Informatik

Vorlesung

Einführung in die Programmiersprache

Visual Basic

Prof. Dr.-Ing. Thomas Wiedemann
Fachgebiet Informatik / Mathematik

Überblick zur 6. Vorlesung

- Einführung und kurze Historie
- Grundlegende Vorgehensweise
- Grundlegender Aufbau von VisualBasic-Programmen
- Grundbefehle
  - if then else
  - Do / While
  - for ( )

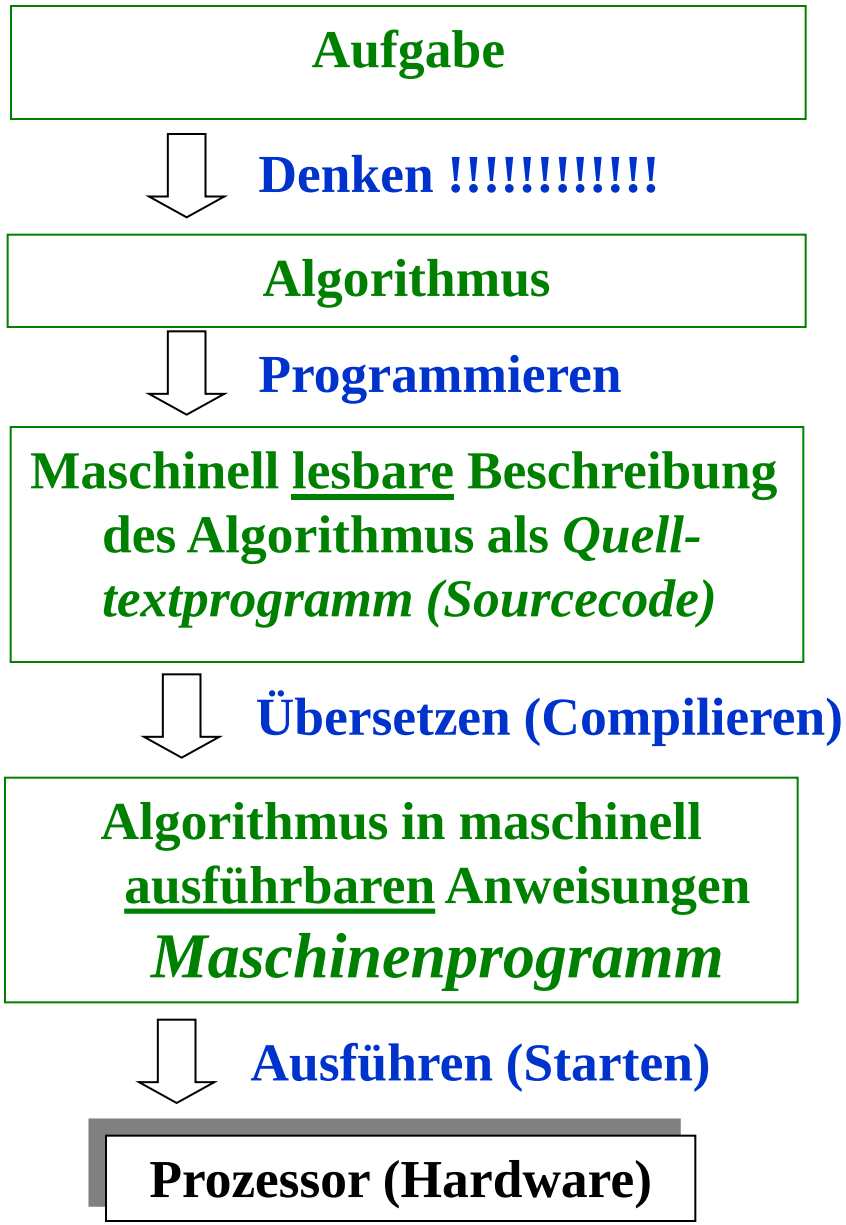
Entwicklung der Programmiersprachen

- zu Beginn direkte Maschinenbefehle (Assembler)
  - sehr aufwändig und fehleranfällig, dafür extrem effizient und schnell
- erste Hochsprachen ab 1960
  - FORTRAN für naturwissenschaftliche Berechungen (auch heute noch)
  - COBOL für betriebswirtschaftliche Programme, PL1, ADA

Problem: Sprachen entweder sehr Prozessor-nah oder Anwendungsnah :

- C / C++ sind sehr hardwarenah, leider aber relativ kompliziert
  - bilden aktuellen Industriestandard (siehe Stellenanzeigen)
  - sinnvoll für komplexe Projekte mit hohen Anforderungen
- Alternative: Visual Basic von Microsoft für Aufgaben mit mittlerer Komplexität und hohen Anforderungen an Stabilität und Komfort
  - Optimum für diesen Kurs bzgl. Leistung und Programmieraufwand

Vom Algorithmus zum Programm (Wiederholung)



Bis hier sind alle Arbeiten unabhängig von der konkreten Umsetzung (Rechner, Betriebssystem, Compiler).

In der Regel noch unabhängig von konkretem Rechner und Betriebssystem. Jedoch Compilerspezifisch !!!

Prozessor-, Rechner- und Betriebssystemspezifisch !!!.

