Using Mathcad Debugging Functions as a Teaching Tool

Kenneth P. Brannan[1] and John A. Murden[2]

Abstract – Survey results from students taking a computer applications class using Mathcad as a programming language indicate that loops and subscripted variables are difficult topics. In the earlier versions of Mathcad used in the class, it had not been possible to demonstrate with the software package during the early stages of instruction on loops how variables change during the line-by-line execution of a loop. With Mathcad 13, additional debugging tools were included that could be used to monitor variables during the execution of a loop. Student responses in the survey showed that the new debugging tools helped to improve understanding of loops and subscripted variables. While the new features were perceived by over two-thirds of the students to be helpful in the teaching and learning of loops and subscripted variables, other teaching methods such as doing weekly assignments, preparing for weekly tests, "playing computer," and working with on-line electronic workbook files were found to be more valuable aids for learning. Students identified the most difficult topic to be nested loops, which is currently being taught as the last topic in the programming sequence. However, three of the relatively difficult topics were taught in the first two weeks, which may contribute to the difficulty experienced by students when they first encounter loops. A review of previously used languages indicated that those languages had features that facilitated the teaching and learning of loops in the early stages of teaching loops.

Keywords: programming, Mathcad, debugging tools, loops

INTRODUCTION

Based on experience gained from a combined forty-plus years of teaching programming in several languages, it appears that one of the critical points at which students encounter difficulty is when loops are introduced. The difficulty escalates when subscripted variables (arrays) are used in the loops. To teach these features in Mathcad, the authors have employed several techniques, including explanations of example programs, flowcharts, pseudocode, hands-on electronic workbook, and frequent tests and assignments. No single technique has eliminated the barriers to learning, particularly for weaker students.

When Mathcad was selected a decade ago as the programming language, a technique was lost that had been available in languages used previously – the ability to demonstrate during the early stages of instruction what happens to loop and non-subscripted variables during line-by-line execution of a loop. With earlier versions of Mathcad, only after arrays were introduced could students see actual output associated with a function, and by then many weaker students were struggling to catch up. With Mathcad 13, several new debugging functions have been added that allow the user to see annotated line-by-line output as the loop is executed. In addition, the debugger allows execution to be suspended to examine output at a given point in the loop, and the user may step through the loop interactively, observing the output during each pass through the loop. The purpose of this paper is to describe how this new feature has been incorporated into an introductory programming class and assess whether the feature has improved the teaching and learning of loops and subscripted variables.

---

[1] The Citadel, Civil and Environmental Engineering Dept, 171 Moultrie St, Charleston, SC 29409-0225, ken.brannan@citadel.edu

[2] The Citadel, Civil and Environmental Engineering Dept, 171 Moultrie St, Charleston, SC 29409-0225, john.murden@citadel.edu

BACKGROUND

When faculty recall programming from their student days, when FORTRAN was the indisputable king of the engineering classroom, what often comes to mind are punch cards, card readers, and long lines of students waiting on the output from batch processed programs. Most engineering educators would not readily swap today's powerful programming tools for a return to those days. However, it is occasionally instructive to take a backward glance to make sure that valuable components of yesterday's teaching toolbox are not lost in the enthusiastic scramble for more powerful tools of the future.

For a number of years, the authors have noticed that their students appear to have difficulty with programming at the point where loops are introduced. Although loops have always presented a challenge (regardless of the language), the "loop challenge" seems to have intensified in recent years. One possible explanation for this observation is that many students in the late 1980s and early 1990s received more programming training prior to beginning college than current students have had. Yet, based on the authors' experience, prior programming experience has not always guaranteed that a student would have a greater advantage in a college programming class. Another possibility was that something must have changed in the way loops are currently being presented compared to the way they were presented in the past. Dusting off a few old programming textbooks and course notes revealed that indeed some seemingly small but interesting changes had taken place over the years.

Department Programming — Past and Present

Over the past two decades, several changes have been made in the department's programming language. During the late 1980s both FORTRAN and BASIC were taught on a VAX system. In 1990, a transition was made to personal computers, and QuickBASIC was selected as the language. QuickBASIC served the department well for the next four years. In the fall of 1994, the department switched to C++, a move that provided multi-platform capability and supported Object-Oriented Programming. C++ was replaced with Mathcad in the fall of 1996. Mathcad not only provided an environment in which programming could be taught, but it included a wide variety of mathematical and graphical tools, symbolic processing, and text editing.

