'Explain in Plain English' Questions: Implications for Teaching

Laurie Murphy
Pacific Lutheran University
Tacoma, Washington
lmurphy@plu.edu

Renée McCauley
College of Charleston
Charleston, SC 29424
mccauleyr@cofc.edu

Sue Fitzgerald
Metropolitan State University
St. Paul, MN 55106
sue.fitzgerald@metrostate.edu

ABSTRACT
This paper reports on the replication of a study of novice programmers, looking for relationships between ability to 'explain in plain English' the meaning of a code segment and success in writing code later in the semester. This study explores the question in a different learning environment and qualitatively evaluates 'explain in plain English' responses to identify implications for teaching. Statistical results from this study are similar to those of the earlier work. Results highlight students' fragile knowledge, particularly for students excluded from the primary analyses by a set of screening questions, and suggest the need for assessment and instruction of basic concepts later into the term than instructors are likely to expect.

Categories and Subject Descriptors
K.3.2 [**Computing Milieux**]: Computers and Education - Computer and Information Science Education

General Terms
Human Factors

Keywords
Computer science education research, qualitative research methods, mixed methods, explain in plain English.

1. INTRODUCTION
Computer science faculty members have long been baffled by the difficulty many novices have when learning to program. Data exist to support the assertion that, after completing their introductory programming courses, many intermediate students can neither read nor write code [6, 11]. In an effort to explain this phenomenon, Whalley et al. [14] hypothesized that "a vital step toward being able to write programs is the capacity to read a piece of code and describe [its function]." Building on this notion, Lopez et al. [10] found a correlation between student ability to explain a piece of code using plain English and their ability to write similar code. Lopez et al. suggested that there is a hierarchy of programming related tasks. "Knowledge of programming constructs forms the bottom of the hierarchy, with 'explain in plain English', Parson's puzzles and the tracing of iterative code forming one or more intermediate levels in the hierarchy."

In 2011, Corney, Lister and Teague (CLT) [3] performed a longitudinal study of novice programmers, looking for relationships between ability to 'explain in plain English' the meaning of a code segment, and success in writing code later in the semester. CLT found that students who could not correctly explain code early in the term were more likely to have difficulty writing code throughout the remainder of the semester. This paper reports preliminary findings of a replication of the CLT project designed to determine if similar results would be observed in a different environment. Additionally, this study extends the work of CLT by qualitatively evaluating students' 'explain in plain English' responses to identify implications for teaching. It is part of a larger study aimed at assessing the impact of teaching students to explain code over the course of a semester.

Results highlight students' fragile knowledge and suggest the need for assessment and instruction of basic concepts later into the term than instructors are likely to expect. Context and pedagogic practice appear to play a positive role in encouraging relational reasoning. Statistical results from this smaller study are similar to those of CLT [3], This work also examines the performance of students excluded from the primary analyses by a set of basic screening questions.

This paper is organized as follows: Section 2 summarizes the background research. Section 3 gives details of the methodology of this study. The findings are specified in Section 4. Teaching implications are addressed in Section 5. Finally, Section 6 presents suggestions for future work.

2. BACKGROUND
A series of recent studies have investigated the relationship between novice programmers' ability to write and explain code [7, 8, 9, 10, 13, 14]. These studies are characterized by their use of 'explain in plain English' questions, which present students with a code segment and require them to provide a brief, natural language description of what the code segment accomplishes. (See [5] for the complete list of 'explain in plain English' questions with solutions.) Responses to questions were analyzed in terms of the Structure of the Observed Learning Outcome (SOLO) taxonomy developed by John Biggs and Kevin Collis [1, 2]. Table 1 lists the SOLO reasoning categories with a brief description of how explanations of code segments could be categorized (adapted from [14]).

Lopez et al. [10] showed a link between *relational reasoning* and an ability to write code. These results were confirmed in replication studies [7, 8, 13].

The studies discussed above are similar in that they all elicited student responses at the end of the semester, and the 'explain in plain English' questions involved iteration and arrays.