Allgemeine Vorgehensweise bei der Programmierung

- Nachfolgende Erklärungen sind noch allgemeingültig
- Konkrete Bedienung der Programme ist systemabhängig

Ablauf der Programmentwicklung

- Entwicklung des Algorithmus (auf Papier oder mit Hilfsmitteln)
- Erstellung eines Projektes zur Ablage aller Entwicklungsdateien eines Programmes (enthält Sourcecode, kleine Grafikbibliotheken (Icons), internationale Texte, Steuerdateien, ...)
- Schreiben des eigenen Sourcecodes (oder Zusammenkopieren aus Hilfetexten oder Beispielen - ist unter Beachtung des Copyrights meist legitim und zur Vermeidung von Schreibfehlern ratsam !!!) und Ermittlung benötigter Funktionsbibliotheken
- Compilieren und Programmtest (in der Regel iterativer, mehrfach ablaufender Prozeß)
  - Fehler 1. Art : Syntaxfehler : Sourcecode entspricht nicht den Regeln
  - Fehler 2. Art : Laufzeitfehler : Source entspricht den Regeln, führt aber bei der Ausführung zu Problemen (Überlauf, Schutzverletzung, IO-Probleme ...)
  - Fehler 3. Art : logische Fehler: Programm läuft, führt aber NICHT zu den gewünschten Ergebnissen - Test und Fehlersuche durch Beispiele (auch WORST CASE !!!)

  Eigene THESE: Programmieren ist 95% Fehlersuche ! !!

Allgemeine Eigenschaften von Visual Basic (VB)

- BASIC wurde in den 80er Jahren als Sprache für Programmieranfänger entwickelt
- durch Zeilennummern und viele GOTO´s damals in Mißkredit geraten
- Bill Gates schätzt die Sprache noch bis heute für schnelle Entwicklungen !
- bereits unter MS-DOS eine erste Basic-Version GW-Basic
- unter Window 3.0 als Visual Basic 1.0 (Visual steht für visuelle Entwicklung)
- starke Verbesserungen unter Windows 95 und Windows NT – Visual Basic 4.0
- als stark Kundenorientiertes Werkzeueg für die Windowsentwicklung etabliert
- heute in MS-Office als **Visual Basic für Applications (VBA)** integriert !
- gleiche Projekte sind mit C/C++ um den Faktor 2-5 aufwändiger !
- HEUTE ist Visual Basic eine moderne Sprache mit sehr großer Funktionalität

- **Vorteile von VB :**

+ gut lesbar und wartbar

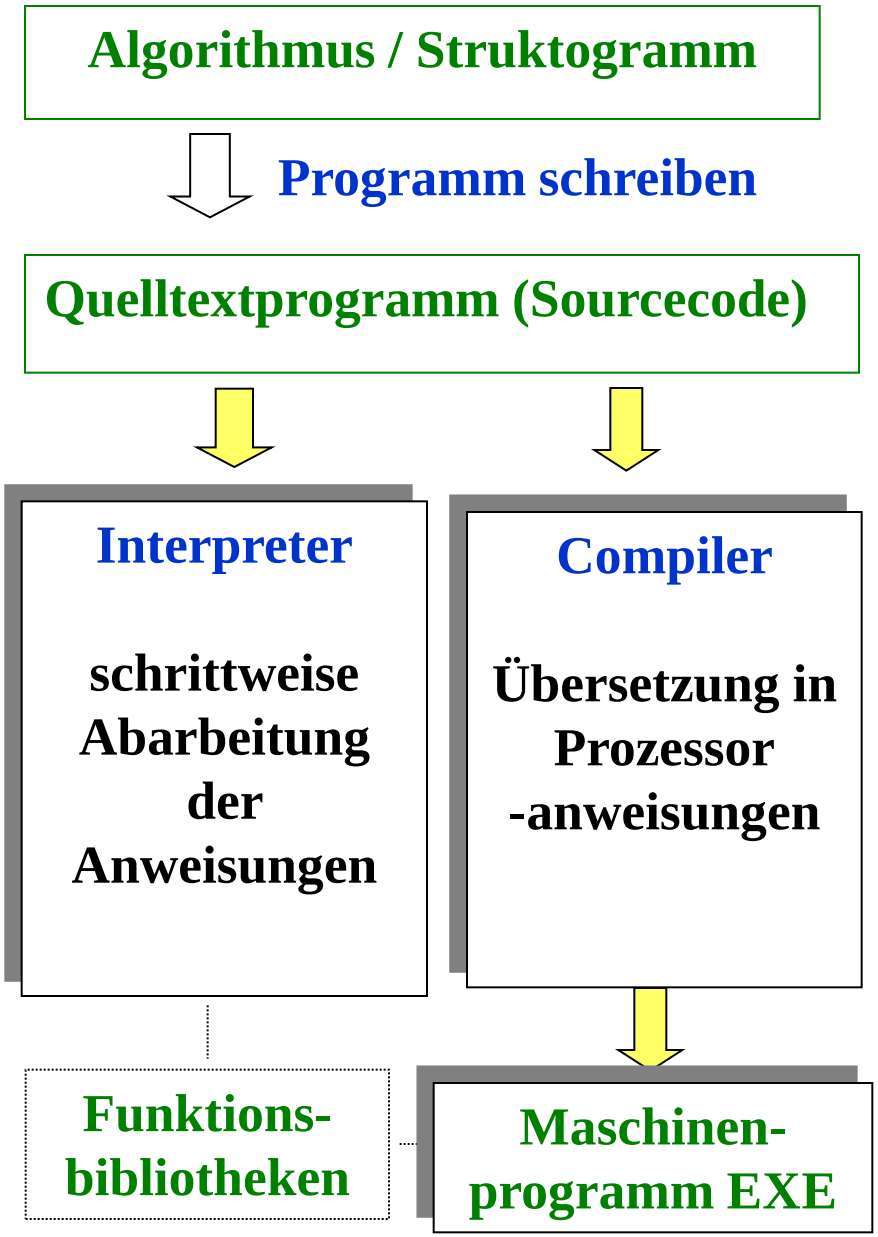
+ schnell erlernbar / gutmütig

Nachteile von VB :

- etwas langsamer als C/C++

- keine Hardwareprogrammierung

Ausführung von Visual Basic -Programmen -



Der Ablauf ist VB-spezifisch ! Teilweise Abweichungen bei anderen Programmiersprachen.

