ingenieur
wissenschaften
htw saar

Digital Skills für Ingenieur*innen

7. Vorlesungstag

„Einführung in Makros“

ingenieur
wissenschaften
htw saar

Einführung in Makros

Operatoren

Ein Operator ist ein Zeichen, das eine mathematische oder logische Operation kennzeichnet oder ausführt und dabei wie eine Funktion ein Ergebnis liefert. Ein Beispiel hierfür ist der * Operator, der zwei Zahlen multipliziert. Die von einem Operator verarbeiteten Werte werden als Operanden bezeichnet. Operatoren besitzen unterschiedliche Prioritäten, wobei ein Operator mit Priorität 1 die höchste Prioritätsstufe einnimmt.

ingenieur wissenschaften htw saar

Einführung in Makros

Übersicht Operatoren und ihre Priorität

Priorität	*Operator*	*Typ*	*Beschreibung*
1	Not	Unär	Bitweises oder logisches Nicht/Not
1	-	Unär	Negation/Negatives Vorzeichen
1	+	Unär	Positives Vorzeichen
2	^	Binär	Potenzierung
3	*	Binär	Multiplikation
3	/	Binär	Division

ingenieur
wissenschaften
htw saar

Einführung in Makros

Übersicht Operatoren und ihre Priorität

Priorität	*Operator*	*Typ*	*Beschreibung*
4	Mod	Binär	Restklassen Division
5	\	Binär	Ganzzahlige Division
6	-	Binär	Subtraktion
6	+	Binär	Addition und String-Verkettung
7	&	Binär	String-Verkettung
8	Is	Binär	Referenzvergleich
8	=	Binär	Gleich

ingenieur wissenschaften htw saar

Einführung in Makros

Übersicht Operatoren und ihre Priorität

Priorität	*Operator*	*Typ*	*Beschreibung*
8	<	Binär	Kleiner
8	>	Binär	Größer
8	<=	Binär	Kleiner gleich
8	>=	Binär	Größer gleich
8	<>	Binär	Ungleich
9	And	Binär	Bitweises UND für Zahlen, Logisches UND für Boolean

Einführung in Makros

Übersicht Operatoren und ihre Priorität

Priorität	*Operator*	*Typ*	*Beschreibung*
9	Or	Binär	Bitweises ODER für Zahlen, Logisches ODER für Boolean
9	Xor	Binär	Exklusives ODER, bitweises für Zahlen, logisch für Boolean
9	Eqv	Binär	Äquivalenz, bitweises für Zahlen, logisch für Boolean
9	Imp	Binär	Implikation. Bitweise für Zahlen, logisch für Boolean

Einführung in Makros

Unäres Plus (+) und Minus (-)

Unäre Operatoren besitzen die höchste Priorität und werden von rechts nach links evaluiert. Es ist in Basic gestattet, Leerzeichen zwischen einem unären Operator und seinem Operanden zu platzieren. Während ein Pluszeichen als Vorzeichen wenig nützlich ist und lediglich betont, dass eine Konstante nicht negativ ist, wird es ansonsten schlichtweg ignoriert. Im Gegensatz dazu repräsentiert ein Minuszeichen als Vorzeichen die numerische Negation.

Einführung in Makros

Unäres Plus (+) und Minus (-)

```
Sub Unaer
    Dim Ganzzahl1 As Integer
    Dim Ganzzahl2 As Integer
    Ganzzahl1 = - 1
    Ganzzahl2 = -1
End Sub
```

Code Listing 1: unaer.bas

Einführung in Makros

Potenzierung (^)

Die Potenzierung ermöglicht die Verwendung sowohl ganzzahliger als auch Fließkommaexponenten. Der Potenzierungsoperator kann nur bei einem ganzzahligen Exponenten mit einer negativen Zahl kombiniert werden.

