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 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 snapping 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 his lack of a strong vision 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 had his 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.

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 shows how one might see a little code 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 15-20 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 rewrite the menu options to keep them consistent 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 at least 7 new lines to process the next 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—

### RESTAURANT MENU
- Burger King
- Chinese House
- Dunkin' Donuts
- Friendly's
- Hong Kong
- Pizza King
- Peking Manor
- McDonald's
- Sizzler

Enter the number of the restaurant you would like information on:

Your choice: 3

![Figure 1. A menu of restaurants](image1.png)

### Dunkin' Donuts
Dunkin' Donuts offers a wide variety of fresh donuts 24 hours a day. Coffee comes with any order. Dunkin' Donuts offers the finest coffee in town.

Press the NEXT key to return to the menu.

![Figure 2. Information displayed for menu item 3.](image2.png)

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-
scripted variables, or arrays, optles and optdescs (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 255-byte strings, respectively. Program lines 15-39 store the restaurant 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 added the new restaurant the third name in the list, its corresponding information would have to be placed third in the information.

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.

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 print statements at lines 18 and 28 have a somewhat opaque syntax, but they basically do the information for each 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.

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 iterations 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 optionnum at line 72. This number is then used in the Boolean expression at line 73 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 correct for subsequent printing at the top of the screen. And line 82 displays the correct option description stored in array optdescs.

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 15-27, separating the new name from the 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 less complex 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 generalization algorithms usually 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.
**There 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 database and displayed algorithmically.**