CS 5142
Scripting Languages
8/30/2013
End User Programming (VBA)
About VBA

• Visual Basic for Applications
  – Visual = create GUI by drag‘ n’ drop
  – Basic = simple, for end users
  – for Applications = embedded in Word, Excel,
    Powerpoint, AutoCAD, Corel Draw, Acrobat

• Object-oriented, event-driven

• Aimed at end user programming
  – Automate task user does by hand otherwise
  – Same niche as Emacs Lisp, AppleScript
Related Languages

- **VB6**
  - Same language as VBA, different libraries
  - “End of life”, not supported on new Microsoft platforms

- **VBScript**
  - Subset of VB 6.0, for server-side scripts

- **VB.NET**
  - Not backward compatible, based on CLR
How to Write + Run Code

• Open Word, PowerPoint, or Excel
• Visual Basic Editor (Alt-F11)
  – Immediate window (edit-eval-print loop)
  – Code editor, including auto-completion
  – Debugger, object browser
  – Help system
• Macro Recorder
  – Recorded macros are useful code blue-print
• Customization
  – Add hand-written code to toolbars or menus
Lexical Peculiarities

- Case insensitive
- Line break sensitive
  - One statement, multiple lines: _
  - One line, multiple statements: :
  - Single-line comments: ’, Rem
- Literals
  - Boolean: True, False
  - Date: #April 20, 2008#, #10:15pm#
  - Variant: Empty, Error, Nothing, Null
  - Character: vbCr, vbTab, vbLf, Chr(149), …
Types

- Variant
  - String
    - (variable) (fixed-length)
  - Array
    - (numeric)
  - Object
    - (user-defined)
- Boolean
- Date
- (primitive)
- (composite)
- (numeric)
- (integral)
- (fixed-point)
- (floating-point)
- Byte
- Integer
- Long
- Decimal
- Currency
- Single
- Double
- Enum
# Variable Declarations

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit, untyped</td>
<td><code>fruit = &quot;apple&quot;</code></td>
<td></td>
</tr>
<tr>
<td>Implicit, typed</td>
<td><code>fruit$ = &quot;apple&quot;</code></td>
<td>![Type declaration characters](% &amp; @ ! # $)</td>
</tr>
<tr>
<td>Explicit, untyped</td>
<td><code>Dim fruit</code></td>
<td>![Explicit declaration keywords](Dim, Private, Public, Static)</td>
</tr>
<tr>
<td>Explicit, typed</td>
<td><code>Dim f As String</code></td>
<td>![Enforce](Option Explicit)</td>
</tr>
<tr>
<td>Constant</td>
<td><code>Public Const pi_ As Byte = 3</code></td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td><code>Dim a1()</code></td>
<td>![Optional](As type)</td>
</tr>
<tr>
<td></td>
<td><code>Dim a2(10)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>Dim a3(5 To 10)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>Dim a4(10, 5)</code></td>
<td></td>
</tr>
</tbody>
</table>
Gradual and Dynamic Typing

• *Gradual typing*: optional in declaration
• If not specified: *Variant*
  – *Dynamic typing*: check at the last moment whether runtime value legal for operation
  – E.g., same variable can hold an *Integer* at one time, and a *String* at another

• If specified:
  – *Static typing*: check at compile time
  – Compiler can optimize time+space
Arrays

• Indexing with round parentheses “( )”

• Statements
  – ReDim [Preserve] id ([new sizes/bounds])
  – “Preserve” keeps old values
  – Erase id

• Functions
  – IsArray(id)
  – LBound(id[, dim]), UBound(id[, dim])

• Options
  – “Option Base 1” overrides default base-0 indexing
# Concepts

BNF = Backus Naur Form

<table>
<thead>
<tr>
<th>Construct</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal</td>
<td>concrete code</td>
<td><strong>Courier Bold</strong></td>
</tr>
<tr>
<td>Non-terminal</td>
<td>placeholder</td>
<td><em>Times Italic</em></td>
</tr>
<tr>
<td>Ellipsis</td>
<td>omitted code</td>
<td>...</td>
</tr>
<tr>
<td>Rule</td>
<td>non-terminal definition</td>
<td><em>Lhs ::= Rhs</em></td>
</tr>
<tr>
<td>Alternative</td>
<td>choose one</td>
<td>*Alt1</td>
</tr>
<tr>
<td>Optional</td>
<td>zero or one times</td>
<td>[Square brackets]</td>
</tr>
<tr>
<td>Repeat</td>
<td>zero or more times</td>
<td>Kleene star*</td>
</tr>
<tr>
<td>Repeat</td>
<td>one or more times</td>
<td>Kleene plus+</td>
</tr>
<tr>
<td>Grouping</td>
<td>treat as unit</td>
<td>(parentheses)</td>
</tr>
</tbody>
</table>
Input and Output