From a teaching perspective, the change to QuickBASIC was relatively seamless. The language was similar enough to FORTRAN that essentially the same approach to teaching programming could be used. When the change to C++ was being considered, there were some concerns that the transition may not be smooth. Some of commands were strikingly different (e.g., ***cout*** instead of ***PRINT*** or ***WRITE***), the structure had a different appearance (e.g., use of braces), the greater flexibility of the language was accompanied by more complexity, and most of the faculty members were not familiar with C++. Nevertheless, the fundamental features of the language needed in an introductory programming language were similar to previously used languages, which permitted the course to be taught in much the same way as before. As will be pointed out later, there were some features associated with the more modular approach to the language that caused a slightly different approach, but adjustments were quickly made and the reasons were eventually forgotten. Students appeared to accept C++ and performed much the same as when programming with the previous languages.

Mathcad offered so many valuable tools that the initial problem from a teaching standpoint was how to fit the desired programming elements into the course syllabus along with the other features. Fundamental programming elements were comparable to C++, but since all programming had to be accomplished within Mathcad functions, it was not possible for students to monitor values of variables as they changed within a loop without first learning how to use subscripted variables. To address this, students were taught in initial classes on looping how to manually keep up with the variables whose values changed in loops. Later, when subscripted variables were introduced, a matrix could be generated in a function that returned a printout of values generated in a loop. As time passed, however, the authors noted that students continued to have difficulty with loops, and they annually made adjustments in how loops were taught in an effort to improve the course.

With the release of version 13, Mathcad added enhanced debugging features that allow immediate monitoring of loops. It is not necessary to teach subscripted variables to employ the debugging features. These new debugging features were incorporated into the course during the fall semester of 2007. In order to illustrate why these Mathcad debugging features were felt to offer potential for improving the teaching and learning of loops, it is helpful to

consider how loops could be introduced in the languages used prior to Mathcad. This is discussed in the sections below, including examples of programs in FORTRAN, C++, and Mathcad that can be used to introduce looping.

The FORTRAN Unconditional *GO TO* Statement

One of the most fundamental programming commands in the FORTRAN language that may be used for program flow control is the unconditional ***GO TO*** (or ***GOTO***) statement. This statement may be readily used to introduce loops to novice students [1]. Figure 1 illustrates a simple FORTRAN program for exploring the impact of slope on the flow rate of water in a pipe, using the Manning equation. The Manning's roughness coefficient is a constant 0.013 for the 21-inch pipe and the slope varies from 0.001 to 0.004. GNU FORTRAN G77 was used for the program.

Figure 1. FORTRAN Program Illustrating GO TO Loop

```
C234567890
      REAL N,Q,SLOPE,AREA,HR

      WRITE(*,*)' Slope  Flow (cfs)'
      WRITE(*,*)

      SLOPE = 0.001
      N = 0.013
      AREA = (3.141593*(21./12.)**2)/4.
      HR = (21./12.)/4.

  100 READ *, SLOPE
      Q = (1.49/N)*(HR**(2./3.))*SLOPE**(1./2.)*AREA
      WRITE(*,10) SLOPE,Q
      GO TO 100

   10 FORMAT(F7.3,F9.1)
      STOP
      END
```

```
 Slope  Flow (cfs)

 0.001      5.0
 0.002      7.1
 0.003      8.7
 0.004     10.0
```

From a teaching and learning perspective, students of the 80s and before, for example, could easily follow the program execution through the loop, which was controlled by the "GO TO 100" statement, which transferred control to statement number 100. When the program was executed on a personal computer, the ***READ*** statement caused execution to pause for the user to type in a number for the slope. As soon as the value for the slope was entered, values for the slope and flow were printed. The four lines of output shown in Figure 1 do not include the values for the slope typed in by the user.

Prior to the availability of personal computers, it was still possible to use essentially the same illustration to introduce loops. Instead of typing in values for the slope at a computer keyboard, a number of punch cards with the input values for slope were read during program execution. When the cards ran out, an execution error occurred and execution was terminated. While this procedure was not considered good programming practice, it nevertheless was a simple way to introduce loops without requiring the student to understand additional control structures. Although the execution error was displayed on the output, the student could clearly see how values for slope and flow changed with each pass through the loop. Further, minimal training was needed for a student to begin to work with loops using this feature.