CLT, however, attempted to assess relational reasoning among novices earlier in the term, prior to the introduction of loops. They investigated loop-less code to determine if relational reasoning was a factor in understanding this type of code segment. They identified code that swaps the contents of two variables as

SIGCSE'12, February 29–March 3, 2012, Rayleigh, NC, USA.

the simplest piece of non-iterative code requiring relational reasoning and used swapping to see if students would struggle with 'explain in plain English' questions as subjects had in earlier studies using loops and arrays. The CLT study involved novice programmers in an introductory programming course who were given non-graded tests during weeks 3 and 5 of semester. The students were told that the test results would be used to guide future teaching. Like earlier 'explain in plain English' studies, data was also collected at the end of the term.

Table 1. SOLO Reasoning Categories

Category	Description
Relational	Provides a summary of what the code does in terms of the code's purpose.
Multistructural	A line-by-line description is provided of all the code.
Unistructural	Provides a description for only one portion of the code
Prestructural	Substantially lacks knowledge of constructs or is unrelated to the question

One of the 'explain in plain English' questions given to students at week 3 presented code that swapped two values. (See Appendix for the test questions.) CLT found that students who could not correctly explain, at week 3, that the code accomplished a swap, performed worse throughout the rest of the semester on various tasks (writing a swap, explaining a sort, explaining code that computed the product of even numbers, tracing a swap, writing code to reverse a string) than students who could explain a swap.

At week 5, participants were asked to write code that swapped two variables. CLT found that students who could not write a swap, at week 5, performed worse on various tasks (explaining a sort, explaining code that computed the product of even numbers, tracing a swap, writing code to reverse a string) throughout the remainder of the semester. Another week 5 question asked students to explain a sort. They found that students who could not explain a sort at week 5 performed consistently worse on later tasks (writing a swap, explaining code that computed the product of even numbers, tracing a swap, writing code to reverse a string) than students who could explain a sort.

CLT suggest that, very early on in the semester, introductory students divide into two groups, those who can think relationally and those who cannot. They suggest that this is the result of current pedagogic practice, not an innate weakness of students.

3. METHODOLOGY

This study adapted CLT's study [3] for use with introductory Java students at a small liberal arts university in the United States by translating their test questions from Python to Java. The questions were given in graded assessments during the spring 2011 semester. When the graded tests were returned the test questions were reviewed by the teacher and the importance of understanding the questions was emphasized. (Revised questions are included in the Appendix.)

3.1 Participants

Participants were 29 undergraduate students (12 female, 17 male) enrolled in a traditional "objects later" introductory Java programming course. Following the process conducted in CLT, data from eleven participants were eliminated from the quantitative analyses due to failure to correctly answer three screening questions (questions 1 through 3 on Test 1, see Appendix). Responses from two other students who did not complete all three question sets were also excluded from the quantitative analysis.

Data from 16 participants were analyzed quantitatively and data from all 29 were considered qualitatively.

3.2 Data Collection

CLT administered their ungraded tests at weeks 3 and 5 and at the end of the semester as part of the final exam. To conserve limited class time and encourage students to respond to the questions seriously, the current study incorporated all study questions into regular graded assessments. CLT's Test 1 questions were added to an exam during the third week of the term, Test 2 questions were added to a quiz during week six, and Test 3 questions were included in an exam given at week nine of a 15 week semester. The research study questions made up 5% of the total points possible for each of the exams, and 25% of the points on the quiz. Although Test 3 was administered before the end of the semester, students in the current study had completed all of the material addressed in those questions by week nine of the term.

3.3 Data Coding

From the students' perspective, the study questions were graded like all other questions on the assessment and they received partial credit in some cases. For analysis purposes, researchers discussed each of the partially correct responses and came to a consensus as to whether they should be counted as correct or incorrect. These were then used to replicate the quantitative analysis presented in the CLT study that found correlations between answers on different test questions.

