Preparing Teachers to Integrate Computer Programming
Into Mathematical Problem Solving

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By
David D. P. Ely, B.A., M.Ed.

Education Teaching and Learning

The Ohio State University
2016

Dissertation Committee:
Dr. Patricia Brosnan, Advisor
Dr. Christopher Stewart
Dr. Lin Ding
Copyright by

David D. P. Ely

2016
Abstract

Responding to the call of the Association for Computing Machinery (ACM) and the CS/10K Project for the development of models to train K – 12 Computer Science teachers, this study examined the effectiveness of a 3-week intensive summer workshop to prepare secondary mathematics teachers to integrate computer programming into mathematical problem solving. The workshop, using the C programming language, was built off of pedagogical principles that emphasized hands-on primary learning, a well-crafted sequential set of learning tasks, applications from mathematics, extensive group work, minimal lecturing, numerous discussions, and a focus on problem solving and logic instead of syntax,

Specifically, the study examined if the workshop could effectively prepare and motivate secondary mathematics teachers to integrate computer programming into the study of classical probability problem solving. The teacher participants were measured on 1) their ability to solve probability problems with and without computer programming, 2) their self-efficacy to teach programming and probability, and 3) their change in motivation levels to integrate computer programming into their teaching of mathematics.

Mathematics teachers from grades 7 -14 were the target audience for this workshop as they were deemed the most likely teachers to benefit from incorporating
computer programming into the classroom. In order to connect with these teachers, classic probability was the chosen mathematical discipline for exploring computer programming. Probability is now taught at grades 6 and beyond in mathematics classrooms, yet many teachers are uncomfortable with the subject. Experimental probability through programming models and simulations offers an alternate route to the traditional, and sometimes unattainable, theoretical probability models.

Using measurement tools of probability tests, surveys, and journal submissions, the study concluded that the workshop had 1) a significantly positive impact on improving the teacher’s performance of classical probability problems, 2) a significant increase in teachers’ comfort levels in using programming for probability problem solving, and 3) a significant increase in comfort and motivation to incorporate programming teaching. A follow-up on one of the teachers found he had successfully implemented portions of the workshop curriculum in a middle school class just two months after completing the workshop.
Dedication

For my mother Jeanette, whose endless support has made all of the difference
I wish to thank my advisor Dr. Patricia Brosnan for guiding me through the long and arduous Ph.D. process and for believing in my ideas and supporting my convictions. I wish to thank Dr. Lin Ding for inspiring me in the classroom and challenging my understanding of science. I wish to thank Dr. Christopher Stewart for over seven years of support, which he selflessly offered long after the focus of my studies changed from computer science to mathematics education.
Vita

2004 ..................B.S. Mathematics & Computer Science, Lewis & Clark College
2006 ..................M.Ed. Mathematics Education, Portland State University
2006 to present ........Licensed Advanced Mathematics Teacher, Oregon
2006 – 2008 ........Secondary Mathematics Teacher, West Linn High School, Oregon
2008 – 2011 ........Graduate Teaching and Research Associate, Department of Computer and Information Sciences, The Ohio State University
Summer 2014............Adjunct Instructor, Department of Mathematics & Statistics, Portland State University.
2011-2015 ............Graduate Teaching Associate, Intern Supervisor, Department of Curriculum and Instruction, The Ohio State University
2015 – present ........Sabbatical Replacement, Department of Mathematics and Computer Science, Lewis & Clark College

Publications
 Sciences (HICSS) (Peer-to-Peer Ecommerce Systems and Applications Minitrack), 2004


Fields of Study

Major Field: Education Teaching and Learning

Specializations: STEM Education and Doctoral courses in Computer Science
Table of Contents

Abstract.................................................................................................................................................. ii
Dedication .................................................................................................................................................. iv
Acknowledgments ....................................................................................................................................... v
Vita............................................................................................................................................................. vi
List of Tables .............................................................................................................................................. xiii
Chapter 1: Introduction ......................................................................................................................... 1
 Why Mathematics Teachers ................................................................................................................... 4
 Teacher Workshop ..................................................................................................................................... 4
 Problem Statement ................................................................................................................................... 7
  Why probability ....................................................................................................................................... 8
  Theoretical vs. experimental methods ..................................................................................................... 10
  Modeling vs. simulation .......................................................................................................................... 12
  Programming vs. statistical-software .................................................................................................... 14
 Plan ......................................................................................................................................................... 16
Chapter 2: Review of the Literature ....................................................................................................... 17
 Challenges to Learning Probability ......................................................................................................... 17
 Teachers’ Preparedness to Teach Probability .......................................................................................... 21
Ease-Of-Use vs. Learning Benefits of Technology and Programming ...................... 23

Simulation/modeling software .................................................................................. 24

General-purpose programming tools ....................................................................... 28

Effective Training for Teaching Computer Programming........................................... 32

Challenges to effective teacher preparation............................................................... 32

Offering more effective preparation through CGI...................................................... 35

Training teachers in computer science .................................................................... 39

Probability and Computer Programming in Common Core State Standards ........... 41

Summary ..................................................................................................................... 44

Chapter 3: Design Description .................................................................................... 46

Subjects ....................................................................................................................... 47

Workshop Description ............................................................................................... 48

Theoretical framework............................................................................................... 49

Environment, length, and typical day ........................................................................ 51

Instructional programming tasks .............................................................................. 53

I/O ............................................................................................................................... 56

Mathematics constructs ............................................................................................ 56

Conditional statements ............................................................................................. 57

Loops .......................................................................................................................... 58

Arrays ......................................................................................................................... 60

Measurement Tools, Data Collection Methods, and Analysis................................... 62
Tests ................................................................................................................................................. 63

Surveys ................................................................................................................................................ 66

Journals/portfolios ................................................................................................................................. 68

Data analysis plans ............................................................................................................................... 70

Theoretical framework for qualitative data analysis ................................................................. 72

Phenomenology ................................................................................................................................. 72

Why phenomenology for workshop analysis ............................................................................... 73

Analysis plans for each measurement tool ............................................................................... 75

Analyzing Tests ............................................................................................................................... 75

Analyzing Surveys – Likert Scale items ....................................................................................... 75

Analyzing journals/portfolios and open-ended survey responses ............................................ 76

Chapter 4: Results ............................................................................................................................... 78

Comparison of Pretests and Posttests ......................................................................................... 78

McNemar Test results ....................................................................................................................... 79

Reliability and validity of tests ..................................................................................................... 88

Survey Results ...................................................................................................................................... 91

Paired Comparison Survey Questions ......................................................................................... 93

1. “Comfort writing a program” analysis ...................................................................................... 95

2. “Comfort using a program to do/teach mathematics” analysis ............................................. 96

3. “Comfort teaching programming” analysis ............................................................................. 96
4. “Belief programming should be essential for mathematics education”

analysis. ........................................................................................................... 96

5. “Motivation to integrate programming into teaching mathematics”

analysis ............................................................................................................. 97

Characteristics and beliefs of workshop participants .................................. 97

Perceptions of future programming and problem solving improvement .... 99

Factors based on Characteristic and Background Questions ...................... 101

Reliability and validity .................................................................................... 106

Journals Analysis ............................................................................................. 109

Workshop Structure ....................................................................................... 111