As more control structures were introduced, they could either be combined with the ***GO TO*** statement or used independently to build upon a student's initial experience with loops. For example, in Figure 2 a Block ***IF*** structure is used to increment values for the slope and terminate the loop when all values of the slope have been used in the computations.

While the ***GO TO*** statement allowed loops to be introduced in an interactive, simple to follow way, with time the statement was frowned upon more and more as poor practice as modular programming structures became the standard. This is clearly shown in the following quotation from a FORTRAN textbook published in 1991:

> Unrestrained use of the ***GO TO*** statement is the primary cause of FORTRAN code that resembles spaghetti: numerous branchings up and down in a program that result in code that is unreadable and unalterable. Be extremely conservative in using this statement. [2]

Figure 2. FORTRAN Program Illustrating GO TO Loop with Block IF Structure

```
C234567890
      REAL N,Q,SLOPE,AREA,HR

      WRITE(*,*)' Slope  Flow (cfs)'
      WRITE(*,*)

      SLOPE = 0.001
      N = 0.013
      AREA = (3.141593*(21./12.)**2)/4.
      HR = (21./12.)/4.

  100 Q =
(1.49/N)*(HR**(2./3.))*SLOPE**(1./2.)*AREA
      IF (SLOPE .LE. 0.004) THEN
           WRITE(*,10) SLOPE,Q
           SLOPE = SLOPE + 0.001
           GO TO 100
      ENDIF

   10 FORMAT(F7.3,F9.1)
      STOP
      END
```

```
 Slope  Flow (cfs)

 0.001      5.0
 0.002      7.1
 0.003      8.7
 0.004     10.0
```

The FORTRAN *DO* Loop and C++ *for* Loop

A powerful FORTRAN feature for generating a loop is the ***DO***-loop structure. ***DO*** loops have been a part of the language since the earliest versions. The initial statement in the structure establishes a beginning value, ending value, and increment for the loop variable. Single and nested loops may be used to meet a variety of computational needs. Figure 3 illustrates how nested loops may be used to explore the impact of both slope and Manning's roughness coefficient on the flow. Integers were used for the loop variables S and MN, which were used in turn to assign values for the slope (SLOPE) and the Manning's roughness coefficient (N).

Regardless of whether or not the unconditional ***GO TO*** or other control statements are used to introduce the concept of loops, students may follow how variables change in ***DO*** loops by placing an output statement inside the loop. In Figure 3, a WRITE statement placed in the inner loop allows a student to see changes in the loop variables S and MN, the corresponding values of SLOPE and N, and how the value of the flow, Q, changes with SLOPE and N.

Figure 3. FORTRAN Program Illustrating Nested Do Loops

```
C234567890
      INTEGER MN,S
      REAL N,Q,SLOPE,AREA,HR

      WRITE(*,*)' S  Slope  MN  Manning''s n  Flow (cfs)'

      DO 200 S = 1,4,1
         SLOPE = REAL(S)/1000.
         WRITE(*,*)

         DO 100 MN = 11,13,1
            N = REAL(MN)/1000.
            AREA = (3.141593*(21./12.)**2)/4.
            HR = (21./12.)/4.
            Q = (1.49/N)*(HR**(2./3.))*SLOPE**(1./2.)*AREA
            WRITE(*,10) S,SLOPE,MN,N,Q
  100       CONTINUE

  200    CONTINUE

  10 FORMAT(I3,F7.3,I4,F10.3,3X,F9.1)
     STOP
     END
```

```
  S  Slope  MN  Manning's n  Flow (cfs)

  1  0.001  11     0.011         5.9
  1  0.001  12     0.012         5.4
  1  0.001  13     0.013         5.0

  2  0.002  11     0.011         8.4
  2  0.002  12     0.012         7.7
  2  0.002  13     0.013         7.1

  3  0.003  11     0.011        10.3
  3  0.003  12     0.012         9.4
  3  0.003  13     0.013         8.7

  4  0.004  11     0.011        11.9
  4  0.004  12     0.012        10.9
  4  0.004  13     0.013        10.0
```

Before the switch to Mathcad, the authors took for granted that this would be a standard feature available in any language. Similar output could be obtained with loops in QuickBASIC and C++. Figure 4 illustrates a ***for*** loop in a C++ program similar to the FORTRAN ***DO*** loop in Figure 3. GNU C++ was used for the program.

