Grundlagen der Informatik

Vorlesung

Aufbau und Strukturierung von VisualBasic-Anwendungen

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

Überblick zur Vorlesung

- Überblick zur Anwendungsentwicklung
  - Typen von Anwendungen
  - Kopplung von Bedienoberflächen und Programm
  - Entwicklung komplexer Anwendungen

- Aufteilung komplexer Programme in Module
  - Funktionsbegriff
  - Bibliotheken
  - allgemeine Arten der Wertübergabe
  - Sichtbarkeit von Funktionen und Variablen

Überblick zur Anwendungsentwicklung

Typen von Anwendungen (geordnet nach Historie)

- einfache Kommandozeilenoperationen (keine Anwendung an sich)
- Terminalanwendungen auf Großrechnern (siehe alte Bankprogramme)
  - Input Tastatur -> Großrechner -> Output Terminalbildschirm
  - Terminal ist „dumm“ – nur Eingabe und Ausgabe, Intelligenz nur auf Großrechner
  - heute Wiederkehr als „Thin Clients“ im Internetbereich (Server = Großrechner)
- Textbasierte Kommandozeilenprogramme (teilw. auch interaktiv)
  - Anwendung liest/schreibt aus und in Dateien oder verwaltet 40x80 Zeichen-BS
  - Bsp.: alle MS-DOS-Programme, fdisk, Systemprogramme
  - Typ heute zunehmend wieder als Internet-Serviceprogramme in Gebrauch !!!
- Grafisch-interaktive Programme (unter Windows o.ä. wie Mac / Linux-KDE)
  - Komplexe Ein- und Ausgaben (Tastatur, Maus, 3D-Grafikkarte)
  - kein direkter Zugriff der Anwendung auf Rechnerhardware

Vorteile von grafisch-interaktiven Oberflächen wie Windows

- grafische Bedienoberflächen (engl. Abk. GUI) entsprechen den visuellen Fähigkeiten des Menschen am besten (Arbeitsplatzprinzip mit Werkzeugen)
- vor Windows mussten GUI´s immer komplett NEU entwickelt werden :
  - jeweils anwendungsspezifische Definition von Bedienelementen
  - dadurch keine einheitlichen Bedienstandards
  - kaum Unterstützung der Grafikausgabe aus Druckern oder Plottern (jede Applikation musste eigene Treiber bereitstellen)

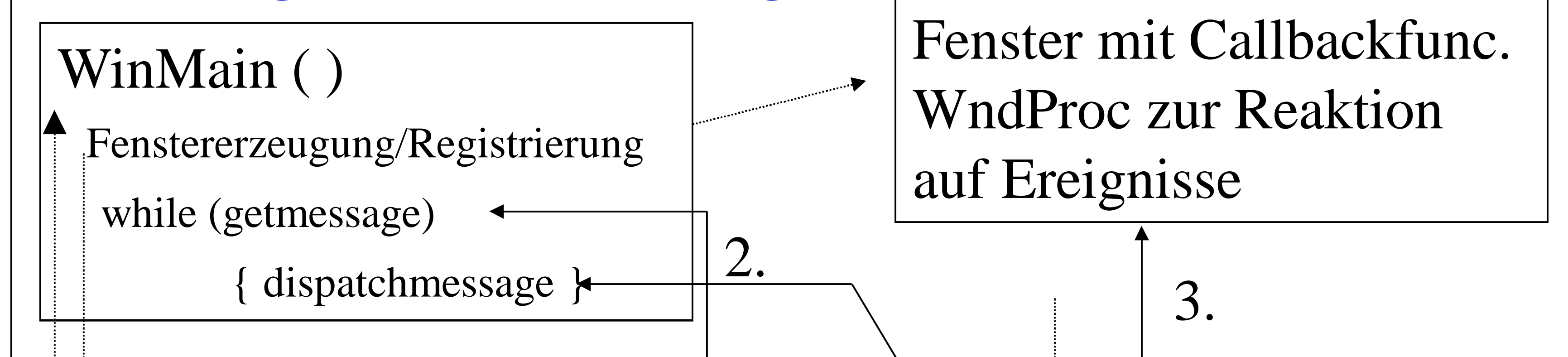
Ziel der Windowsentwicklung Ende der 80er Jahre war