Comparison to previous coursework ......................................................... 111

Content structure ............................................................................................ 112

Pedagogy ........................................................................................................... 116

Learning Processes .......................................................................................... 121

Programming strategies ................................................................................ 121

Probability Strategies .................................................................................... 126

Personal Outcomes ......................................................................................... 130

Emotional reaction .......................................................................................... 130

Progression of comfort ............................................................................... 133

Future applications ........................................................................................ 135
Chapter 5: Conclusions ................................................................. 138

Post-Workshop Implementation: An Interview with Terry ....................... 138

Study Conclusions ........................................................................ 141

Study Limitations ........................................................................ 146

Future Research ........................................................................... 148

APPENDIX A: Programming Tasks and Coding .................................. 155

APPENDIX B: Pretest and Posttest .................................................. 185

APPENDIX C: Pre-Survey and Post-Survey .................................... 190

APPENDIX D: Journal data referenced by line number ..................... 195
List of Tables

Table 1: Teachers’ Backgrounds ................................................................. 48
Table 2. Sequential Programming Tasks ..................................................... 55
Table 3. Rater I & II Scores of Pretest & Posttest Questions 1=correct, 0= incorrect .... 81
Table 4. McNemar Test on pretest and posttest scores for Dice, Chips and Defective ... 83
Table 5. Inter-Rater Reliability on pretests and posttests ................................... 89
Table 6. Raw Data of Surveys ........................................................................ 92
Table 7. Paired Comparisons of Pre-Survey and Post-Survey Responses.............. 95
Table 8. Teachers’ Beliefs on Use of Technology on Pre-Survey.......................... 99
Table 9. Expectation to Program in Future and Belief that Problem Solving Improved100
Table 10. Factors Affecting Teachers’ Increased Comfort Levels in Writing Programs103
Table 11. Factors Affecting Teachers’ Increased Comfort Level in Using Programming to Do or Teach Mathematics ............................................................................... 104
Table 12. Factors Affecting Teachers’ Increased Comfort Level in Teaching a Programming Class or Unit ..................................................................................... 104
Table 13. Factors Affecting Teachers’ Increased Positive Beliefs that Programming is Essential in Mathematics Education ........................................................................ 104
Table 14. Factors Affecting Teachers’ Increased Motivation to Add Programming to Mathematics Classes Taught................................................................. 105

Table 15: Sequential Programming Tasks (repeated)......................................................... 156
List of Figures

Figure 1. Pretest Questions ........................................................................................................ 64
Figure 2. Supplemental questions to pretest ........................................................................... 64
Figure 3. Likert responses for survey ......................................................................................... 67
Figure 4: Comfort level questions on Post-Survey .................................................................... 68
Figure 5. Portfolio and journal instructions and prompts ......................................................... 69
Figure 6. Sample of imperfect simulation code for defective probability problem ........... 86
Figure 7. Thematic Hierarchical Structure for Journal Analysis ............................................. 110
Engaging students in mathematics can be a challenge for even the strongest and most experienced teachers. In an effort to increase students’ interest and understanding of mathematics, the modern mathematics classroom often integrates technology into the curriculum, and for good reason, as young people today are often fluent in the use of powerful, multi-purpose technologies, such as smart phones, tablets, and personal computers, before ever stepping foot in a classroom that utilizes said technologies. The modern mathematics classroom is using technology at an unprecedented level, and the National Council of Teachers of Mathematics in their position statement strongly urges the use of technologies in mathematics education:

It is essential that teachers and students have regular access to technologies that support and advance mathematical sense making, reasoning, problem solving, and communication. Effective teachers optimize the potential of technology to develop students’ understanding, stimulate their interest, and increase their proficiency in mathematics. When teachers use technology strategically, they can provide greater access to mathematics for all students. (“Technology in Teaching and Learning Mathematics”, 2011)
Yet, computer programming, the basic, driving force behind making technology useful, continues to largely be ignored in K-12 education. Fluency in computer programming bends technology to meet the specific needs of the task at hand, which may be crucial in an endeavor as specific and nuanced as mathematical problem solving. While technology usage has become widespread across academia, general purpose programming of the technology is still primarily considered a niche discipline (Armoni, 2011). By neglecting computer programming in the typical mathematics classroom we may be ignoring a powerful means to drastically increase students’ engagement in and understanding of mathematics.

This study attempts to expand upon the premise that mathematics education and computer programming can and should have a synergistic relationship. This relationship can be leveraged to not only meet national goals for increasing computer programming at the secondary level, but also ongoing goals of the mathematics education community for exploring mathematics in new and deeper ways (Stephenson, Gal-Ezer, Haberman, & Verno, 2005).

Enrollment in mathematics and sciences are on the increase in secondary schools in the United States. However, in many secondary schools computer programming classes are non-existent or have been in decline over the last 20 years, according to the 2009 NAEP Report on America’s High School Graduates, (Nord, et al., 2011). In an effort to address this problem, The Association for Computing Machinery (ACM) and the National Science Foundation (NFS) have spawned an initiative called the CS/10K Project, which aimed “to develop an effective high school curriculum that will be taught
by 10,000 well-prepared teachers in 10,000 high schools by 2015” (Cuny, 2010, p. 1). In 2005, the Computer Science Teachers Association (CSTA) recognized a need for computer science teachers at the K-12 level that were accredited computer science educators with a mastery of content knowledge. “New high school computer science teachers should be required to have completed an undergraduate degree in computer science or a comparable degree program” (Stephenson, Gal-Ezer, Haberman, & Verno, 2005, p.12). However, as was further noted, computer science majors rarely enter the teaching profession.

Through the CS/10K Project, Jan Cuny states, “The greatest challenge will be scaling the teacher preparation opportunities and support to reach 10,000 computing teachers by 2015. Too few high schools have computing teachers with formal CS training and many have no computing teacher at all” (Cuny, 2010, p.2). With 2015 now in the past and without these national goals yet met, is a new tactic necessary to fill schools with highly qualified computer programming educators? It may be unrealistic to expect that budget-strapped high schools will hire computer science specialists with degrees in computer science to teach a full load of computer science courses. But what if a solution is already in place?

This study advances the hypothesis that training mathematics teachers in computer programming offers an avenue towards meeting the CS/10K Project goals, as well as the recommendations of the National Council for Teachers of Mathematics (NCTM) for strengthening mathematical problem solving and using technology in mathematics education (NCTM, 2000).
Why Mathematics Teachers

Consider the following reasons why training mathematics teachers in computer programming could be a beneficial approach to the inclusion of programming into secondary level education:

1. Every secondary school in America already has licensed mathematics teachers, thus eliminating the need to hire new staff at potentially great expense.

2. Mathematics teachers are already trained in a discipline with a strong tradition of problem solving and logical reasoning; skills essential to programming.

3. Computer programming may be of great value to doing and understanding mathematics and vice versa.

4. Some school districts will not hire full time computer science teachers, but they might hire someone who can teach 1-2 programming classes as part of their 5-6 course load.

If mathematics teachers are the key to injecting computer science into high schools, then it is essential that programming is tied to their interests and comfort areas. In other words, it is imperative that they are shown the positive, powerful impact computer programming can have on teaching and understanding mathematics.