In addition, the package used in the 1990s for teaching C++ (Borland Turbo C++) had several debugging features that were ideal for teaching loops. The programming environment had a debugging window called a watch window. Any variable could be selected for monitoring in the watch window. A program user could single step through the program and watch the value of the selected variable change as each new step executed. Alternatively, a breakpoint could be set on a given line of the program, and the program could be executed up to the breakpoint and then single stepped through the next section.

Figure 4. C++ Program Illustration Nested For Loops

```
#include <iostream>
#include <math.h>
#include <iomanip>

using namespace std;

int main()
{
  int mn,s;
  float n,q,slope,area,hr;

  cout << "  s  Slope  mn  Manning's n  Flow (cfs)" << endl;

  for (s=1; s <= 4; s=s+1)
  {
      slope = (float)s/1000.;
      cout << endl;

      for(mn=11; mn <= 13; mn=mn+1)
      {
          n = (float)mn/1000.;
          area = (3.141593*pow((21./12.),2))/4.;
          hr = (21./12.)/4.;
          q = (1.49/n)*pow(hr,(2.0/3.0))*pow(slope,(1.0/2.0))*area;
          cout<< setw(3) << s << setw(7) << setprecision(3) << slope
              << setw(4) << mn << setw(10) << setprecision(3) << n;
          printf("%12.1f\n",q);
      }
  }
return(0);
}
```

```
  s  Slope  mn  Manning's n  Flow (cfs)

  1  0.001  11     0.011         5.9
  1  0.001  12     0.012         5.4
  1  0.001  13     0.013         5.0

  2  0.002  11     0.011         8.4
  2  0.002  12     0.012         7.7
  2  0.002  13     0.013         7.1

  3  0.003  11     0.011        10.3
  3  0.003  12     0.012         9.4
  3  0.003  13     0.013         8.7

  4  0.004  11     0.011        11.9
  4  0.004  12     0.012        10.9
  4  0.004  13     0.013        10.0
```

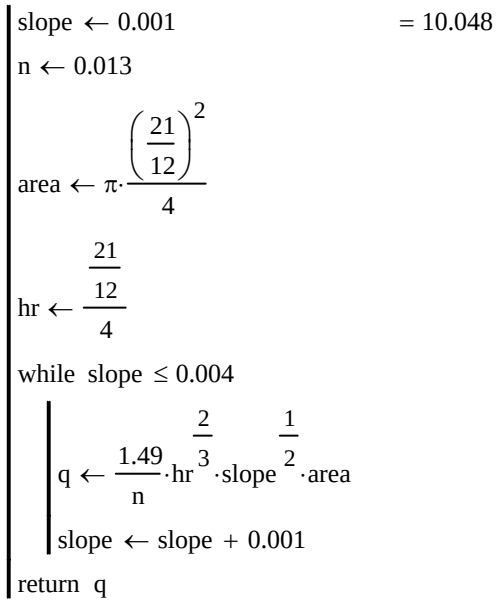
Although nothing analogous to the unconditional ***GO TO*** statement was used in the C++ version of the course, students did not appear to be at a disadvantage when loops were introduced in C++ because the utility of the watch

window more than offset the lack of a simple feature for introducing loops. The debugging feature added what was at the time a new dimension to teaching and learning of loops.

Mathcad Loops

Programs in Mathcad are written using a feature called the Mathcad programming operator. Figure 5 shows a Mathcad program that from a computational viewpoint works similarly to the FORTRAN program in Figure 2. The long vertical bar on the left of the program shows the extent of the program. The shorter vertical bar shows the extent of the ***while*** structure. In this program, the slope increments through four values from 0.001 to 0.004 and four corresponding values for the flow, q, are computed in the loop. However, in the first twelve versions of Mathcad, there were no output features that could be used to monitor a set of values for each pass through the loop without storing the values in a matrix. Therefore, until students learned matrices, only one value could be returned by the program, as illustrated by the number 10.048 (the value of the flow, q, following the final pass through the loop) to the right of the equals sign in Figure 5.

Figure 5. Mathcad Program – Only One Value from Loop Displays



Beginning with version 13 of Mathcad, two new debugging features were added that permit students to see values of variables produced in a loop. One feature is called the ***trace*** function. The ***trace*** function can display both text and values generated in a loop in a special window referred to as a trace window. When the ***trace*** function is incorporated into a program, execution proceeds without interruption, displaying all output from the ***trace*** function in the *trace* window. The second feature is the ***pause*** function, which works exactly like the ***trace*** function except that the user can step through the program, displaying one line of output from the ***pause*** function at one time. The ***pause*** function is analogous to using a breakpoint in the Turbo C++ programming environment. Figure 6 shows the same program as shown in Figure 5 with the ***trace*** function added. Output in the trace window is shown below the program.