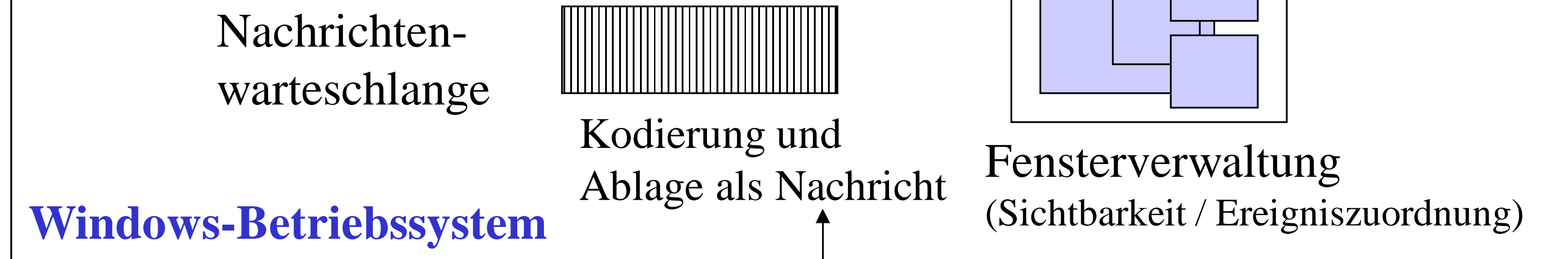
- die Definition einer zusätzlichen Softwareschicht zur Unterstützung einer einheitlichen grafischen Benutzerschnittstelle (wesentliche Vorarbeiten durch Xerox und Apple)
- einheitliche Ansteuerung aller Hardwarekomponenten über universelle Treiber

Grundprinzip von Windows und ähnlichen Systemen:

- Verwaltung aller Ein- und Ausgaben <u>durch das System</u> über Meldungssystem
- gleichzeitige Darstellung mehrerer Anwendung in Fenstern (Multitasking)
- Windows ruft bei Anwendungen entsprechende Ereignisse auf (Callback´s)

Das Nachrichtensystem von WINDOWS



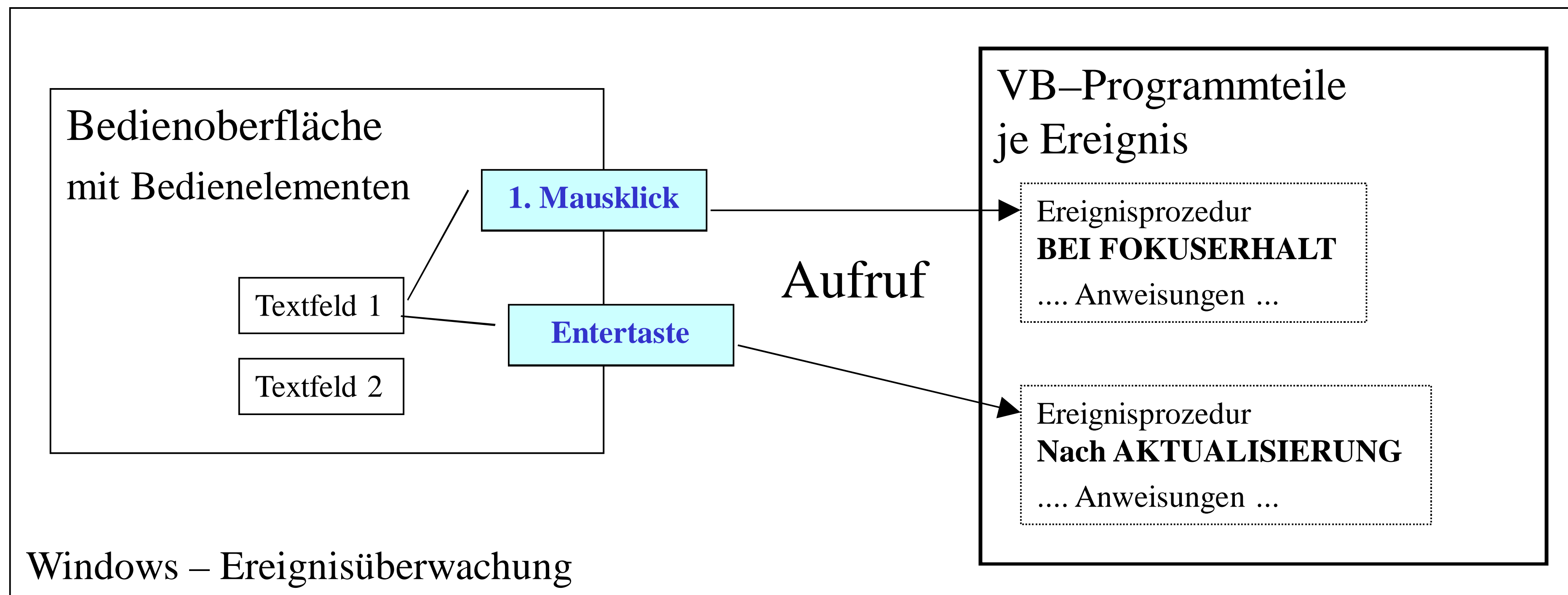


1. Hardwareereignis (z.B. Mausklick)

Hardware (Bildschirm, Tastatur, Maus, Massenspeicher, Drucker, ...)

