TrainingNews
Volume VIII, Number 11 July 1987

THE CBT CRAFTSMAN

On Creativity In Programming

Scratch an excellent course and underneath you will find a beautiful design, not to mention an elegant body of program code.

Jesse Heines

My editors and I had considerable trouble coming up with a title for my May 1987 column on Paul Russell. The title that we used, "On CBT and Creativity," came out of a number of suggestions, including my own. When I read the title and article in their final form, however, I felt that we may have used the term "creativity" too restrictively. That column focused on creative design. This month I'd like to focus on creative programming.

I have often stated that you can't make a good picture from a poor negative, i.e., you can't implement a good course from a poor design. Yet even with a good negative, the resultant picture can be disappointing. Interestingly enough, the photographer who originally snapped the shutter may not be the best technician to make the print in the darkroom. To use another analogy, this time from the world of music, a work's composer may not be its best performer or conductor. Consider this excerpt from the program notes about Tchaikovsky's Fifth Symphony from a recent concert by the Atlanta Symphony:

"At the first two performances in St. Petersburg in November 1888, audiences applauded but the critics found the work disappointing, calling it an unworthy successor to the Fourth Symphony. His brother, Modest Tchaikovsky, felt that the reason was Tchaikovsky's poor showing as a conductor. Early in his career, Tchaikovsky had been terrified of the podium, having to hold on to his beard while conducting, he said, to keep his head from flying off. Later, after he heard the work well played and well received in Hamburg, he was able to write to his nephew Vladimir Davidov, 'The Fifth Symphony was magnificently played, and I like it far better now, having had a bad opinion of it for some time.'

The programming of a CBT course is to its design as the performance of a musical score is to its composition. Code can be hacked together or it can be implemented with elegance. The former seldom achieves the full range of interactions planned by the designer and is always difficult to update. The latter is a close reflection of the designer's intentions and allows small changes to be made without extensive reprogramming.

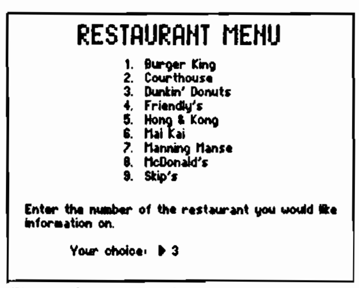

Figure 1. A menu of restaurants

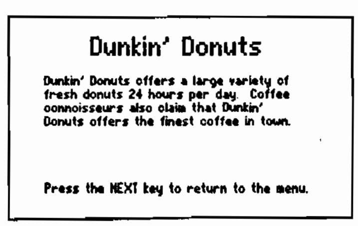

Figure 2. Information displayed for menu item 3.

Let us consider a common interaction that occurs in many CBT courses: a student selects an item from the menu, the computer displays information pertaining to that item, and the student is then asked to press the RETURN key to return to the menu. Figure 1 is an example of a typical menu for this type of interaction. This example is a list of restaurants, and the student is asked to type a number to receive information about the corresponding restaurant. Figure 2 shows the information displayed for the third restaurant on the menu, "Dunkin' Donuts."

There are basically two ways to program this interaction. First, we can "hard-code" the menu options and their corresponding information displays. In TenCORE*, this approach can be programmed as shown on the left side of page 8. The code shown here might seem a little foreign if you are not all familiar with computer programming, but the discussion below will enable you to follow the logic of it, whether or not you understand the commands.

The "hard-coding" in this approach occurs in two places: first at lines 15-26 where the restaurant names are displayed and then at lines 51-78 where the corresponding information is displayed. Now consider what has to be done in this hard-coded approach if we wish to add a restaurant to the list. First, we must renumber the menu options to keep them consecutive both in lines 18-26 where the list is displayed and in lines 51-78 where response processing takes place. Second, we must insert an entire response processing selection of at least 7 new lines to process the new option (see lines 61-67). This is a considerable amount of overhead considering that lines 52-56, 62-66, and 70-74 are all almost identical—the only difference being the string that is packed into variable *censtrng* at lines 54, 64, and 72.