We also conducted qualitative analyses on some of the test questions. This paper describes our assessment of the 'explain in plain English' exercises from Test 1 which required students to "explain a swap of two variables" and Test 2 which required students to "explain a sort of three values". Test 1 'explain a swap' questions were categorized based on the types of errors students made. We categorized Test 2 'explain a sort' answers according to the SOLO taxonomy based on previous work by Whalley et al. [14] (see Table 1). Responses to this question were categorized independently by two researchers. Three researchers worked together to come to consensus on the appropriate categorization of all responses. These results are discussed in 4.2.

4. FINDINGS

4.1 Quantitative Analysis

This section summarizes this study's quantitative results and compares them with those found in CLT [3]. Due to the small sample size, the numbers are not statistically significant. However, they are indicative of trends similar to those found by CLT. Below we refer to this study's results using SS, for Small Study.

4.1.1 Screening

CLT used the first three questions on Test 1, those designed to assess students' understanding of variables, assignment and simple tracing, as screening questions for their study. Students who did not answer all three screening questions correctly (144, or 63% of their students) were eliminated from the statistical analysis. The argument for doing this was to screen students who did not understand the necessary foundational concepts required to understand swapping. This is similar to excluding subjects who do not have an adequate understanding of English from a study where the questions are posed in English.

For comparison, we used the same procedure of removing students who could not answer the screening questions (11, or 41%) from our analysis. Tables 2-4 compare results from the two studies.

Table 2 shows similar results for students who passed the screening questions in both studies, with our SS students performing somewhat better on the three explanation questions (swap, sort and product of evens) and string reversal, but slightly lower on writing a swap.

Table 2. The percentage of students who answered each question correctly

Question		SS n=16	CLT n=83
Test 1	Explain a swap	63%	47%
Test 2	Write a swap	69%	73%
	Explain a sort of 3 variables	69%	48%
Test 3	MCQ, explain the product of even nums	94%	76%
	Trace a swap	88%	89%
	Write code to reverse a string	75%	59%

4.1.2 Explain a Swap

Table 3 compares the performance of students who could correctly explain a swap on Test 1 with those who could not for each institution. It shows how many in each group correctly answered the subsequent test questions. For example, only 33% of the small study (SS) students who were initially unable to explain a swap were able to write a swap on Test 2. Again, results between the two studies are similar. The most notable differences are on the Test 2 questions for students who could not explain a swap on Test 1; a lower percentage of the SS students were able to write a swap while more were able to explain a sort than those in the CLT study. This may be due in part to the fact that the Test 1 questions were reviewed and the importance of being able to understand and explain code was emphasized when the tests were returned to the SS students.

4.1.3 Write a Swap

Table 4 compares the performance of students who were able to write a swap on Test 2 with those who could not, again showing the percentage that were able to correctly answer subsequent questions. Here the performance for the SS students who incorrectly wrote the swapping code is considerably better than that found by CLT. Again, this suggests that reviewing and emphasizing earlier questions may improve student understanding. Differences in the timing of the tests and the nature of the assessments (graded vs. ungraded) may also have been factors.

4.1.4 Impact of Early Confusion about Variables

Students from our study who did not appear to understand swapping early in the term were able to recover fairly well. This led us to wonder about the students who had been screened from the statistical analysis early on, those who appeared to be confused about fundamental concepts such as variables and assignment. How did they fare as the term progressed?

Table 5 compares SS students who were unable to answer all three screening questions correctly on Test 1 with those who could. Here we see that these students not only did much worse at explaining a swap on Test 1, but that they also continued to struggle with swapping and sorting on Test 2. They seemed to recover somewhat on Test 3 on the 'explaining a product' question, although this question was multiple choice, and particularly on tracing a swap. However, they performed much worse than students who answered the screening questions correctly when it came to altering a loop to reverse a string.

4.2 Qualitative Analysis

As noted above, students were tested in weeks 3, 6 and 9 to see if their ability to explain the purpose of code segments was related to their ability to write code correctly. We took a closer look at several test questions to see what patterns emerged.

The following subsections present a qualitative analysis of the ways in which students went wrong on the two early 'explain in plain English' questions. They reveal misconceptions that appear to impede students' early attempts at relational reasoning.