Es ist wichtig zu beachten, dass die Potenzierung eine niedrigere Priorität als die Negation hat. Daher wird der Ausdruck -1^2 fälschlicherweise zu 1 ausgewertet. Im Basic erfolgt die Auswertung mehrerer Exponenten (z.B., 2^3^4) von links nach rechts ((2^3)^4), während der mathematische Standard die Priorität von rechts nach links festlegt (2^(3^4)).

Einführung in Makros

Multiplikation (*) und Division (/)

Die Priorität von Multiplikation und Division ist identisch.

```
Sub ExampleMultDiv
    Print "2*4     = " & 2*4          REM 8
    Print "6/2.0   = " & 6/2.0        REM 3
    Print "-5/2    = " & -5/2         REM -2.5
    Print "4*4/2   = " & 4*4/2        REM 8
End Sub
```

Code Listing 2: mult_div.bas

Einführung in Makros

Rest nach Division (Mod)

Der Modulo-Operator wird auch als „Rest nach Division“ bezeichnet. Wenn beispielsweise 7 Mod 2 ausgeführt wird, ergibt dies 1, da 7 durch 2 gleich 3 ist, mit einem Rest von 1. Vor der Berechnung werden alle Operanden auf ganze Zahlen gerundet.

Einführung in Makros

Rest nach Division (Mod)

```
Sub ExampleMod
    6    Mod    13    REM    6
    13   Mod    6     REM    1
    12   Mod    6     REM    0
    12.8 Mod    6     REM    0
    6.5  Mod    3     REM    1
    -23  Mod    8     REM    -7
    23   Mod    -8    REM    7
    -23  Mod    -8    REM    -7
End Sub
```

Code Listing 3: mod.bas

Einführung in Makros

Ganzzahlige Division (\)

In einer ganzzahligen Division erfordert die Operation zwei Ganzzahl-Werte, und das Ergebnis ist ebenfalls eine Ganzzahl. Konstante numerische Operanden des ganzzahligen Divisionsoperators werden vor der Ausführung der Operation auf Ganzzahlen abgeschnitten.

Einführung in Makros

Ganzzahlige Division (\)

In einer ganzzahligen Division erfordert die Operation zwei Ganzzahl-Werte, und das Ergebnis ist ebenfalls eine Ganzzahl. Konstante numerische Operanden des ganzzahligen Divisionsoperators werden vor der Ausführung der Operation auf Ganzzahlen abgeschnitten.

```
Sub ExampleIntDiv
    7\2          REM   3
    8\3          REM   2
   -7\2          REM   -3
   --8\3         REM   -2
End Sub
```

Code Listing 4: int_div.bas

Einführung in Makros

Addition (+), Subtraktion (-) und String-Verkettung (& und +)

Die Priorität von Addition und Subtraktion ist gleich, wobei diese höher ist als die des Operators für die String-Verkettung. Beim Addieren numerischer Werte ist Vorsicht geboten, da der Operator + auch für die Verkettung von Strings verwendet werden kann. Wenn der erste Operand des Operators + eine Zahl ist und der zweite ein String ist, wird der String in eine Zahl umgewandelt. Im Gegensatz dazu wird, wenn der erste Operand des Operators + ein String ist und der zweite eine Zahl ist, die Zahl in einen String umgewandelt.

Einführung in Makros

Addition (+), Subtraktion (-) und String-Verkettung (& und +)

Die Priorität von Addition und Subtraktion ist gleich, wobei diese höher ist als die des Operators für die String-Verkettung. Beim Addieren numerischer Werte ist Vorsicht geboten, da der Operator + auch für die Verkettung von Strings verwendet werden kann. Wenn der erste Operand des Operators + eine Zahl ist und der zweite ein String ist, wird der String in eine Zahl umgewandelt. Im Gegensatz dazu wird, wenn der erste Operand des Operators + ein String ist und der zweite eine Zahl ist, die Zahl in einen String umgewandelt.

```
Print    236      + "3"    REM 239  (Numerisch)
Print    "236"    + 3      REM 2363 (String)
```