Visual Basic kann in 2 Modi arbeiten:

Für schnelle Entwicklungszyklen wird der Quelltext von einem **INTERPRETER** schrittweise analysiert und ausgeführt. Dies erlaubt sehr schnelle Entwicklungszyklen, ist jedoch durch den Interpretationsaufwand etwas langsamer.

Fertige Programme können durch einen **COMPILER** direkt in Anweisungen des Prozessors übersetzt werden. Dabei sind erhebliche Geschwindigkeitsoptimierungen möglich ! Optional kann auch ein Zwischencode (P-Code) generiert werden, welcher schon relativ nahe am Maschinencode orientiert ist und dabei aber Speicherplatz spart. Compilierte Programme müssen zur Laufzeit nicht mehr analysiert werden und sind daher schneller !

Allgemeine Regeln bei der VisualBasic-Programmierung

Verwendbare Zeichen für eigene Bezeichner (Variablen, Funktionen ...)

- Buchstaben A-Z a-z - auch deutsche Umlaute (trotzdem bitte vermeiden)
- **Visual Basic unterscheidet nicht zwischen Groß- und Kleinschreibung**

  Berechne() ist die gleiche Funktion wie berechne()
- Ziffern 0 – 9 , der Unterstrich _

Nicht zulässig für Bezeichner sind :

- Sonderzeichen ! ? " ' ( ) [ ] { } + - < > \ / . , : ; * & # % ^ ~ |
- Alle nicht druckbaren Zeichen (0-31) der ASCII-Codetabelle (z.B. Neue Zeile)
- Leerzeichen können verwendet werden, erfordern jedoch dann [mein Name]

Bezeichner

- dienen zur Bezeichnung von Datentypen, Variablen, Konstanten und Funktionen
- **definiert als Folge von Buchstaben und Ziffern**
- **das erste Zeichen muß ein Buchstabe sein !!!!**
- da Leerzeichen als Trennzeichen fungieren, wird statt Leerzeichen in Bezeichnern meist ein Unterstrich eingesetzt

Arbeit mit Bezeichnern und Trennzeichen

Korrekte Bezeichner :

alpha i anzahl_Personen a1 a2 b23 b24 b2o : Rem Korrekte Namen

Inkorrekte Bezeichner :

1wert #Wert ?Ähnlichkeit : Rem falsch !

ergibt in allen 3 Fällen die Meldung „SYNTAXFEHLER“ oder ähnliche Fehlermeldungen

Trennzeichen

- dienen zur Trennung einzelner Morpheme und Bezeichner
- verwendbar als Trennzeichen sind: Leerzeichen, Tabulator, Zeilenumbruch, Kommentar, in der englischen Literatur auch als White Spaces bezeichnet
- für den Interpreter / Compiler zählt nur die Existenz mindestens eines Trennzeichens, weitere werden ignoriert und können damit beliebig kombiniert werden !

Festlegungen für die weitere Arbeit

Zur Unterscheidung von Erklärungen und Programmbeispielen auf den weiteren Folien werden folgenden Festlegungen getroffen:

- Allgemeine Erklärungen werden mit der Schriftart Times Roman dargestellt
- Programme und Ausschnitte davon werden mit der Schriftart ARIAL dargestellt :

a = b + c

(erkennbar an schnörkelloser Schrift und fetteren Buchstaben)

- Beispielnamen oder Beispieltexte werden zusätzlich KURSIV / Blau dargestellt und können/müssen von Ihnen angepasst oder entsprechend ausgewählt werden :

Msgbox "*Der Wert ist zu klein !*"

- Besonders relevante Programmteile oder Erklärungen werden ROT eingefärbt und/oder unterstrichen (siehe Beispiel) . Die Unterstreichung dient nur für die Kennzeichnung auf einfarbigen Ausdrucken und ist NICHT programmrelevant !

- Hinweis: Ähnliche Festlegungen finden Sie auch in jeder Dokumentation zu Programmsprachen. Lesen Sie diese zuerst und orientieren Sie sich danach !

Kommentare

- enthalten beliebigen Text und werden durch Compiler ignoriert
- Einleitung mit REM (engl. Remark = Anmerkung) oder '
- Beispiel.:

```
Rem     Das ist ein Demoprogramm !
```

- Nach Anweisungen ist vor dem REM mit : eine Trennung zu vollziehen !
- auch nach Code:

```
brsumme = nsumme *  mwst   :  Rem Berechnet Mwst
```

- TIP: Mit Kommentaren können Codeteile kurzzeitig auch ausgeblendet werden, ohne diese löschen zu müssen (spart Arbeit und Zeit)

```
REM   Testausdruck1(a,b,c)
```

Schlüsselworte

- Schlüsselworte sind Identifikatoren, welchen in VB bereits eine **feste Bedeutung** zugeordnet wurde
- kann als grundlegender Befehlsvorrat von VB aufgefasst werden
- alle größeren VB-Programme basieren auf diesen Grundbefehlen oder auf externen Programmbausteinen
- **eine Neudefinition oder Verwendung von Identifikatoren für eigene Variablen ist nicht zulässig !**

- **Es gibt ca. 40 Schlüsselworte in VB. Die genaue Anzahl hängt von der Version der Programmiersprache ab.**
- **Wichtige Schlüsselworte werden auf den Folien rot unterstrichen dargestellt. Alle wesentlichen Schlüsselworte werden im Rahmen dieser LV erklärt und angewendet !**

Variablen

Variablen dienen zur Speicherung von Daten.

