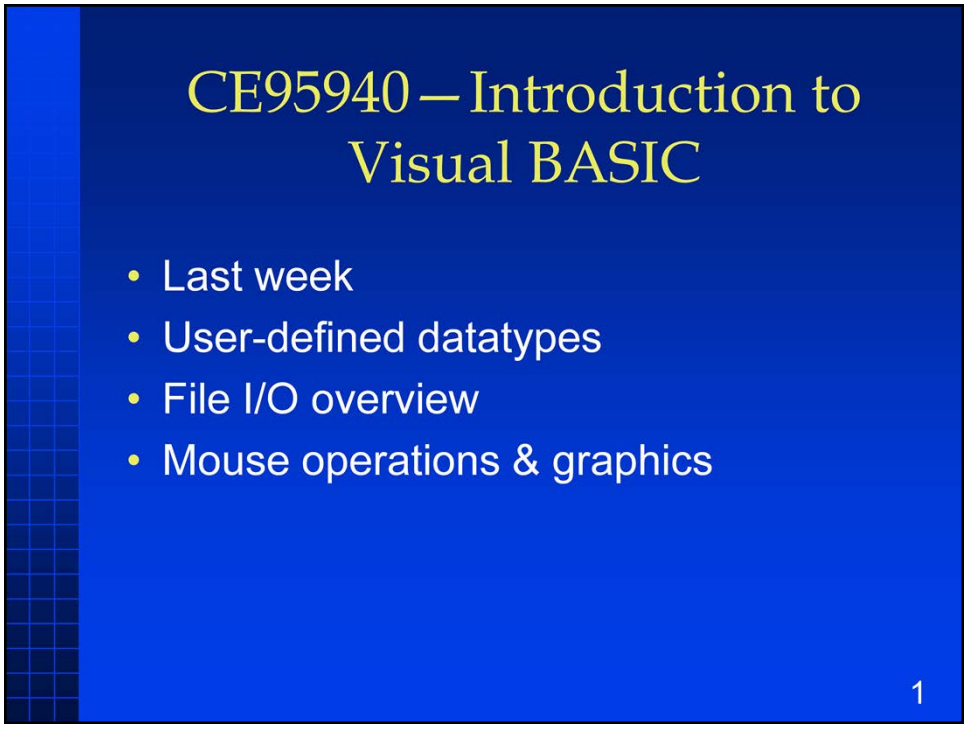
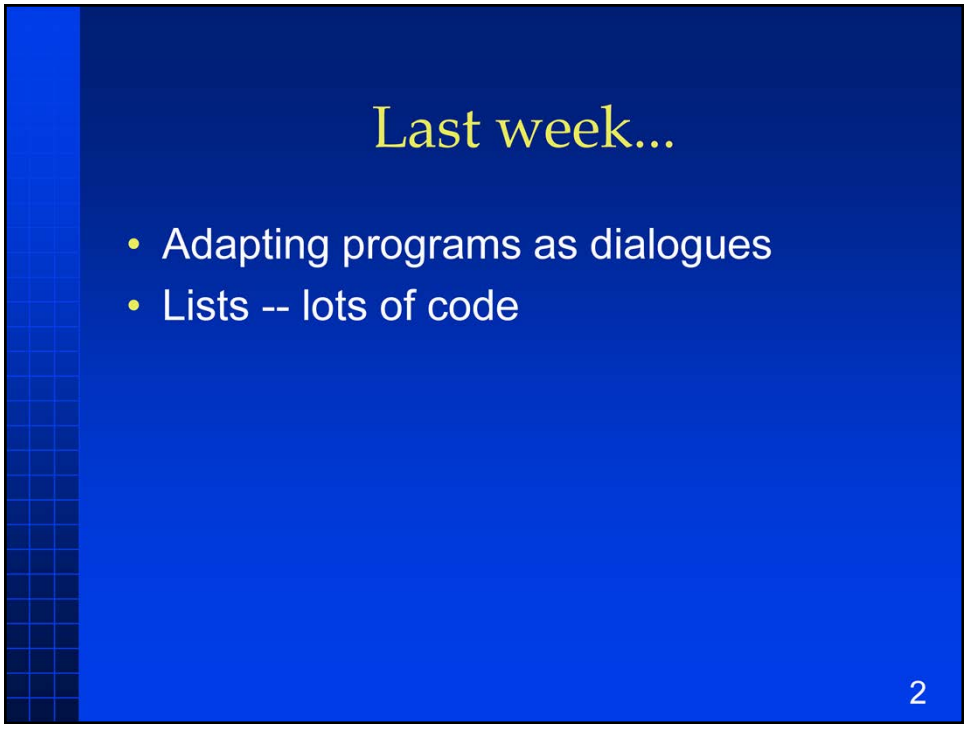


Review common questions and problems from last week



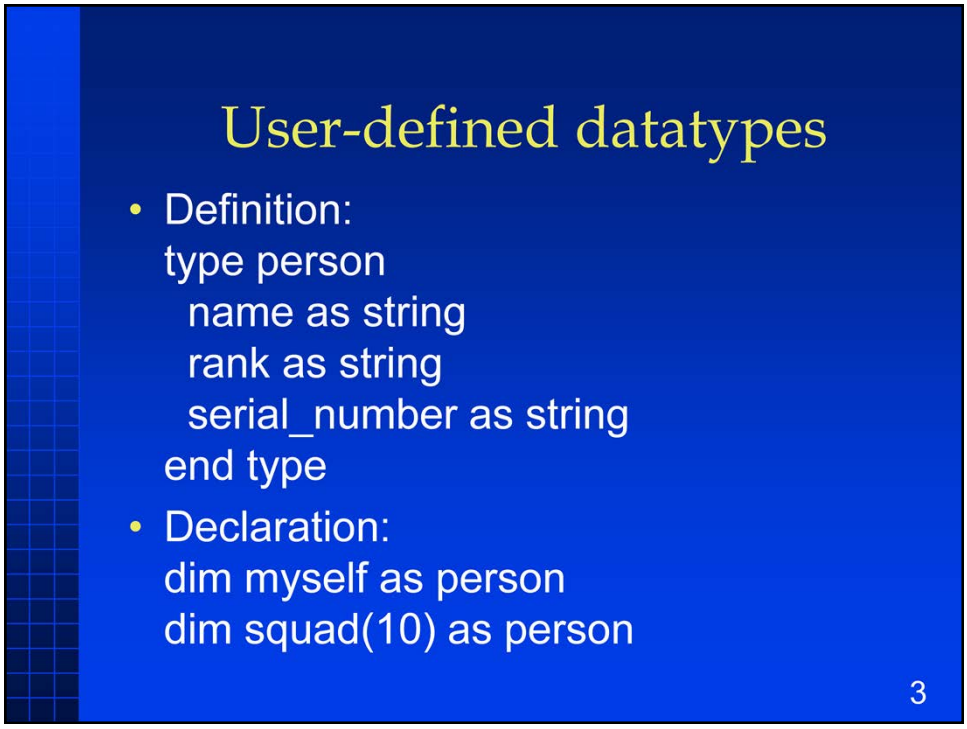
Exercise: adapt an independent program for use as a dialogue