It should be noted that programming classes do not normally use the Mathcad programming operator in exactly the same way as shown here. In class assignments, a program is usually part of a function definition. To see the output

Figure 6. Mathcad Program – Values from Loop Displayed in Trace Window.

slope ← 0.001 = 10.048

n ← 0.013

$$area \leftarrow \pi \cdot \frac{\left(\frac{21}{12}\right)^2}{4}$$

$$hr \leftarrow \frac{\frac{21}{12}}{4}$$

trace("slope flow (cfs)")

trace(" ")

while slope ≤ 0.004

$$q \leftarrow \frac{1.49}{n} \cdot hr^{\frac{2}{3}} \cdot slope^{\frac{1}{2}} \cdot area$$

trace("{0} {1}", slope, q)

slope ← slope + 0.001

return q

```
slope      flow (cfs)

0.001  5.02413074945107
0.002  7.1051938450094
0.003  8.70204972191835
0.004  10.0482614989021
```

from the function, the function is evaluated. The form shown in Figures 5 and 6 was selected for better comparison with the FORTRAN and C++ examples.

INCORPORATING THE MATHCAD DEBUGGING FEATURE INTO CLASSES

Typically, two sections of a required course, Computer Applications for Civil and Environmental Engineering, are taught during each fall semester. Each class is taught by one of the authors, but both authors attend all classes. The presence of two instructors is especially helpful for answering questions during hands-on exercises and instruction. Voluntary team-teaching in this manner has been practiced by the authors for over a decade. Although each class is taught by a different instructor, an effort is made to keep teaching similar with respect to the format of the classes and the information presented.

Teaching of loops involves material covered over seven lessons of instruction, with each lesson designed to last one week. The course is a two credit-hour course (one hour lecture, two hours laboratory) with three one-hour classes (Monday/Wednesday/Friday) per week. Topics are introduced on the first day, students receive assignments on the second day, and students submit assignments on the third day and take a weekly test. Lessons related to loops include:

- *Iteration (range variables) and graphing*. Mathcad *range variables* are similar to the parameters of a ***DO*** or ***for*** loop; they are variables that may change values, beginning at an initial value and incremented by a

specified amount until a final value is reached. They are very useful in serving as the independent variable of a graph. Although *range variables* are not loops, ***for*** loops are defined through the use of a range, so this is the initial step in teaching loops.

- *Programming with the **if** statement*. Additional experience with *range variables* is gained by working with ***if*** statements. Students do not encounter true loops in this lesson.

- *Programming with the **for** statement*. Students encounter loops for the first time using the ***for*** statement. Once students are familiar with ***for*** statements, the ***break*** and ***continue*** operators are discussed. In addition, the ***trace*** and ***pause*** functions are introduced to students through interactive exercises.

- *Programming with the **while** statement*. Work with loops is continued using a different structure – the ***while*** loop.

- *Vectors and vector operations*. Students are introduced to subscripted variables through the concept of the vector, which is a variable with a single subscript. The lesson also includes inserting external data files arranged as a vector into worksheets. Any loops required in this lesson are single loops.

- *Programming with vectors*. Students learn to search data in a vector using nested loops.

- *Programming with matrices*. Students learn to search data in a matrix (multiple rows and columns) using nested loops.

The ***trace*** and ***pause*** features were incorporated into all lessons that involved loops, beginning with the lesson on the ***for*** statement. To help determine if the degree of emphasis in a class (students spending more time evaluating computer programs including the ***trace*** and ***pause*** functions and incorporating these features into programs), the ***trace*** and ***pause*** features were purposely emphasized in Section 2 of the class more than they were emphasized in Section 1. Other aspects of the class were treated similarly. Students were not tested on the ***trace*** and ***pause*** features. (This is the way the course was taught using the C++ watch window and breakpoints. Since this had been very successful, the same methodology was followed for instruction in Mathcad.)