A better, more creative way to program this interaction is shown next to the other example.

This approach is considerably different. Instead of "hard-coding" the restaurant names and their corresponding information, we set up a small database. First we define some local variables in lines 4-10. The database itself consists of two sub-

---

Jesse M. Heines is an assistant professor of computer science at the University of Lowell in Lowell, Massachusetts. He is the author of Screen Design Strategies for Computer-Assisted Instruction *as well as numerous articles on courseware development. Dr. Heines provides training and consultation on computer-based training, develops custom training programs on contract, and writes "The CBT Craftsman" every other month.*

scripted variables, or arrays, *optitles* and *opdescs* (I would indeed prefer to have used longer, more descriptive variable names, but TenCORE* restricts variable names to eight or fewer characters). These arrays store the option titles (restaurant names) and option descriptions (their corresponding information) in 20-byte and 256-byte strings, respectively.

Program lines 15-39 store the restaurant data in the database variables. The loop that begins at line 17 and ends at line 39 is executed once for each restaurant in the database. The two *packc* statements at lines 18 and 28 have a somewhat opaque syntax, but they basically store the information for one restaurant in the correct array elements on each pass through the loop. Note that each piece of data is separated from the next simply by a semicolon. This will be important later in the discussion.

In summary, one should see that the extra effort needed to write a creative, algorithmic program will pay dividends as the program grows both in size and complexity.

Once the data is stored, it can be displayed algorithmically rather than in a hard-coded manner. For example, look at how the menu options are displayed using a loop at lines 50-53. Line 51 says "display the value of the loop index, *k*, as a number, follow this with a period and two spaces, and then display the indexed option title as an alphanumeric string." There is no reference to a specific, hard-coded string here. This loop will work no matter what the option strings contain. And the loop itself will be executed as many times as necessary, controlled by the value of the constant *noptions* on line 50.

Likewise, response handling is performed algorithmically rather than in a hard-coded manner. First, the student's response is stored as a number in variable *optionum* at line 72. This number is then used in the Boolean expression at line 76 to determine whether the student's response is in the range of 1 to the number of options, inclusive. If it is, the code at lines 76-82 handles all valid options. Line 79 packs the correct option title into variable *censtrng* for subsequent centering at the top of the screen. And line 82 displays the correct option description stored in array *opdescs*.

Now consider what has to be done in this algorithmic approach if we wish to add a restaurant to the list. First, we must increase the constant at line 5 that stores the number of options. Second, we add the new restaurant name to the list at lines 19-27, separating the new name from the others by a semicolon. Note that no renumbering is necessary, because the options are numbered "on the fly" by the loop at lines 50-53. Finally, we add the restaurant information to the list at lines 29-38 in a position that corresponds to where we add the restaurant name. That is, if we made the new restaurant the third name in the list, its corresponding information would have to be placed third in the information list. Again, note that we only need to separate the new information from the others by a semicolon and that no renumbering is necessary. (The numbers that appear after the double dollar signs on lines 29, 33, and 35 are comments for clarity.) This update procedure is clearly much easier than that required for the hard-coded approach. In addition, only the new menu data is being added; no overhead is required. The loop at lines 50-53 does not need modification, and neither does the response processing code at lines 76-82. As the menu grows, therefore, this approach will clearly result in much less code than a hard-coded approach. In addition, this same algorithm can be used for other menus simply by changing the data in the arrays.