- Jede Variable benötigt einen entsprechenden Speicherplatz !
- Bei VB sollte dieser vor der ersten Anwendung definiert werden ! Falls nicht, reserviert VB automatisch Speicher, was bei unbeabsichtigten Schreibfehlern problematisch wird !
- Die generelle Syntax (Schreibweise) für die Speicherreservierung ist :

Dim *Variablenname* as *Datentyp*

- damit reserviert VB den Speicher (Dim ist Hauptschlüsselwort = Dimensionierung)
- das Schlüsselwort as trennt den Variablennamen vom Datentype
- Eine mehrmalige Definition einer Variablen im gleichen Bereich ist NICHT ZULÄSSIG !
- Bei Bedarf können auf einer Zeile mehrere Variablendefinitionen aufgeführt werden.
- Dabei ist mit Komma zu trennen :

- Variablendefinitionen können an jeder Stelle im Programm erfolgen !
- Es ist aber üblich und entspricht einem guten Programmierstil, alle Variablen möglichst in einer Gruppe am Anfang eines Programmes oder einer Funktionsgruppe zu konzentrieren. Dies spart Zeit bei der späteren Suche nach diesen Definitionen !

Verfügbare Datentypen für Variablendefinitionen

- **In Entsprechung zu den allgemein üblichen Datentypen in der Informatik (siehe VL Datenstrukturen) verfügt VB über folgende Standartdatentypen :**

für ganze Zahlen :

Dim a as integer :Rem ganze Zahl mit 16 Bit => bis +/- 32.000

Dim a as long :Rem ganze Zahl mit 32 Bit => bis +/- 2 Mrd.

für Gleitkommazahlen (rationale Zahlen)

Dim a as single :Rem einfache Gleitkommazahl ca. 7 Stellen genau

(mit Wertebereich +/-3,4 E 38 bis +/-1,4 E -45)

Dim a as double :Rem doppelt genau mit ca. 15 Stellen

(mit Wertebereich +/- 1,79 E308 bis +/- 4,9 E-324

für Texte

Dim s as string : Rem für beliebig viele Zeichen (max. 64 Kbyte)

Für beliebige Datentypen:

Dim v as variant : Rem Zur Ablage aller möglichen Datentypen

-> variant - nach Möglichkeit vermeiden, da sehr rechenzeitintensiv !!

Konstanten

Konstanten bleiben während der Zeit der Ausführung unverändert

- dienen zum Bereitstellen bekannter Werte wie PI oder einen festen Anzahl
- Der Versuch einer Änderung wird als Fehler erkannt.
- Als Schlüsselwort dient const
- Schreibweise:

```
const  anzahlp  = 100     : Rem ganze Zahl
const mwstsatz   = 0.19          : Rem normale Dezimalzahl – rational
```

```
‘ ACHTUNG:  Infolge des amerikanischen Ursprungs ist für
‘ gebrochene Zahlen der Punkt statt Komma zu verwenden !
const Meldungstext1  = “Alles in Ordnung“
```

- VB verfügt intern bereits über eine große Anzahl (einige 100) von vordefinierten Konstanten. Diese helfen bei der Lösung häufiger Aufgaben oder definieren Auswahlmöglichkeiten bei Funktionsaufrufen.

Anweisungen sind eine Kombination von Operatoren, Konstanten und Variablen.

Die Ergibtanweisung

- ist die häufigste Form der Anweisung in Programmen
- rechte Seite definiert eine Berechnung und weist das Ergebnis der linken Seite zu

```
y =  x * 300 + z
```

- Mehrere Anweisungen können auf einer Zeile, durch : getrennt, plaziert werden:

```
x = a * 200  :  a =  b * c  :  Rem Mehrfachanweisungen
```

- Es wird immer von links nach rechts ausgewertet ! Auch vor Kommentaren muß ein : gesetzt werden, da diese als normale Anweisungen ausgewertet werden !
- bei gleichzeitiger Verwendung einer Variable links und rechts wird der alte Inhalt überschrieben:

```
y = y +10  :   Rem  alter y-Wert wird überschrieben
```

Grundaufbau von Visual Basic - Programmen

```
Rem  Demoprogramm

function Test1()

Dim a as integer, b as integer
Dim c as double

a = 20 :   b= 40
c = ( a * b  ) / 100

Msgbox  "c="  &  c
Debug.print  "Berechnung beendet"
end function
```