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

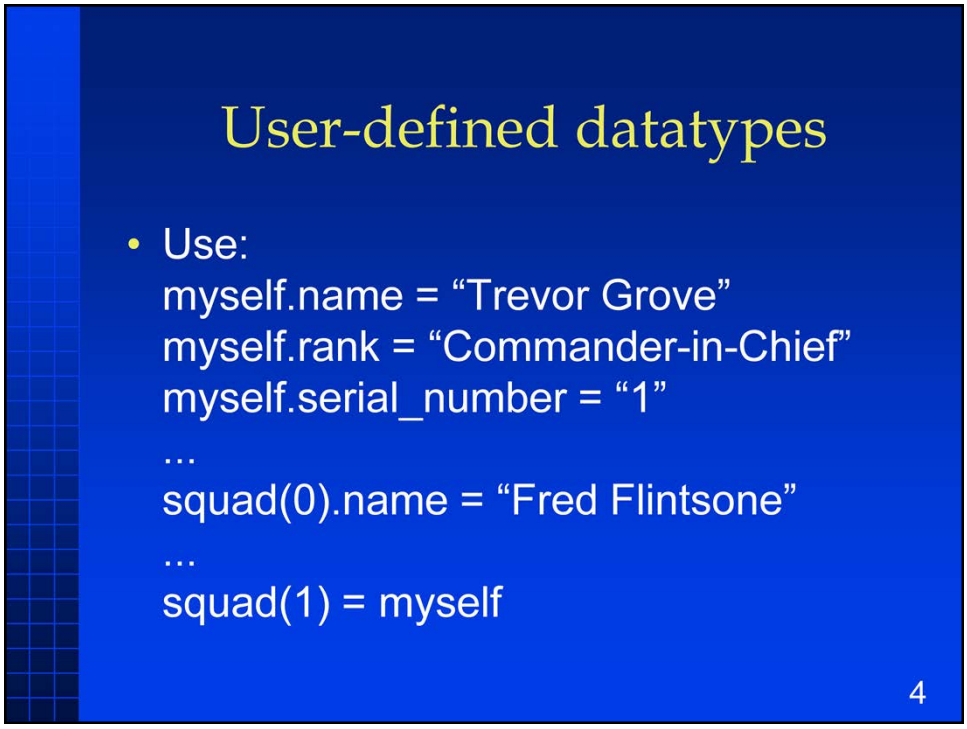


data-structuring techniques
analogous to C structs, Pascal records, (COBOL, PL/1, Asssembler)

