Variablen in Visual Basic

Sinn von Variablen

Es ist etwas umständlich, immer den vollen Objektnamen anzugeben (z.B. txtEingabe.Text oder noch schlimmer Data1.DataSource = "C:\db\db.mdb"), dies ist erforderlich, wenn man z.B. bei dem mit dem Inhalt eines Textfeldes rechnen will. Wie auch im richtigen Leben gibt es auch in Visual Basic eine Möglichkeit den Objekten sozusagen kürzere und besser zu merkende "Spitz"-Namen zu geben. Diese "Spitz"-Namen heißen Variablen.

Was sind Variablen genau

Wenn ich soeben von einem Spitznamen sprach, so ist dies eher bildlich gemeint. Denn eine Variable ist ein symbolischer Name für einen Speicherbereich im Hauptspeicher. In diesem Speicherbereich werden die der Variable zugewiesenen Daten gespeichert.
Ein Arbeitsspeicher besteht aus Registern, genau genommen müsste man als Programmierer nun die Speicheradresse angeben. *Dies kann man auch heute noch, jedoch nicht mit Visual Basic. Die Programmiersprache C und C++ bieten einem noch die Möglichkeit auf die Vergabe des Speicherbereichs Einfluss zu nehmen. Doch dies lassen wir mal außer Acht.* Also eigentlich spricht man die Speicheradresse an, jedoch ist diese schwer einzusetzen. Wie kompliziert solche Speicheradresse sind zeigen manche kryptischen Fehlermeldung von Windows.
Darum gibt man dieser Speicheradresse einen Namen, den Variablennamen, ähnlich einer Domain für die IP-Adresse des Servers.

Varialen deklarieren

Bevor man jedoch die Variable verwenden kann, muss man Visual Basic sagen, dass es diese gibt und um welche Art von Variablen es sich handelt. Es gibt zwei Möglichkeiten, dies zu tun, das implizite und das explizite Deklarieren.
Das erste Verfahren bedeutet, dass Visual Basic die Variable automatisch deklariert (d.h. Visual Basic teilt sich selber mit, dass es die Variable gibt und welche Art von Variable es ist), wenn ihr etwas zugewiesen werden soll.
Das explizite Deklarieren bedeutet, dass man vor der Zuweisung von Werten auf die Variable, die Variable vereinbaren (also einbinden) muss; wie das funktioniert folgt gleich.

Warum sollte man explizit deklarieren

Jetzt wird man sich fragen, warum man denn nicht immer die erste Methode verwendet, Visual Basic macht doch alles für einen und man braucht nichts zu tun. Doch dies hat einen großen Nachteil, denn Visual Basic wählt den denkbar schlechtesten Datentypen (Der Datentyp bestimmt die Art und Größe der Variable, die im Hauptspeicher (Stack) reserviert wird) Variant, dieser Datentyp ist ein Allzweck-Datentyp der sowohl Zahlen als auch Zeichenfolgen (Wörter) aufnehmen kann und dies hat seinen Preis (er beansprucht 16 Byte (für Zahlen) oder 22 Byte + 1 Byte pro Zeichen (für Zeichenfolgen) (zum Vergleich: Integer 2 Byte (für Zahlen) String 1 Byte pro Zeichen (für Zeichenfolgen)). Also, das implizierte deklarieren geht zwar schneller und ist bequemer, aber man sollte dem Hauptspeicher des Benutzers zu liebe, die Variablen "von Hand" explizit deklarieren.

Wie deklariert man explizit?
Das explizite Deklarieren funktioniert so: DIM *Variablenname* AS *Datentyp.*
DIM steht für Dimensionierung, also heißt die Deklaration übersetzt dimensioniere die Variable *Variablenname* als *Datentyp*. Zum Variablenname ist noch etwas zur Konvention (Gesetz zur Namensgebung) zu sagen: Man kann entweder m (steht für Memory (Speicher)) als Präfix voranstellen um zu zeigen, das es sich um eine Variable handelt, oder man kann den Datentyp als Präfix verwenden (z.B. int für Integer oder strg für String).

Namensgebung bei Variablen
Jeder Variablenname muss mit einem Buchstaben beginnen, der Name darf höchstens 256 Zeichen lang sein (Es ist jedoch nicht sinnvoll lange Namen zu verwenden), er darf keine Sonderzeichen (wie .,;!$%&#) und Leerzeichen enthalten, er darf nur einmal im Gültigkeitsbereich vorkommen und nicht ein von Visual Basic reserviertes Schlüsselwort sein und wie schon erwähnt sollte man zur besseren Identifizierung einen Präfix (entweder aus m oder aus drei Buchstaben des Datentypen) voranstellen.