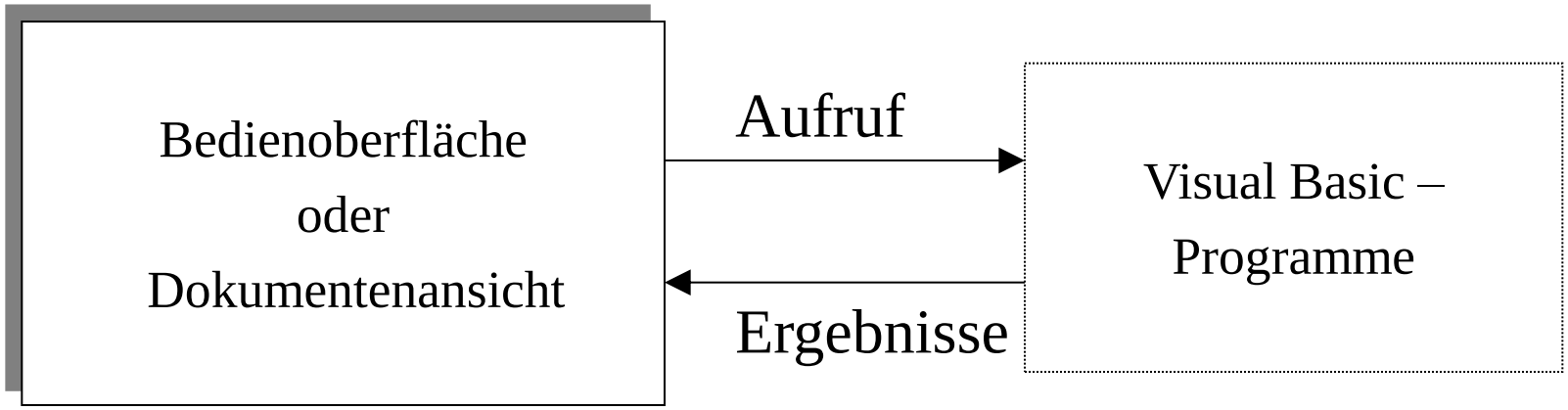
- Programmkopf mit Kommentar

- **Kopf einer Funktion**
- **Start des Funktionsrumpfes**
  - Deklaration von Variablen
  - Vergabe von Anfangswerten
  - Anweisungen (einfache Zuweisungen und Berechnung)
  - Ausgabe als Dialogbox auf dem Bildschirm
  - interne Systemausgabe
  - **Ende des Funktionsrumpfes**

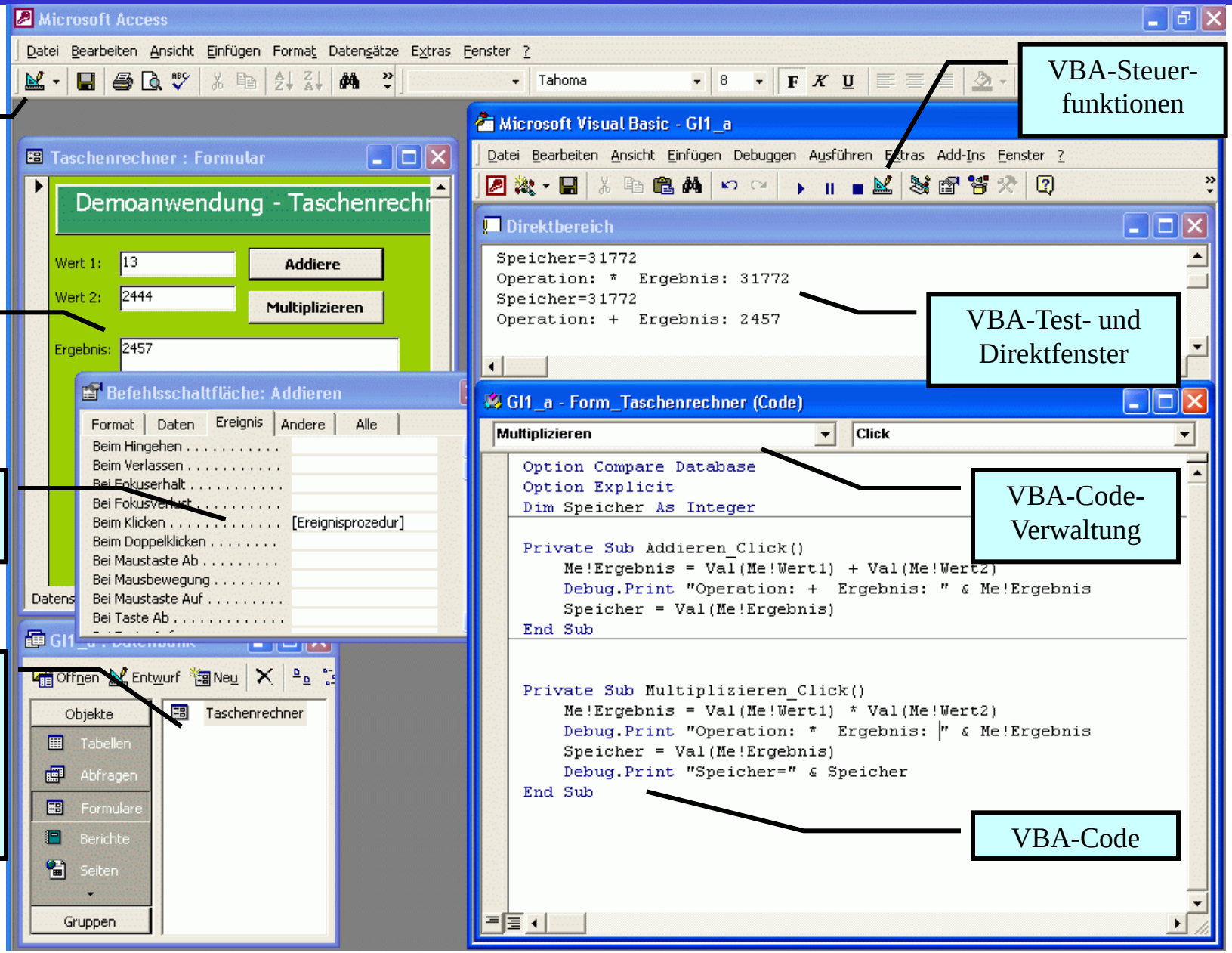
Das Programm kann einfach durch Drücken der Funktionstaste F5 gestartet werden !