Kopplung von Bedienoberfläche und Programm aus VB-Sicht

- Windows überwacht den Mauszeiger und die Maustasten (+Tastatur)
- bei einer Aktion wird Zuordnung zu einem aktiven Fenster bestimmt
- entsprechend der Aktion werden Ereignisse aktiviert, welche bei Bedarf mit Anweisungen hinterlegt werden können



Die wichtigsten Ereignisse unter Visual-Basic (Windows)

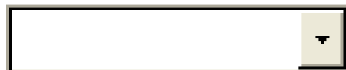
Ereignis (immer mit „Beim .. „) - Auslösung / Einsatz / Anwendungen (siehe Bsp.)

- **Fokuserhalt / Fokusverlust** **-** wenn das Element den Cursor erhält (durch Weitergehen vom vorherigen / Anklicken) – meist nur für Kontrollzwecke
- **Hingehen / Verlassen** – analog zu Fokus, jedoch mit einigen Ausnahmen
- **vor / nach Änderung** **-** wenn der alte Inhalt GEÄNDERT wurde, sehr wichtig für das Auslösen von automatischen Neuberechnungen - kann dem Anwender viel Arbeit und Probleme ersparen !!
- **Klicken / Doppelklicken** **-** bei Tasten das WICHTIGSTE Ereignis – führt zum Aufruf des Programms, welches der Taste zugeordnet ist
- **Maus-AB / AUF / Bewegung** **-** zur genauen Kontrolle der Maus – (Spielereien)
- **Taste AUF / AB** **-** zur genauen Kontrolle der Tasteneingaben – (selten nötig)

Spezielle Ereignisse:

- **Bei Zeitgeber (nur Formulare)** **-** wird von einem Timer ausgelöst, welcher mit einem Intervall in Millisekunden gesteuert wird -> alle zeitabhängigen Prozesse
- **Bei Größenänderung** - wenn ein Formular in der Größe geändert wird, zur automatischen Anpassung von GUI´s an unterschiedliche Auflösungen

Die wichtigsten Formularelemente unter Visual-Basic (Windows)

	Name (falls nicht links) /Einsatz/ Anwendungen (siehe Bsp.)
Bezeichnungsfeld	• nur Beschriftung
Textfeld	• für alle Eingaben und Ausgaben, verfügt über volle Funktionalität als Editor, Cut&Paste
Befehlsschaltfläche	• als Taste zur Auslösung von Vorgängen
Umschalttaste	• als rastende Umschalttaste mit 2 Zuständen
	• **Pull-Downliste** (dt. Kombinationsfeld) zur Auswahl von Werten aus einer aufklappenden Liste
Option1 Option2	• **Optionsfelder** (engl. Radio-Buttons) für Auswahl von eindeutigen Zuständen
Kontrollkästchen	• für JA/NEIN-Optionen
	• **Image** : Bild-Grafiken und grafische Basiselemente

Hunderte weitere Komponenten sind käuflich erwerbbar ...

Die wichtigsten Eigenschaften von Formularelementen

- **Jedes Element (auch das Formular selbst) verfügt über Eigenschaften**
- **Über** Me!Elementname.Eigenschaft **kann auf diese lesend oder teilweise auch schreibend zugegriffen werden.**

Die wichtigsten Eigenschaften sind (ohne Anspruch auf Vollständigkeit):

- **Visible :** Sichtbarkeit / zum Verbergen von Elementen
- **Links / Oben** : definiert die Position des linken oberen Ecke des Elementes
- **Breite / Höhe** : definieren die Breite und Höhe des Elementes
- **Hintergrund-, Rahmen- , Textfarbe** : entsp. Farbe (Rot Grün Blau)
- **Font** : Schriftarteneinstellung mit einer Werteliste
- **Format / Eingabeformat** : spezielle Formatierungsvorschriften für die Anzeige und Eingabe
- **Aktiviert / Gesperrt**: erlaubt Anklicken / Modifizieren des Feldinhalts

```
Bsp.:   if  Me!Saldo  <0 then    Me!Saldo.BackColor =  RGB(255, 0,0)
```

Entwicklung und Test von Programmen

Programmentwicklung im historischen Kontext :