Teacher Workshop

An annual graduate course/teacher workshop was developed to offer secondary mathematics teachers an experience that could empower them to integrate computer programming into their mathematics classes. This teacher workshop is offered during
summer terms through a masters’ program for secondary level and community college mathematics teachers in the mathematics and statistics department at a large public university in the Pacific Northwest. This teacher workshop utilizes a curriculum and pedagogy that was specifically designed to prepare secondary school mathematics teachers to teach computer programming while simultaneously exposing these teachers to mathematics in non-traditional, computer-assisted ways that they, as well as their own K-12 students, may find fun, engaging, and approachable. The workshop merges mathematics problems with carefully crafted sequential programming tasks to keep the class rooted as much as possible in mathematical exploration. Syntax is de-emphasized to minimize the tedium and frustration of learning a new programming language.

For example, the teachers are presented with a problem such as determining the probability that two randomly placed rooks on a chessboard can “capture each other.” The teachers first determine how to numerically represent the concept of two rooks “capturing each other” by representing the chessboard with a matrix. They then reformulate the problem in more mathematical terms that will be conducive to programming, by asking, what is the probability that on an 8 x 8 square grid, two randomly chosen positions are on the same row or the same column? Teachers then work alone or in pairs to write a computer program to determine the experimental probability. Syntax is provided “as needed” by the workshop instructor, but only after teachers see a need for a new tool.

The 3-week workshop operates on the philosophy that a combination of mathematics and programming can accomplish far more together than alone. The daily
approach to learning in the workshop is based around a constructivist theory of learning wherein lecturing is kept to a minimum and syntax is introduced only on a “need to know basis” as need for new tools becomes necessary to program models for more advanced mathematics problems. This pedagogical approach is in direct response to the concern that “[t]he difficulties that students have in elementary computer science studies are often attributed to the need to spend too much time on the syntax of low-level procedural languages like Pascal and C” (Ben-Ari, 2001, p. 55).

Teachers are directly engaged with mathematics problems through programming tasks while a computer is in front of them. The consequences for every line of code that the teachers write are revealed as they test the code on the computer and analyze the results to understand what programming decisions are fruitful and why. This emphasis in interaction between programmer, program, and concept, facilitates the constructivist approach as the teachers generate their own meaning making.

The development of the teacher workshop is a collaborative effort that continues to be adjusted each summer to better meet the needs and interests of the teachers. Modifying a successful curriculum that was developed by a numerical analyst and computer scientist for undergraduate computer science majors originally created the teacher workshop curriculum. Computer scientists and mathematics educators lead the workshops.
Problem Statement

The purpose of this study is to investigate how the workshop effectively prepares and motivates mathematics teachers to integrate computer programming into their mathematics classes.

To evaluate the “effectiveness” of the workshop, this study explores the following three research questions:

1. How does the workshop’s computer programming experience impact the teachers’ content knowledge?
2. How does the workshop experience impact the teachers’ self-efficacy to teach both computer programming and probability?
3. How does the workshop experience impact the teachers’ motivation to integrate computer programming into mathematics curricula?

These criteria for evaluating the workshop were chosen to evaluate “effectiveness” based on the premise that they are “necessary but not sufficient” criteria for translating teachers’ workshop experiences into real results where these teachers are integrating programming into the actual mathematics classes that they teach. Established teachers need to be convinced that computer programming has utility in their mathematics classrooms, will be engaging for their students, and is something they are confident that they can teach. These three criteria can be considered “internal factors” for effectiveness as they all address internal, personal change in the teacher subjects. Internal factors were well suited for this study because they could be capably researched within the research environment that the workshop provided. In order to sufficiently
evaluate effectiveness, “external factors” that could impact the bottom line of having teachers implement programming into their teaching, such as school administrative support, technology support, and time and budget constraints, should also be researched, but are beyond the scope of this study.

Problem solving heuristics using computer programming were introduced in the workshop in large part due to the belief that they could help many teachers feel more confident in their own probability-based problem solving abilities and that they could imagine their own students being affected similarly. In this way, the workshop attempts to model the utility that computer programming can bring to mathematical problem solving and in turn generate the confidence and excitement teachers would need to integrate computer programming into their own mathematics classrooms.

Ultimately, the goal will be to extend this line of research to explore the use of computer programming to other domains of mathematics. Narrowing the focus of this study from mathematical problem solving to problem solving in probability was chosen in hopes of yielding stronger and more conclusive results.

Why probability

Probability was specifically chosen as the branch of mathematical problem solving focus for this study for several reasons:

1. Probability is playing a larger part in the secondary mathematics curriculum since the publication of the NCTM Principles and Standards for School Mathematics (NCTM, 2000), and yet teachers are often uncomfortable teaching it and may not be prepared. Prior mathematics education research
(Peard, 1996) suggests that K-12 mathematics teachers often share many of the same misconceptions on probability as their students. In a 2001 study, Watson found that “[t]here was no indication that the teachers had any particular expertise in probability and statistics” (Watson, 2001, p. 313). Shaughnessy argues that teachers’ confidence toward probability can affect teaching:

“The success of the NCTM’s ambitious standards recommendations will ultimately depend upon teachers. What are elementary and secondary teachers’ conceptions of and attitudes towards stochastics [probability and statistics]? ... We need to gather information from teachers…” (Shaughnessy, 1992, p. 489)

2. Probability reasoning can be counter-intuitive and theoretical results alone are often unconvincing. Results achieved via traditional counting methods or through the development of probability distributions for solving probability problems can be corroborated by accurate, experimental means that use an alternative computational modeling method (the Monte Carlo method) for which computer programming is well suited.

3. The exploration of engaging, real-world probability problems can be too advanced for many high school students using theoretical methods, but can be easily explored by a competent computer programmer using Monte Carlo probability modeling methods.
Theoretical vs. experimental methods

Probability is a mathematical subject that deals with the likelihood that some event will occur. Calculating probability is typically done in one of two ways; either theoretically or experimentally. Theoretical methods involve counting theory and understanding probability distributions, which can be challenging for students and teachers alike. Another way to calculate probability is empirically, with an experimental approximation. This experimental approach approximates the theoretical probability by defining probability as the number of favorable occurrences of an event divided by the total number of occurrences or trials. The law of large numbers can then be used to argue that with a sufficiently large enough number of trials, the probability achieved experimentally will approach the theoretical value. This experimental approach is known as the Monte Carlo method and is well suited for computer use, as the computer is capable of generating the results for millions of trials very quickly.

The Monte Carlo method is highly used in real-world probability problem solving mainly because experimental models are often easier to develop than theoretical ones. In 1946, world-famous Mathematician, Stanislaw Ulam used the first general purpose, electronic computing machine, ENIAC, to carry out computations to test the feasibility of nuclear weapons being developed for the Manhattan Project. While working with ENIAC he realized that the computer could make answering probability questions using simulation techniques very practical.

