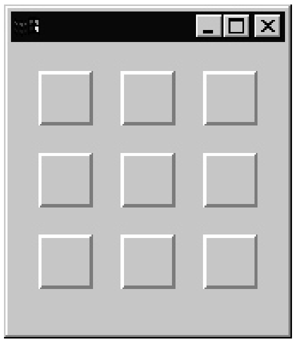
CSc 451, Spring 2003
Assignment 9
Due: Thursday, April 24 at 18:00

Problem 1. (16 points) `sequence.icn`

In this problem you are to write a program named `sequence` that is a guessing game based on the Palm Pilot game "Lock Down" by Digivello[1]. The idea of the game is simple: guess a sequence of 1-9 randomly chosen digits. The interface is a keypad of nine buttons:



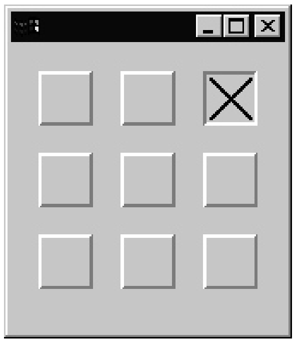
The buttons are blank but logically numbered like a typical telephone keypad, with button 1 in the upper left and button 9 in the lower right.

`sequence` takes one argument, which is a number from 1-9 indicating how many digits are to be guessed. For example, the command

```
    sequence 3
```

might generate a 3-digit sequence such as 417, 928, or 345. Each digit may only appear once in a sequence.

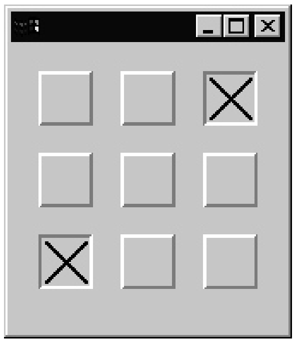
The user presses buttons, hopefully in a methodical way, to find the first digit. When the correct digit is pressed, the button toggles to display an "X". For example, suppose the sequence is 379. When the user presses the third button, they'll see this:



(Other buttons pressed before the third button will simply rebound.)

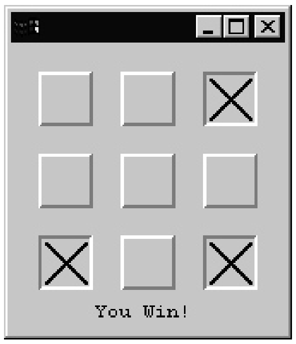
[1]Who may in turn have based it on a game by somebody else...

If the user then luckily presses the seventh button, they'll see this, reflecting his or her progress:



If instead of pressing the seventh button they'd pressed the second button, then both buttons would reset to their initial, unpressed state, and the user would start over again. (And hopefully the first thing the user would do is press the third button because it's already been determined to be the first button in the sequence.)

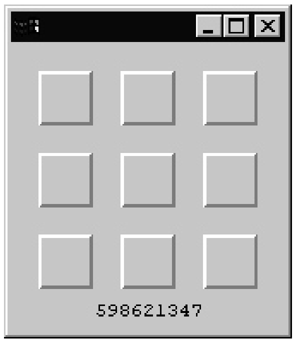
If the user continues his or her lucky streak and presses the ninth button, they win, and see this:



```
You Win!
```

If `sequence` is run with no argument, a default of three digits is used.

To facilitate debugging, if the first (or only) argument is `-d` the sequence is displayed. '`sequence -d 9`' might do this:



```
598621347
```

In general, the behavior is this: The command '`sequence N`' starts the program and randomly generates a sequence of N unique digits from 1 through 9. The user then presses buttons and as long as the sequence matches the chosen sequence, the buttons stay down. If there is deviation from the sequence then all buttons reset and the user must try again to enter the complete sequence.

Your solution should have buttons that are all the same size and evenly spaced. Hint: You may find it easier to roughly place the buttons with VIB and manually edit the generated initialization data rather than adjusting the button properties with VIB. (In case it isn't obvious, use toggling "`xbox`" buttons.)