“fields”, “elements” with structure

type defines shape; declaration of variable independent

type definitions must occur in code modules, not form modules



dotting operator

array index, then dotting

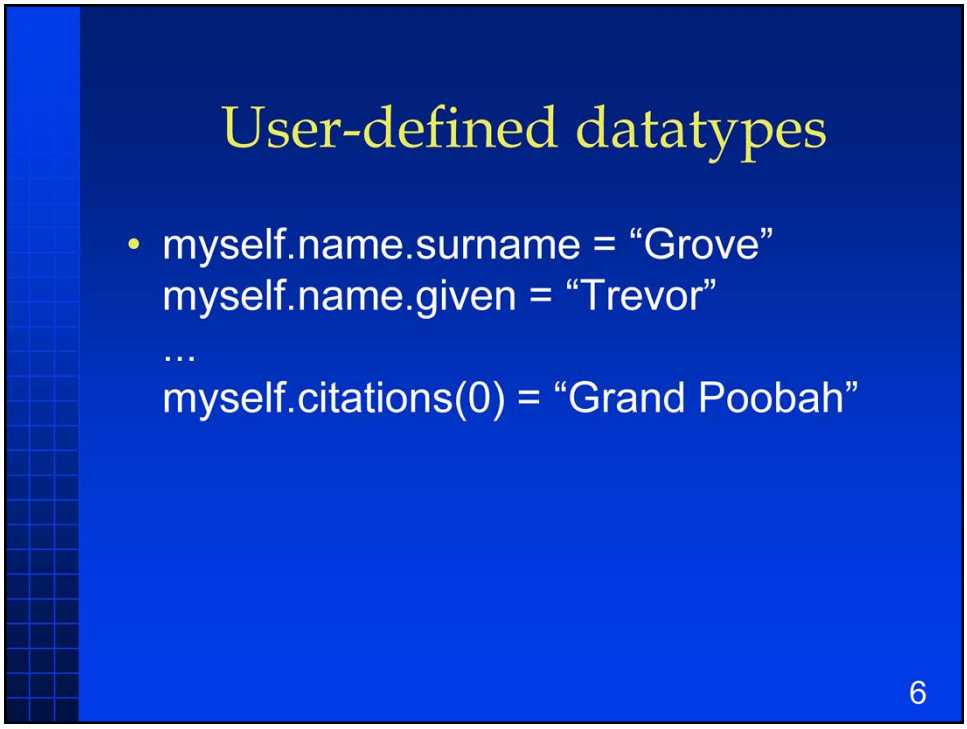
assign entire structures

User-defined datatypes

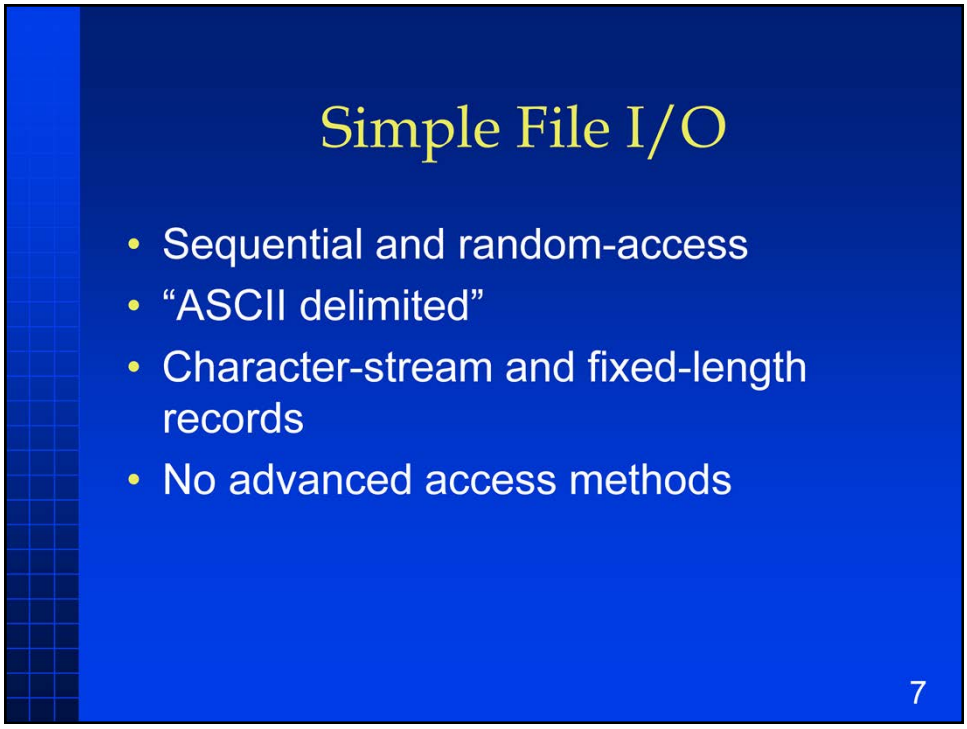
- Nesting:

```
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
```


types inside types: user-defined, array, etc



sequence of dotting operations



Typical I/O system modelled after DOS file facilities

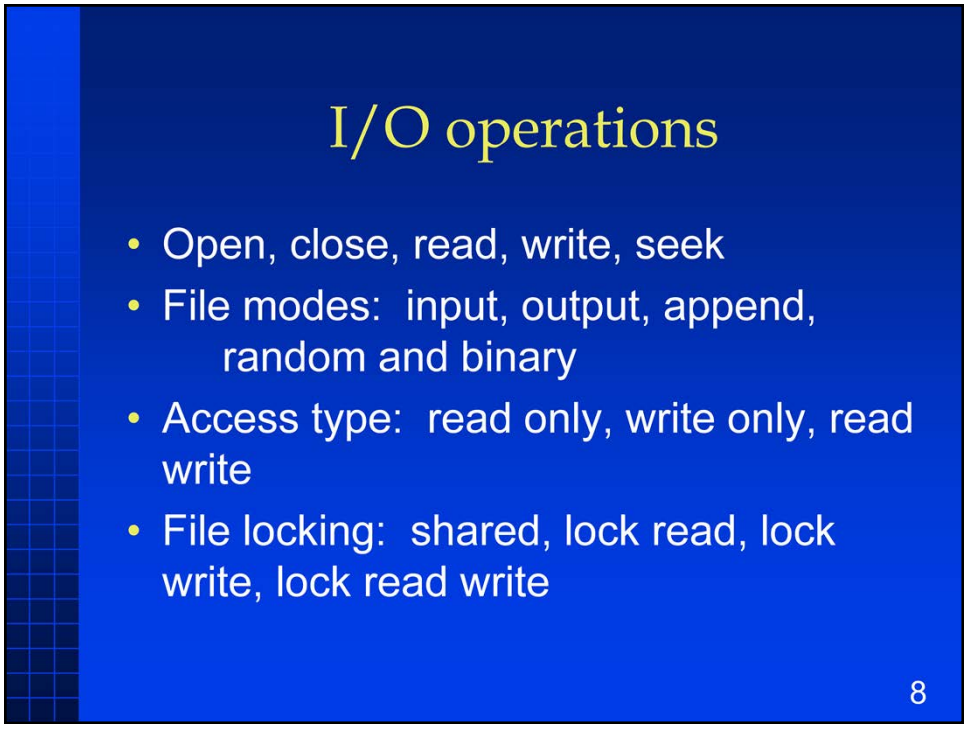
simple sequential files use character-delimited files (blanks, commas, quoted strings)

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

no ISAM etc, networking facilities from DOS only

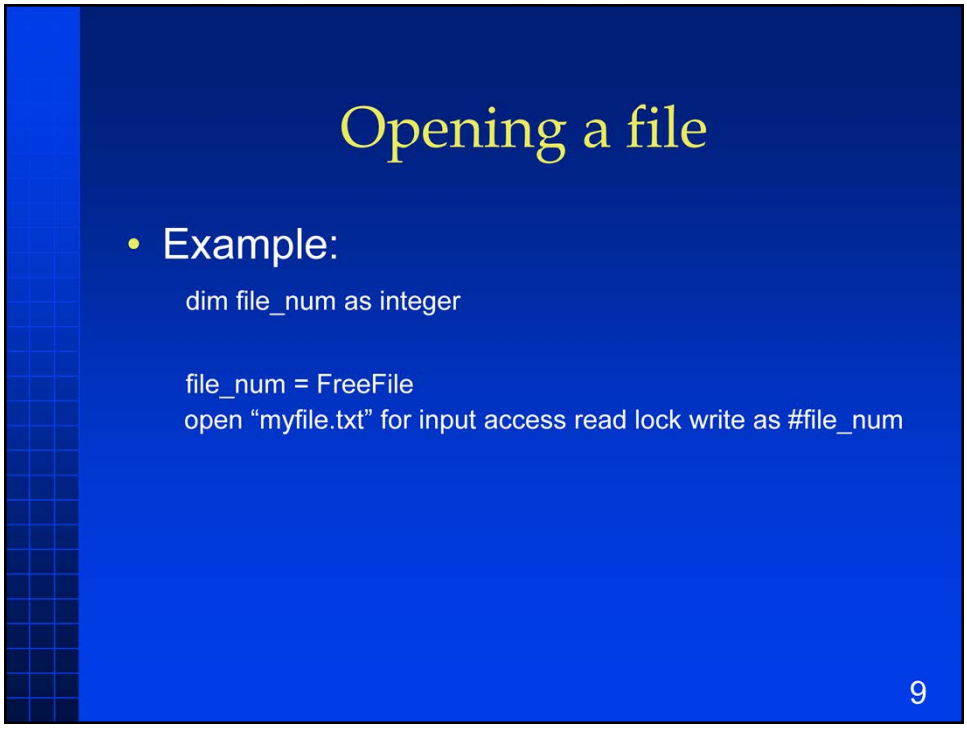
databases replacing other than sequential?

third-party stuff.