- erste Programme umfassten nur wenige Dutzend Befehle
- rasch zunehmende Anforderungen erforderten schnell einige Tausend Befehle
- Programmierung wurde schnell zur Domäne von Experten mit starker Orientierung auf Programmierung und wenig Anwendungswissen
- die sehr große Anzahl von Freiheitsgraden bei der Programmierung (Namen der Variablen, Umsetzung mit verschiedenen Schleifentypen, Sequenz der Befehle, Art der Funktionsparameter) führt auch heute noch zu einem individuell sehr differenzierten Stil bei der Programmierung : Es ist teilweise einfacher und effizienter, einen komplexen, existierenden Algorithmus selbst noch einmal neu zu programmieren als den vorhandenen Sourcecode zu modifizieren ! (eigene Erfahrung!)
- die traditionelle Abfolge von Sourcetextänderung, Compilieren und Ausführen mit Einzeltools dauerte bis zu einige Minuten (auch als Turn-around-Zeit bezeichnet)
- einige Unarten erster Programmiersprachen, wie das sehr häufige Springen innerhalb des Programms mit dem Goto-Befehl (gibt es in VB fast nicht mehr), führte zu sehr schwer verständlichen Programmen
- alle Faktoren zusammen führten Ende der 60iger Jahre zur **"Softwarekrise"**

Die Softwarekrise

- die wachsenden Aufgabenstellungen waren mit der herkömmlichen Programmierweise nicht mehr beherrschbar
- typische Problemsymptome (auch heute teilweise noch auftretend) :
  - mehrfaches Überziehen der Kosten- und Zeitbudgets, Projektabbrüche
  - instabile Anwendungen (Abstürze, unerklärliche Fehler -> Windows ...)
  - Falsche Entscheidungen beim Grunddesign (z.B. Sicherheit beim IE)
  - Änderungen und Wartung der Programme sehr problematisch

Ursachen:

- Die komplexen Zusammenhänge großer Programme waren für den einzelnen Programmierer nicht mehr überschaubar.
- Bei Projekten mit mehreren Programmieren führte der zunehmende Verwaltungsoverhead schon bei kleinen Gruppen zu einer schnell absinkenden Effizienz.
- Auftraggeber und Programmierer leben und denken in verschiedenen Welten und verstehen sich teilweise nicht !
- Einige "Experten" sahen eine Komplexitätsgrenze für Software aufziehen !

Die Bewältigung der Softwarekrise

- Die großen Rationalisierungseffekte der EDV und damit verbundene Geldmittel forcierten die theoretische und praktische Forschung nach Auswegen.

Die Softwarekrise wurde zumindest teilweise entschärft durch :

- bessere Programmiersprachen zur **Strukturierten Programmierung**
- **Integrierte Entwicklungswerkzeuge** zur Senkung der Turn-around-Zeiten
- sehr starke **Modularisierung der Software** (siehe diese VL Teil 2)
- ein effizientes, **EDV-bezogenes Projektmanagement**

Hinweis: Da die auslösenden Faktoren der Softwarekrise immer noch vorhanden sind, kann bei einer traditionellen Programmierung ohne oder bei inkorrekter Anwendung der oben aufgezählten Lösungsmethoden immer noch eine Krise bei der Anwendungsentwicklung auftreten ! Die Existenz der obigen Prinziplösungen garantiert keinen Erfolg im konkreten Projekt !

Die strukturierte Programmierung

- Der übermäßige Gebrauch des Sprungbefehls goto führte zu relativ schlecht lesbarem Code (auch Spagetti-Code genannt) – siehe Beispiel rechts
- der Schweizer Niklas Wirth propagierte die Einführung einer geringen Anzahl von Grundkonstrukten als vollständigen Ersatz für den goto-Befehl und entwickelte dazu die Sprache Pascal
- Ziel: bessere Übersichtlichkeit und Wiedererkennung
- alle modernen Sprachen verwenden heute analog:
  - if ( ) then { } else { }
  - While { }   do { } while   for ( ) {}
  - der Goto-Befehl ist zwar teilweise noch verfügbar, wird aber kaum noch verwendet
  - als Ersatz für goto sind die Befehle break und continue zu betrachten