- `Debug.Print "Hello, world!"
- `Application.StatusBar = "Hello, world!"
- `MsgBox "Hello, world!"
  - `MsgBox(prompt[, buttons][, title]) As Long
  - Buttons: 0 vbOkOnly, 1 vbOkCancel, 2 vbAbortRetryIgnore, 3 vbYesNoCancel, 4 vbYesNo, 5 vbRetryCancel
  - Result: 1 vbOk, 2 vbCancel, 3 vbAbort, 4 vbRetry, 5 vblgnore, 6 vbYes, 7 vbNo

- `userName = InputBox("Who are you?")
  - `InputBox(prompt[,title][,default][,xpos][,ypos]) As String
### Operator Characterization

<table>
<thead>
<tr>
<th>Operator</th>
<th>Arity</th>
<th>Precedence</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>(...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>1</td>
<td></td>
<td>Call; Indexing</td>
</tr>
<tr>
<td>+, -</td>
<td>2</td>
<td>L</td>
<td>Identity, negation</td>
</tr>
<tr>
<td>* /</td>
<td>2</td>
<td>L</td>
<td>Multiplicative</td>
</tr>
<tr>
<td>\</td>
<td>2</td>
<td>L</td>
<td>Integer division</td>
</tr>
<tr>
<td>Mod</td>
<td>2</td>
<td>L</td>
<td>Modulus</td>
</tr>
<tr>
<td>+, -</td>
<td>2</td>
<td>L</td>
<td>Additive, subtraction</td>
</tr>
<tr>
<td>&amp;</td>
<td>2</td>
<td>L</td>
<td>String comparison</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>2</td>
<td>L</td>
<td>Bit shift</td>
</tr>
<tr>
<td>=, &lt;=&gt;, &lt;, &lt;=, &gt;, &gt;=, Is</td>
<td>2</td>
<td>L</td>
<td>Comparison</td>
</tr>
<tr>
<td>Not</td>
<td>1</td>
<td></td>
<td>Negation</td>
</tr>
<tr>
<td>And, Or, Xor, Eqv, Imp</td>
<td>2</td>
<td>L</td>
<td>Logic (not all same precedence)</td>
</tr>
<tr>
<td>[Set] ... = ...</td>
<td>2</td>
<td></td>
<td>Assignment statement</td>
</tr>
</tbody>
</table>

**Arity:**
- 1 = unary
- 2 = binary

**Precedence:** from high to low

**Associativity:**
- L = left
- R = right
## Concepts

### Arity, Precedence, Associativity

<table>
<thead>
<tr>
<th>Arity</th>
<th>Number of operands</th>
<th>2 - 2</th>
<th>unary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedence</td>
<td>Binding strength</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Associativity</td>
<td>Grouping direction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Arity**
  - unary
  - binary

- **Precedence**
  - Binding strength: $2 + 2 * 2$
  - $(2 + 2) * 2$
  - $2 + (2 * 2)$
  - * has higher precedence than +

- **Associativity**
  - Grouping direction: $2 / 2 / 2$
  - $(2 / 2) / 2$
  - $2 / (2 / 2)$
  - / is left-associative

Precedence and associativity in programming usually follows the conventions from math.
## Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
<th>Associativity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(...)</td>
<td></td>
<td></td>
<td>Subexpression; Call; Indexing</td>
</tr>
<tr>
<td>^</td>
<td>2</td>
<td>L</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>+, −</td>
<td>1</td>
<td></td>
<td>Identity, negation</td>
</tr>
<tr>
<td>*, /</td>
<td>2</td>
<td>L</td>
<td>Multiplicative</td>
</tr>
<tr>
<td>\</td>
<td>2</td>
<td>L</td>
<td>Integer division</td>
</tr>
<tr>
<td>Mod</td>
<td>2</td>
<td>L</td>
<td>Modulus</td>
</tr>
<tr>
<td>+, −</td>
<td>2</td>
<td>L</td>
<td>Additive; String concatenation</td>
</tr>
<tr>
<td>&amp;</td>
<td>2</td>
<td>L</td>
<td>String concatenation</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>2</td>
<td>L</td>
<td>Bit shift</td>
</tr>
<tr>
<td>=, &lt;&gt;, &lt;=, &gt;, &gt;=, Is</td>
<td>2</td>
<td>L</td>
<td>Comparison</td>
</tr>
<tr>
<td>Not</td>
<td>1</td>
<td></td>
<td>Negation</td>
</tr>
<tr>
<td>And, Or, Xor, Eqv, Imp</td>
<td>2</td>
<td>L</td>
<td>Logic (not all same precedence)</td>
</tr>
<tr>
<td>[Set] ... = ...</td>
<td>2</td>
<td></td>
<td>Assignment statement</td>
</tr>
</tbody>
</table>
## Control Statements