Table 3. The performance of students, broken down according to the Test 1 'explain a swap' question

Test 1 Explain a swap		SS Wrong (n=6)	SS Right (n=10)	CLT Wrong (n=44)	CLT Right (n=39)	CLT χ^2 test
Test 2	Write a swap	33%	90%	57%	92%	p = 0.001
	Explain a sort of 3 variables	67%	70%	36%	62%	p = 0.03
Test 3	MCQ, explain the product of even nums	83%	100%	64%	90%	p = 0.01
	Trace a swap	83%	90%	82%	97%	p = 0.03
	Write code to reverse a string	50%	90%	41%	79%	p = 0.001

Table 4. The performance of students, broken down according to the Test 2 'write a swap' question

Test 2 Write a swap		SS Wrong (n=5)	SS Right (n=11)	CLT Wrong (n=22)	CLT Right (n=61)	CLT χ^2 test
Test 2	Explain a sort of 3 variables	40%	82%	14%	61%	p = 0.001
Test 3	MCQ, explain the product of even nums	100%	91%	59%	82%	p = 0.03
	Trace a swap	80%	91%	68%	97%	p = 0.001
	Write code to reverse a string	60%	82%	27%	70%	p = 0.001

Table 5. The performance of students, broken down according to the Test 1 screening questions (n = 27)

Test 1 Screening Questions		SS	
		Wrong (n=11)	Right (n=16)
Test 1	Explain a swap	27%	63%
Test 2	Write a swap	18%	69%
	Explain a sort of 3 variables	18%	69%
Test 3	MCQ, explain the product of even nums	73%	94%
	Trace a swap	82%	88%
	Write code to reverse a string	27%	75%

4.2.1 Explain a Swap

We examined the most vulnerable students, those who could not pass the screening questions. Of the 13 students who did not answer all three screening questions correctly (including the two students who did not take all three tests), we observed the following about their answers to the 'explain a swap' question on Test 1:

- Four answered the question correctly. Closer inspection revealed that three of those students only missed one screening question, suggesting they likely did have a reasonably good grasp of variables and assignment and probably just made a careless mistake.
- Three of them indicated that the swap caused all three variables to be equal, which, interestingly, is what the screening questions they answered incorrectly actually did. These students were attempting to grasp the code on a relational level, but probably did not trace the code with example values in order to be sure of its actual purpose.
- Three left the question blank or gave incomplete or incomprehensible responses, suggesting a complete lack of understanding, even though an example swap using different variables was explained just above.
- Two said the statements restored the variables to their original values, ignoring the instructions that stated "NOTE: Tell what the second set of three lines of code do all by themselves. Do NOT think of those second three lines as being executed after the first three lines of code."
- One appeared to believe that assignments happened from left to right, given the following response: *The second set of data states that a = c once program begins because it assigns a->b, b->c, c->a*

Of the students who passed the screening questions, 10 of the 16 correctly described a swap, although one of them provided a multistructural (line-by-line) explanation as well. Errors included:

- Three students gave incorrect multistructural responses that simply described the code line-by-line. E.g., *a is assigned the value in b then b is assigned the value in c then c is assigned the value in a.*
- One gave a unistructural description, stating the code's purpose was to assign the value of c.
- One thought the code assigned the same value to all three variables.
- One gave a nonsensical answer.

Overall, the screening questions appear to have effectively identified those students who had a good understanding of variables and assignment. However, they also appear to have unnecessarily excluded some students. A slightly less stringent threshold for exclusion is indicated.

4.2.2 Explain a Sort

On Test 2, students were asked to explain a sorting code segment. We analyzed this test question by categorizing each answer as relational, multistructural, unistructural or prestructural using the SOLO taxonomy [1]. Whalley et al [14] performed a similar analysis on their data using the definitions given in Table 1.

Two researchers categorized student responses to this test question. Later all three authors negotiated these categorizations to consensus. Although the researchers reached consensus, it must be noted that the exact meanings of the SOLO designations of prestructural, unistructural, multistructural and relational as applied to computer programming are open to interpretation.