The idea for what was later called the Monte Carlo method occurred to me when I was playing solitaire during my illness. I noticed that it may be much more
practical to get an idea of the probability of the successful outcome of a solitaire game by … experimenting with the process and merely noticing what proportion comes out successful rather than to try to compute all the combinatorial possibilities which are an exponentially increasing number so great that, except in very elementary cases, there is no way to estimate it. This is intellectually surprising, and if not exactly humiliating, it gives one a feeling of modesty about the limits of rational or traditional thinking. In a sufficiently complicated problem, actual sampling is better than an examination of all the chains of possibilities. (Ulam, 1976, pp. 196-7)

Today, Ulam’s premise that simulation is a necessary component in solving sufficiently complex probabilistic problems is widely accepted. Furthermore, simulation is becoming recognized as an important part of probability education. Consider this recent quote from Shaughnessy, former president of the National Council of Teacher of Mathematics and expert on probability education, in 2013:

The current belief among researchers is that simulation approaches to probability and re-randomization techniques for hypothesis testing help change our students’ thinking about probability and statistics… However, what is needed in our field now are statistical experiments—randomized design studies—that can test the hypothesis that simulation and re-randomization approaches can improve students’ probabilistic and statistical conceptions…There is a growing movement in statistics education to (finally) seriously introduce simulation and re-randomization approaches to statistical decision making, both for secondary
students and intro college stats students. Some of us in statistics education have been promoting this approach for decades, so it is not like this is a new idea. The ICME 12 stats sessions ended with some challenges to the young statistics educators who were in attendance to design some experiments to test out the growing body of evidence that informal inference is a better approach to introduce statistical decision making than appealing to theoretical probability distributions. (Rossman & Shaughnessy, 2013, p. 22)

Despite the important role modeling and simulation play in real-world probability problem solving and the increased attention it has recently received in mathematics education, learning how to use the computer to effectively solve probability problems through modeling and simulation is not being taught at the secondary level.

**Modeling vs. simulation**

It is important to distinguish the difference between the term *modeling* and the term *simulating* as these two terms are often erroneously used synonymously. While the concepts are closely related, they are not the same.

*Modeling* is the act of creating a model. A model is a construct that represents some system and is intended to approximate the system as closely as possible, while at the same time remaining simple and versatile enough to be manipulatable; an important feature that is often not doable or practical with the system itself, hence the need for a model. A simulation, however, is simply the operation of a model, with the term *simulating* meaning the iteration of a simulation (Maria, 1997). In simulating, a user changes input variables to a model and uses the output that results to make inferences
about the solution to a problem that the system described, but by creating a model the user is developing the heuristics for solving the problem.

Simulating is recognized as a valuable learning tool and is used in many academic disciplines. When presented as a pedagogical practice however, it can have multiple meanings. Sometimes simulation as a pedagogical practice refers to one of the following:

1. The practice of creating a model and then running simulations on that model. In this pedagogical approach, users will advance their understanding of the system by progressively improving the model’s representativeness of the system based on the results obtained through simulating.

2. The practice of running simulations on a pre-created model. In this pedagogical approach, it is assumed that simulating is being done on a pre-created model that is already believed to be an appropriate representation of the system. Users will often learn in a black-box setting, meaning that they have no control over changing the model and do not have any understanding of how the model works, in other words, the model is a mysterious black-box. Users will advance their understanding by comparing the relationship to the only thing that they control, input to the model, with the output from the model generated by the specific input.

The second pedagogical approach to learning through simulating, where the model is a black-box, can be a valuable practice for problem solving, however it is not the type of simulating that Stanislaw Ulam called for and is not the pedagogical approach employed in the teacher’s workshop.
Both approaches will allow one to answer a probability problem, but the black-box approach relies on the use of a pre-constructed model already existing, being accessible, and being a suitable fit for solving the probability problem at hand. In contrast, only the approach where learners are engaged in model creation and modification, trains learners to tackle novel problems.

*Programming vs. statistical-software*

Teacher participants in the study’s workshop used computer programming to create models for solving probability problems. Teachers programmed in the C programming language, which is a widely used, general-purpose programming language, and one that their secondary students may likely encounter if they take an introductory college computer science class.

Using the modeling approach to create models using a general-purpose programming language, shows the teachers that probability problems can be solved without depending on knowing or identifying theoretical probability distributions. In the workshop, the programming tools that are used to solve probability problems are never more complex than basic loops, conditional statements, arrays, and a general-use, uniformly distributed, random number generator.

One might ask, what is the value of using a general-purpose programming language to teach modeling in probability, over powerful, statistical programming languages such as R, Minitab, or Fathom? (R, 2012; Minitab, 2010; Fathom, 2015). The answer lies in the type of background understanding in probability that general-purpose
programmers will need versus those who choose to program with statistics tools that are
found in statistical software.

Those who reach for statistical software and/or programming languages to model
probability problems, often do so for access to pre-made, statistical tool-sets that allow
them to program their models using complex theoretical probability distributions.
Programmers that use these tools require a great deal of training in theoretical probability
to know when, where and how to use these tools in their programs. This defeats the
purpose of the workshop, which is to warm teachers to the idea that they can empower
their secondary mathematics students to solve interesting, real-world probability
problems without having to first have a strong background in theoretical probability.

It is possible to use statistical software languages without using theoretical
probability tool-sets, in which case, programs written in one of those languages would be
only superficially different from a program written in C, as they would only differ
syntactically. In this case, the choice to use C or a statistical software language would
not impact the goals of the workshop and so the instructor can feel free to use whichever
language they prefer. It is worth noting however that C is a compiled language that will
almost certainly run a large number of simulation iterations far faster than an interpreted
statistical software language like R. Large numbers of simulation iterations are important
for obtaining accurate experimental probabilities through simulation. The workshop,
therefore, offers an innovative approach to teaching and learning problem solving in
probability, through the integration of general-purpose programming.
Plan

A three-week workshop course was developed to contribute to the advancement of computer programming integration into secondary level mathematics curricula, by attempting to train and motivate current secondary level mathematics teachers in the use of computer programming for mathematical problem solving. This experimental workshop was studied using a mixed methods approach to begin investigating how this workshop effectively trains secondary mathematics teachers in new mathematical problem solving strategies and stimulates their motivation and confidence levels in integrating computer programming into their own secondary level mathematics courses.
Five questions will be explored through review of relevant literature:

1. What does the literature suggest about the challenges to learning probability?
2. What does the literature suggest about teachers’ preparedness to teach probability?
3. What does the literature suggest about how tradeoffs between a technologies’ ease-of-use and its learning benefits impact the integration of computer programming into mathematics instruction?
4. What does literature suggest about training teachers to teach computer programming?
5. What do the Common Core State Standards say about probability education and computer programming?

Challenges to Learning Probability

Probability is a subdomain of mathematics that is often challenging for learners to understand, at even a fundamental level, as learners need to reason in ways that may be counter-intuitive and in conflict with their world-view. The research team of Garfield and Ahlgren suggest that there are three primary reasons why learners struggle to develop an understanding of the fundamental concepts in probability:
… students appear to have difficulties developing correct intuition about fundamental ideas of probability for at least three reasons. First, many students have an underlying difficulty with rational number concepts and proportional reasoning, which are used in calculating, reporting, and interpreting probabilities… Second, probability ideas often appear to conflict with students’ experiences and how they view the world… [and] Third, many students have already developed a distaste for probability through having been exposed to its study in a highly abstract and formal way. (Garfield & Ahlgren, 1988, p. 47)

Garfield and Ahlgren recommend that to overcome these difficulties, instruction should be grounded in activities and simulations, and not in abstraction. Instruction should “create situations [for learners] requiring probabilistic reasoning that correspond to the students’ view of the world” (Garfield & Ahlgren, 1988, p. 48).