The text "`You Win!`"and the digits displayed with `-d` need not be horizontally centered, although the reference version roughly does so.

If the user wins, typing a "`q`" then exits the program.

You may assume that the arguments are always valid.

To get run-to-run randomization, '`link random`' and then inside main, call `randomize()`.

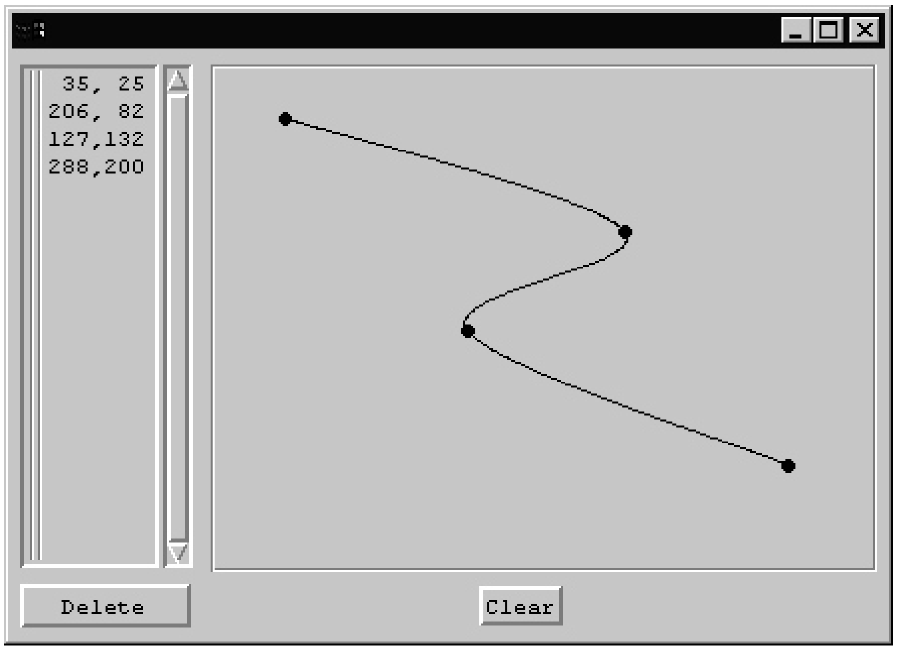
Here is some code that produces a shuffled string of the digits from 1 to 9:

```
digits := &digits[2:0]
every 1 to ?20 do
    ?digits :=: ?digits
```

Hint: Watch out for infinite recursion when implementing the error-resets-all-buttons behavior. Note that with the Windows implementation infinite recursion sometimes produces a program fault dialog and no traceback from the interpreter.