Table 6 summarizes the categorization of the 27 student answers using the SOLO taxonomy. The table shows the number of answers which fall into each category as well as the percentage of answers falling into each category. The vast majority were relational (70%); the students provided a summary of what the code did in terms of its purpose, although only 58% of these answers were correct as shown in the right column of Table 6. All prestructural and unistructural answers were incorrect as were most of the multistructural answers.

Table 6. SOLO categorization of answers to 'Explain a Sort'

Category	Number (%)	Correct (%)
Relational	19 (70%)	11 (58%)
Multistructural	4 (15%)	1 (25%)
Unistructural	3 (11%)	0 (0%)
Prestructural	1 (4%)	0 (0%)

Students went wrong in a number of ways.

- Many provided a plain English explanation but incorrectly deduced the purpose of the code.

  It prints the variable y1, y2, & y3 values in reverse order [Relational]

  It assigns y1 the highest value of the three variables [Relational]

  It prints y3 first, then the y2, finally the y1 because each step allowed a swap [Multistructural]

  To print y2 first, y2 second the[n] y1 [Unistructural]

- A few provided code instead of plain English [Prestructural]
- Some answers were correct and explained in plain English, but still reflected a step-by-step understanding

  The code prints y1, y2, & y3 after having made y1 hold the largest value, y3 the smallest and y2 the in between value. [Multistructural]

- Yet other answers were partially correct but subtly wrong, describing only part of the answer:

  It swaps the smaller values of two variables [Unistructural]

- Sadly, a few were simply incomplete.

  swap y1 and y2 when [Unistructural]

5. IMPLICATIONS FOR TEACHING

For some disciplines, introductory courses cover loosely related topics over a term, which means a misconception about one topic is unlikely to impede students' understanding of later material. The cumulative nature of programming, however, means that early misunderstandings can have disastrous consequences later in the course. 'Explain in plain English' questions appear to be an effective means for gauging students' level of abstract thinking.

We discuss here some possible implications of our findings on teaching practice.

5.1 Role of context and review

For students who demonstrated an understanding of variables and assignment, CLT found statistically significant correlations between ability to explain a swap at week 3 and write a swap at week 5, with performance on those tasks and on other programming tasks later in the term. Although analysis for statistical significance was not possible for this study, similar trends were seen for SS. At both institutions students who were able to explain swapping and sorting performed better later in the term. However, SS students who were *not* able to explain a swap appear to have recovered and performed better than the CLT group as the term progressed.

While our results should be viewed cautiously due to the small dataset, they suggest that pedagogic practice and context may somewhat mitigate difficulties for those students who catch on to the basics (i.e., variables and assignment) early in the term. Not only were the early swapping questions reviewed and emphasized for the SS students, but their overall learning context is also highly supportive; these students attend very small classes (15-25 students) held in a computer classroom that facilitates hands on practice, much of it done in pairs. They also participate in weekly two hour lab sessions conducted by the instructor and an undergraduate TA. Despite these advantages, for students who do not understand the basic concepts at week 3, recovering and succeeding in the course appears to be difficult.

5.2 Applying innovative teaching practice to identify and address fragile knowledge

As teachers we often assume that students understand the basics, particularly as the course progresses. However, the test data reveal confusion about basic concepts well into the semester, and suggest explanation questions could be useful for identifying students with fragile knowledge early in the term. Such students could be encouraged to visit tutors or directed to online visualizations or video podcasts to help them master these basic concepts. Common errors should be addressed explicitly and could be used in paired or small group discussions or as distracters for multiple-choice questions. These MCQs could be presented in the context of *Just in Time Teaching* [12] as a way of assessing students' understanding of basic concepts before covering more advanced concepts during class; or using *Peer Instruction* [4], which gives students an opportunity to discuss their responses in small groups as a way of challenging potentially faulty mental models.

