i første omgang Err.Number. Men for overhovedet at kunne fange koden, skal vores fejlhåndteringsrutine startes, når der opstår en fejl. Dette gøres med udtrykket

```
On Error
```

On Error er en hændelse, der indtræffer, når en fejl opstår. Den er ikke nævnt under On-nøgleordet i artiklen om hændelser, da den KUN bruges til fejlhåndtering. Sætningen efterfølges af en instruktion om, hvad der skal gøres i tilfælde af fejl. Her er de forskellige muligheder:

```
On Error Resume
On Error Resume Next
On Error Goto
```

`On Error Resume` betyder at kodens afvikling fortsætter med samme linje, som fejlen opstod i, mens `On Error Resume Next` medfører at koden afvikles fra linjen efter den linje, hvor fejlen opstod. Den første anvendes så sjældent, at det grænser til aldrig. Den anden anvendes heller ikke ret tit, men der er en række situationer, hvor den er effektiv. Antag fx at man skrevet en kode, der skal lukke en projektmappe og derefter afslutte Excel.

```
Sub Luk_og_Afslut()
    Workbooks("Mappe2.xls").Close SaveChanges:=False
    Application.Quit
End Sub
```

Ovenstående kode lukker projektmappen Mappe2.xls uden at gemme eventuelle ændringer og afslutter Excel. Imidlertid vil den fejle i første linje, hvis mappen allerede ER lukket. I dette tilfælde ville det være relevant, at bruge Resume Next, så den bare fortsætter med at lukke Excel, selv om den relevante mappe, allerede er lukket. Her behøver man ikke at kommunikere en fejlmeddelelses til brugeren :"*Du skal åbne mappe2,.xls så den kan blive lukket igen".* Koden kunne derfor se således ud:

```
Sub Luk_og_Afslut()
    On Error Resume Next
    Workbooks("Mappe2.xls").Close SaveChanges:=False
    Application.Quit
End Sub
```

On Error GoTo giver mulighed for at springe til sin Fejl-rutine. Normalt anses GoTo udtryk i kode for dårlig programmering, men netop i forbindelse med fejlhåndtering, vil det være OK. Sætningen kunne fx lyde:

```
On Error GoTo Fejl
```

"Fejl" er en såkaldt label eller et linjemærke. I VBA skiver man disse som navnet på linjemærket, efterfulgt af et kolon, så der, hvor fejlrutinen skal stå, skriver man altså

```
Fejl:
```

Og så kommer koden, der håndterer fejlene. Læg ivørigt mlærke til, at linjemærker ikke kan indrykkes. De vil altid stå helt til venstre. Lad mig vende tilbage til det første eksempel

```
Sub Fejl2()
    On Error GoTo Fejl
    Dim Ans As Integer
    …
    Ans = InputBox("Indtast et tale mellem 1 og 20000")
Fejl:
    …
End Sub
```

Vi skal nu have skrevet fejlhåndteringstrutinen. Den nemme, men ikke særligt effektive løsning er, at vise brugeren de fejlmeddlelser, som kommer for VBA:

```
Sub Fejl2()
On Error GoTo Fejl
    Dim Ans As Integer
    Ans = InputBox("Indtast et tale mellem 1 og 20000")
Fejl:
    Msgbox "Fejl " & Err.Number & ": " & Err.Description
End Sub
```

Imidlertid er det ikke særligt sigende for brugeren, men vi opnår dog, at han ikke ender i et kode-debugging vindue. Bedre er det, at skrive noget kode, der håndtere de enkelte fejltyper. Er der bare en enkelt, eller et par stykker som her, kan det gøres med If…Then…Else sætninger, men er der flere er det mere effektivt at gøre det med Select Case.

```
Sub Fejl2()
On Error GoTo Fejl
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Ans = InputBox("Indtast et tale mellem 1 og 20000")
Fejl:
    Fejlnummer = Err.Number
    Select Case Fejlnummer
        Case Is = 6
            Msgbox "Du kan ikke indtaste så store tal. Værdien skal være mellem
1 og 20.000"
        Case Is = 13
            Msgbox "Du kan kun indtaste tal mellem 1 og 20.000"
    End Select
End Sub
```