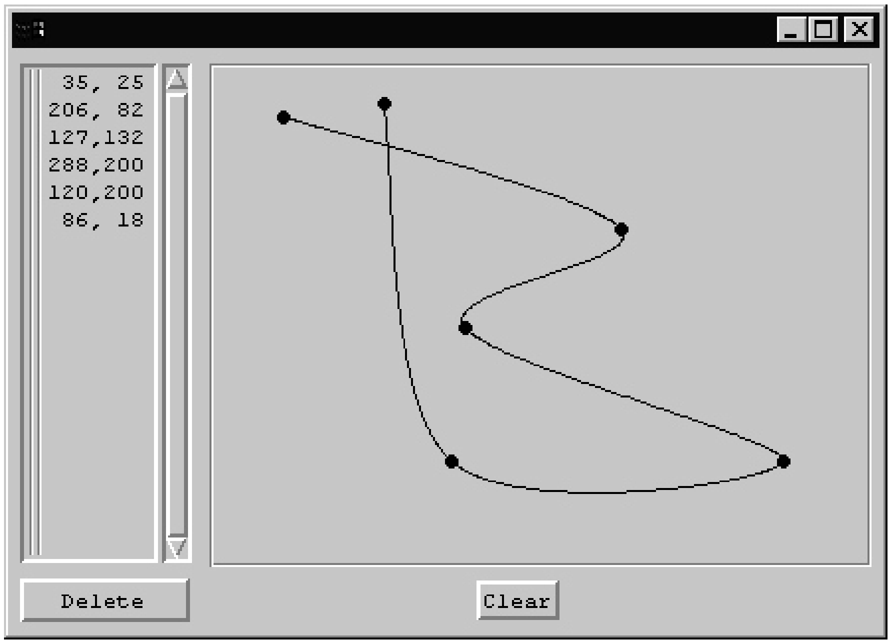
Problem 2. (25 points) `ced.icn`

In this problem you are implement a simple curve editor. `ced` is always run with no arguments. Here is a mid-run snapshot:



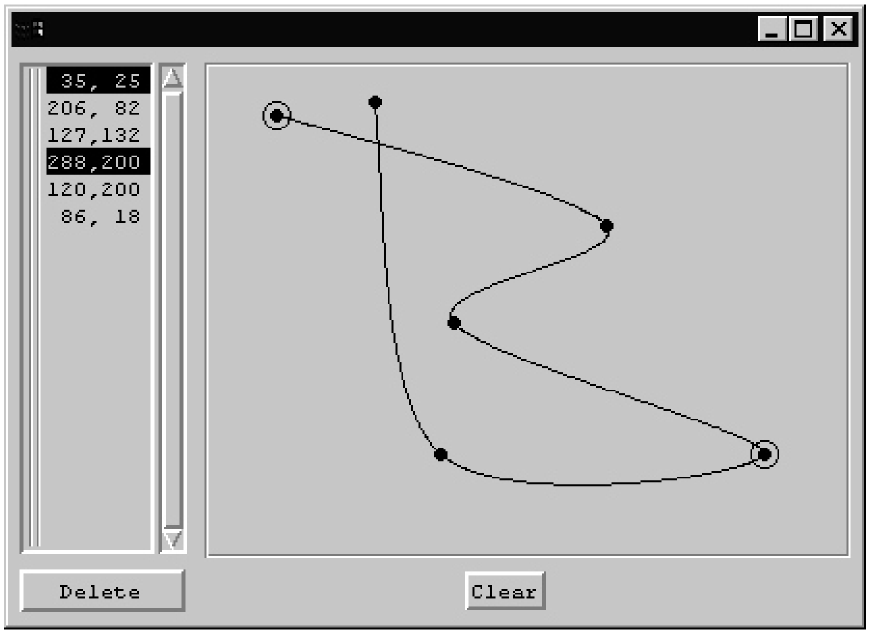
```
 35, 25
206, 82
127,132
288,200
```

On the left is a text list with the coordinates of the current curve's four points. On the right is a region displaying the curve. Left clicks in the curve region add points at the position of the clicks. Here is a snapshot after adding two more points:

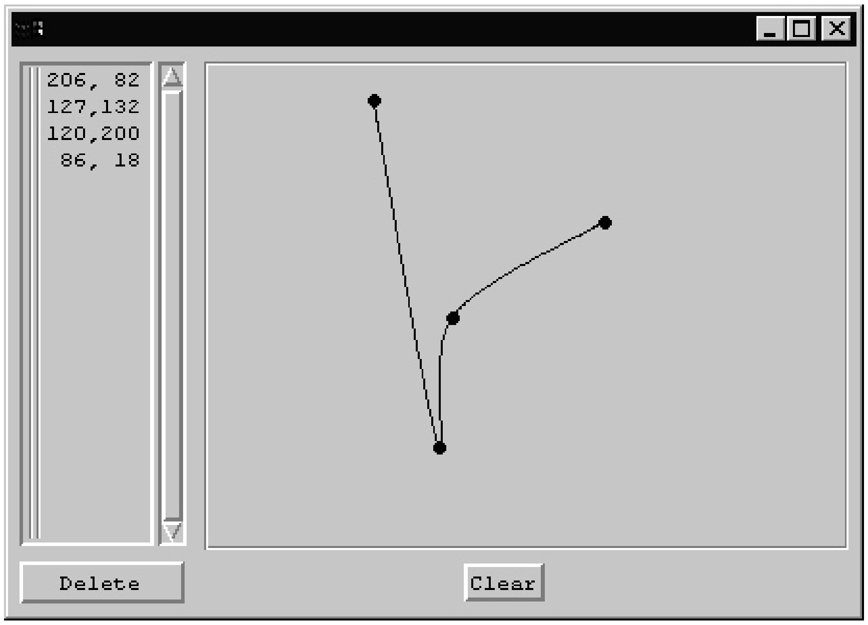


```
 35, 25
206, 82
127,132
288,200
120,200
 86, 18
```