In summary, one should see that the extra effort needed to write a creative, algorithmic program will pay dividends as the program grows both in size and complexity. A single algorithm can often be written to handle a large variety of similar situations, thus reducing overall size. In addition, creative, generalized algorithms actually reduce program complexity by centralizing functionality. Why have to debug a separate subroutine for each menu when you can have one generalized menu handler for the entire course? The generalized menu handler will undoubtedly be better tested, offer more functionality, and provide superior human factors to any series of repetitive menu subroutines. Programmers, like musicians, should use the full range of their creative skills to implement course authors' compositions. □

TenCORE is a registered trademark of Computer Teaching Corporation. In this language, each command line begins with a command word, which tells the system what to do. The command word is followed by a command argument, which tells the system the parameters needed for that command. The lines are numbered only for reference. TenCORE does not require line numbering.

```
1 *
2 * This unit displays a list of restaurants, allows the user
3 * to select one, and displays data on that restaurant.
4 *
5 screen cga          $$ set screen to medium resolution color
6 spacing variable    $$ use variable pitch when displaying text
7 *
8 * Show the menu title
9 *
10 zero     censtrng                      $$ clear global string variable censtrng
11 pack     censtrng,,RESTAURANT MENU     $$ pack "RESTAURANT MENU" into censtrng
12 do       center(160,180)               $$ center string in censtrng around
13                                        $$   column 160 on line 180 (pixels)
14 *
15 * Display the menu options
16 *
17 at       120,165
18 write    1.  Burger King
19          2.  Courthouse
20          3.  Dunkin' Donuts
21          4.  Friendly's
22          5.  Hong & Kong
23          6.  Mai Kai
24          7.  Manning Manse
25          8.  McDonald's
26          9.  Skip's
27 *
28 * Prompt for user input
29 *
30 at       50,zy-25    $$ position cursor to x = 50 pixels and
31                      $$                    y = 25 pixels below the current y
32 write    Enter the number of the restaurant you would like
33          information on.
34
35                    Your choice:  $$$
36 *
37 * Get user input
38 *
39 blanks                     $$ allow a NEXT key press by itself
40 arrow                      $$ pause for student input
41 holdok                     $$ don't print the standard system "ok" message
42 erase    zx,zy+10; 319,0   $$ erase the error message functional area
43 *
44 * Provide feedback
45 *
46 answer                                $$ respond to a NEXT key press by
47 .        write   Enter a number before $$   itself
48 .                pressing NEXT.
49 .        judge   ignore                $$ ignore student entry and return to
50                                        $$   the arrow (line 40) for another
51 answer   1                             $$ respond to an answer of "1"
52 .        erase                         $$ erase entire screen
53 .        zero    censtrng
54 .        pack    censtrng,,Burger King
55 .        do      center(160,180)
56 .        at      65,160              $$ display information message
57 .        write   Burger King offers flame-broiled "Whopper"
58 .                hamburgers and a nice salad bar in a fast-
59 .                food environment.  You may eat in or take
60 .                out.
61 answer   2                             $$ respond to an answer of "2"
62 .        erase
63 .        zero    censtrng
64 .        pack    censtrng,,Courthouse
65 .        do      center(160,180)
66 .        at      65,160
67 .        write   The Courthouse offers pub-style lunches
68 .                and dinners in a club environment.
69 answer   3                             $$ respond to an answer of "3"
70 .        erase
71 .        zero    censtrng
72 .        pack    censtrng,,Dunkin' Donuts
73 .        do      center(160,180)
74 .        at      65,160
75 .        write   Dunkin' Donuts offers a large variety of
76 .                fresh donuts 24 hours per day.  Coffee
77 .                connoisseurs also claim that Dunkin'
78 .                Donuts offers the finest coffee in town.
```

