

# CE95940 – Introduction to Visual BASIC

- Last week
- User-defined datatypes
- File I/O overview
- Mouse operations & graphics

Review common questions and problems from last week

## Last week...

- Adapting programs as dialogues
- Lists -- lots of code

2

Exercise: adapt an independent program for use as a dialogue

Not easy -- wish there was a "make VBX" button

Exercise created a copy; can also use in place

Implications for software engineering and development team strategy.

global.bas not reserved

## User-defined datatypes

- Definition:  
type person  
  name as string  
  rank as string  
  serial\_number as string  
end type
- Declaration:  
dim myself as person  
dim squad(10) as person

3

data-structuring techniques

analogous to C structs, Pascal records, (COBOL, PL/1, Assembler)

“fields”, “elements” with structure

type defines shape; declaration of variable independent

type definitions must occur in code modules, not form modules

## User-defined datatypes

- Use:  
    `myself.name = "Trevor Grove"`  
    `myself.rank = "Commander-in-Chief"`  
    `myself.serial_number = "1"`  
    ...  
    `squad(0).name = "Fred Flintstone"`  
    ...  
    `squad(1) = myself`

4

dotting operator

array index, then dotting

assign entire structures

## User-defined datatypes

- Nesting:  
type full\_name  
    surname as string  
    given as string  
end type  
type person  
    name as full\_name  
    rank as string  
    citations(10) as string  
end type

5

types inside types: user-defined, array, etc

## User-defined datatypes

- `myself.name.surname = "Grove"`  
`myself.name.given = "Trevor"`  
...  
`myself.citations(0) = "Grand Poobah"`

sequence of dotting operations

## Simple File I/O

- Sequential and random-access
- “ASCII delimited”
- Character-stream and fixed-length records
- No advanced access methods

7

Typical I/O system modelled after DOS file facilities

simple sequential files use character-delimited files (blanks, commas, quoted strings)

lower-level access for byte-stream access (random)

no ISAM etc, networking facilities from DOS only

databases replacing other than sequential?

third-party stuff.

## I/O operations

- Open, close, read, write, seek
- File modes: input, output, append, random and binary
- Access type: read only, write only, read write
- File locking: shared, lock read, lock write, lock read write

8

typical operations

some features require DOS support (SHARE), designed for use in multiprocessing or networking environments



# Opening a file

- Example:

```
dim file_num as integer
```

```
file_num = FreeFile
```

```
open "myfile.txt" for input access read lock write as #file_num
```

file\_num == file reference, handle; Freefile returns available file number

# is required part of text

## Random access

- Fixed-length records
- Functions get, put (records or characters)
- Seek (record or character)
- Fixed-length strings (eg `string*100`), user-defined types
- `Len` function

10

fundamental operations: get, put, seek

if file is binary, then characters; otherwise fixed-length records

record lengths defined by constituent types, can have fixed-length strings and user-defined types to help define record shape

use version of `Len` function to return size of entry

using help: beware of difference between seek method (db object) and seek statement (does seek) and seek function (returns current position)

won't deal with random-access in this course

## Fixed-length records

- C struct or Pascal record
- Example:  
type person  
    name as string\*50  
    phone\_number as string\*12  
    age as integer  
end type
- length is 64

11

length is  $50 + 12 + 2$

standard type sizes defined in “visual basic data types” (index entry “types”)

can use fixed-length string declarations for ordinary variables,  
questionable use?

## Delimited sequential files

- Input, output, append
- input vs input #
- print # vs write #
- Example:

12

input # reads blank, comma-delimited fields, quoted strings; stream-oriented

Input reads raw characters (space, comma, NL); must specify number of characters to read

write # generates delimiters, print # does not  
automatic type conversions (numeric to char)

## Writing a file

- `dim file_num as integer`  
`dim name_resp as string, ph_resp as string`  
  
`file_num = FreeFile`  
`open "rolodex.txt" for output access write lock read`  
`write as #file_num`  
`name_resp = InputBox$( "Name?" )`  
`ph_resp = InputBox$( "Phone number"`  
`write #file_num, name_resp, ph_resp`  
`close #file_num`
- **File record appears as:**  
`"Trevor Grove", "519 888 4679"`

13

Suppose typical responses

Write# places quotes and commas suitable for input#

close particularly important for writing -- flushes files and update directory entries

difference between close function for files and close method for db objects

## Reading a file

- `dim name_in as string, ph_in as string`  
`dim file_num as integer`

```
file_num = FreeFile
open "rolodex.txt" for input access read lock write as
#file_num
do while not Eof( file_num )
    input #file_num, name_in, ph_in
    ' process
loop
close #file_num
```

14

input number uses delimiters

strings containing blanks enclosed in quotations, otherwise quotations optional

various numeric formats, generally delimited by non-numeric characters

detail rules, see Input # function

typical processing loop; explain while statement

beware of difference between EOF function and EOF property of DB objects

## Mousing & graphics

- Mouse down, up, move
- Interactive graphic indicators

15

VB provides low-level interface to mouse events

perform actions based upon mouse movements -- draw lines, position or objects

popup menus via rmb new Windows feature -- next week

<<<drag&drop handled separately (soon)>>> not

not talking about "business graphics" -- charts, graphs etc. These are handled by add-on control packages (e.g. professional version has VBgraph)