The user may delete points by selecting them in the list and then pressing the `Delete` button. In addition to the list's inversion of selected points, they are circled in the drawing region. (If a point is deselected, the circle disappears.) Here the user has selected two points to delete:

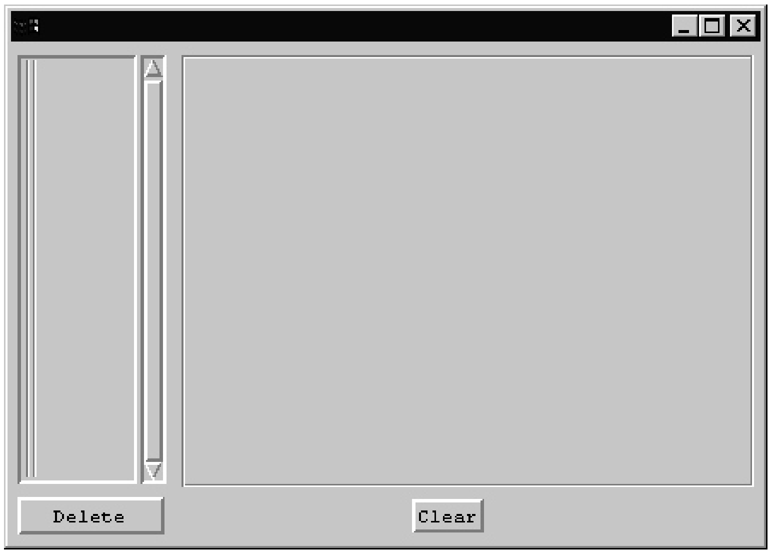


Here is the result after pressing the `Delete` button:



New points are always added to the end of the list. Adding a point clears any current selection(s) from the list and repositions the list so that the first point is in view. Be sure that clicking outside the region doesn't cause a point to be added.

Pressing `Clear` deletes all points, producing the same result as a freshly started `ced`:



The interface layout for your solution does not need to match the reference version. All that is required is that it have a text list, a drawing area of reasonable size, and `Clear` and `Delete` buttons.

The coordinates should be formatted as shown, right justifying each value in a field of width three.

Represent each point with a filled circle of radius 3. Use a circle with a radius of 7 for selected points. Use `DrawCurve![x1, y1, x2, y2, ...]` to actually draw the curve. In some cases the curve may swing outside the drawing area. If so, be sure it is appropriately clipped.

Despite the name "editor" there is no provision to save one's work.

Miscellaneous

Aside from code appearing in the class handouts, in the texts, or in messages from the instructor you may not seek, study, refer to, use, incorporate, etc., any existing solutions or portions thereof for any problem.

Reference Versions

Both UNIX and Windows reference versions of `sequence` and `ced` are in `/home/cs451/a9`. Notify the instructor if the behavior of a reference version seems to contradict or extend a problem specification.

Deliverables

Use `turnin` with the tag `451_9` to submit your solutions for grading. The deliverables for this assignment are `sequence.icn` and `ced.icn`.