Nu vil koden teste på hvilken fejl, der er opstået og såfremt det er 6 vil den fortælle at tallet er for stort, er det en tekst, vil den fortælle, at der kun kan indtastes tal. Rutinen lider dog af endnu et par mangler. For det første kan der måske opstå fejl, som man ikke har forudset og som man derfor ikke tester for. Derfor bør man altid have en Else sætning i sin fejlrutiner. Denne skal så håndtere de fejl, som vi ikke fik tænkt på i første omgang.

```
Sub Fejl2()
On Error GoTo Fejl
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Ans = InputBox("Indtast et tale mellem 1 og 20000")
    Exit Sub
Fejl:
    Fejlnummer = Err.Number
    Select Case Fejlnummer
        Case Is = 6
            Msgbox "Du kan ikke indtaste så store tal. Værdien skal være mellem
1 og 20.000"
        Case Is = 13
            Msgbox "Du kan kun indtaste tal mellem 1 og 20.000"
        Case Else
            Msgbox "Der er opstået en uforudset fejl"
    End Select
End Sub
```

Teksten i Case Else er måske ikke den mest geniale, men den er bedre end ingenting. Læg også mærke til, at jeg har indsat linjen

Exit Sub lige inden fejlrutinen. Det er for at forhindre at denne afspilles hver gang koden køres, også når der ikke opstår fejl. I det konkrete tilfælde vil brugeren ikke opdage, at koden køres, men i andre tilfælde vil man nok undre sig over at få vist fejlmeddelelser, selv om alting er gjort helt rigtigt.

Jamen så er vi jo klar, ikke? Nej ikke helt. I ovenstående tilfælde sker der ikke andet, end at brugeren får en fejlmeddelelse, som han kan klikke OK til, og så afsluttes koden, så brugeren kan starte forfra. Det er imidlertid ikke altid den mest hensigtsmæssige løsning. Så vi kan indbygge i vores kode, hvad der skal ske hvis…. Også her har vi forskellige muligheder, der ligner dem, som vi så for On Error sætningen.

```
Resume
Resume Next
Resume Linjemærke
End
```

`Resume` genoptager koden fra den linje, der fejlede. `Resume Next` fra linjen efter den fejlende linje, `Resume Linjemærke` fortsætter koden fra det specificerede linjemærke og `End` afslutter procedure. Den reviderede kode sikrer at så længe, der kun er tale om fejl 6 eller fejl 15, vil inputbocksen blive vist igen og igen indtil, der indtastes en "lovlig" værdi.

```
Sub Fejl2()
On Error GoTo Fejl
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Ans = InputBox("Indtast et tale mellem 1 og 20000")
    Exit Sub
Fejl:
    Fejlnummer = Err.Number
    Select Case Fejlnummer
        Case Is = 6
            Msgbox "Du kan ikke indtaste så store tal. Værdien skal være mellem 1 og 20.000"
            Resume
        Case Is = 13
            Msgbox "Du kan kun indtaste tal mellem 1 og 20.000"
            Resume
        Case Else
            Msgbox "Der er opstået en uforudset fejl"
            End
    End Select
End Sub
```

Imidlertid er det ikke helt koden ikke hel god endnu. Prøv fx at afspille den, og så klik på Annuller-knappen i stedet for på OK. Så vises

Og det bliver den ved med, til der faktisk er tastet en lovlig værdi. Vi må altså også tage højde for, at brugeren kan fortryde sin oprindelige start af programmet.

`I artiklen om Kommunikaton med Makroer forklarer jeg, hvordan et Klik på Annuller knappen i en Input boks kan håndteres.` Denne metode kan imidlertid ikke anvendes her, da den behandler et Klik på Annuller på samme måde som et klik på OK, og det ønsker vi ikke i denne sammenhæng. Vi må derfor gå en anden vej og bruge Metoden InputBox i stedet for Funktionen InputBox.

```
Sub Fejl2()
On Error GoTo Fejl
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Ans = Application.InputBox("Indtast et tale mellem 1 og 20000")
    If Ans = False Then
        Exit Sub
    End If
    Exit Sub
Fejl:
    Fejlnummer = Err.Number
    Select Case Fejlnummer
        Case Is = 6
            Msgbox "Du kan ikke indtaste så store tal. Værdien skal være mellem 1 og 20.000"
            Resume
        Case Is = 13
            Msgbox "Du kan kun indtaste tal mellem 1 og 20.000"
            Resume
        Case Else
            Msgbox "Der er opstået en uforudset fejl"
            End
    End Select
End Sub
```