Wertezuweisung
Nachdem die Variable festgelegt ist, kann man ihr Werte des Datentyps zuweisen. Das geht z.B. so: mVariN = txtVariable.Text, die Anweisung liest man so: Der Variable VariN wir der Inhalt des Textfeldes txtVariable zugewiesen. Also: Man liest Wertezuweisungen immer von rechts nach links.

Sonderfall Textfeld
Zum Thema Textfeld und Rechnen: Leider liefert das Steuerelement Textfeld als Rückgabewert eine Zeichenfolge, wenn man diese einfach einer "Zahlen"-Variable zuweist, bekommt man die Fehlermeldung "Typenunverträglichkeit", also muss man den Inhalt vorher in einen "Zahlen"-Datentypen konvertieren, dies geht mit dem Voranstellen vom Schlüsselwort Val. Das funktioniert so: mVariZahl = Val(txtVariable.Text). Also gar nicht so schwer!
Wenn man das Ergebnis der Rechnung dann wieder zurück in einem Textfeld ausgeben will, muss man auch den ganzen Weg wieder rückwärts gehen, dies geht mit dem Voranstellen des Schlüsselwortes Str. Das funktioniert so: txtVariable2.Text = Str(mVariZahl).

Arrays
Arrays stellen eine Sonderform der Variablen dar, es handelt sich um Datenfelder. Es werden also mehrere gleichartige Variablen mit dem gleichen Namen auf einmal definiert. Alle Elemente innerhalb des Datenfeldes haben den gleichen Datentyp und die Anzahl der Datenelemente muss schon bei der Definition angegeben werden.
Die Definierung funktioniert folgendermaßen:
Dim strMitarbeiter (1 to 5) as String
Der erste Teil (bis zur Klammer) gleicht der gewöhnlichen Variablendefinition, in der Klammer steht dann, wie groß das Datenfeld sein soll (d.h. wie viele gleichartige Variablen erstellt werden sollen). Der Schluss ist wieder normal.
Zugewiesen wird wie folgt:

strMitarbeiter(2) = "Müller"

In der Klammer steht der Index, die ist die Nummer der reservierten Variablen. Die

Zuweisung ist also bis auf die Angabe des Indexes identisch mit der gewöhnlichen Variablenzuweisung.
Auch der Aufruf der Array-Variable ist einfach:
Print strMitarbeiter(4)

Wieder muss nur der Index zusätzlich angegeben werden.

Ein- und Ausgabefunktionen in Visual Basic

Wozu braucht man Ein- und Ausgabefunktionen?

Um Eingaben vom Benutzer annehmen zu können, braucht man eine Eingabemöglichkeit. Natürlich kann man für diesen Zweck auch einfach ein Textfeld auf dem Formular aufziehen, doch dies hat den Nachteil, dass das Steuerelement immer sichtbar ist, bzw. ein neues Formular dafür angelegt werden muss. Außerdem muss man die Inhalte des Textfeldes konvertieren, wenn man ihn einer "Zahl"-Variable zuweisen will.
Ähnlich sieht es auch mit der Ausgabe aus. Es ist viel zu umständlich für jede Meldung eine neue Form anzulegen.

InBox-Funktion für die Eingabe

Für die Eingabe gibt es eine Eingabefunktion in Visual Basic, diese heißt InputBox. Wie bei allen Funktionen können der InputBox-Funktion Argumente(Parameter) zugewiesen werden, z.B. für den Inhaltstext.
Eine InputBox-Funktion wird folgendermaßen verwendet:
Variable = InputBox(Beschriftung, Titel, Standartinhalt des Dialogs)

Die Funktion wird also direkt einer Variable zugewiesen, die Beschriftung bestimmt, was für ein Text über dem Eingabefeld steht, der Titel legt die Überschrift des Dialogfeldes fest und der Standartinhalt legt fest, was standardmäßig schon in dem Feld als Vorgabe steht.