... and so on for the other six restaurants

```
79 no                                     $$ respond to an "incorrect"
80 .        write   Please enter a number  $$   entry
81 .                between 1 and 9.
82 .        judge   ignore
83 endarrow                               $$ end response processing
84 *
85 * Get NEXT key to repeat menu
86 *
87 zero     censtrng
88 pack     censtrng,,Press the NEXT key to return to the menu.
89 do       center(160,30)
90 next     «zunit»          $$ execute the current unit again after the
91                           $$   student presses NEXT
```

```
1 *
2 * This unit implements the restaurant menu as a database.
3 *
4 define  local            $$ begin local variable definitions
5 noptions = 9             $$ number of menu options (a constant)
6 k,2                      $$ loop index (a 2-byte integer)
7 optionum,2               $$ option number chosen by user (a 2-byte integer)
8 optitles(noptions),20    $$ option titles (an array of 20-byte strings)
9 opdescs(noptions),256    $$ option descriptions (an array of 256-byte strings)
10 define  end             $$ end local variable definitions
11 *
12 screen  cga             $$ set screen to medium resolution color
13 spacing variable        $$ use variable pitch when displaying text
14 *
15 * Set up the database
16 *
17 loop     k ∈ 1,noptions              $$ loop for the number of options
18 .        packc   k; optitles(k);; ; ; $$ store option titles in array
19 .                Burger King;
20 .                The Courthouse;
21 .                Dunkin' Donuts;
22 .                Friendly's;
23 .                Hong & Kong;
24 .                Mai Kai;
25 .                Manning Manse;
26 .                McDonald's;
27 .                Skip's;
28 .        packc   k; opdescs(k);; ; ;  $$ store option descriptions in array
29 .                Burger King offers flame-broiled "Whopper"       $$ 1
30 .                hamburgers and a nice salad bar in a fast-
31 .                food environment.  You may eat in or take
32 .                out.;
33 .                The Courthouse offers pub-style lunches          $$ 2
34 .                and dinners in a club environment.;
35 .                Dunkin' Donuts offers a large variety of         $$ 3
36 .                fresh donuts 24 hours per day.  Coffee
37 .                connoisseurs also claim that Dunkin'
38 .                Donuts offers the finest coffee in town.;
```

... and so on for the other six restaurants

```
39 endloop
40 *
41 * Show the menu title
42 *
43 zero     censtrng
44 pack     censtrng,,RESTAURANT MENU
45 do       center(160,180)
46 *
47 * Display the menu options
48 *
49 at       120,165
50 loop     k ∈ 1, noptions
51 .        write   «s,k».  «a,optitles(k)»
52 .        at      120,zy-10
53 endloop
54 *
55 * Prompt for user input
56 *
57 at       50,zy-15
58 write    Enter the number of the restaurant you would like
59          information on.
60
61                    Your choice:  $$$
62 *
63 * Get user input
64 *
65 blanks                       $$ allow a NEXT key press by itself
66 arrow                        $$ pause for student input
67 holdok                       $$ don't print the standard system "ok" message
68 erase    zx,zy+10; 319,0     $$ erase the error message functional area
69 *
70 * Provide feedback
71 *
72 store    optionum                               $$ store entry in a variable
73 .        write   Enter a number and             $$ respond to a "null" entry
74 .                then press NEXT.
75 .        judge   ignore
76 ok       optionum ≥ 1 $and$ optionum ≤ noptions  $$ respond to all valid
77 .        erase                                  $$   option number entries
78 .        zero    censtrng
79 .        pack    censtrng,,«a,optitles(optionum)»
80 .        do      center(160,180)
81 .        at      65,160
82 .        showa   opdescs(optionum)
83 no                                              $$ respond to an "incorrect"
84 .        write   Please enter a number          $$   entry
85 .                between 1 and 9.
86 .        judge   ignore
87 endarrow                                        $$ end response processing
88 *
89 * Get NEXT key to repeat menu
90 *
91 zero     censtrng
92 pack     censtrng,,Press the NEXT key to return to the menu.
93 do       center(160,30)
94 next     «zunit»          $$ execute the current unit again after the
95                           $$   student presses NEXT
```

Here are two ways to program a common CBT interaction in which a student selects an item from a menu, the computer displays relevant information, and the student is then asked to press the RETURN key to return to the menu. On the left, the menu options and their corresponding information displays are hard-coded. On the right, they are stored in a data base and displayed algorithmically.