Konold, Polltsek, Well, Lohmeier, and Lipson support task-based learning that gets learners engaged in reasoning that will evoke commonly held misconceptions of probability concepts, saying that “one way to produce conceptual change is to create situations for which the answers based on a particular incorrect intuition produce cognitive conflict” (Konold, Polltsek, Well, Lohmeier, & Lipson, 1993).

Consider the following three examples of problems that researchers used to identify commonly held misconceptions in basic probabilistic reasoning.

Konold et al. (1993) used the four-heads problem to highlight a misconception that learners may hold about the meaning of an independent variable. The problem states:
A fair coin is flipped four times, each time landing with heads up. What is the most likely outcome if the coin is flipped a fifth time?

a) Another heads is more likely than a tails

b) A tails is more likely than another heads

c) The outcomes (heads and tails) are equally likely.

(Konold, et al., 1993, p. 397)

In a research study of 25 undergraduate students enrolled in remedial mathematics at the University of Massachusetts, Konold et al. (1993) discovered that 30% of students’ reasoning led to an incorrect response. Of these 25 students, 22% demonstrated a common misconception known as the gambler's fallacy where students who know that the probability of either a heads or tails is $\frac{1}{2}$ believe that because heads has been flipped 4 times already, tails is now “due to come up.” Students that hold this misconception may not understand that flipping a coin is an independent event or may find their understanding of independent events at conflict with their intuitive belief that tails must be “due” now in order to move towards an experimental probability that will approach the theoretical probability of $\frac{1}{2}$.

In a second example Fischbein and Schnarch (1997) studied the effects that age had on probabilistic misconceptions. While their final results showed that some probability misconceptions decreased and some increased as learners aged, the following question showed that a high percentage of learners from grades 5, 7, 9, 11 and college continued to be confounded by the misconception common to the following problem. Their example states:
Suppose one rolls two dice simultaneously. Which of the following has a greater chance of happening?

a) Getting the pair 5-6
b) Getting the pair 6-6
c) Both have the same chance  (Fischbein & Schnarch, 1997, p.98)

The common misconception is that both have the same chance. Fischbein and Schnarch’s study involved 20 subjects from each of the grade levels 5, 7, 9, and college and 70%, 70%, 65%, 75%, and 78% of subjects at each respective grade levels answered that both the pairs 5-6 and 6-6 have the same chance. Learners that hold this misconception may be confusing the ordered pairs (5, 6) and (6, 5), which are not equivalent, with the set \{5, 6\}, which is equivalent to the set \{6, 5\}. This misconception may lead learners to erroneously assign equal likelihood to both the \{5, 6\} and \{6, 6\} sets.

The third example is from a question in the literature first suggested by Shaughnessy (1992) and then modified and researched by Peard (1996). The following question highlights the misconception associated with “…the assumption of equal likelihood when none exists” (Peard, 1996, p.441).

The spinner shown is spun once. What is the probability that yellow will show?

Explain your reasoning.

(Peard, 1996, p.441)
Of the fifty pre-service elementary school teachers studied in Peard’s research, thirty six gave an answer of 1/2 or 2/4, thirteen gave an answer 1/3, and one gave an answer of 2/3. Answers to this problem revealed that while many adult teachers are able to reason about probability numerically, a high proportion do not have a thorough understanding that unequal likelihood of outcomes will impact the theoretical probability of an event.

Teachers’ Preparedness to Teach Probability

Multiple research studies suggest that many teachers struggle to understand basic probability concepts. A study by Begg and Edwards (1999) showed that approximately 66% of elementary school teachers understood the concept of equally likely events and that less understood independent events. Another study, conducted by Carnell (1997), investigated 13 pre-service middle school teachers’ understanding of conditional probability. Carnell found that each of the 13 pre-service teachers held at least one misconception that involved confusion between conditionality and causality (Carnell, 1997). In a 2001 study in Australia, Watson conducted a study of 15 elementary teachers and 28 secondary teachers and showed that the secondary teachers were significantly more confident than elementary teachers in basic probability measurement. However, these secondary teachers still struggled to understand other concepts in probability. The study revealed that secondary teachers struggled to interpret ratios as odds or as part-whole probability (Watson, 2001).

In 1996 Robert Peard studied five specific misconceptions that pre-service primary teacher education students had on basic probability concepts. Peard identified a
set of 12 questions, modified from research literature on probability, which he administered to 50 year-two B.Ed. students at Queensland University. An analysis of these questions revealed the following errors in probabilistic reasoning:

- the false assumption of equal likelihood;
- independence of events;
- awareness of counter-intuitive probabilities;
- belief in other fallacies

(Peard, 1996, p.437-438)

The data showed that the assumption of equal likelihood when there was none, occurred in 36% of the responses for Peard’s question 6: *If I throw a pair of fair dice is it harder to get a pair of sixes than it is to get a five and a six? Why or why not* (Peard, 1996, p.441) and 46% for question 12:

*A deck of 21 green and yellow cards are shuffled and dealt twice.*

*The following sequences are observed:*

**A:** YYYGGYYGGYYGGGGGYYY

**B:** GGYGYGYYGYGYYGYGGGYGY

*Which sequence would represent the better shuffled (more random) deck?*

*A, B, or are both equally random? Why?* (Peard, 1996, p. 442)

An analysis of how mathematically rigorous the quality of a response was showed that in only 17% of all responses was the quality sufficient enough to demonstrate understanding of probability principles via a quantitative explanation. “A full 60% of all
responses were made without any evidence of analysis or with an incorrect use of probability principles” (Peard, 1996). Peard concludes by noting that all of the 50 students had experienced some sort of formal education in probability and suggests that more education on probability should be required in their pre-service education.

Ease-Of-Use vs. Learning Benefits of Technology and Programming

The teaching and learning of mathematics is an ever-evolving process in classrooms throughout the nation. In the modern, digital age, the use of technology has been near the forefront of this evolution. The National Council of Teachers of Mathematics has identified technology as an essential tool in the 21st century mathematics classroom for helping to make mathematics accessible for all students (Technology in Teaching and Learning Mathematics, 2011). When coupled with expert guidance from mathematics teachers, technology allows students to “support and extend mathematical reasoning and sense making, gain access to mathematical content and problem-solving contexts, and enhance computational fluency” (The Role of Technology in the Teaching and Learning of Mathematics, 2008, p. 1). However, when it comes to programming technology as a means to enhance mathematical thinking, teachers may often be skeptical that the value added to the exploration of mathematics will be worth the time investment and expected difficulty of learning to computer program.

In order to maximize the effectiveness of computer programming and incentivize mathematics teachers to integrate it into their classrooms, programming must be both simple enough for students to use effectively and robust enough to engage students in in-depth learning. Ironically, simplicity and robustness seem often to exist in diametric
opposition, as the easier technology is to use (or program), often the less versatile and powerful it is.

This section of the literature review will explore the tradeoffs between ease-of-use and in-depth learning as they manifest in simulation/modeling software versus general-purpose programming; both promising approaches for using technology to teach and learn probability through computer modeling. The studies presented on simulation software tools go beyond the exploration of ease-of-use versus in-depth learning by also examining the effectiveness of programming with simulation software for learning concepts in probability. No studies were found that use a general-purpose programming language, in conjunction with a de-emphasis on specialized stochastic tools, for enhancing the ability to solve problems in probability and so the research presented on general-purpose programming will only focus on ease-of-use versus in-depth learning and a general applicability to learning mathematics.