Forskellen på Metoden InputBox og Funktionen InputBox er, at funktionen, som blev anvendt i første omgang ved klik på Annuller returnerer en tom tekststreng og dette betragtes som en fejl Kontrollen overgives derfor til fejlrutinen. Et klik på Annuller i Metoden InputBox returner derimod den boolske værdi False, og dette udløser ikke en fejl. Der er dermed mulighed for at teste på variablens indhold, og forlade proceduren, hvis den er False. Det sker i de markerede linjer. Nu er vi altså nået så langt, at vi får vist en fejlmeddelelse, hvis der indtastes en værdi, som udløser en fejl, men ikke, hvis vi klikker på Annuller. Imidlertid er vi stadig ikke helt på plads. Koden tester jo kun på kørselsfejl, men hvad med logiske fejl? Det vender jeg tilbage til nedenfor, da heller ikke dette håndteres i fejlrutinen.

En generel fejlhåndteringsrutine

Som man måske kan se, kan det bleve forholdsvist beværligt at skulle kode denne type fejlrutiner i alle sine procedurer. I stedet kan man én gang for alle kode en generel fejlhåndteringsrutine, som de enkelte procedurer så kan kalde, når fejl opstår. Så kan man koncentrere sig om at håndtere logiske fejl i procedurerne.

En generel fejlhåndteringsrutine, kan godt blive lige så specifik som en speciel, men det vil sjælden ske. I stedet for at teste på hver enkelt fejlkode for sig, vil man snarere "gruppere" ensartede fejl, som der så skal reageres ens på. En genrel fejlhåndteringsrutine, skal oprettes som en Function, ikke som en Sub, da den skal kunne kaldes af de enkelte procedurer med det fejlnummer, som opstår i den kaldende procedure. Ofte vil det være en fordel at kode fejlrutinen i et andet modul end det, hvor de "almindelige" procedurer befinder sig. Denne Function skal have derfor have et enkelt argument, fx

```
Function Fejlrutine(Fejlnummer) As Integer
```

Nu skal vi så i gang med at kode sleve rutinen, og ofte vil det være en fordel at kode fejlrutinen i et andet modul end det, hvor de "almindelige" procedurer befinder sig. Endnu engang anvender vi en Select Case. Se næste side.

```
Function Fejlrutine(Fejlnummer)
    Dim MinFejl As Integer
    Select Case Fejlnummer
        Case Is = 6
            Msgbox "Du har indtastet et tal, der ligger uden for de angivne
grænser"
            MinFejl = 1
        Case Is = 7
            Msgbox "Der er ikke mere hukommelse. LUk alle unødvendige
programmer"
            MinFejl = 1
        Case Is = 11
            Msgbox "Du forsøger at dividere med 0. Dette er ikke tilladt"
            MinFejl = 1
        Case Is = 13
            Msgbox "Du har brugt en datatype, som ikke er tilladt."
            MinFejl = 1
        Case Is = 48, 49, 51
            Msgbox "Der er opstået en alvorlig programfejl nummer: " _
               & Err.Number & crlf _
            & " " & Err.Description & crlf & " Kontakt help desk."
            MinFejl = 2
        Case Else
            Msgbox "Der er opstået en fejl, som ikke kan rettes. _
               Programmet afsluttes"
            MinFejl = 2
       End Select

       Fejlrutine = MinFejl
End Function
```

Koden her undersøger for nogle specifikke fejl, og håndterer disse. Desuden tager den højde for uforudsete fejl. Det er selvfølgelig muligt at indsætte alle mulige specifikke fejlnumre i koden. Det, der er vigtigt er at MinFejl skal sættes til samme tal, afhængigt af, hvordan den kaldende procedure. I ovenstående skal 1 håndteres med Resume mens 2 håndteres med End. Det er selvfølgelig også muligt at lade andre koder gøre andre ting, fx kan 3 betyde Exit Sub, 4 kan betyde End osv.