Code Listing 5: prio_add_sub_ver_1.bas

Einführung in Makros

Addition (+), Subtraktion (-) und String-Verkettung (& und +)

Der Operator für die String-Verkettung versucht, die Operanden in Strings zu konvertieren, falls mindestens ein Operand ein String ist.

```
Print 133      & "7"     REM 1337 (String)
Print "133"    & 7       REM 1337 (String)
Print 133      & 7       REM Um zu funktionieren,
                         REM muss wenigsten ein String dabei sein! (Leerstring)
```

Code Listing 6: prio_add_sub_ver_2.bas

Einführung in Makros

Addition (+), Subtraktion (-) und String-Verkettung (& und +)

Das Kombinieren von String-Manipulationen und numerischen Operationen kann zu ungewöhnlichen Ergebnissen führen, insbesondere aufgrund der Tatsache, dass die String-Verkettung mit dem Operator "&" eine geringere Priorität hat als der Operator "+".

Einführung in Makros

Addition (+), Subtraktion (-) und String-Verkettung (& und +)

Das Kombinieren von String-Manipulationen und numerischen Operationen kann zu ungewöhnlichen Ergebnissen führen, insbesondere aufgrund der Tatsache, dass die String-Verkettung mit dem Operator "&" eine geringere Priorität hat als der Operator "+".

```
Print 13 + "3" & 7          REM 167     Erst die Addition,
                            REM         dann die Stringkonvertierung
Print 13 & "3" + 7          REM 1337    Erst die Addition,
                            REM         aber der erste Operand ist ein String
Print 13 & 3 + "7"          REM 1310    Erst die Addition,
                            REM         aber der erste Operand ist ein Integer
```

Code Listing 7: prio_add_sub_ver_3.bas

Einführung in Makros

Logische und bitweise Operatoren

Jeder logische Operator formuliert eine grundlegende Frage, die entweder mit True (wahr) oder False (falsch) beantwortet werden kann.

Einführung in Makros

Logische und bitweise Operatoren

Jeder logische Operator formuliert eine grundlegende Frage, die entweder mit True (wahr) oder False (falsch) beantwortet werden kann.

```
If neuerKunde = True Then
    'Code, der ausgeführt, wenn ein neuer Kunde angelegt wird.
Else
    'Code, der ausgeführt, wenn der Kunde schon angelegt ist.
End If
```

Code Listing 8: bool_example.bas

Einführung in Makros

Logische und bitweise Operatoren

Jeder logische Operator formuliert eine grundlegende Frage, die entweder mit True (wahr) oder False (falsch) beantwortet werden kann.

x	*y*	*x And y*	*x Or y*	*x Xor y*	*x Eqv y*	*x Imp y*
1	1	1	1	0	1	1
1	0	0	1	1	0	0
0	1	0	1	1	0	1
0	0	0	0	0	1	1

Einführung in Makros

Logische und bitweise Operatoren

Jeder logische Operator formuliert eine grundlegende Frage, die entweder mit True (wahr) oder False (falsch) beantwortet werden kann.

In bestimmten logischen Ausdrücken ist es eigentlich nicht erforderlich, alle Operanden auszuwerten. Zum Beispiel kann bei einem Ausdruck wie „False And True“ bereits nach Auswertung des ersten Operanden „False“ mit der Auswertung gestoppt werden, da die zweite Operator keine Rolle mehr spielt (das Ergebnis kann nur „False“ sein). Dieses Prinzip ist als Kurzschlussauswertung bekannt. Bedauerlicherweise wird dies in Basic nicht unterstützt, und es erfolgt die Auswertung aller Operanden.

Einführung in Makros

Shift Operatoren

In Basic sind direkte Operationen zum Verschieben der Bits eines Bytes nach links oder rechts, wie sie in anderen Programmiersprachen üblich sind, nicht verfügbar, da die erforderlichen Operatoren fehlen.