Simulation/modeling software

Several software packages exist that let users model and/or simulate probability problems. Two of the more popular packages used in K-12 classrooms are Fathom and TinkerPlots. At the University of Kassel, in Germany, Fathom is an important tool for a required introductory course on probability and statistics for future mathematics teachers in the department of mathematics and informatics. In this course, simulation techniques are developed in tandem with the exploration of probability concepts. Students model probability problems both mathematically and by simulation via the Fathom software (Maxara & Biehler, 2006). The results of both approaches are then analyzed against each
other in hopes of supporting a greater understanding of probability. The course creators, Maxara and Biehler, are committed to Fathom as a simulation tool in the introductory course, noting that:

“The software Fathom achieves best our criteria for a software tool that supports both the learning and the use of probability and statistics in problem solving…” (Maxara & Biehler, 2006, p. 1).

However, Maxara and Biehler identify that students do have difficulty using Fathom due to the nature of the software. The course introduces three different versions of simulation for modeling that Fathom offers; a simultaneous simulation, a sequential simulation, and simulation with sampling. To visualize these three different versions, imagine flipping fair coins. Simultaneous simulation would be appropriate for flipping two coins at the same time. Sequential simulation would be appropriate for flipping a single coin once and then flipping it again in a sequence. Simulation with sampling would be getting data on coin flips by randomly sampling from a collection of data on coin flips. Maxara and Biehler discovered that students struggle to identify the correct Fathom simulation version to apply to properly simulate a given probability problem. The researchers also note that when using Fathom, “[d]epending on the type of simulation, some events or random variables can be defined only in a complicated way or they cannot be defined at all” (Maxara & Biehler, 2006, p. 2). Generalizing and/or modifying Fathom simulation models also proved to be problematic for students as it is sometimes not feasible, depending on the simulation version being used. Maxara and Biehler give the following example:
… the waiting time until the first occurrence of a “T” [a coin flip resulting in tails] can be easily defined by little modifications of the sampling-based simulation but not at all with the other two types. (Maxara & Biehler, 2006, p. 2)

*Fathom* uses a construct called a *measure* that also can be difficult for students to understand and use for modeling probability, because a *measure* is limited in the way that it can represent some probability concepts. For example, a *measure* can be used to represent an *event* as a true or false Boolean expression of some occurrence, but it cannot represent an *event* as a subset of a sample space (Maxara & Biehler, 2006).

After running the course for the first time, Maxara and Biehler conducted interviews with students that revealed that in-depth understanding of probability remained limited. The researchers found those students’ knowledge and use of language somewhat improved with regard to simulations in *Fathom* but the improvement was unsatisfactory with regard to understanding probability concepts (Maxara & Biehler, 2006). The researchers concluded that probability misconceptions were more prevalent than expected after the course and that while the simulations worked well for showing the fallacy of misconceptions within a given context, the simulations alone were not enough to eliminate the misconception entirely. To address these issues, Maxara and Biehler changed their tasks to give more exploratory emphasis to probability concepts and to more often compare simulation solutions with theoretical solutions. To address misconceptions better, tasks were modified to address misconceptions before, during, and after the simulation tasks by comparing the reasoning behind the misconceptions to the reasoning behind the theory and simulations. In the end Maxara and Biehler’s study
suggests that it is not the simulation itself that improves probabilistic reasoning, but the discussion that simulation stimulates that well-prepares students’ minds to let go of the intuitive thinking that leads to misconception and consider more correct ways of probabilistic reasoning.

The software *TinkerPlots* is also a popular probability simulation-modeling tool, but is more likely to cater to a younger audience than *Fathom*, and is often recommended for use at the middle school grade levels. Researchers Kazak and Konold (2010) studied how specially designed instructional tasks might use *TinkerPlots* to help middle school students’ better understand the concepts of chance events using simulation. In introductory lessons on probability, at the middle school level, it is a common task for students to predict and then explain the outcomes of data collected via physical simulation. Kazak and Konold note that physical simulations generate cognitive conflict for students that can be used as the impetus for deeper exploration:

Having noticed a misfit between the actual data and their expectations, students typically begin to question their initial expectation. This motivates the need to collect more data and, in turn, the need to using the simulation capabilities of the computer. (Kazak & Konold, 2010, p. 5)

Kazak and Konold found *TinkerPlots* to be a valuable tool for extending physical simulation and visualizing the results of the simulation upon large numbers of trials. Kazak and Konold say that “[a]nalyzing the situation with more data spurs students to develop new conjectures about the situation” (Kazak & Konold, 2010, p. 5).
By using the large volume of data that *TinkerPlots* simulations provide, the researchers were able to introduce students to concepts of sample space and the law of large numbers through an in-depth empirical analysis.

We introduce the sample space as both an explanation for the results they see as well as the basis for predicting what they will see as they collect more data… students tend to focus on the expected distribution based on the sample space and the simulation data. With this new perception [the *Tinkerplots* simulation] they come to see these deviations as ‘noise.’ By varying the sample size… students can observe the noise increasing or decreasing. This helps them develop a general understanding of the Law of Large Numbers… In these investigations, the computer tool plays a central role. (Kazak & Konold, 2010, p. 5)

The literature does not clearly suggest why Kazak and Konold were more successful than Maxara and Biehler in using their respective simulation tools for effective in-depth learning of probability and simulation concepts. However, it is possible that Kazak and Konold reported greater success because their study integrated technology use into higher-quality instructional tasks that were better aligned to clearer and simpler learning objectives.

*General-purpose programming tools*

Few people would argue against the computer being a transformational educational tool for how we learn about the world today. Computers are essential technology in virtually all scientific fields of study and for it to be a useful tool a program is required that shapes the computer for the necessary task. However, despite the value
of computer programs to the in-depth learning at the cutting edge of science, the impetus for computer programming integration in the K-12 classroom seemed to peak and then quickly fade with the introduction of the personal computer in the 1980s (Resnick et al., 2009). During this time of programming enthusiasm, languages like Basic and Logo were used by thousands of schools to teach students how to program. The Logo programming language was even designed by Seymour Papert, specifically to teach computer programming concepts to children. Why then, has teaching children to program lost favor in schools when the computer itself has become a critical resource in the lives of many children? Resnick et al. (2009) identified multiple factors for the decline of programming in the K-12 curricula:

1) Early programming languages were too difficult to use, and many children simply couldn’t master the syntax of programming.

2) Programming was often introduced with activities … that were not connected to young people’s interests or experiences.

3) Programming was often introduced in contexts where no one could provide guidance when things went wrong – or encourage deeper explorations when things went right. (Resnick et al., 2009, p. 63)

In 2006, a new programming language called Scratch was developed to address these factors that may have contributed to the decline of programming inclusion. Scratch is a programming language built to be intuitive to learn and robust to use. A block interface is used to help users see how common programming control structures logically build a program and to eliminate the need for tricky syntax that is often a barrier of entry.
for new users. The interpreted environment of Scratch means that users need only to click on a top stack of blocks for code to immediately begin executing. Scratch also allows users to change the stack of blocks during program execution. Combining this interpreted environment with the easy ability to swap blocks in and out of a stack allows new users to experiment without the need for meticulous planning. This “tinkerers” approach to coding can make Scratch easier and faster to learn and use.