6. FUTURE WORK

This work was conducted in the context of a larger study designed to examine the impact of exposing students to 'explain in plain English' questions over a term. In addition to the results reported here, students in this study were also given a set of more complex 'explain in plain English' questions involving loops and arrays on their final exam. Their performance on those questions, and on code writing questions, will be compared with those of students from earlier semesters at the same institution who were not given 'explain in plain English' test questions throughout the term.

'Explain in plain English' questions may well prove to be an effective way to gauge the point at which student thinking moves from concrete to abstract. A study of the impact of innovative teaching practices such as Just in Time Teaching or Peer Instruction in combination with 'explain in plain English' questions may shed light on students' ability to overcome early misconceptions.

7. ACKNOWLEDGMENTS

We extend our thanks to Raymond Lister for so generously sharing his problem sets.

8. REFERENCES

[1] Biggs, J. & Collis, K. 1982. *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, New York, NY.

[2] Biggs, J. SOLO Taxonomy. Retrieved May 6, 2011 from http://www.johnbiggs.com.au/solo_taxonomy.html

[3] Corney, M., Lister, R. & Teague, D. 2011. Early relational reasoning and the novice programmer: Swapping as the "Hello World" of relational reasoning. *Proceedings of the 13th ACE Conference* (ACE '11). Australian Computing Society.

[4] Crouch, C. H. & Mazur, E. 2001. Peer instruction: Ten years of experience and results. *American Journal of Physics* 69.

[5] Lister, R. Explain in Plain English Questions: Examples and Answers. 2010. Retrieved May 19, 2011 from http://www.staff.it.uts.edu.au/~raymond/braceletace2010.

[6] Lister R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, E., Sanders, K., Seppälä, O., Simon, B., Thomas, L. 2004. A multi- national study of reading and tracing skills in novice programmers. *SIGCSE Bull.* 36,4 (December 2004), 119-150.

[7] Lister, R., Clear, T., Simon, Bouvier, D. J., Carter, P., Eckerdal, A., Jacková, J., Lopez, M., McCartney, R., Robbins, P., Seppälä, O., & Thompson, E. 2009. Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer. *SIGCSE Bulletin* 41:4, 156-173.

[8] Lister, R., Fidge C. & Teague, D. 2009. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. In Proceedings *of the14th Annual ITiCSE Conference* (ITiCSE '09). 161-165.

[9] Lister, R., Simon, B., Thompson, E., Whalley, J. & Prasad, C. 2006. Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy, in *Proceedings of the 11th Annual ITiCSE Conference* (ITiCSE '06), 118-122.

[10] Lopez, M., Whalley, J., Robbins, P., & Lister, R. 2008. Relationships between reading, tracing and writing skills in introductory programming, in *Proceedings of the* 4th *ICER Workshop* (ICER'08), 101-112.

[11] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, K., Thomas, L., Utting, I., & Wilusz, T. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bull.* 33, 4 (December 2001), 125-140.

[12] Novak, G., Gavrin, A. & Wolfgang, C. 1999. Just-in-Time Teaching: Blending Active Learning with Web Technology, Prentice Hall, Upper Saddle River, NJ.

[13] Venables, A., Tan, G., & Lister, R. 2009. A closer look at tracing, explaining and code writing skills in the novice programmer. *Proceedings of the ICER Workshop* (ICER'09), 117-128.

[14] Whalley, J., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P. & Prasard, C. 2006. An Australasian Study of Reading and Comprehension Skills in Novice Programmers, using the Bloom and SOLO Taxonomies. In *Proceedings of the 8th ACE Conference* (ACE'06), 243-252.