Nu skal de procedurer, der skal anvende fejlrutinen så gøres klar:

```
Sub Fejl2()
On Error GoTo Fejlkald
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Ans = Application.InputBox("Indtast et tale mellem 1 og 20000")
    If Ans = False Then
        Exit Sub
    End If
    Exit Sub
Fejlkald:
    Fejlnummer = Err.Number
    Select Case Fejlrutine(Err.Number)
        Case Is = 1
            Resume
        Case Is = 2
            Resume Next
        Case Is = 3
            Exit Sub
         Case is 4
            End
    End Select
End Sub
```

Alt fra Fejlkald og proceduren ud, kan bare kopieres til alle andre makroer. I Select Case kaldes fejlhåndteringsfunktionen, med det fejlnummer, som koden fejler med. Rutinen viser

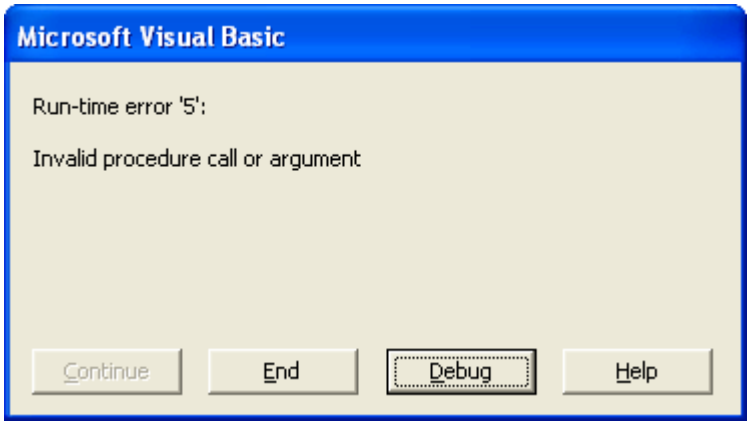
meddelelsesboksen og returnerer derefter en talværdi, som fortæller procedurens fejlområde, hvordan der skal fortsættes efter fejlbehandlingen. Afslutningerne SKAL ligge i de enkelte procedurer.

NB! En generel fejlhåndteringsrutine kan ikke blive lige så præcis, som en specifik, knyttet til et bestemt modul, da flere forskellige fejl kan returnere samme fejlkode. En liste over de mulige kørselsfejlkoder, med tilhørende beskrivelse af fejlene, kan findes ved at starte VBA hjælpen. Søg efter Err Object. Når hjælpen til dette er åben, klikkes på See also. Vælg nu Trappable Errors. Så vises listen over fejl. Der er i alt 120 fejl, men mange af dem er ret sjældne og vil ikke skulle kunne håndteres specielt, men kan i stedet håndteres under Case Else.

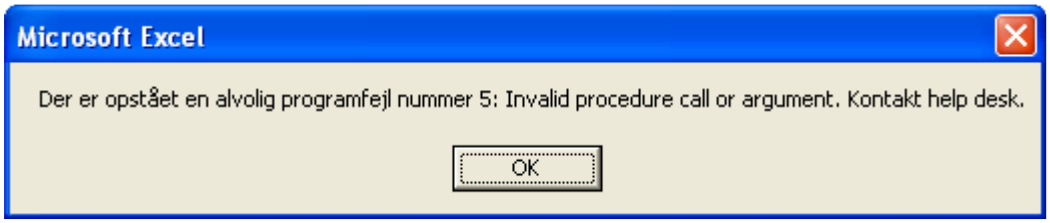
Imidlertid sker det, at vores fejlmeddelelse i fejlhåndteringsrutinen bliver for "generel", og så må vi ty til andre metoder. Koden jeg har brugt indtil videre, "gør ingenting". Den tildeler bare en værdi til en variabel. Men hvis vi nu antager, at denne makroens funktion var at uddrage kvadratroden af tallet i denne variabel. I så fald kunne koden se således ud:

```
Sub Fejl2()
On Error GoTo Fejlkald
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Ans = Application.InputBox("Indtast et tale mellem 1 og 20000")
    If Ans = False Then
        Exit Sub
    End If
    Msgbox Sqr(Ans)
    Exit Sub
Fejlkald:
    Fejlnummer = Err.Number
    Select Case Fejlrutine(Err.Number)
        Case Is = 1
            Resume
        Case Is = 2
            Resume Next
        Case Is = 3
            End
    End Select
End Sub
```