MsgBox-Funktion für die Ausgabe

Die Ausgabe funktioniert sehr ähnlich, hier heißt die Funktion MsgBox. Eine solche Funktion wir folgendermaßen verwendet:
MsgBox(Beschriftung, Button-Art, Titel, Name der Hilfe-Datei zu diesem Dialog)

Die Funktion kann also direkt aufgerufen werden, die Beschriftung legt fest, was die Meldung besagen soll, die Button-Art legt fest, ob es sich z.B um eine Fehlermeldung handelt, der Titel legt die Überschrift des Dialogs fest, wenn es sich um eine Fehlermeldung handeln soll, kann man auch noch den Namen der Hilfedatei ange-

ben, die das Problem näher beschreibt und vielleicht auch Lösungsvorschläge bietet.

Auswahlstrukturen in Visual Basic

Was sind Auswahlstrukturen

Mit einer Auswahlstruktur kann die Reihenfolge gesteuert werden, in der Anweisungen ausgeführt werden. So kann man die Ausführung von Anweisungen an Bedingungen knüpfen. Wenn etwas passiert, dann tue dies. Ähnlich wird die Auswahlstruktur auch geschrieben, sie wird so verwendet:
If *Bedingung* Then
Anweisung
End If
Also wie schon oben erwähnt, heißt dies ins Deutsche übersetzt: Wenn die Bedingung wahr ist, dann tue dies und beende die Auswahlstruktur danach. Das End If darf nichts vergessen werden, da sonst eine Fehlermeldung erscheint.
Für *Bedingung* wird ein bedingter Ausdruck (wie z.B. ***Text1.Text = ""***) eingesetzt, wenn diese Bedingung wahr ist, dann wird das ausgeführt, was in der *Anweisung* steht (z.B. ***MsgBox(„Textfeld ist leer“)***).

Auswahlstruktur mit If...Then

Das obige ist eine einseitige Auswahlstruktur, nun gibt es noch eine zweiseitige Auswahlstruktur. Hier wird für den Fall, dass die erste Bedingung nicht wahr ist, eine alternative Anweisung ausgeführt. An die obige Anweisung wird dann noch Else hinzugefügt, dies funktioniert so:
If ***Bedingung*** Then
Anweisung1
Else
Anweisung2
End If

Um das obige Beispiel mit dem leeren Textfeld aufzugreifen, könnte nun die *Anweisung2* MsgBox("Textfeld ist voll") heißen.

Auswahlstruktur mit Select Case

Wenn sie sehr viele Anweisungen haben ist es sehr umständlich immer die nur wenig abgeänderte Bedingung immer ganz ausschreiben zu müssen. Für solche Fälle bietet Visual Basic die "Select Case"-Struktur an. Hier wird in der erste Zeile festgelegt, worauf sich die folgenden Bedingungen beziehen sollen, die Struktur wird folgendermaßen verwendet:
Select Case ***Variable***
Case ***Wert1***
Anweisung1

Case ***Wert2***
Anweisung2
End Select

Die Variable kann auch ein Eigenschaftswert, wie z.B. List1.ListIndex sein. Wenn der Wert nach dem Case mit dem Variablenwert übereinstimmt, also, z.B. der 2. Listenpunkt ausgewählt wurde, dann wird die Anweisung ausgeführt (Kleiner Hinweis am Rande: Das Listenfeld fängt mit 0 an zu zählen, so dass der erste Case-Fall 0 heißt). Auch hier ist es wichtig die Struktur mit End Select abzuschließen um eine Fehlermeldung zu vermeiden.

Schleifen

Wozu Schleifen?

Mit Schleifen kann man einfach und schnell Anweisungen mehrmals ausführen. Dabei kann man entweder eine Anzahl der Ausführungen festlegen oder die Ausführungszahl an einer Bedingung festmachen. So kann man das Programm anweisen die Anweisungen solange auszuführen, wie eine Bedingung erfüllt wird.

For-Schleifen

Mit For-Schleifen kann man das Programm anweisen, die Anweisung eine bestimmte Anzahl zu wiederholen. Deshalb kann man sie auch als bedingte Wiederholungsschleife bezeichnen.
Eine For-Schleife ist wie folgt aufgebaut:
For *i = 1 To 10*
Anweisung
Next i

In dem sogenannten Schleifenkopf (For...) steht die Bedingung die Schleife 10mal auszuführen (1 bis 10). Der erste Wert stellt hierbei den Anfangswert dar und der Wert nach dem to den Endwert. Beim Endwert wird begonnen und auf die Zählervariable i immer 1 aufgeschlagen, bis die Variable den Endwert erreicht hat. Im sogenannten Schleifenkörper steht die Anweisung die bei der Ausführung wiederholt werden soll. Im Schleifenfuss schließlich wird die Schleife abgeschlossen mit der Schlüsselphrase Next i.