Upcoming: graphics: simple shapes, combine with mouse to do outlining (rubber-banding), line tracing, freehand lines -- one example of things that can be done

## PictureBox object

- Display images (like Image)
- Methods: line, circle, pset, point, cls
- CurrentX, CurrentY
- AutoRedraw property
- Sample

16

picturebox is one of two objects that support drawing, other is form

line: draw a line or rectangle

circle: circle/ellipse or arc

pset: draw point; point returns colour of current point

cls: clear

currentx, currenty: current points, can be reset (but no visible effect until something is drawn)

persistent bitmaps: Windows handles redrawing: use AutoRedraw

advantage: much easier (redraw sometimes impossible); disadvantage: consumes resources (memory)

most applicable to Printer

SavePicture function, form.print method



## PB sample

- create PB, set autoredraw, line to 1000,1000, QB(1..6)
- use help to get syntax
- add “b” to make box; add “f” to fill
- add button that tacks on small box: “-step(50,50), b)
- experiment with box args; reset currenty
- demonstrate no autoredraw

17

this slide is not shown

```
picture1.Line -(1000, 1000), QBColor(5)
picture1.Line -(1000, 1000), QBColor(5), B
picture1.Line -(1000, 1000), QBColor(5), BF
```

```
picture1.Line -Step(100, 100), QBColor(4), B
```

```
For i = 1 To 5
    picture1.Line -Step(100, 100), QBColor(4), B
Next i
```

```
For i = 1 To 5
    picture1.Line -Step(100, 100), QBColor(4), B
    picture1.CurrentY = picture1.CurrentY - 100
Next i
```

```
Circle (1000, 1000), 300, , 0, 2 * 3.141592653589, .25
```

## Mouse events -- down

- MouseDown on specific object, captures subsequent mouse events
- Sub *control*\_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
- Typically used to initialize operations
- Example

18

Mouse clicks very generic, special-purpose apps require more details interaction

Most objects get mouse events, we'll look only at PictureBox

Capture: i.e. only one object at a time gets events

button says which one: bit-mask for left, middle, right

shift indicate state of ctrl-alt-shift keys -- bitmask

constants defined in constant.txt

x, y -- current position of mouse

Example: show mouse location

form, picturebox, labels x: y:

```
pb mousedown x.caption = x; y.caption = y
```

skeleton in lect4\mouse

## Mouse events - up

- Occurs when button released
- Sequence: down, up, click, dblclick
- Used to finish/restore/terminate operations
- Sub *control\_MouseUp* (Button As Integer, Shift As Integer, X As Single, Y As Single)
- Example

19

add up-time refresh to previous (copy code to mouseup)

## Mouse events -- move

- Occur whenever mouse positioned over object
- Typically use state indicator to decide relevance of event

20

frequency is irregular: depends on system load, speed of movement  
lots of mouse\_move events happening all the time -- must act on only the ones we want

typical sequence: down sets switch, move interrogates and acts accordingly, up resets

example: copy caption updates to move -- unconditional update  
add state code and interrogate -- only when pressed

cute Example: (lect4\button)

```
buttondown: button_moving = True
```

```
    org_x = X
```

```
    org_y = Y
```

```
buttonmove: If (button_moving) Then
```

```
    command1.Top = command1.Top + (Y - org_y)
```

```
    command1.Left = command1.Left + (X - org_x)
```

```
End If
```

```
buttonup: button_moving = False
```

## Of mice and methods

- Combine graphical methods with mouse operations
- Examples

21

Example: new project, form.autoredraw (skeleton in lect4\freehand)

Form.mousedown toggle on, up toggle off

move: pset (x,y),qbc(4)

line (0,0)-(x,y)

line -(x,y)

Enhance to true freehand:

down: tracking = True

startx = x

starty = y

first = True

move: If tracking Then

  If (first) Then

    form1.Line (startx, starty)-(x, y), QBColor(4)

  Else

    form1.Line -(x, y)

  End If

  first = False

End If

## Line stretching

- Colour XORing -- DrawMode
- DrawStyle -- solid, dotted, dashed
- Example

22

skeleton in lect4\stretch

```
down:tracking = True
```

```
    startx = x  starty = y
```

```
    first = True
```

```
    form1.DrawStyle = 2 ' dots, 1 is dash
```

```
    form1.DrawMode = 7 ' xor
```

```
move: If tracking Then
```

```
    If (first) Then
```

```
        form1.Line (startx, starty)-(x, y), QBColor(2)
```

```
        lastx = x; lasty = y
```

```
    Else
```

```
        form1.Line (startx, starty)-(lastx, lasty), QBColor(2)
```

```
        form1.Line (startx, starty)-(x, y), QBColor(2)
```

```
        lastx = x; lasty = y
```

```
    End If
```

```
    first = False
```

```
End If
```

-----> continued next page



## Summary

- VB file processing
- Graphics methods of PictureBox, Form
- Mouse operations

23

up: tracking = False

form1.Line (startx, starty)-(lastx, lasty), QBColor(2)

form1.DrawMode = 13 ' use the colour as is (copypen)

form1.DrawStyle = 0 ' solid line

form1.Line (startx, starty)-(x, y), QBColor(3)

=====

Summary:

File processing OK, real apps use a database probably

Graphics methods; could be used for business graphics, but too tedious

-- use a 3rdparty package

Mouse -- interesting, of questionable use unless involved with graphical applications