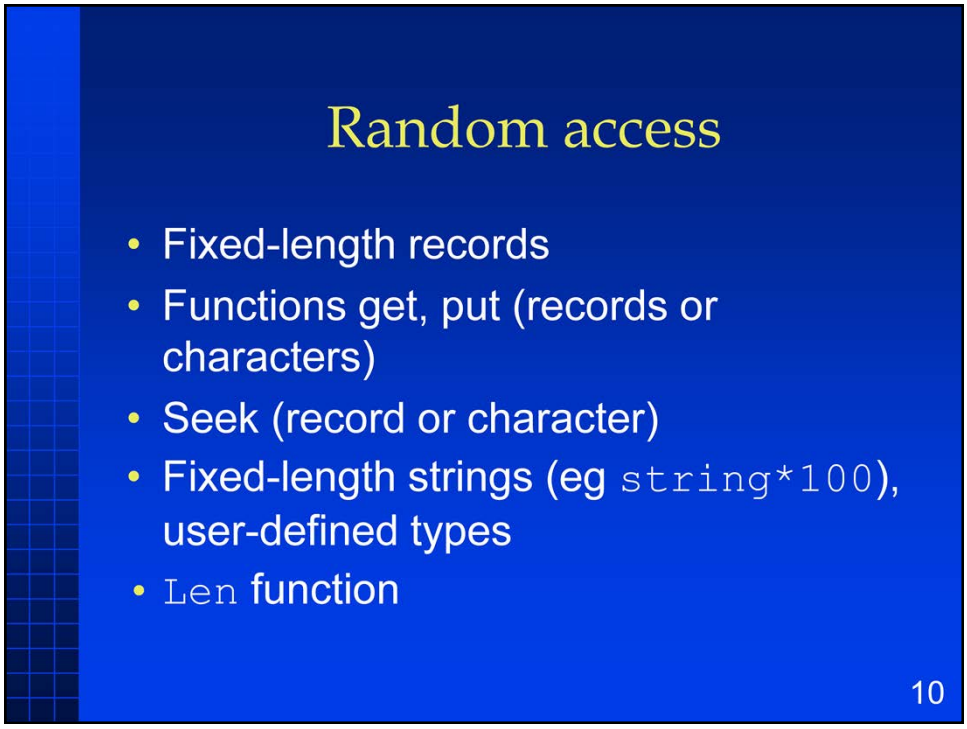
typical operations

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



file_num == file reference, handle; Freefile returns available file number

is required part of text



fundamental operations: get, put, seek

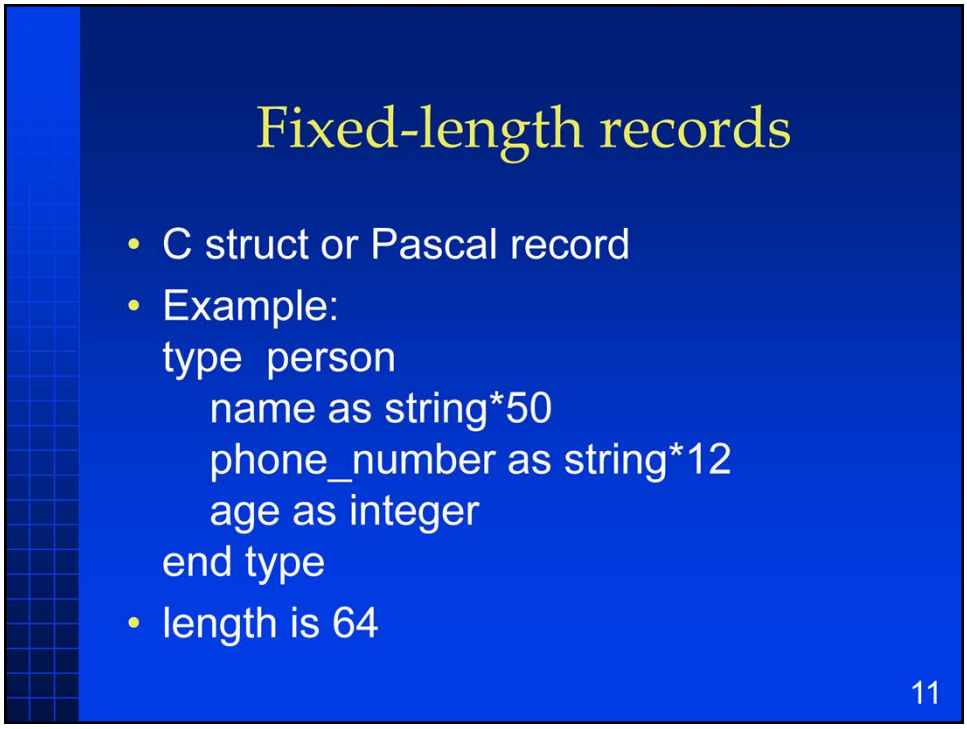
if file is binary, then characters; otherwise fixed-lenght records

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 entiry

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

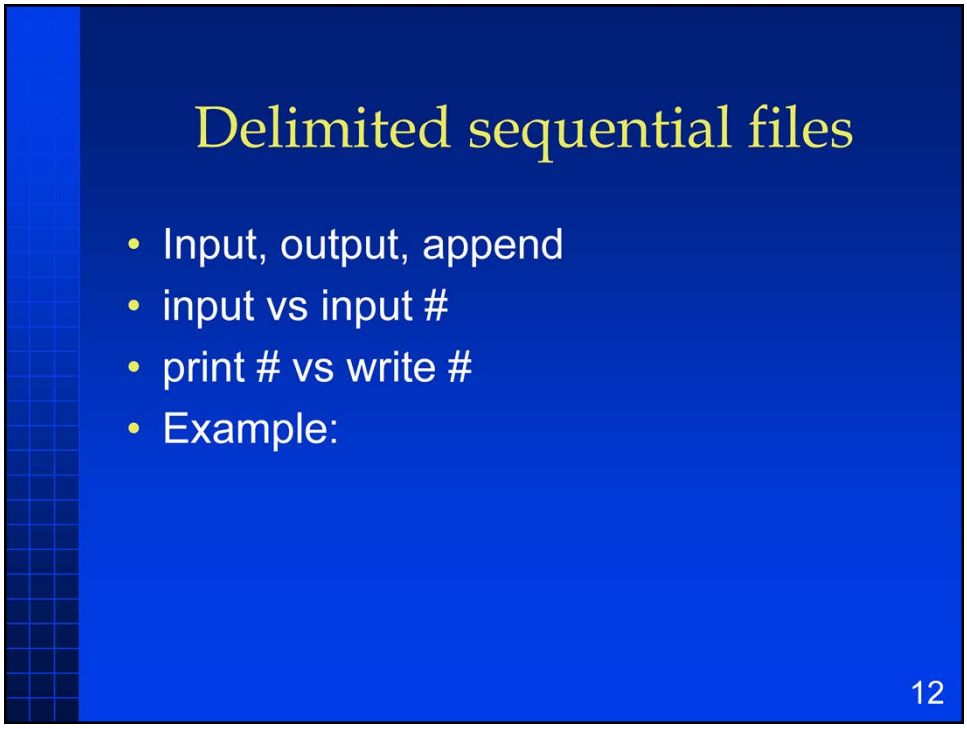
won't deal with random-access in this course



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?