Altes Basic-Unstrukturiert

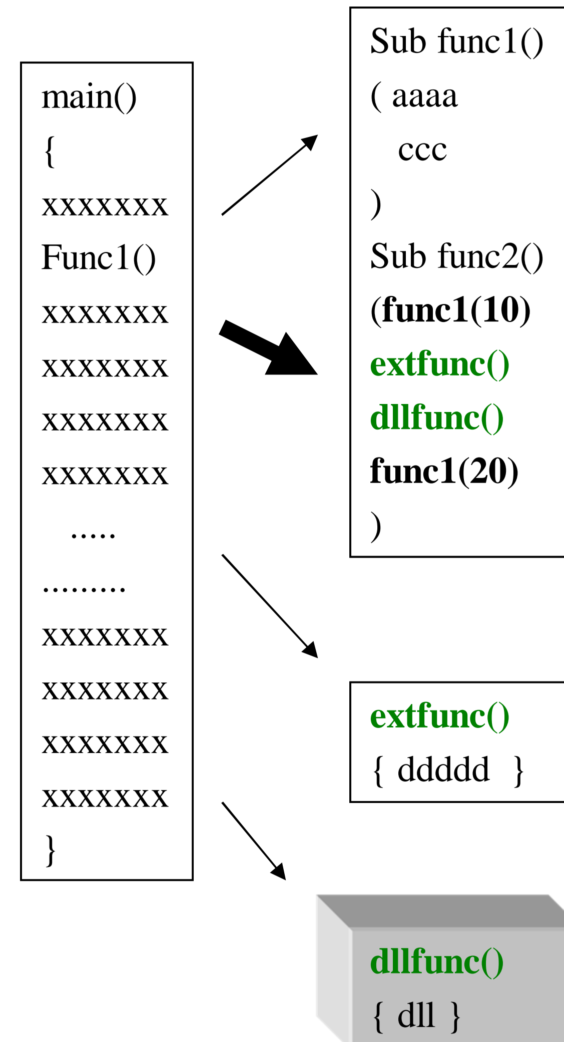
```
10:    c = 1;  a = 6;
20:    c = c + a;
30:    if (c<30) goto 20
40:    printf( "%i", c);
50:    a = a-1;
60:    if (a<=1) goto 90
70:    c=0;
80:    goto 20
90:    exit;
```

Visual Basic -Strukturiert

```
c = 0;  a = 6;
while (a > 1)
{ a = a-1;   c=0;
    while ( c < 30)
     {   c = c + a;  }
     printf( "%i", c);
}
```

Allgemeine Optionen der Zerlegung von Anwendungen

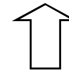
- innerhalb eines Quelltextes durch Gliederung in Funktionen und Prozeduren
- anwendungsorientierte Zusammenfassung ähnlicher Funktionen in Funktions-bibliotheken innerhalb der eigenen Anwendung (Module) oder in zugekauften Komponenten
- Programmteile (unter Windows Dynamic Link Library "DLL") mit der Möglichkeit des Aufrufes von außen
- Kommunikation zwischen eigenständigen Programmen über das Netzwerk (**aktuelle Entwicklung -> Webservices und** Anwendungsserver : Reisebuchungen ..)



Funktionen und Prozeduren

- analog zur Mathematik erhält eine Rechenvorschrift oder ein Algorithmus einen Namen und kann mit verschiedenen Parametern aufgerufen werden :

Mathematik	Informatik
$y=f(x) = \sin(x)$	$y = \sin(0.3)$ $y = \sin(180)$

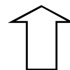


Name der Funktion

- Eine Funktion liefert genau EINEN WERT ZURÜCK !
- Eine Prozedur arbeitet analog wie eine Funktion, GIBT ABER NICHTS ZURÜCK !

Nur in der Informatik ! (kein Pendant in der Mathematik)

Motor_einschalten(Motornummer, Zeitdauer)

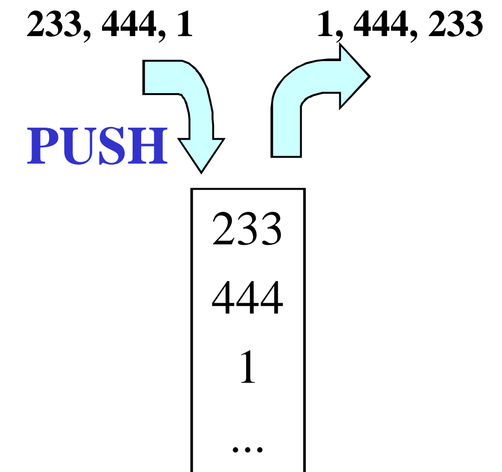


Name der Prozedur

- Aus technischer Sicht handelt es sich bei Funktionen und Prozeduren um **Unterprogramme,** d.h. um untergeordnete, von außen aufrufbare Programmteile.

Arbeitsweise des Prozessors beim Aufruf von Unterprogrammen

- da unbekannt ist, wie oft (auch rekursiv) ein Unterprogramm aufgerufen wird, existiert in allen Prozessoren ein spezieller Speicherbeich: der STACK (englisch Stapelspeicher)
- der Stack arbeitet nach dem LIFO-Prinzip: Last in -First out
- es gibt nur zwei Stackbefehle : PUSH legt Daten auf den Stack, POP holt die obersten Daten wieder vom Stack



Vorgehensweise des Prozessors beim Aufruf von Unterprogrammen

- Prozessor legt zuerst alle Aufrufparameter auf den Stack
- als letztes kommt die aktuell ausgeführte Adresse als Zahlenwert auf den Stack (= Rücksprungadresse)
- Prozessor springt zur Adresse des Unterprogramms
- Unterprogramm arbeitet und Auswertung der Parameter der Werte auf dem Stack und legt Ergebnis am Ende ebenfalls auf den Stack
- der Rücksprungbefehl am Ende des Unterprogramms holt die Rücksprungadresse wieder von Stack und springt dieser wieder an