Einführung in Makros

Shift Operatoren

In Basic sind direkte Operationen zum Verschieben der Bits eines Bytes nach links oder rechts, wie sie in anderen Programmiersprachen üblich sind, nicht verfügbar, da die erforderlichen Operatoren fehlen.

In C und Python werden beispielsweise die Operatoren "<<" und ">>" für solche Zwecke verwendet. In LibreOffice Basic muss eine entsprechende Funktionalität manuell implementiert werden.

Einführung in Makros

Ablaufsteuerung

Die Ablaufsteuerung bezieht sich darauf, welche Codezeile als nächstes ausgeführt wird. Der Aufruf einer Subroutine oder Funktion stellt eine einfache Form der nicht bedingten Ablaufsteuerung dar. Komplexere Ablaufsteuerungen umfassen Verzweigungen und Schleifen. Ablaufsteuerung ermöglicht die Gestaltung komplexer Abläufe in Makros, die sich je nach aktuellem Datenzustand ändern können.

In Basic sind bedingte Verzweigungsanweisungen wie "wenn x, dann tue y" verfügbar. Schleifenanweisungen bewirken, dass bestimmte Codebereiche im Programm wiederholt werden. Mit Schleifenanweisungen kann ein Bereich entweder eine festgelegte Anzahl von Malen durchlaufen werden oder bis eine bestimmte "exit"-Bedingung erfüllt ist.

Einführung in Makros

Einzeiliges If

Die If-Konstruktion führt abhängig von einem Ausdruck einen Codeblock aus. Die grundlegendste (einzeilige) Form einer If-Anweisung ist wie folgt:

```
If Bedingung Then Anweisung
```

Code Listing 9: if_einzeilig.bas

Einführung in Makros

Mehrzeiliges If

Wenn mehrere Anweisungen erforderlich sind, muss die Struktur aus mehreren Zeilen bestehen und mit der Zeile "End If" (oder "EndIf") abgeschlossen werden. Selbstverständlich ist diese Form auch mit nur einer Anweisung möglich, um die Lesbarkeit zu verbessern:

```
If Bedingung Then
    Anweisung 1
    [Anweisung 2]
End If
```

Code Listing 10: if_mehrzeilig.bas

Einführung in Makros

If Then Else

Die ElseIf-Anweisung ermöglicht die Abfolge mehrerer If-Anweisungen für verschiedene Tests. Der Codeblock der ERSTEN wahren Bedingung wird ausgeführt. Falls keine der anderen Bedingungen True ist, wird der Else-Block ausgeführt.

```
If Bedingung Then
        Anweisungen
[ElseIf Bedingung Then]
        Anweisungen
[Else]
        Anweisungen
End If
```

Code Listing 11: if_else.bas

Einführung in Makros

Immediate If

Die IIf-Funktion (Immediate If = unmittelbares If) gibt abhängig von einer Bedingung einen von zwei Werten zurück. Sie können als Argumente sowohl einfache Datentypen als auch Funktionen verwenden. Die Rückgabewerte der einfachen Datentypen und Funktionen werden dabei von IIf zur Rückgabe herangezogen.

```
obj = IIf (Bedingung, AusdruckWennWahr, AusdruckWennFalsch)
```

Code Listing 12: immediate_if.bas

Einführung in Makros

Immediate If

Vor der Ausführung einer Routine werden alle Argumente ausgewertet. Da IIf eine Funktion ist, werden alle ihre Argumente im Voraus ausgewertet. Das bedeutet, dass auch alle als Argumente verwendeten Funktionen vor der eigentlichen Ausführung von IIf ausgeführt werden. Im Gegensatz dazu wird bei einer If-Anweisung der von der Bedingung abhängige Code erst ausgeführt, wenn die Bedingung erfüllt ist.

```
obj = IIf (Bedingung, AusdruckWennWahr, AusdruckWennFalsch)
```