The research team of Maloney, Peppler, Kafai, Resnick, and Rusk (2008) performed a research study to empirically support the claim that Scratch was both easy to use and suitable for teaching children how to program a computer. The study was conducted at an after-school center known as the Clubhouse and it analyzed the Scratch programming projects of children ranging from ages 8 to 18.

Maloney et al. collected data for 18 months to “track the extent to which programming concepts were taking root in the Clubhouse culture over time” (Maloney et al., 2008, p. 3). Out of the 536 projects created by Clubhouse children during this data collection process, 425 projects were scripted, meaning that they were comprised of a sequence of commands that were sequentially executed by the interpreter. These 425 projects were the focus of the researchers as they were the projects that could be analyzed for the use of programming concepts. Below are the programming concepts that Maloney et al. searched for and the percentage of the 425 scripted projects that used the concept:

- User Interaction -- 53.6%
- Loops -- 51.8%
- Conditional Statements -- 26.1%
Communications and Synch. -- 24.7%

Boolean Logic -- 10.8%

Variables -- 9.6%

Random Numbers -- 4.7%

(Maloney et al., 2008, p. 4)

The heavy use of user interaction and loops was unsurprising to the researchers who noted that video games and animation projects were extremely popular with Clubhouse members and were necessary programming constructs in these types of projects. In order to program simulations of most probability problems, learners may not need much of an understanding of user interaction or communications and synch (inter-object communication), but will need to use all of the other programming concepts.

The importance of random number generation to probabilistic simulation and the results from this data, make a compelling case that programming simulations may not be easily intuited by learners. The researchers found that; “… Boolean operations, variables, and random numbers are concepts that are not easily discovered on one’s own” (Maloney et al., 2008, p. 3). What this research does not make clear is how often the low usage of these concepts was due to a lack of conceptual understanding, a lack of understanding of Scratch constructs, some mixture of both, or a lack of need for these concepts in the projects students chose to pursue. However, researchers did have encouraging encounters with Clubhouse members indicating that with some minimal guidance, students could quickly assimilate these concepts and incorporate them into their code:
When Mitchel Resnick, on a visit to the Clubhouse, showed him [a struggling Clubhouse member] how to use variables, he immediately saw how they could be used to solve his problems and thanked Mitchel repeatedly for the advice. (Maloney et al., 2008, p. 3)

Effective Training for Teaching Computer Programming

*Challenges to effective teacher preparation*

This section of the literature review will explore the difficulties that teacher preparation programs face in preparing teachers in reform-based instruction for both mathematics and computer science education. An instructional approach known as Cognitively Guided Instruction (CGI) will be reviewed as a promising pedagogy for producing well-prepared teachers.

The tenacity with which teachers hold on to early formed beliefs about mathematics instruction is a significant obstacle to producing well-prepared new teachers in reform based pedagogy. These early beliefs take shape before they ever begin teacher preparation training. Often times these early beliefs continue to shape teachers’ instructional practices even after being trained in more modern, reform-based, classroom practices. The research of Benken and Wilson highlights this problem well.

Benken and Wilson (1998) conducted an ethnographic study of a preservice secondary teacher named Leslie, focused around her beliefs about mathematical instruction. The researchers used three different categories, *Problem-Solving*, *Platonist*, and *Instrumentalist*, to characterize a person’s view of mathematical instruction. Only the problem-solving category promotes a philosophy of instruction that aligns with
NCTM (1989) and mathematical reform recommendations that are more often becoming the standard in mathematics classrooms across the country. Of the Problem-Solving view, Benken and Wilson say:

According to the Problem-Solving view, mathematics is seen as a continually growing field of human creation that develops through conjectures, the generation of patterns, proofs and questioning. The results of mathematics remain open to revision. Teachers with Problem-Solving view are likely to think of themselves as facilitators who allow students opportunities to explore, generate and solve problems, and construct their own understandings. Problem-Solving teachers approach teaching in ways that stimulate classroom discourse, encourage student investigation and cooperative learning, and create an environment necessary for students to make connections within mathematics and to the world around them (Benken & Wilson, 1998, p. 4).

Work with Leslie revealed early on that her view on mathematics and mathematics instruction were more consistent with that of a Platonist. She believed “that mathematics was sequentially ordered and that doing mathematics involved applying steps in order and mastering” (Benken & Wilson, 1998, p. 5). Algorithmic-drill instruction was at the core of her teaching philosophy and Leslie saw her role as that of an explainer of information. After being exposed to and trained in mathematics reform philosophies, Leslie explained that she now thought that “meaning is more important than formulas of answers” (Benken & Wilson, 1998, p. 5) and espoused a philosophy of a
more conceptual and interconnected view of mathematics that was more consistent with
the mathematical reform based philosophies she was learning.

Despite receiving training in reform based instructional practices and expressing a
desire to teach in ways that were compatible with that philosophy, Leslie struggled during
her student teaching to involve students in reform based practices. Leslie felt compelled
to drill students on algorithms before allowing them to investigate and explore and failed
to incorporate student observations and thought processes in reviewing subject matter.
Leslie’s lessons were very teacher-centric unlike the student-centric emphasis valued in
reform-based teaching. Leslie’s tendency to let deep-rooted, previously held beliefs
about teaching influence her instructional practices, even after receiving contrary
training, is a well-documented phenomenon in preservice teachers and has been
corroborated by the research of Beswick (2006), and Biza, Nardi, and Zachariades
(2007).

Since the 1989 mathematics reform recommendations by NCTM, the desire for
student-centric pedagogy has started to become the classroom standard. In these
classrooms, lessons are expected to be designed around students as active participants in
their own learning. However, research by Alice Artzt (1999) suggests that teacher-
centric instructional practices may be an inevitable starting point in a new teacher’s
developmental process. Artzt theorizes that teacher development may require maturation
through a set of stages where the first stage is grounded not in student-centric, but rather
teacher-centric instructional practices. Artzt describes new teachers in the first stage of
development as being “driven by the belief that students learn best by receiving clear information transmitted by a knowledgeable teacher” (Artzt, 1999, p. 143-144).

Artzt’s (1999) research used this theory to explore the impact that pre and post lesson reflection would have on transitioning student teachers to higher stages of development, where the shift on teaching would become more student-centric. These stages would be characterized by “helping students build on what they understand,” by helping students “take greater responsibility for their own learning…[and by giving students opportunities that will engage them in]…constructing deep and full mathematical understanding” (Artzt, 1999, p.144). Artzt confirmed that indeed student teachers did naturally implement very teacher-centric lessons that were not consistent with reform based instruction, but that thorough pre and post lesson reflection that engages student teachers in sharing, reexamining, and questioning their knowledge and beliefs, they could progress out of the teacher-centric stage.