THE SURVEY AND STUDENT RESPONSES

In an effort to help assess the impact of the new debugging features on the students' learning of several topics related to the looping constructs, variables with a single subscript (vectors in Mathcad) and variables with multiple subscripts, or matrices, students in the two Fall 2006 sections of the Computer Applications in Civil and Environmental course completed the survey shown in the Appendix. The first two questions were intended to establish the students' perspective on the difficulty of eight selected topics associated primarily with loops and subscripted variables. The remaining four questions in the survey asked students to comment on the helpfulness or rank the benefit of various tools or techniques used in the course. The average responses are reported in Table 1 by course section and for the combined enrollment.

Not surprisingly and by a large margin, the students rated *nested-loops* as the most difficult of the eight topics in the list (position eight). After nested loops, there were three topics that were very close in average rank within one section and more spread in the other. The combined ranking placed the ***while**-loop* (with the associated conditionals) at position seven, the *break* and *continue* operators at position six and *matrices* (variables with multiple subscripts) at the fifth position on the difficulty list. The remaining four topics were ranked by the students as ***for**-loops* in position four, *vectors* (variables with a single subscript) *used in loops* at number three, ***if** statements in multi-line functions* in position 2, and the least difficult of the topics was *vectors as a stand-alone topic*. The ***if** statement topic* and *vectors as a stand-alone topic* are not directly associated with loops, but were included to help determine if

1	Please rank the following Mathcad features or topics in order of increasing difficulty to understand: (1 - least difficult to 8 most difficult)	Average Ranking		
		Sec 01 n = 17	Sec 02 n = 15	Combined n = 32
	for loops	4.1	4.3	4.2
	while statement	5.1	4.8	5.0
	Nested loops	7.2	6.5	6.8
	Vectors – use of single subscript variables	2.8	3.6	3.2
	Matrices – use of multiple subscript variables	4.3	4.9	4.6
	If statement in multi-line functions (conditionals)	4.0	3.1	3.6
	Use of subscripted variables in loops	3.3	4.1	3.7
	The break and continue operators in loops	4.9	4.9	4.9
2	How difficult were loops to understand during the first week that you encountered them. [Rate 1 to 5 most difficult]	3.5	3.4	3.4
3	Did the trace or pause function help your general understanding of loops?	Yes – 65%	Yes – 53%	
4	Did the trace or pause function help your understanding of subscripted variables in loops?	Yes – 35%	Yes – 67%	
	Those answering Yes to #3 **or** #4	Yes – 71%	Yes – 67%	Yes – 69%
5	Using a scale of 1 to 5, how useful were the following techniques or features in helping you to understand loops in general. (5 most helpful)	Sec 01	Sec 02	Combined
	Flowcharts	3.1	3.4	3.3
	Using pencil and paper to track the value of variables through a loop	3.6	3.9	3.8
	Using the trace window with the pause or trace function	2.8	3.3	3.1
	Doing a Mathcad assignment	4.4	4.4	4.4
	On-line lessons	4.3	3.7	4.0
	Preparing for weekly tests	3.9	4.2	4.0
6	Using a scale of 1 to 5, how useful were the following techniques or features in helping you to understand subscripted variables in loops. (5 most helpful)	Sec 01	Sec 02	Combined
	Flowcharts	3.0	3.0	3.0
	Using pencil and paper to track the value of variables through a loop	3.9	4.1	4.0
	Using the trace window with the pause or trace function	2.6	3.2	2.9
	Doing a Mathcad assignment	4.2	4.2	4.2
	On-line lessons	4.1	3.6	3.9
	Preparing for weekly tests	3.8	4.2	4.0

Table 1 Summary of Student Responses

students considered looping to be more difficult than other topics taught during the same period of time. The overall rankings indicate that this is the case.

Question 2 on the survey provides additional information on how difficult students perceived working with loops to be during their first week. Students responded with difficulty rating of 3.4 of 5.0, indicating that students perceived working with loops to be moderately difficult from the beginning of instruction on loops. This tends to confirm the previous ranking for the relative difficulty of the early topics (single ***for***-loops, ***break*** and ***continue*** operators, and the ***while***-loop) obtained from first question on the survey. Over the seven-week period, then, the initial topics on looping were perceived to be difficult, the difficulty level decreased following the lesson on ***while*** loops, and the difficulty level then increased toward the end when *nested loops* were encountered. Ideally, the difficulty level should begin at a level that promotes learning and increase gradually as students gain more experience with the concepts.