Code Listing 12: immediate_if.bas

Einführung in Makros

Immediate If

Vor der Ausführung einer Routine werden alle Argumente ausgewertet. Da IIf eine Funktion ist, werden alle ihre Argumente im Voraus ausgewertet. Das bedeutet, dass auch alle als Argumente verwendeten Funktionen vor der eigentlichen Ausführung von IIf ausgeführt werden. Im Gegensatz dazu wird bei einer If-Anweisung der von der Bedingung abhängige Code erst ausgeführt, wenn die Bedingung erfüllt ist.

```
obj = IIF(x <> 0, 1/x, 0)   REM Division-durch-Null-Fehler
```

Code Listing 13: iif_dd0f.bas

Einführung in Makros

Choose

Die Funktion Choose ähnelt der Funktion IIf darin, dass sie abhängig vom Wert ihres ersten Arguments eines der nachfolgenden Argumente zurückgibt. Der entscheidende Unterschied besteht darin, dass Choose mehr als zwei Rückgabeargumente haben kann. Des Weiteren ist das erste Argument kein boolescher Wert, sondern ein Index in Form einer Ganzzahl. Dieser Index gibt an, welches der potenziell vielen Argumente zurückgegeben werden soll.

```
obj = Choose(Ausdruck, Select_1[, Select_2, ... [,Select_n]])
```

Code Listing 14: choose.bas

Einführung in Makros

Choose

Bei Choose werden alle Argumente vor der Ausführung ausgewertet (genau wie bei IIf), unabhängig davon, welches davon zurückgegeben wird. Die Auswahlargumente können dabei sowohl Ausdrücke als auch Funktionsaufrufe sein.

```
Obj = Choose(1, "eins", "zwei", "drei", "vier")
```

Code Listing 15: choose_example.bas

Einführung in Makros

Select Case

Die Select Case-Anweisung weist Ähnlichkeiten mit einer If-Anweisung auf, die mehrere ElseIf-Blöcke enthält. Allerdings wird lediglich ein Bedingungsausdruck definiert, der anschließend auf Gleichheit mit verschiedenen Werten überprüft wird.

```
Select Case bedingungs_ausdruck
    Case case_ausdruck1
        Anweisungsblock1
    Case case_ausdruck2
        Anweisungsblock2
    Case Else
        Anweisungsblock3
End Select
```

Code Listing 16: select_case.bas

Einführung in Makros

Select Case

Der bedingungs_ausdruck wird mit jedem case_ausdruck verglichen, und der Block von Anweisungen nach der ersten Übereinstimmung wird ausgeführt.

```
Select Case bedingungs_ausdruck
        Case case_ausdruck1
                Anweisungsblock1
        Case case_ausdruck2
                Anweisungsblock2
        Case Else
                Anweisungsblock3
End Select
```

Code Listing 16: select_case.bas

Einführung in Makros

Select Case

Der optionale Block Case Else wird aktiviert, wenn keine der Bedingungen erfüllt ist. Es tritt kein Fehler auf, wenn keine der Bedingungen zutrifft und kein Case Else definiert ist. In diesem Fall gibt es einfach keine auszuführenden Aktionen.

```
Select Case bedingungs_ausdruck
    Case case_ausdruck1
        Anweisungsblock1
    Case case_ausdruck2
        Anweisungsblock2
    Case Else
        Anweisungsblock3
End Select
```

Code Listing 16: select_case.bas

Einführung in Makros

Select Case

Oftmals besteht ein Case-Ausdruck einfach aus einer Konstanten, beispielsweise Case 4" oder Case "Hallo".

```
Select Selector
    Case 4
        Print "Vier"
    Case "Hallo"
        Print "Hallo"
End Select
```