Det nye er den fremhævede linje. Selv om vi indtaster et tal, og dette tal ligger inden for grænserne for et heltal, ville koden fejle, hvis vi indtastede et negativt tal. Denne fejl opstår imidlertid ikke når værdien bliver tildelt variablen, men først når vi prøver at udføre beregningen. Uden fejlhåndteringsrutine ville vi få denne fejlmeddelelse:



Og med ville vi få denne:



Ingen af dem fortæller brugeren, hvad det er, han eller hun har gjort galt. Fejl 5 er typisk fejl, som brugeren ikke selv kan gøre noget ved, men sådan er det ikke i dette konkrete tilfælde. Den må altså håndteres på anden måde, fx på en måde, der ligner den, hvormed vi håndterede et klik på Annuller, det vil sige ved en test i selve proceduren.

```
If Ans <=0 Then
    MsgBox "Du kan ikke beregne kvadratroden af et tal, der er mindre eller lig
0"
End If
```

Logiske fejl

Logiske fejl er fejl, som ikke kan fanges af fejlhåndteringsrutinen, fordi de ikke udløser en kørselsfejl. Hvis vi af forskellige grunde, virkelig vil sætte en begrænsning på 20.000 på ovenstående; at vi altså ikke vil tillade beregning af kvadratroden af tal over 20.000, kan vi slet ikke fange tal mellem 20.001 og 32.767. Det er vi altså også nødt til at håndtere selv på samme måde som ovenfor:

```
If Ans >20000 Then
    MsgBox "Funktionen accepterer ikke tal større end 20.000"
End If
```

Nu kan vi så skrive alle vore logiske tests sammen ved at erstatte Ifsætningerne med en Select Case.

```
Sub Fejl2()
    On Error GoTo Fejlkald
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Ans = Application.InputBox("Indtast et tale mellem 1 og 20000")
    Select Case Ans
        Case Is = False
            Exit Sub
        Case Is <= 0
            Msgbox "Du kan ikke beregne kvadratroden af et tal, der er mindre
eller lig 0"
            Exit Sub
        Case Is > 20000
            Msgbox "Funktionen accepterer ikke tal større end 20.000"
            Exit Sub
    End Select

    Msgbox Sqr(Ans)
    Exit Sub
Fejlkald:
    Fejlnummer = Err.Number
    Select Case Fejlrutine(Err.Number)
        Case Is = 1
            Resume
        Case Is = 2
            Resume Next
        Case Is = 3
            End
    End Select
End Sub
```

Det virker næsten upåklageligt. Det eneste problem er, at i de sidste to tilfælde vil vi gerne opnå at få vist inputboxen igen, i stedet for at brugeren skal starte proceduren helt forfra igen på grund af en tastefejl. Den nemmeste løsning er at oprette et linjemærke lige før linjen, der viser inputboksen. Derefter erstattes Exit Sub i de sidste to Case sætninger med Goto Linjemærke. Så kan diverse meddelelsesbokse selvfølgelig "pyntes" med knapper mm., og koden kan kommenteres. Men det vil jeg springe over i denne omgang, da det allerede er omtalt i andre artikler, så jeg afslutter bare med den færdige kode:

```
Sub Fejl2()
    On Error GoTo Fejlkald
    Dim Ans As Integer
    Dim Fejlnummer As Integer
    Dim varX As Boolean

Forfra:

    Ans = Application.InputBox("Indtast et tale mellem 1 og 20000")
    Select Case Ans
        Case Is = False
            Exit Sub
        Case Is <= 0
            Msgbox "Du kan ikke beregne kvadratroden af et tal, der er mindre
eller lig 0"
            GoTo Forfra
        Case Is > 20000
            Msgbox "Funktionen accepterer ikke tal større end 20.000"
            GoTo Forfra
    End Select


    Msgbox Sqr(Ans)
    Exit Sub
Fejlkald:
    Fejlnummer = Err.Number
    Select Case Fejlrutine(Err.Number)
        Case Is = 1
            Resume
        Case Is = 2
            Resume Next
        Case Is = 3
            End
    End Select
End Sub
```