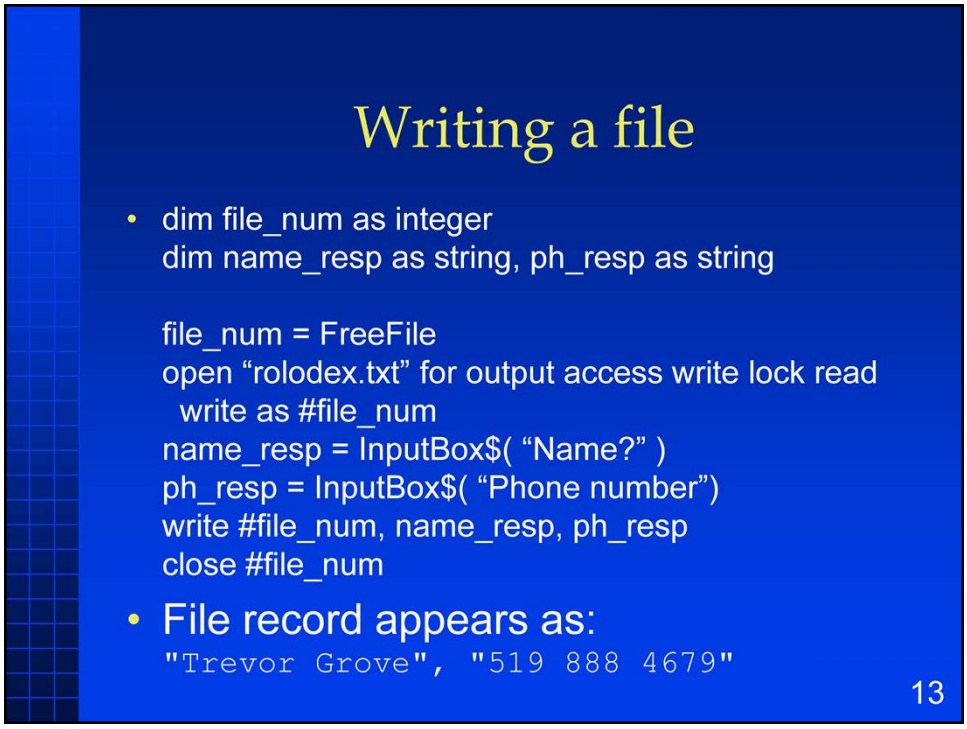


input # reads blank, comma-delimited fields, quoted strings; stream-oriented

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

write # grnerates delimiters, print # does not

automatic type conversions (numeric to char)