<table>
<thead>
<tr>
<th>Conditional</th>
<th>If ... Then ... ElseIf ... Else ... End If</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Select Case expr ... Case Else ... End Select</td>
</tr>
<tr>
<td>Fixed-Iteration loops</td>
<td>For id = expr To expr Step expr ... Next id</td>
</tr>
<tr>
<td></td>
<td>For Each id In expr ... Next id</td>
</tr>
<tr>
<td>Indefinite loops</td>
<td>Do ... Loop</td>
</tr>
<tr>
<td></td>
<td>While expr ... Wend</td>
</tr>
<tr>
<td></td>
<td>Do While expr ... Loop</td>
</tr>
<tr>
<td></td>
<td>Do ... Loop While expr</td>
</tr>
<tr>
<td></td>
<td>Do Until expr ... Loop</td>
</tr>
<tr>
<td></td>
<td>Do ... Loop Until expr</td>
</tr>
<tr>
<td>Unstructured control</td>
<td>GoTo id / line number</td>
</tr>
<tr>
<td></td>
<td>Exit Do</td>
</tr>
<tr>
<td></td>
<td>Exit For</td>
</tr>
<tr>
<td></td>
<td>Exit Function</td>
</tr>
<tr>
<td></td>
<td>On Error GoTo id</td>
</tr>
<tr>
<td></td>
<td>Err.Raise number</td>
</tr>
<tr>
<td></td>
<td>Resume id / line number</td>
</tr>
</tbody>
</table>
Writing Subroutines

• Declaration
  – $.mods*\ Sub id [(arg*)] ... End Sub$
  – $mods*\ Function id [(arg*)][As type] ... End Function$
  – To return a value, assign to function name, else default

• Function modifiers
  – Public, Private: visibility outside module
  – Static: make all locals static

• Arguments: $mods*\ id [(\ )] [As type] [= expr]$

• Argument modifiers
  – Optional: after first, rest must also be Optional
  – ByRef, ByVal: default ByRef
  – ParamArray: must be last, allows for var-args
### Positional vs. Named Parameters

**Sub** `bake(Optional Amount=1, Optional Dish="pizza")` ... End Sub

<table>
<thead>
<tr>
<th>Positional</th>
<th>Named</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bake(2, &quot;cakes&quot;)</code></td>
<td><code>bake(Amount:=2, Dish:=&quot;cakes&quot;)</code></td>
</tr>
<tr>
<td><code>bake(2)</code></td>
<td><code>bake(Amount:=2)</code></td>
</tr>
<tr>
<td><code>bake(, &quot;cakes&quot;)</code></td>
<td><code>bake(Dish:=&quot;cakes&quot;)</code></td>
</tr>
<tr>
<td><code>bake(Amount:=2)</code></td>
<td><code>bake(Dish:=&quot;cakes&quot;, Amount:=2)</code></td>
</tr>
</tbody>
</table>

**Call syntax:**
- How to omit optional parameters
- Whether or not order matters
- Note: parentheses not required
Library Functions

- Simple data conversion: CBool, CByte, CCur, CDate, CDbl, CInt, CLng, CSng, CStr, CVar
- Complex data conversion: Asc, Chr, Format, Hex, Oct, RGB, QBColor, Str, Val
- String: InStr, InStrRev, LCase, UCase, Left, Len, LTrim, RTrim, Mid, Right, Space, StrComp, StrConv, StrReverse, Trim
- Math: Abs, Atn, Cos, Exp, Fix, Log, Rnd, Sgn, Sin, Sqr, Tan, IsNumeric
- And many more
Option Explicit

Sub LemonStar()
    Dim S As PowerPoint.Slide
    Set S = ActivePresentation.Slides( ActivePresentation.Slides.Count)
    Dim I As Integer
    For I = 0 To 8
        Dim L As PowerPoint.Shape
        Const Dpi As Integer = 72 ' 72 dots per inch
        Set L = S.Shapes.AddLine( BeginX:=Dpi*5, BeginY:=Dpi*3.75+I*Dpi/8, _
                                   EndX :=Dpi*6, EndY :=Dpi*4.75-I*Dpi/8)
        L.Line.ForeColor.RGB = RGB(I * 31, I * 31, 0)
    Next I
End Sub
VBA Documentation

• Included:
  – Macro recorder
  – Auto-completion
  – Help system
  – Object browser

• Book: Mastering VBA, 2nd edition.

• Online:
  MSDN Library ➔ Development Tools+Languages
  ➔ Visual Studio 6.0 ➔ VB 6.0
Evaluating VBA

Soap-box

Strengths
- Development environment
- Simplicity
- Availability
- Popularity
- Best tool for end users of MS Office

Weaknesses
- Single platform/vendor
- Binary format
- Security
- Missing features
  - Structured exceptions
  - Implementation inheritance
  - Regular expressions
  - Associative arrays
Announcements

• No class on Monday
• Office hours on Wednesday after class
• We have a TA:
  Ozan Irsoy (oirsoy@cs.cornell.edu)
Last Slide

• First homework is due on Friday, Sept. 6th

• Today’s lecture
  – Dynamic typing
  – Precedence and associativity
  – Basics of VBA

• Next lecture
  – Putting the V and the A in VBA
  – Application extension
  – Properties
  – Call-backs