Die Integrierte Entwicklungsumgebung von Visual Basic / VBA

- Visual Basic / VBA-Programme werden unter Windows eingesetzt
- die Programmierschnittstelle befindet sich in der Regel verdeckt hinter der jeweiligen Bedienoberfläche :
  - bei Winword / Excel / Powerpoint erfolgt der Zugang über **„MAKROS“**
  - bei Access ist die VBA-DIE entweder über den Modulbereich oder über die Eigenschaften von Bedienelementen (z.B. Taste) zugänglich
  - bei Visual Basic ist der Zugang wie bei Access über verschiedene Modul-bereiche oder über die Objekteigenschaften möglich



Die VBA – Entwicklungsumgebung von Access



Menüleiste mit Schalter Entwurf / Ansicht

Bedien-formular

Eigenschaften / VBA-Code

Projekt-EXPLORER zeigt alle Objekte des Programms

VBA-Steuer-funktionen

VBA-Test- und Direktfenster

VBA-Code-Verwaltung

VBA-Code

Komponenten der VB-IDE

- Der Projekt-EXPLORER listet alle Bestandteile (Objekte) der Anwendung auf.
- Einfache FORMULARE sind als Bedienschnittstellen für die ersten Übungen ausreichend. Auf diesen können alle bekannten und verfügbaren Windows-Bedienelemente angeordnet werden (siehe Folgeseiten) !
- Das EIGENSCHAFTENFENSTER zeigt zu einem Objekt alle verfügbaren Eigenschaften an und erlaubt Änderungen dieser (siehe VL Objekt-Editor).
- Über die Eigenschaft „Beim Klicken“ wird der integrierte VBA-Editor aufgerufen und ein VBA-Programm kann eingeben werden.
- Der VBA-Editor verfügt über eine Pulldownliste mit allen Programmbestandteilen. Steuertasten kontrollieren die Programmausführung.
- Das Direktfenster dient zur Hintergrundausgabe von Zwischenergebnissen und Kontrollinformationen. Es ist im Normalbetrieb für den Anwender NICHT sichtbar !

Aus- und Eingabe von Daten – Allgemein

Allgemeine Anforderungen

- eine nur auf sich selbst bezogene Berechnung ist kaum sinnvoll

Ausgaben sind notwendig zur

- Darstellung von Ergebnissen
- Information über den Verlauf von Berechnungen
- teilweise auch zur Fehlersuche

Eingaben dienen

- zur Eingabe von variablen Berechnungsparametern
- zur Steuerung der Berechnungsablaufes

Aus- und Eingaben unter Visual Basic

Unter VB ist die Ein- und Ausgabe relativ stark an die jeweilige Bedienoberfläche und den Zweck der Ein-/Ausgabe gekoppelt !

- Formularfelder dienen am häufigsten zur komfortablen Eingabe von Werten und zur einfachen Anzeige von Ergebnissen (siehe Bsp.-Formular S. 10)
- Aufklappende Dialogboxen dienen für Warnmeldungen oder für spezielle Eingaben, wie z.B. die Auswahl eines Dateiverzeichnisses auf der Festplatte
- Ausgaben auf das Direktfenster können zur Protokollierung für den Entwickler dienen, auch Manipulationen an einzelnen Werten des Programms sind durch einfache Zuweisungen möglich
- Datei-basierte Operationen stellen eine weitere Gruppe von Aus- und Eingabe dar, welche nur innerhalb des VB-Programms abläuft (siehe gesonderte Vorlesung Dateien)

Automatische Datentypumwandlungen bei der Aus- und Eingabe

- Bei fast allen Aus- und Eingaben werden nur Texte verarbeitet.
- Um auch mit Zahlen arbeiten zu können, sind Typumwandlungen von Text nach Zahl und zurück notwendig

Allgemeine Regeln:

- Bei einer Ausgabe von Zahlen werden diese meist automatisch in Text konvertiert und dann ausgegeben.
- Texte können mit dem &-Operator aneinandergefügt werden:

  "Der Wert ist " & i - ergibt bei i=3 "Der Wert ist 3"
- Falls die automatische Konvertierung nicht wie gewünscht arbeitet, kann mit der Funktion format(Wert, Formatangaben) eine explizite Konvertierung erfolgen : text = format(Wert, Formatangaben)
- Bei Eingaben ist mit val( ) der entsprechende Zahlenwert zu ermitteln :

  i = val(Eingabetext)

Aus- und Eingaben über Formularfelder

Formularfelder sind eine sehr flexible und komfortable Ein-/Ausgabeoption :

- jedes Formularfeld hat einen eindeutigen Namen, z.B. Nettobetrag
- das aktuelle Formular wird mit dem Schlüsselwort **ME** angesprochen
- andere Formulare werden mit **Forms("Formularname")** referenziert
- die Formularreferenz ergibt gekoppelt mit einem „!"und dem Feldnamen den Verweis auf das Formularelement :

  Me!Nettobetrag oder **Forms("Rechnung")!Nettobetrag**

- in der Regel kann lesend und schreibend zugegriffen werden :

  Me!Bruttobetrag = Me!Nettobetrag * 1.16

Vorteile / Nachteile von Formularfeldern

- Komfortabel und leistungsfähig für alle Standardein- und ausgaben
- leider hoher Platzbedarf durch ständige Anzeige auf Bildschirm

Aus- und Eingaben mit Dialogboxen - Msgbox

Dialogboxen sind sinnvoll bei seltenen oder sehr speziellen Ein- und Ausgaben

Meldungsboxen

- Grundbefehl **Msgbox** (engl. Message Box)
- einfache Textmeldung: Msgbox (“Achtung : Wert ist zu klein !“)
- Meldung mit Zusatzdaten Msgbox (“Zaehler hat den Wert“ & i)
  - (der Operator & koppelt den Text und den Wert miteinander)
- Meldungen mit Rückgabewert i = MsgBox("Abbrechen?", vbYesNo)
  - Die Angabe vbYesNo stellt eine Konstante dar und führt zur Anzeige von:

  - der Rückgabewert liefert einen Code für Ja/Nein
  - Meldungsboxen sind MODAL, d.h. sie halten den Lauf des Programms an !
- Das Schlüsselwort Msgbox kann mit weiteren Parametern aufgerufen werden, welche den Titel des Fensters und Hilfe-relevante Daten einstellen ...