Werteübergabe an Funktionen

Probleme:

- Funktionen werden von verschiedenen Stellen aufgerufen (Verwendung fester, globaler Übergabeadressen sehr unschön und gefährlich - potentielle Konflikte zwischen den verschiedenen Aufrufern)
- mehrfacher, rekursiver Aufruf der gleichen Funktion möglich
- bis auf einfache, mathematische Funktionen reichen die Rechenregister des Prozessors zur Werteübergabe nicht aus

Lösung:

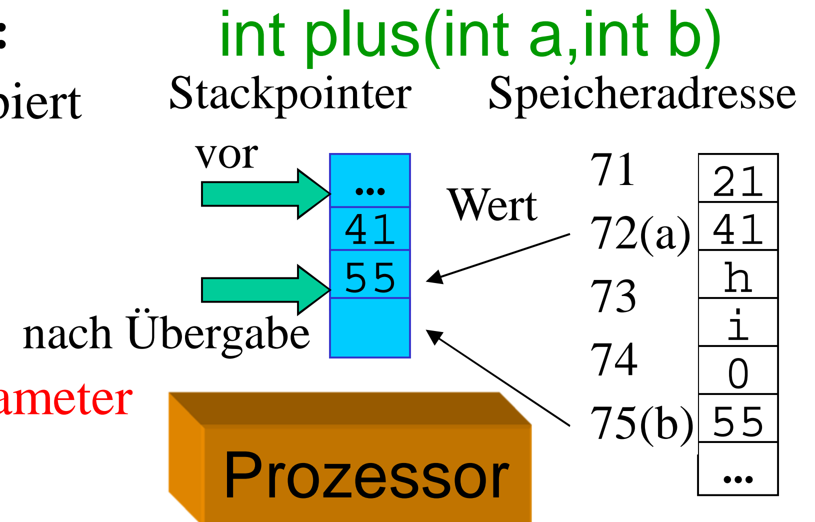
- Verwendung des Rechnerstacks auch als genereller Übergabebereich für Datemn
  - STACK = **Stapel**speicher nach dem LIFO-Prinzip - der zuletzt mit PUSH aufgelegte Werte geht zuerst mit POP wieder runter
  - Stack ist einfach realisiert als Pointer auf einen Speicherbereich, welcher bei PUSH runter und bei POP hochgezählt wird (Speicherüberlauf oder Verletzung fremder Speicherbereiche möglich -> STACK OVERFLOW ERROR !)

- Merke: Neben den Funktionsübergabewerten wird auch die Adresse des aufrufenden Programmes auf den Stack gelegt. Nach Beendigung der Funktion wird dieser Wert wieder in den Programmzeiger des Prozessors geladen. Eine irrtümliche Manipulation dieses Wertes ist meist „tödlich“ !
- Die meisten Hackerangriffe verwenden Programmfehler bei der Stackverwaltung !

Parameterübergabe by value / by reference

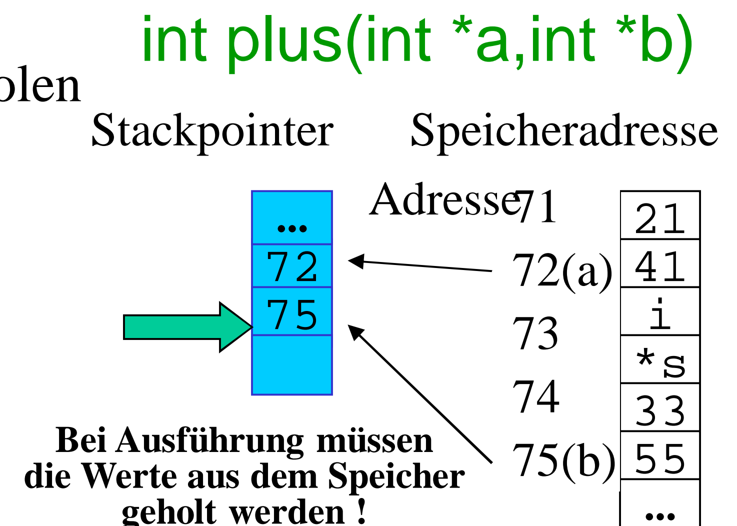
Parameterübergabe „als Wert“ (engl. **by value**) **:**

- Wert der Variable wird direkt auf den Stack kopiert
- schnell (Stackspeicher ist im Prozessor)
- Originalwert bleibt erhalten !!!
  - Gut: keine Nebeneffekte
  - Negativ: keine Wertrückgabe über den Parameter möglich (nur über Funktionswert)