The remaining items in the survey asked students about the benefit of six tools, techniques or activities used during the course related to their understanding of loops in general and of subscripted variables in loops. Questions 2 and 3 addressed whether the students believed that Mathcad's ***trace*** and ***pause*** features were valuable in helping students to understand loops and subscripted variables. Over two-thirds of the students (69%) expressed that the debugging features were helpful in helping them to learn either loops or subscripted variables or both. As noted previously, these features were intentionally emphasized more in Section 2 than in Section 1. The additional emphasis may have paid off by the time that subscripted variables were introduced, as almost twice the percentage of students in Section 2 believed that they benefited from the debugging features as compared to the percentage in Section 1.

Students were asked in Questions 5 and 6 to assign a helpfulness rating from 1 to 5 for each of the six listed items associated with understanding loops or understanding subscripted variables. The average "helpfulness ratings" were used to rank order these six items for each section and the combined enrollment in the lists presented in Table 2. While 69% of the students responded in Questions 3 and 4 that the new debugging functions, ***trace*** and ***pause***, contributed to their understanding of loops or subscripted variables, there were several other teaching methods that were considered to be more valuable. As shown in Table 2, students rated "doing a Mathcad assignment" as the most helpful activity for understanding both loops in general and subscripted variables in loops. Preparing for the weekly tests, using paper and pencil to track the values of the variables in a loop while "playing computer," and the on-line access to the electronic workbook files prepared for each week's lesson were closely grouped to fill out the four most helpful items.

Even though flow charts appear ahead of the debugging features, the average "helpfulness ratings" for flow charts is almost numerically equivalent to the ***trace*** and ***pause*** ratings. However, the authors agreed that in incorporating the new debugging features into the course, flow charts had not been emphasized as much as in previous years. How a topic is presented can significantly impact how valuable students perceive it to be. An excellent illustration of this is occasionally recounted by a colleague [3]. While teaching a transportation class this colleague somewhat apologetically introduced the subject of funding, legislation and the history of transportation, leaving the impression with the students that these subjects would be boring. Following the course, this is exactly what the students reported they experienced with this material. In future years, he changed his approach, demonstrating enthusiasm for the subject. Students responded in kind as they reported on the course at the end of the semester. Moral: in future years the authors look forward to representing not only flow charts, but the ***trace*** and ***pause*** features also with enough enthusiasm that students will inherit the maximum benefit from these features.

CONCLUDING COMMENTS

Mathcad has been a valuable tool for students at The Citadel for over ten years. With each new version, the potential of this package has continued to expand for a variety of uses. The new debugging tools introduced in Version 13 add to the package both debugging and a way to enhance teaching and learning of loops and subscripted variables.

Through the survey, the students verified that loops and subscripted variables were difficult to learn and that the debugging tools contributed another dimension for improving their understanding. Although not as helpful as other

teaching methods such as doing weekly assignments, preparing for weekly tests, "playing computer," and working with on-line electronic workbook files, the debugging features were still a useful addition to the instructor's toolkit.

The survey also provided the authors with information that may help improve the course by facilitating the teaching of loops and subscripted variables. In the first two weeks of instruction on loops, three features are currently presented that are among the most difficult topics for students. It may be worthwhile to rearrange topics (e.g., ***break*** and ***continue*** operators and ***while*** loops) so that the difficulty level gradually increases instead of having the most difficult topics all at either the beginning or end of the time spent on loops and subscripted variables.

Using a scale of 1 to 5 (5 most helpful) rate how useful were the following techniques or features in helping you to understand loops in general		
Sec 01	**Sec 02**	**Combined**
Doing an assignment in Mathcad — 4.4	Doing an assignment in Mathcad — 4.4	Doing an assignment in Mathcad — 4.4
On-line lessons — 4.3	Preparing for weekly tests — 4.2	On-line lessons — 4.0
Preparing for weekly tests — 3.9	Using pencil and paper, track the value of variables through a loop — 3.9	Preparing for weekly tests — 4.0
Using pencil and paper, track the value of variables through a loop — 3.6	On-line lessons — 3.7	Using pencil and paper, track the value of variables through a loop — 3.8
Flow charts — 3.1	Flow charts — 3.4	Flow charts — 3.3
Using the trace window with the pause or trace function — 2.8	Using the trace window with the pause or trace function — 3.3	Using the trace window with the pause or trace function — 3.1
Using a scale of 1 to 5 (5 most helpful) rate how useful were the following techniques or features in helping you to understand subscripted variables in loops		
Sec 01	**Sec 02**	**Combined**
Doing an assignment in Mathcad — 4.2	Preparing for weekly tests — 4.2	Doing an assignment in Mathcad — 4.2
On-line lessons — 4.1	Doing an assignment in Mathcad — 4.2	Preparing for weekly tests — 4.0
Using pencil and paper, track the value of variables through a loop — 3.9	Using pencil and paper, track the value of variables through a loop — 4.1	Using pencil and paper, track the value of variables through a loop — 4.0
Preparing for weekly tests — 3.8	On-line lessons — 3.6	On-line lessons — 3.9
Flow charts — 3.0	Using the trace window with the pause or trace function — 3.2	Flow charts — 3.0
Using the trace window with the pause or trace function — 2.6	Flow charts — 3.0	Using the trace window with the pause or trace function — 2.9