APPENDIX

TEST 1
(Adapted from Corney, Lister & Teague, Exam 1 Spring 2011 at 3 weeks)

For the questions below, you may write down any work on this exam paper, except in the answer boxes. Write ONLY your answer in the answer boxes. (5 points)

1. In the boxes provided below, write the values in the variables after the following code has been executed:

   ```
   r = 2;  s = 4;  r = s;
   ```
   {Displayed vertically here to save space.}

2. In the boxes provided below, write the values in the variables after the following code has been executed:

   ```
   P = 1;  q = 8;  p = q;  q = p;
   ```
   {Displayed vertically here to save space.}

3. In the boxes provided below, write the values in the int variables after the following code has been executed:

   ```
   x = 5;  y = 3;  z = 7;
   ```
   {Displayed vertically here to save space.}

   ```
   x = z;  y = x;  z = y;
   ```

4. The purpose of the following three lines of code is swap the values in variables a and b:

   ```
   c = a;  a = b;  b = c;
   ```
   {Displayed vertically here to save space.}

   The three lines of code below are the same as the lines above, but in a different order:

   ```
   a = b;  b = c;  c = a;
   ```
   {Displayed vertically here to save space.}

   In one sentence that you should write in the box below, describe the purpose of those second set of three lines. NOTE: Tell what the second set of three lines of code do all by themselves. Do NOT think of those second three lines as being executed after the first three lines of code.

   {BOX OMITTED}

5. *{QUESTION OMITTED –same swap as above for variables i, j and k}*

TEST 2
(Adapted from Corney, Lister & Teague, Quiz 3 Spring 2011 at 6 weeks)

1. (2 point) Suppose you have two integer variables, called `p` and `q`. **In the box below write code to swap the values in those two variables**. You may declare and use any extra variables required to make the swap. Give each extra variable a meaningful name that reflects its purpose. *{BOX OMITTED}*
2. (1 point) *{QUESTION OMITTED – tracing a simple nested if}*
3. (2 points) If you were asked to describe the purpose of the code below, a good answer would be "*It prints the smaller of the two values stored in the variables* `a` *and* `b`".

   ```
   if (a < b) System.out.print(a);
   else  System.out.print(b);
   ```

   In one sentence that you should write in the empty box below, describe the purpose of the following code.

   Do **NOT** give a line-by-line description of what the code does. Instead, tell us the purpose of the code, like the purpose given for the code in the above example (i.e. "*It prints the smaller of the two values stored in the variables* `a` *and* `b`").

   Assume that the variables y1, y2 and y3 are all variables with integer values.

   In each of the three boxes that contain sentences beginning with "`Code to swap the values …`",assume that appropriate code is provided instead of the box – do **NOT** write that code.

   ```
   if (y1 < y2)
       Code to swap the values in y1 and y2 goes here.
   if (y2 < y3)
       Code to swap the values in y2 and y3 goes here.
   if (y1 < y2)
       Code to swap the values in y1 and y2 goes here.
   System.out.println(y1);
   System.out.println(y2);
   System.out.println(y3);
   ```
   {BOX OMITTED}

TEST 3
(Adapted from Corney, Lister & Teague, Exam 2 Spring 2011 at 9 weeks)

1. (2 points) Which best describes the purpose of the following function definition? You may assume the method has access to the `Scanner` object `keyboard`.

   ```
   public static intdoSomethingWithNumbers( ) {
           int total = 1;
           System.out.print("Please input an integer: ");
           response = keyboard.nextInt();
           while (response != 0) {
                   if (response % 2 == 0)
                      total = total * response;
                   System.out.print("Please input an integer: ");
                   response = keyboard.nextInt();
           }
           return total;
   }
   ```

   a) It does not do anything as the body of the while loop never executes
   b) It returns the product of all numbers entered
   c) It returns the product of all even numbers entered
   d) It returns the product of all odd numbers entered

2. (1 point) What do the variables **`value1`**,**`value2`** and **`value3`** hold after the following Java code is executed? Assume that they are all int variables.

   ```
   value1 = 10; value2 = 15; value3 = value1; value1 = value2;
   value2 = value3;
   ```

   {Code displayed horizontally here to save space. Labeled boxes for answers omitted}

3. (2 points) The following Java code copies a string:

   ```
   String source = "the cat sat on the mat";
   String target = "";
   for (inti = 0; i<source.length(); i++)
           target = target + source.charAt(i);
   ```

   Rewrite this code snippet so the target string contains the source string in reverse. E.g., "abc" becomes "cba". *{BOX OMITTED}*