Andere Zählmuster von For-Schleifen

Wenn man ein anderes Zählmuster als 1,2,3 verwenden will, kann man einen anderen Startwert definieren, dazu wird statt der 1 To ... z.B. 3 To ... geschrieben. Man kann auch die Zählschritte beeinflussen. Dies funktioniert mit dem Schlüsselwort Step. Eine Anweisung mit Step sieht dann folgendermaßen aus:
For ***i = 5*** to ***25*** Step ***5***
Print i
Next i

Bei dieser Anweisung wird von 5 ab in fünfer Schritten bis 25 gezählt (also 5,10,15,20,25), dies geht auch mit Dezimalzahlen. Bei Dezimalzahlen aber darauf achten, dass man statt dem trennenden Komma eine Punkt verwendet (also NICHT 0,5 SONDERN 0.5).

Exit For

Mit Exit For kann man die Schleife vorzeitig beenden um auf ein Ereignis zu reagieren. z.B. so:

```
For i = 1 To 3
Pass = InputBox("Geben Sie Ihr Passwort ein, oder Ende zum Beenden")
If Pass = "Ende" Then
Exit For
Next i
```

Die Schleife wird 3 mal wiederholt oder vorzeitig beendet, wenn das korrekte Passwort eingegeben wird.

Schleifen mit Do While Loop

Mit Do While Loop-Schleifen kann man bei der Anzahl der Wiederholungen der Anweisungen auf Bedingen reagieren. Eine Do While-Schleife wird solange ausgeführt, solange die Beding erfüllt ist.

Die Do While Loop Schleife ist folgendermaßen aufgebaut:

Do While *Bedingung*
Anweisung
Loop

Wie bei der For-Schleife steht die Bedingung im Schleifenkopf, die Anweisung im Schleifenkörper (ist aber jeder Schleifenart so) und Loop schließt die Schleife ab.

Achtung bei Do While Schleifen

Do-Schleifen können bei falscher Schleifenbedingung Endlosschleifen werden, das heißt, die Bedingung kann nie erfüllt werden und die Schleife wird dann unendlich oft ausgeführt. Im schlimmsten Falle kann dies zum Absturz eines Computers führen (So arbeiten auch Hacker die einen Internetserver lahm legen wollen. Sie schreiben eine Endlosschleife die den Server zum Absturz bringt). Deshalb sollte man entweder auf eine sinnvolle Schleifengestaltung achten oder eine Austrittsbedingung einbauen (z.B. eine Zahl die die Schleife beendet).

Schleifen mit Do Loop Unti

Eine Do Loop Until-Schleife wird solange ausgeführt, bis ein Ereignis eintritt, das die Schleife beendet. Die Do... Loop Until-Schleife ist folgendermaßen aufgebaut:

Do
Anweisung
Loop Until *Bedingung*

Auf den ersten Blick ist schon zu erkennen, dass sich diese Schleifenart grundlegend von den vorherigen Arten unterscheidet. Sie ist eine geschlossene Schleife, da die Bedingung im Schleifenfuss steht. Deshalb wird diese Schleife mindestens einmal ausgeführt.

Auch bei dieser Schleife ist auf eine erfüllbare Bedingung zu achten, da ansonsten wie bei der Do While Loop-Schleife eine Endlosschleife erzeugt wird.

Funktionen

Warum Funktionen?

Man stellt sich natürlich die Frage warum man Funktionen einsetzen soll, wo doch schon andere Mittel bekannt sind und diese (vermeintlich) einfacher und besser sind. Doch Funktionen haben Vorteile gegenüber der normalen Schreibweise von Anweisungen.
Sie verringern den Programmieraufwand in dem Programm, da man vorgefertigte Funktionen verwenden kann und sich so nicht mit der komplizierten Herleitung und Programmierung des Problems beschäftigen muss. Außerdem werden Fehlerquellen vermindert, da man die Anweisungen nur noch einmal schreiben muss und so den gewöhnlichen Quelltext vom logischen trennen kann. Natürlich wird der Quelltext auch durch den Einsatz von Funktionen übersichtlicher.