Table 2 Ranked by Students' "Averaged Helpfulness Rating"

References

[1] Hammond, Robert H., William B. Rogers, and John B. Crittenden, *Introduction to Fortran 77 and the Personal Computer*, McGraw-Hill Nook Company, New York, 1987.

[2] Borse, G. J., *FORTRAN 77 and Numerical Methods for Engineers*, Second Edition, PWS-Kent Publishing Company, Boston, 1991.

[3] Dion, T. R., Personal Communication, 2006.

John Alden Murden

Since 1989, J. A. Murden has served on the faculty in Civil and Environmental Engineering at The Citadel and currently holds the rank of Associate Professor. He earned his B.S., M.S. and Ph.D. degrees in Civil Engineering from Clemson University in 1977, 1984 and 1987 respectively. Dr. Murden's current interests include modeling civil engineering systems, experimental mechanics, computer applications, and improving sophomore and junior courses in civil engineering. He brings industrial experience in aerospace structures, nuclear engineering and shipboard structures into his courses.

Kenneth P. Brannan

Kenneth P. Brannan is Professor and Head of Civil and Environmental Engineering at The Citadel. He was President of the Southeastern Section of ASEE during 1998-1999, co-recipient of the Thomas C. Evans Instructional Paper Award for 1990, and co-recipient of the Best Paper Award at the 2005 ASEE Annual Conference. He earned B.C.E. and M.S. degrees from Auburn University and the Ph.D. from Virginia Tech. A registered professional engineer, Dr. Brannan has interests in freshman engineering education, computers in engineering education, water supply, and wastewater treatment systems.

APPENDIX – STUDENT SURVEY AND RESULTS

CIVL 209 – LOOP & SUBSCRIPTED VARIABLE ASSESSMENT

1) Please rank the following Mathcad features or topics in order of increasing difficulty to understand. Place a 1 in the blank corresponding to the least difficult Mathcad feature, a 2 in the blank corresponding to the next most difficult, etc.
(Rank 1 – 8 use each number only once)
__ for loops
__ while statement
__ nested loops
__ vectors — use of single subscript variables
__ matrices — use of multiple subscript variables
__ if statement in multi-line functions — conditions
__ use of subscripted variables in loops
__ the break and continue operators in loops

2) Using a scale of 1 to 5, how difficult were loops to understand during the first week that you encountered them (**5 is the most difficult**)? 1 2 3 4 5

3) Did the trace or pause function help your general understanding of loops?
Yes No

4) Did the trace or pause function help your understanding of subscripted variables in loops?
Yes No

5) Using a scale of 1 to 5, how useful were the following techniques or features in helping you to understand loops in general (**5 is the most helpful**)?
Less - More
1 2 3 4 5 — Flow charts
1 2 3 4 5 — Using pencil and paper, tracking the value of variables through a loop
1 2 3 4 5 — Using the trace window with the pause or trace function
1 2 3 4 5 — Doing an assignment in Mathcad
1 2 3 4 5 — On-line lessons
1 2 3 4 5 — Preparing for weekly tests

6) Using a scale of 1 to 5, how useful were the following techniques or features in helping you to understand subscripted variables in loops (**5 is the most helpful**)?
Less - More
1 2 3 4 5 — Flow charts
1 2 3 4 5 — Using pencil and paper, tracking the value of variables through a loop
1 2 3 4 5 — Using the trace window with the pause or trace function
1 2 3 4 5 — Doing an assignment in Mathcad
1 2 3 4 5 — On-line lessons
1 2 3 4 5 — Preparing for weekly tests