Parameterübergabe „mittels Adresszeiger“
(engl. by **reference**) **:**

- die Adresse der Variable wird auf den Stack gelegt
- langsamer, Prozessor muß Wert erst aus Speicher holen
- Originalwert kann über Adresszugriff verändert werden !!!
  - Gut: Wertrückgabe möglich
  - Negativ: gefährliche Nebeneffekte möglich



Definition und Verwendung von Funktionen in Visual Basic

- Eine Funktion muß **deklariert** werden **mit dem Schlüsselwort function und nachfolgend dem Funktionsnamens, eventueller Parameter mit Typangaben und dem Rückgabetyps**
- Die Rückgabe von Werten an die aufrufende Funktion erfolgt durch Zuweisen des Ergebnisses an den Namen der Funktion.
- Die Funktion wird beendet durch die Schlüsselwörter **end function**
- Deklarationsbeispiel:

```
Function   meineFunktion  ( Wert1 as integer,  Wert2 as float) as  integer
   . . .
   meineFunktion = ergebnis : rem = return (aus and. Sprachen)
End Function
```

- Bei Verwendung (Aufruf) der Funktion sind nur der Funktionsname und die konkreten Parameter OHNE Typangaben zu schreiben , auch Schachtelungen sind möglich :

```
Bsp.1:     x = meineFunktion (  12,   f )
Bsp.2:     y = meineFunktion (  j ,   sin(f*0.3)  )
```

Definition und Verwendung von Prozeduren in Visual Basic

- Eine Prozedur entspricht einer Funktion ohne Rückgabewert !
- Eine Prozedur muß **deklariert** werden **mit dem Schlüsselwort Sub und nachfolgend dem Prozedurnamen und eventueller Parameter mit Typangaben.**
- Die Prozedur wird beendet durch die Schlüsselwörter **end sub**
- Deklarationsbeispiel:

```
Sub   meineProzedur ( Wert1 as integer,  Wert2 as float)
    . . .  : rem Anweisungen
End sub
```

- Bei Verwendung (Aufruf) der Prozedur sind nur deren Name und die konkreten Parameter OHNE Typangaben zu schreiben , auch Schachtelungen sind möglich :

```
Bsp.1:        meineProzedur (  12,   f )
Bsp.2:        meineProzedur (  j ,   sin(f*0.3)  )
```

Regeln bei der Verwendung von Funktionen und Prozeduren

Regeln und Hinweise:

- Die Zuordnung von aufrufenden Variablen zu den Funktionsparametern ergibt sich allein aus der Reihenfolge.
- Der Interpreter / Compiler prüft, ob zwischen der Deklaration und dem Aufruf
  - **die Anzahl der Parameter übereinstimmt**
    - falls zu wenige Parameter angegeben sind, erscheint **„Argument ist nicht optional“ und die Aufrufstelle wird markiert**
    - falls zu viele Parameter angegeben sind, erscheint **„Falsche Anzahl an Argumenten ... “ und die Aufrufstelle wird markiert**
  - **ob die Typen der Parameter passfähig sind. Dabei sind in gewissen Grenzen automatische Konvertierungen möglich :**
    - kleinere Datentypen werden zur größeren konvertiert
    - Zahlen oder Texte werden gegenseitig konvertiert
    - Falls dabei keine Übereinstimmung existiert und auch keine Konvertierung möglich ist, kommt es zum Fehler **“Typen unverträglich“**

Regeln II

Angaben zum Type der Werteübergabe :

- Bei Visual Basic ist CALL BY RERFERENCE die Standardübergabe, d.h. es wird eine Referenz übergeben, über welche der Originalwert auch geändert werden kann! ***(bei den meisten anderen Programmiersprachen ist Call by Value der Standard)***

- für den Aufruf CALL BY REFRENCE ist das Schlüsselwort ByRef (oder nichts als Option) zu verwenden :

  Sub meineProzedur (Wert1 as integer, **ByRef** Wert2 as float)

- für den Aufruf CALL BY VALUE ist das Schlüsselwort ByVAL zu verwenden :

  Sub meineProzedur (Wert1 as integer, **ByVal** Wert2 as float)

- für Optionale Parameter (siehe auch Fehlerliste) kann die Option Optional angeben werden :

  Sub meineProzedur (Wert1 as integer, **Optional** Wert2 as float)

Typische Gliederung von VB-Programmen mit Funktionen

```
Sub testproc( )
  Dim i As Double, mw As Double
  mw = 0.19
  i = berechne(200, mw):
  Debug.Print "Aufruf1 = " & i
  i = berechne(400, mw)
  Debug.Print "Aufruf2 = " & i
End Sub

Function berechne(wert1 As Integer,
ByVal wert2 As Double) As Double
   Dim erg As Double
   erg = wert1 * (1 + wert2)
   wert2 = 100: rem kein Effekt  bei ByVal !
   Rem Ergebnisrückgabe
   berechne = erg
End Function
```