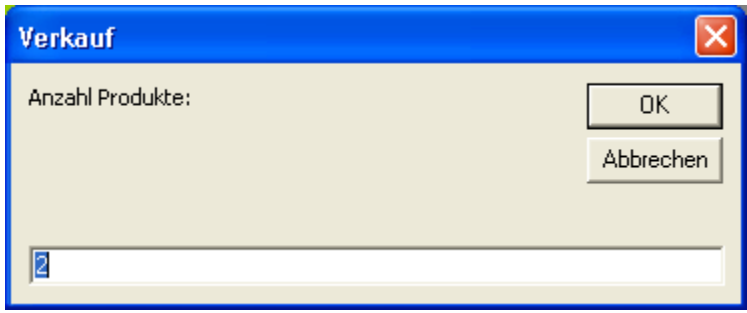
Aus- und Eingaben mit Dialogboxen - Inputbox

Seltene Eingabewerte können mit der Inputbox eingegeben werden:

Eingabedialogbox

- Grundbefehl **Inputbox**
- mit den Attributen : Anzeigetext Titel Vorgabewert ...

```
anzahltext = InputBox("Anzahl Produkte:", "Verkauf", "2")
anzahl = val(anzahltext)
```



  - der Vorgabewert setzt einen Ausgangswert
  - falls nicht abgebrochen wird, übergibt Inputbox den eingegebenen Wert an die linke Variable
  - weitere Attribute für Hilfe und Positionierung
- Inputbox hält ebenfalls den Programmlauf an (modal) und sollte nur für sehr wichtige oder seltene Eingaben eingesetzt werden

Aus- und Eingaben im Direktfenster

Für Testzwecke wird das Direktfenster verwendet !

- Ausgaben im Direktfenster erfolgen mit dem Befehl **Debug.print**
  - (Print stellt dabei eine Methode des Debug-Objektes dar)
- es können beliebig viele Werte ausgegeben werden
  - bei Trennung durch Komma werden Spalten generiert

    Debug.print anzahl, preis, me!brutto : rem Spaltenweise
  - bei Trennung durch Semikolon werden alle Werte aneinander gereiht

    Debug.print anzahl ; preis ; me!brutto : rem aneinander

- Werteänderungen erfolgen im Direktfenster mit einer normalen Zuweisung :

  anzahl = 10

  Me!Brutto = 0

  (Werteänderungen müssen für den weiteren Programmlauf plausibel sein !!)

Steueranweisungen (IF ELSE)

- Die wichtigste Steueranweisung ist der logische Test mit bedingter Ausführung der Folgeoperationen je nach Testergebnis :
  - entspricht den einfachen Prozessorbefehlen zur Verzweigung
  - dient zum Aufbau aller anderen Steuerbefehle
- **Syntax** (vollständige, lange Form)

```
if  Bedingung then
   Anweisungen_wenn_wahr
else
   Anweisungen_wenn_falsch
endif
```

Beispiel:

```
if i > 10 then
   i = 1
else
   i = i+1
endif
```

Was bewirkt dieser Programmcode bei mehrfachem, wiederholten Aufruf ?

Steueranweisungen (IF ELSE) II

- Der ELSE-Zweig kann auch entfallen !

```
if Bedingung then
   Anweisungen_wenn_wahr
endif
```

- Bei wenigen Anweisungen können diese auch direkt hinter THEN / ELSE auf GENAU EINE Zeile geschrieben werden ! Das endif entfällt dann ! (Spart Platz im Quelltext, sonst kein Effekt !!)

```
if  Bedingung then Anw1: Anw1b else Anw2
```

- Es kann auch ein IF-ELSE in ein anderes geschachtelt werden !

Beispiel:

```
if  i > 10 then
      if z > 10 then i =-z else i=1
else
 i = i+1
endif
```

Mögliche Bedingungen

- Als ***Bedingung*** muss ein auswertbarer Ausdruck angegeben werden, welcher einen Wahrheitswert oder eine Zahl ergibt
- Typische Bedingungen sind logische Vergleiche, wobei die einzelnen Glieder auch wiederum aus mehreren Ausdrücken bestehen können :

```
a=b   a>b   a<b   a>=b+c    c<= B+2

a<> b : Rem realisiert UNGLEICH
```

- Bei Vergleichen steht das = nur für die Gleichheit, nicht für eine Zuweisung !!
- Vergleiche können mit den logischen Operationen AND, OR, NOT, XOR auch kombiniert werden :

```
If  (a>  10  AND  a < 30)  OR  (c<1 AND c>0)  then …
```

- Falls nur ein Rechenausdruck angegeben wird, wird ein
  - ein **Wert 0 als FALSCH** und
  - jeder andere Wert als WAHR interpretiert !
  - es gibt 2 vordefinierte Konstanten TRUE (=-1) und FALSE (=0)

```
If  a  then  Msgbox "a ist ungleich 0 !"
```

Steueranweisungen (select case)

- Die SELECT - Anweisung entspricht einer mehrfachen IF-Kette:
  - einfachere und schnellere Programmierung – die Bedingung muss nur einmal angegeben werden, beliebig viele Fälle (= Case) sind möglich
  - übersichtlicher und besser strukturierbar

```
Select Case   Zahlenwert
   Case 1   : Anweisungen
   Case 2   : Anweisungen
   Case 3,4,5 : Anweisungen
   Case 8 to 20 : Anweisungen
   Case else  :  Anweisungen_ansonsten
End select
```

- Es können nur Zahlen (oder Einzelbuchstaben) verarbeitet werden !
- Als Case-Wert können einzelne Werte, Aufzählungen oder ganze Bereiche angegeben werden !
- Das case else dient zum Abfangen von nicht gefundenen Fällen.