Code Listing 17: select_case_example.bas

Einführung in Makros

Select Case

Es ist möglich, mehrere Werte gleichzeitig anzugeben, indem sie durch Kommas getrennt werden, wie in „Case 1, 3, 5“. Das Schlüsselwort "To" ermöglicht die Definition eines Wertebereichs, zum Beispiel „Case 6 To 10“. Offene Wertebereiche können mit „Case < 10“ oder „Case Is < 10“ überprüft werden.

```
Select Case i
    Case 1, 3, 5
        Print "i ist eins , drei oder fünf"
    Case 6 To 10
        Print "i ist ein Wert von 6 bis 10"
...
```

Code Listing 18: select_case_wb.bas

Einführung in Makros

While ... Wend

Durch die Verwendung der Anweisung While ... Wend wird ein Anweisungsblock so lange wiederholt, wie eine bestimmte Bedingung wahr ist. Im Vergleich zur Schleifenanweisung Do While … Loop weist dieses Konstrukt einige Nachteile auf, da While ... Wend keine Exit-Anweisung unterstützt.

```
While Bedingung
    Anweisungsblock
Wend
```

Code Listing 19: while_wend.bas

Einführung in Makros

Do ... Loop

Der Schleifenmechanismus kommt in verschiedenen Formen vor und wird verwendet, um einen Anweisungsblock so lange zu wiederholen, wie eine Bedingung wahr ist oder bis eine bestimmte Bedingung erfüllt ist. Falls die Anfangsbedingung falsch ist, wird die Schleife überhaupt nicht ausgeführt.

```
Do While Bedingung
        Block
        [Exit Do]
        Block
Loop
```

Code Listing 20: while_loop.bas

Einführung in Makros

Do ... Loop

In einer ähnlichen, wenn auch weniger gebräuchlichen Form wird der Anweisungsblock so lange wiederholt, wie die Bedingung falsch ist. Anders ausgedrückt, der Code wird ausgeführt, bis die Bedingung wahr wird. Falls die Bedingung bereits zu Beginn als wahr evaluiert wird, erfolgt keine Ausführung der Schleife.

```
Do Until Bedingung
    Block
    [Exit Do]
    Block
Loop
```

Code Listing 21: do_until_loop.bas

Einführung in Makros

Do ... Loop

Es besteht die Möglichkeit, die Überprüfung der Bedingung ans Ende der Schleife zu setzen, sodass der Anweisungsblock mindestens einmal ausgeführt wird. In der nachfolgenden Form wird die Schleife mindestens einmal durchlaufen und anschließend so lange wiederholt, wie die Bedingung wahr ist.

```
Do
        Block
        [Exit Do]
        Block
Loop While Bedingung
```

Code Listing 22: do_loop_while.bas

Einführung in Makros

Do ... Loop

In der dargestellten Struktur wird die Schleife mindestens einmal durchlaufen und dann wiederholt, solange die Bedingung falsch ist.

```
Do
        Block
        [Exit Do]
        Block
Loop Until Bedingung
```

Code Listing 23: do_loop_until.bas

Einführung in Makros

Aussteigen aus der Do-Schleife

Die Anweisung Exit Do bewirkt das sofortige Schleifenende. Sie ist nur zwischen Do und Loop zulässig. Das Programm wird mit der Anweisung fortgesetzt, die der innersten Loop-Anweisung folgt.

Einführung in Makros

For ... Next

Die For ... Next-Anweisung führt einen Anweisungsblock für eine festgelegte Anzahl von Iterationen wiederholt aus.

```
For zähler = startwert To endwert [Step schrittWert]
    Anweisungsblock1
    [Exit For]
    Anweisungsblock2
Next [zähler]
```

Code Listing 24: for_next.bas

Einführung in Makros

For ... Next

Die Variable "zähler" wird mit dem Wert von "startwert" initialisiert. Bei Erreichen der Anweisung "Next" wird der Variable "zähler" der Wert "schrittwert" hinzugefügt. Falls kein "schrittwert" angegeben ist, wird 1 addiert.

```
For zähler = startwert To endwert [Step schrittWert]
      Anweisungsblock1
      [Exit For]
      Anweisungsblock2
Next [zähler]
```

Code Listing 24: for_next.bas