- Blau – Prozedurgerüst
- Lokale Variablendefinition
- Aufruf der unteren Funktion mit nachfolgender Ausgabe
- Ende der Prozedur
- **Kopf der Funktion mit Parameterdeklaration**
  - Deklaration von Variablen
  - Anweisungen (einfache Berechnungen)
  - Rückgabe des Ergebniswertes

Funktionsaufruf mit ByReferenz

```
Sub testproc( )
  Dim i As Double, mw As Double
  mw = 0.19
  i = berechne(200, mw): ‘ geändert nach Aufruf
  Debug.Print "Aufruf1 = " & i
  i = berechne(400, mw)
  Debug.Print "Aufruf2 = " & i
End Sub

Function berechne(wert1 As Integer,
ByRef wert2 As Double) As Double
   Dim erg As Double
   erg = wert1  * (1 + wert2)
   wert2 = 100:
   berechne = erg:  Rem Ergebnisrückgabe
End Function
```

- Blau – Prozedurgerüst
- Alles anderen Prinzipien wie bisher …
- Ende der Prozedur
- **Kopf der Funktion mit ByReference –Übergabe**
- ByRef kann aufrufenden Programm Werte ändern (dann ggf. gefährlich)
- oder kann bei absichtlicher Verwendung auch beliebig viele Werte zurück geben (dann positiv)

Sichtbarkeit von Funktionen und Variablen

- Zur effizienten Strukturierung ist es sinnvoll, daß die Sichtbarkeit (=Verfügbarkeit) von Variablen und Funktionen genauer definiert werden kann :
- Das Schlüsselwort **public** definiert einen Bezeichner als öffentlich – d.h. im jeweiligen System können alle anderen Komponenten darauf zugreifen.
- Das Schlüsselwort **private** definiert einen Bezeichner als NICHT-öffentlich – d.h. außerhalb des aktuellen Systems ist die Variable oder Funktion NICHT verfügbar !
- Das Schlüsselwort **Global** definiert bei Verwendung anstelle von DIM eine Variable als GLOBAL , d.h. überall verfügbar !
- Allerdings besteht wie bei ByRef-Parametern die Gefahr, daß UNBEFUGTE Funktionen diesen Wert unbemerkt verändern !

Sichtbarkeit von Funktionen und Variablen

```
Dim  a as integer
Global mwst as double



Public Sub testproc( )
  Dim i As Double, mw As Double
  mwst  = 0.16
  i = berechne(200, mw): Debug.Print  i
End Sub

Private Function berechne(wert1 As Integer,
ByRef wert2 As Double) As Double
   Dim erg As Double
   erg = wert1 * (1 + wert2):   wert2 = 100
   berechne = erg
End Function
```

- lokale Variablendefinition mit DIM
- Globale Defintion von mwst
  – ab hier kann in der GESAMTEN Anwendung auf die Mehrwertsteuer zurückgegriffen werden
- Die Funktion Testproc ist öffentlich !
- Diese Funktion ist nur in diesem Modul sichtbar, sonst NICHT !

Hilfsmittel beim Test von Funktionen

Microsoft Visual Ba

Datei Bearbeiten Ansicht Einfügen D

(Allgemein)

```
Private Sub Testproc1_Click()
   Call testproc
End Sub

Sub testproc()
  Dim i As Double, mw As Doubl
  mw = 0.19
  i = berechne(200, mw):
  Debug.Print "Aufruf1 = " & :
  i = berechne(400, mw)
  Debug.Print "Aufruf2 = " & :
End Sub

Function berechne(wert1 As Int
   Dim erg As Double
   erg = wert1 * (1 + wert2)
   wert2 = 100: Rem kein Effek
   Rem Ergebnisrückgabe
   berechne = erg
End Function
```

Direktbereich

```
Aufruf1 = 238
Aufruf2 = 40400
```

- **Breakpoint / Unterbrechungspunkt**
  (Einfügen mit Klick auf Rand oder mit F9)
  => Prozessor hält die Ausführung DAVOR an
  => gelber Pfeil und gelbe Markierung zeigt Position des Prozessors VOR der Ausführung an

Ab Breakpoint weiter mit

- F8 - genau ein Schritt
- Shift-F8 - ein Schritt OHNE in Funktion zu springen (Funktion wird aber ausgeführt)
- F5 – weiter mit voller Geschwindigkeit (ggf. bis zum nächsten Breakpoint)

- **Direktfenster** zur Anzeige der Debug.Print-Ausgaben
  (Anzeige mit STRG+G oder über „Anzeige“)