Typische Anwendung von select case

- Die Select-Anweisung wird meist bei einer größeren Anzahl von Alternativen eingesetzt.
- häufige Anwendung bei der Auswertung von Anwenderkommandos

```
Befehlsnr = ...   //  Eingabe Befehlsnr
Select Case Befehlsnr
   case 1   : call Drucken()
   case 2   : call Eingeben()
   case Else :  Msgbox "Falscher_Befehl"
End Select
```

- Die Anweisungen Drucken und Eingeben stellen den Aufruf von Funktionen dar, welcher die eigentlichen Aktionen ausführen (-> siehe Funktionen)

Steueranweisungen (do while ... loop)

- Universelle Schleife mit frei definierbarer Abbruchbedingung
  - sinnvoll bei unbekannter Anzahl von Zyklen
- **Syntax** (genaue Schreibregel - jede Abweichung führt zum Syntaxfehler)

```
Rem Kopfgesteuerte Schleife
Rem läuft bei falscher Bedingung überhaupt NICHT durch die Anweisungen !!!
Do While  Bedingung
          Anweisungen_wenn_wahr
Loop
```

- Solange die Bedingung richtig ist, werden die Anweisungen ausgeführt !
- Es besteht bei Schleifen potentiell immer die Gefahr einer Endlos-Schleife !
- Innerhalb der Anweisungen sollte potentiell die Bedingung beeinflussbar sein !

Beispiel:

```
i=0
Do while i<100
  Msgbox i*i   :    i = i+1
Loop
```

Steueranweisungen (do until ... Loop)

- Universelle Schleife mit frei definierbarer Abbruchbedingung
  - sinnvoll bei unbekannter Anzahl von Zyklen
  - Im Vergleich zur ersten Schleife ist die Bedingung invertiert, d.h. es wird solange wiederholt, bis (=UNTIL) die Bedingung eintritt !
- **Syntax**

Rem Kopfgesteuerte Schleife
Rem läuft bei richtiger Bedingung überhaupt NICHT durch die Anweisungen !!!

```
Do until Bedingung
         Anweisungen_wenn_wahr
Loop
```

- Solange die Bedingung NICHT RICHTIG ist, werden die Anweisungen ausgeführt !
- Es besteht bei Schleifen potentiell immer die Gefahr einer Endlos-Schleife !
- Innerhalb der Anweisungen sollte potentiell die Bedingung beeinflussbar sein !

```
i=0
Do until  i=100
 Msgbox i*i:i=i+1
Loop
```

Achtung : Gefahr bei größeren Schritten, z.B. i = i+3 !!??

Steueranweisungen - Zählschleife (for ( ))

- Zählschleife mit definierter Anzahl von Zyklen
- **Syntax**

```
For Zaehler=Startwert to Endwert
   Zählaktion(en)
Next  Zaehler
```

- Es besteht kaum die Gefahr einer Endlos-Schleife, da diese Schleife für eine genau definierte Anzahl von Durchläufen verwendet wird.
- Die Startwert-Aktion wird genau einmal am Anfang ausgeführt !
- Die Endbedingung wird nach jeder Schleife geprüft.
- Die Zählaktion wird nach jeder Prüfung und erneuten Schleifenlauf ausgeführt !

```
Dim i as integer
For i=1 to 100
 Debug.Print  i*i
Next i
```

Steueranweisungen - Zählschleife for mit Schrittweite ungleich 1

- Für andere Schrittweiten als 1 kann die Option STEP verwendet werden

Syntax

```
For Zaehler=Startwert to Endwert Step Schrittweite
    Zählaktion(en)
Next  Zaehler
```

- Die Zählaktion wird nach jeder Prüfung und erneuten Schleifenlauf ausgeführt !
- Der Zaehler bewegt sich in Sprüngen mit der Schrittweite fort !
- Es können auch negative Werte für die Schrittweite eingesetzt werden, dann sind die Start- und Endwerte zu vertauschen !

```
Dim i as integer
For i=0 to 100  Step 5
 Debug.Print  i*i
Next i
```

```
Dim i as integer
For i=100 to 0  Step -5
 Debug.Print  i*i
Next i
```

Zusammenfassung

- Visual Basic von Microsoft kann für Aufgaben mit mittlerer Komplexität und hohen Anforderungen an Stabilität und Komfort gut verwendet werden
  - Die Performance ist für normale Büroanwendungen und nicht-zeitkritische Anwendungen (KEINE Anwendung in sicherheitskritischen Bereichen!) ausreichend
  - Bei Bedarf an Hardwareansteuerungen können externe Zusatzprogramme heute über Netzwerkprotokolle (TCP/IP) angebunden werden

- Die Programmiersprache Visual Basic ist
  - gut strukturiert und leicht erlernbar
  - erlaubt sehr schnelle Entwicklungszyklen
  - und lässt sich leicht mit Datenbanken oder Internetapplikationen koppeln