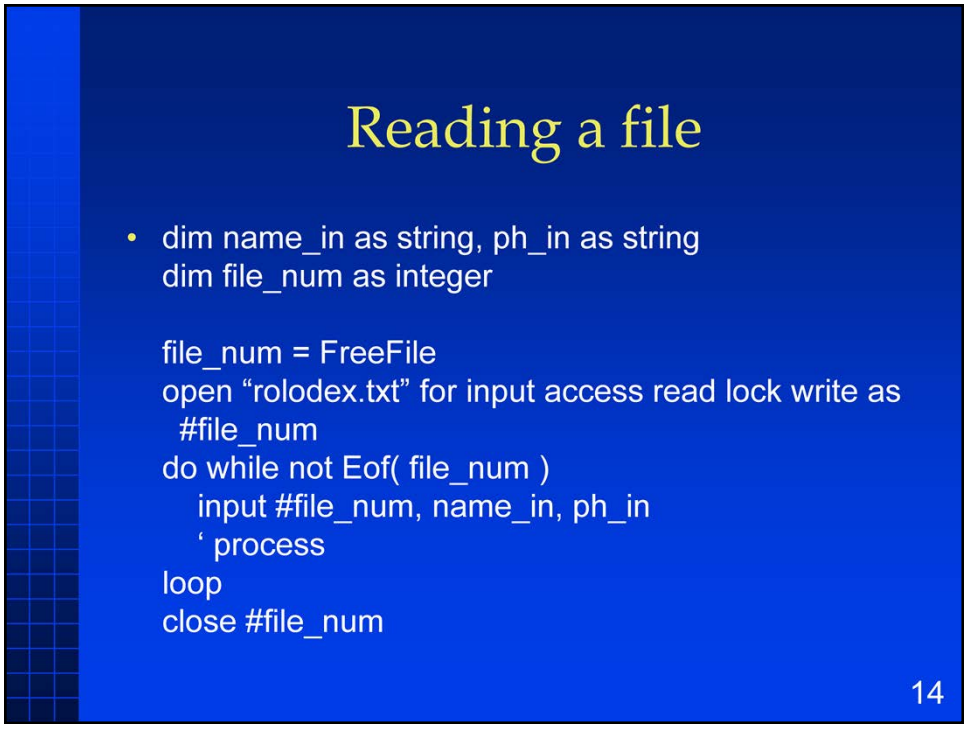


Suppose typical responses

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



input number uses delimiters

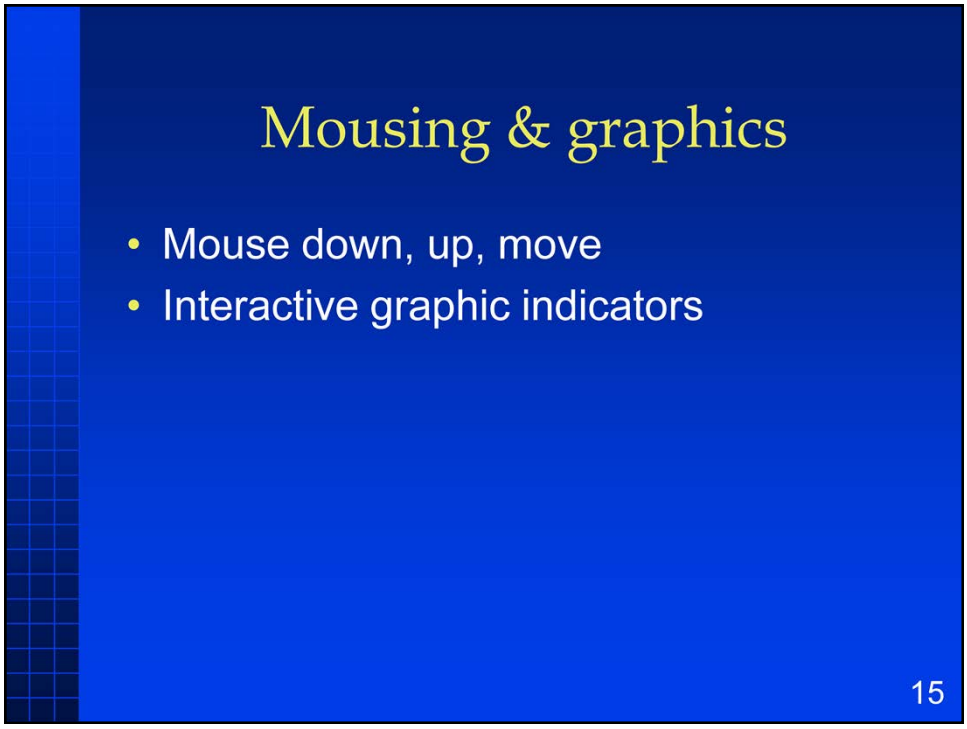
strings containing blanks enclosed in quotations, otherwise quotations optional

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



VB provides low-level interface to mouse events

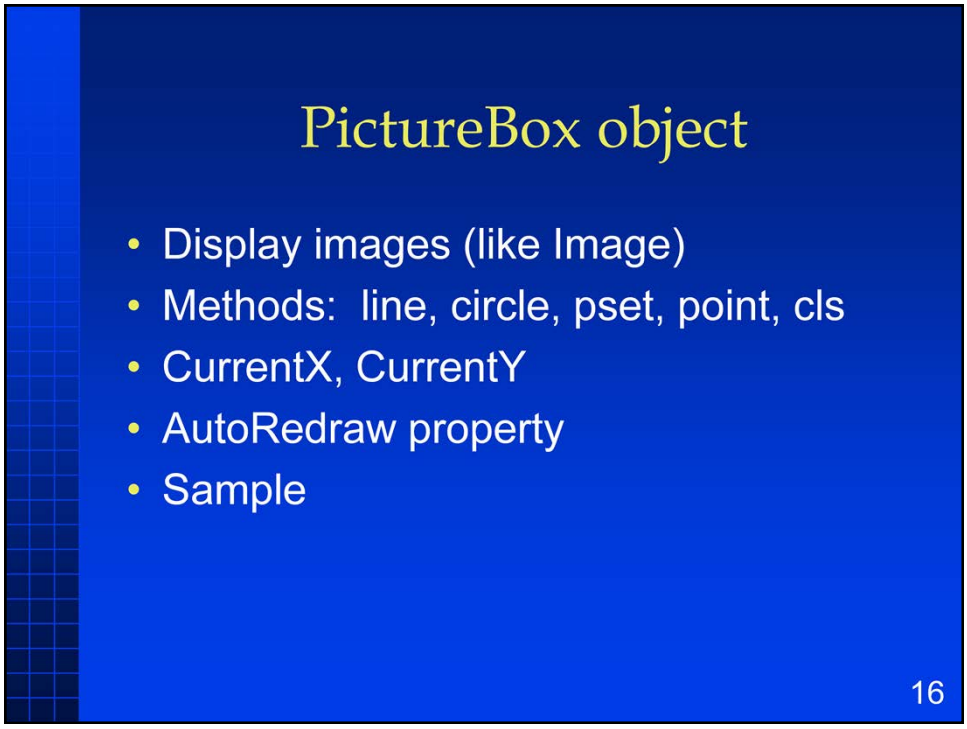
perform actions based upon mouse movements -- draw lines, position or objects

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 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

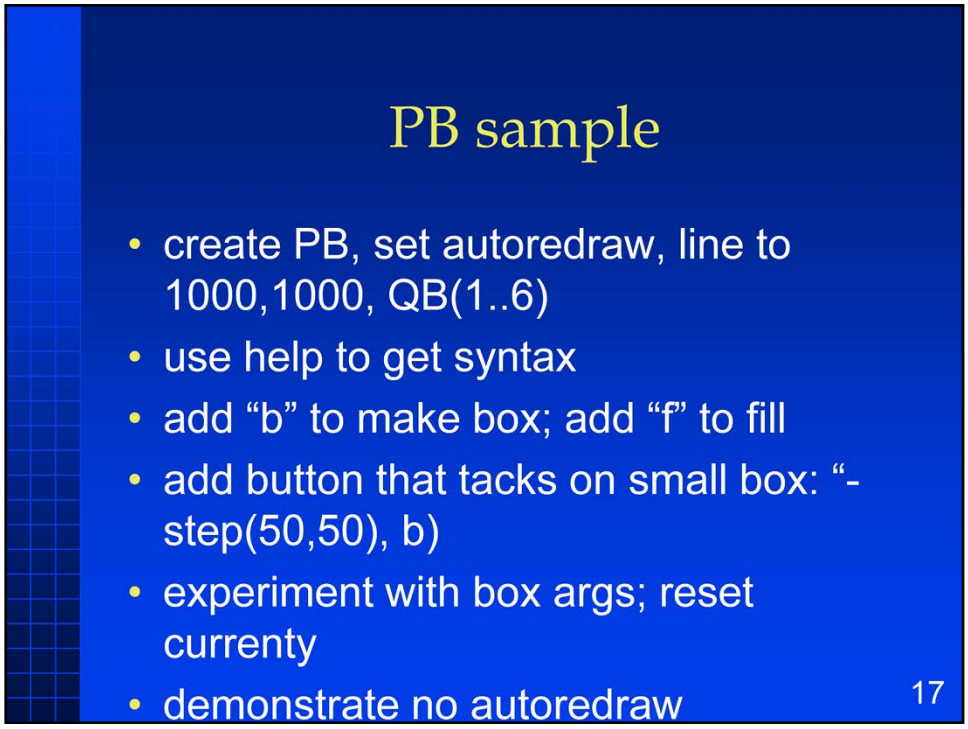
currentx, currenty: current points, can be reset (but no visible effect until something is drawn)