Wie funktionieren Funktionen

Funktionen sind kleine Unterprogramme, die separat vom Quelltext abgehoben geschrieben werden. Man kann in Visual Basic integrierte Funktionen verwenden, dann muss man die Funktionen mit ihrem Namen aufrufen und ihnen in der Regel ein Argument übergeben. Das sieht dann zum Beispiel so aus:
Sqr(Zahl)

Mit der Funktion Sqr lässt sich die Quadratwurzel von einer Zahl ausrechnen, die Zahl wird dann in Klammern hinter den Funktionsnamen geschrieben und so an die Funktion übergeben. Die Berechnen den Wert und übergibt diesen wieder zurück an die Funktion, die genauso lautet wie der Funktionsname.

Eigene Funktionen schreiben

Natürlich kann man auch eigene Funktionen schreiben. Das sieht dann so aus:
Public Function Euro(DM as Currency) as Currency
Euro = DM / 1,9558
End Function

Hier lautet der Name der Funktion Euro. Die benutzerdefinierten Funktionen werden nicht in eine Prozedur (Private Sub ... End Sub) geschrieben, sondern vor, bzw. hinter die Prozeduren der Knöpfe und anderen Objekte.
Jede Funktion beginnt mit den Schlüsselwörtern Public Function. Mit Public legt man fest, dass die Funktion im ganzen Programm Gültigkeit hat. In der Klammer stehen die Attributwerte des Attributs, das an die Funktion übergeben werden soll. Das Attribut ist eine Angabe, die die Funktion zu ihrer Berechnung von außerhalb benötigt um ihre Berechnungen durchführen zu können. In unserem Beispiel fordert die Funktion einen Wert für DM von außen an. Jede Funktion hat einen Rückgabewert, als das Ergebnis, das die Berechnung ergibt, dass an die Stelle zurückgegeben wird, wo es angefordert wird. Den Datentyp für diesen Datentyp gibt man nach der Klammer an. Man kann zwar die Attributsliste weglassen, den Datentyp des Rückgabewerts muss man jedoch immer angeben. Beendet wird die Funktion mit den Schlüsselwörtern End Function.
Daraus ergibt sich folgende allgemeine Definition:
Public Function Funktionsname (Attribut as Datentyp) as Datentyp
Anweisung
End Function

Aufgerufen wird die Funktion, in dem der Funktionsname mit der Übergabe des/der Attribute angegeben wird (z.B. Betrag = Euro(2)).

Prozeduren

Was sind Prozeduren?

Prozeduren unterscheiden sich von Funktionen nur dadurch, dass dem Namen einer Prozedur kein Rückgabewert zugeordnet ist. Sie sind also Funktionen ganz ähnlich; das schließt auch die Gründe ein, warum man Prozeduren einsetzen sollte. Jedoch werden Prozeduren vorwiegend dazu verwendet Benutzereingaben zu verarbeiten, Informationen anzuzeigen oder auszudrucken. Sie können aber auch mehrere Eigenschaften in Abhängigkeit von einer Bedingung verändern.
Der große Unterschied ist der Rückgabewert. Während Funktionen nur einen Rückgabewert haben, können Prozeduren beliebig viele haben.

Syntax von Prozeduren

Prozeduren sind folgendermaßen aufgebaut:
Sub *Prozedurname****(****Parameter****)***
Prozeduranweisungen
End Sub

Jetzt wird jedem ein Licht auf gehen der sich die Struktur der Prozedur ansieht: Das sieht doch so aus, wie der automatisch von Visual Basic generierte Rahmen für den Quelltext von Steuerelement. GENAU! Bei dem von Visual Basic generierten Rahmen handelt es sich um Ereignisprozeduren der Steuerelemente.
Nun gehe ich noch mal genauer auf die Struktur der Prozedur ein. Die Parameter sind optional, d.h. man kann die Prozedur auch ohne Parameter programmieren.

Übergabemöglichkeiten

Man der Prozedur auf verschiedene Weisen Werte übergeben, entweder als Referenz oder als Wert. Bei der Übergabe als Referenz wird der Prozedur eine Variable übergeben, die die Prozedur verändern kann. Bei der Übergabe als Wert, also z.B. ein Name in Anführungszeichen, wird der Wert als Konstante behandelt und kann somit folglich nicht verändert werden.

Prozeduren aufrufen

Mit folgendem Befehl wir die Prozedur aufgerufen:
Prozedurname(Parameter)

Im Falle des Weglassens der Parameter wird die Prozedur also nur durch die Angabe des Namens aufgerufen.