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



this slide is not shown

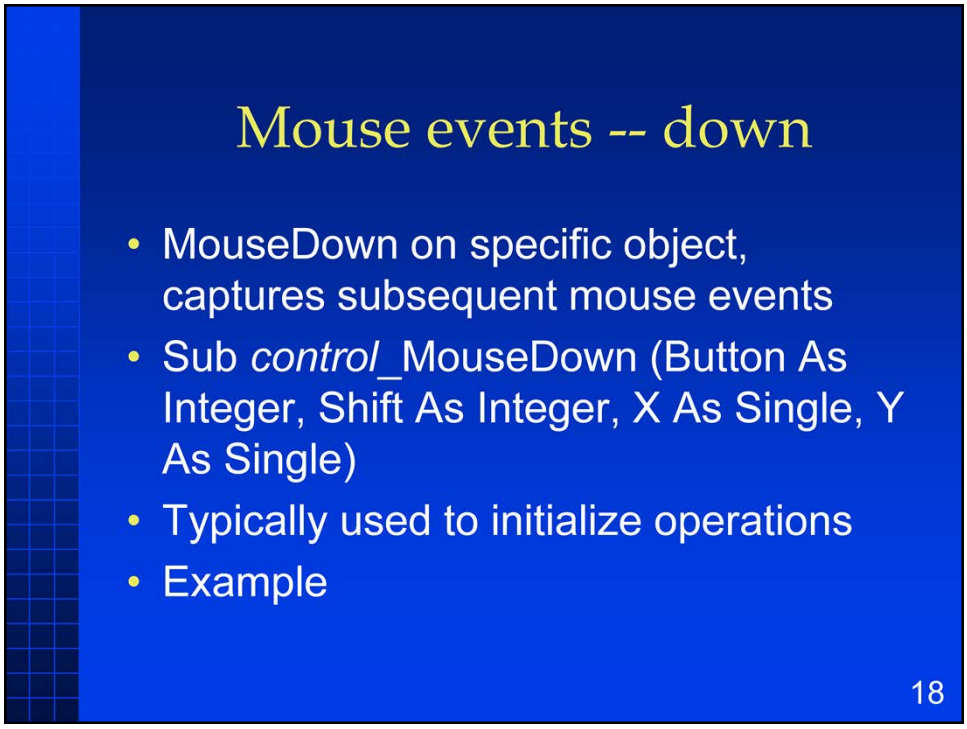
```
picture1.Line -(1000, 1000), QBColor(5)
picture1.Line -(1000, 1000), QBColor(5), B
picture1.Line -(1000, 1000), QBColor(5), BF

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
```

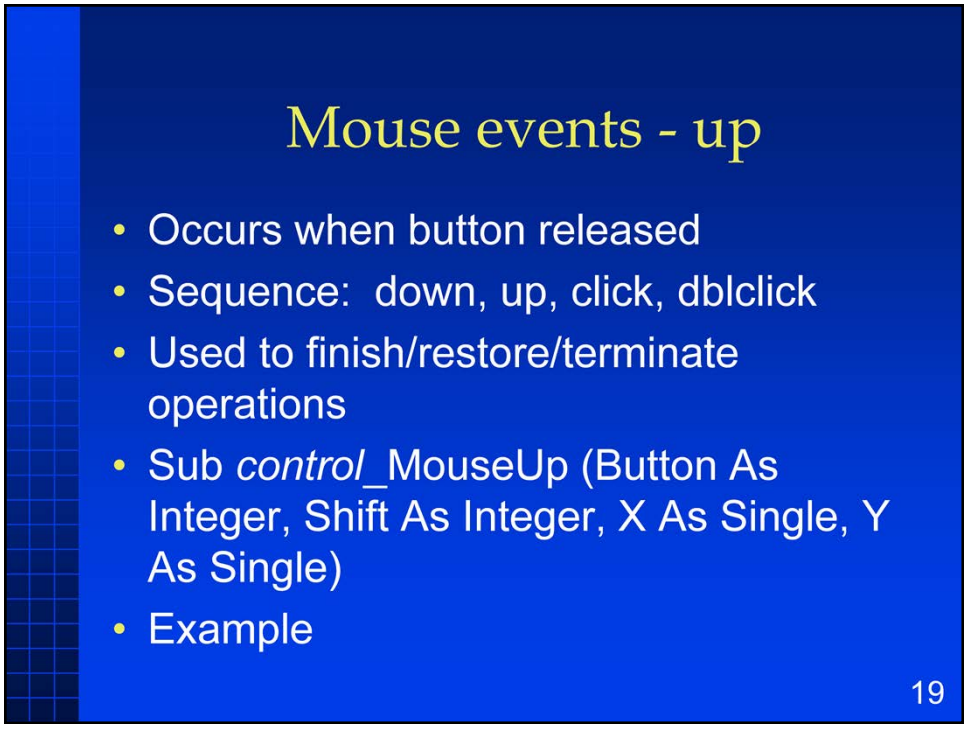


Mouse clicks very generic, special-purpose apps require more details interaction

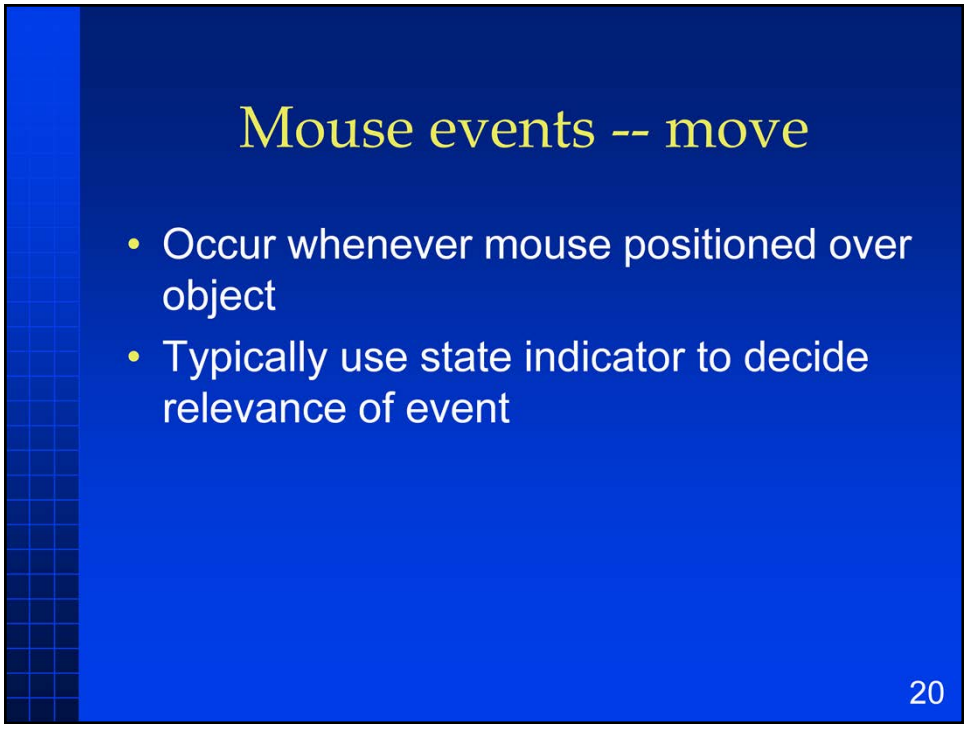
Most objects get mouse events, we'll look only at PictBox
Capture: i.e. only one object at a time gets events
button says which one: bit-mask for left, middle, right
shift indicate state of ctrl-alt-shift keys -- bitmask
constants defined in constant.txt
x, y -- current position of mouse
Example: show mouse location
form, picturebox, labels x: y:

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

sleleton in lect4\mouse



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



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

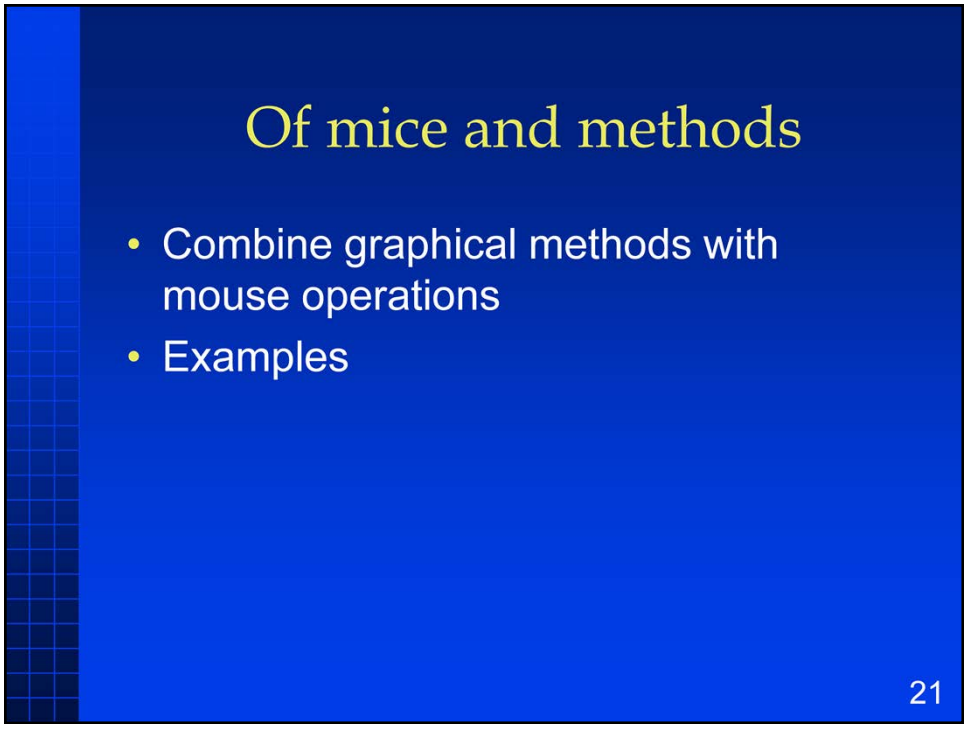
typical sequence: down sets switch, move interrogates and acts accordingly, up resets

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
```



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:

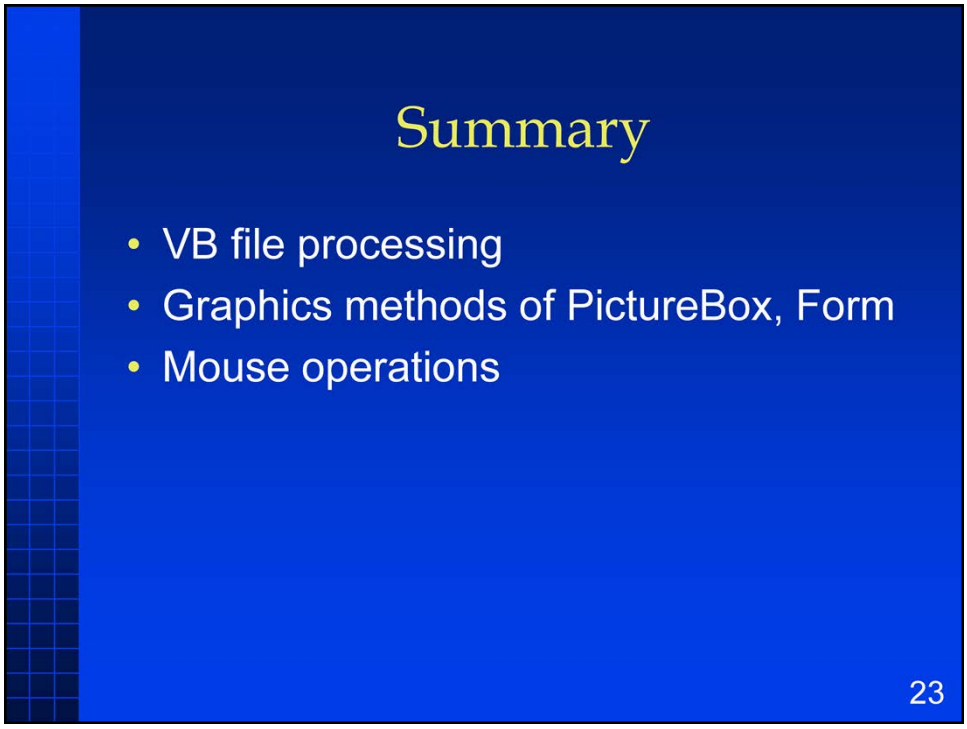
```
down:  tracking = True
       startx = x
       starty = y
       first = True
move: If tracking Then
        If (first) Then
          form1.Line (startx, starty)-(x, y), QBColor(4)
       Else
         form1.Line -(x, y)
       End If
       first = False
```

End If

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



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