Offering more effective preparation through CGI

In order to address the aforementioned challenges to teacher preparation, teacher training programs need to promote instructional strategy that will:

1. Be strongly impactful in reshaping deeply ingrained and teacher-centric teaching beliefs
2. Offer clear benefits for the mathematical achievement of students
3. Endure in the face of curricular and political change in the public school mathematics classroom
In the 1990s the research group spearheaded by Carpenter, Peterson, and Fennema developed and studied a type of instructional pedagogy called *Cognitively Guided Instruction* (CGI) that offers these keys attributes. CGI is an approach to teaching that uses inquiry methods and problem solving situations to engage students in mathematics in a way that allows teachers to understand how students conceptualize mathematics and reason mathematically (Fennema et al., 1996). It gives students a platform to explore and consider multiple types of conceptual models for mathematical ideas and to reason collaboratively as they share their problem solving ideas. While many teachers without CGI training do have some knowledge of how their students think, CGI training helps teachers to sharpen and refine that knowledge and most importantly, make it the focal point of their instructional practices:

… before teachers participate in CGI programs, many have informal, although somewhat unfocused, knowledge about children’s mathematical thinking. They recognize some distinctions among problems and can identify many of the primary strategies often used by children, but seldom do they relate critical problem dimensions to children’s solutions or problem difficulty, nor does their knowledge of children’s thinking play a critical role in planning instruction… Our goal was to help the teachers build on and refocus their informal knowledge so that children’s thinking would become central to their thinking about instruction (Fennema et al., 1996, p. 406 – 407).

CGI has the benefit of being a reform-based approach to teaching that has been extensively studied. Research studies have shown that CGI is a teaching strategy that is strongly impactful in reshaping deeply ingrained, teacher-centric teaching beliefs.
Fennema et al. (1996) conducted a longitudinal study over four years on 21 primary school teachers participating in a CGI teacher development program. Eighteen teachers changed their beliefs and instructional practices dramatically. Instruction evolved from modeling mathematical procedures to guiding children in conversation and mathematical knowledge sharing as they engaged in mathematical problem-solving tasks. Other studies corroborated the impact CGI training had on shifting teacher beliefs and practices to being more reform oriented and reporting that the teachers’ mathematical knowledge increased (Simon & Schifter, 1991; Schifter & Fosnot, 1993; Simon, 1995).

Results from the research study by Fennema et al. (1996) also strongly supported the conclusion that CGI offers marked benefit in the teachers’ students’ mathematical achievement. The study compared student achievement in addition and subtraction in the classrooms of the 21 CGI teachers with achievement in addition and subtraction in the classrooms of a control group of teachers not trained in CGI. The results were staggering as overall student improvement in mathematical conceptual understanding and problem solving ability was higher in each of the 21 CGI teachers’ classrooms as compared to the control group. Even though CGI is well suited to concept and problem solving development, the study also found that overall student improvement in addition and subtraction computational abilities were consistent between the CGI and control groups.

The findings suggest that developing an understanding of children's mathematical thinking can be a productive basis for helping teachers to make the fundamental changes called for in current reform recommendations. (Fennema et al., 1996, p. 403)
In order for CGI to endure in the face of curricular change, it also needs to be versatile. CGI’s versatility is a product of it being a student-centric approach that promotes student and teacher reflection, conceptual growth, and an evolving understanding of how students think that is both generalizable and specific down to the individual student level. By using CGI, it will not matter how curriculum changes influence the knowledge and concepts that students arrive with, because teachers use their evolving understanding of student knowledge to design their future lesson plans. The CGI teacher is trained to elicit student thought as the class lesson unfolds and make the appropriate adjustments to meet students at whatever their current level of mathematical understanding may be. This dynamic approach to lesson planning that is grounded in whatever students’ knowledge and understanding may currently be, should be versatile enough to withstand any future changes to the political climate of the public school mathematics classroom.

The longitudinal nature of the research by Fennema et al. (1996) lends further support to the belief that CGI will have staying power. Four years after being introduced to CGI training, it was still the key teaching practice in many of the teachers’ classrooms: By the end of the [four year] study, the instruction of 90% of the teachers had become more cognitively guided ... This kind of instruction epitomizes the process standards of the reform movement (NCTM, 1989, 1991). The students spent most of their time engaged in true problem solving. Teachers understood and appreciated the variety of solutions that children constructed, and they understood the power of children’s communication about their thinking.
Seventeen of the 21 teachers came to believe more strongly that children could solve problems without being shown procedures for solving them… This belief changed their perception of their role as teachers. They came to believe that their role was not to tell children how to think, but to provide an environment in which children’s knowledge could develop as the children engaged in problem solving experiences and reported on solution strategies. (Fennema et al., 1996, p. 429)

Training teachers in computer science

There is a drastic lack of computer science teacher training programs in the United States. Over half of the States have no state teacher certification nor are there teacher training courses for computer science (Gal-Ezer & Stephenson, 2009). The Association for Computing Machinery (ACM) is “…the world’s largest educational and scientific computing society” and through its Computer Science Teacher Association (CSTA) developed standards for computer science education. One major emphasis in teaching computer science, and particularly computer programming, is the emphasis on problem solving (CSTA K-12 Computer Science Standards, 2011).

The summer programming workshop heeds the call of CSTA by emphasizing mathematical problem solving and providing needed teacher training in programming. The development of the summer workshop is supported by Gal-Ezer and Stephenson’s study that showed that “…[computer science] teachers continue to rank workshop and seminars as the most effective means for delivering professional development to [computer science] teachers” (Gal-Ezer & Stephenson, 2009, p. 4). Teacher development workshops were ranked the most effective while introductory college courses were
ranked least effective with networking, professional conferences, and online resources ranked second, third and fourth respectively.

The summer workshop followed a constructivist approach that was heavily influenced by CGI pedagogy. Constructivism has long been studied and advocated in mathematics education and can be simplistically defined as students actively constructing their own knowledge instead of passively learning from lectures and texts (Noddings, et.al. 1990; Ernest, 1994). As computer science education is a relatively new field, it has been guided by research in mathematics education (Ben-Ari, 2001). While the use of reform-based pedagogy has been a cornerstone of mathematics education for over twenty years, its use in computer science education is less established. However, scant literature exists to support reform-based pedagogy for the teaching and learning of computer science and, by extension, computer programming. Ben-Ari (2001) advocates for the use of constructivism in computer science education, and Armoni (2011) refines this notion by stating that “…CS teacher preparation programs should be rooted in the constructivist point of view” (Armoni, 2011, p. 23).

The workshop adopts a constructivist theory of learning and therefore keeps lecture to a minimum and introduces syntax only on a “need to know basis.” This pedagogical approach directly responds to the concern that introductory computer science courses spend too much time on programming syntax: “The difficulties that students have in elementary computer science studies are often attributed to the need to spend too much time on the syntax of low-level procedural languages like Pascal and C” (Ben-Ari, 2001, p. 55).
Probability and Computer Programming in Common Core State Standards

With the wide-spread adoption of the Common Core State Standards in Mathematics (National Governors, 2010), greater expectations have been placed on schools in the United States to develop students’ understanding of statistics and probability than ever before. It is no longer acceptable that probability education be woefully underserved in middle school and high school mathematics classrooms.

Common Core State Standards introduce statistics and probability to mathematics curricula beginning in middle school. Although the Common Core Standards introduce statistics in the 6th grade, probability is not a topic of focus until the 7th grade.

By the conclusion of the 7th grade, students are expected to understand probability through the exploration of “chance processes” (National Governors, 2010, p.47) with a number between 0 and 1 representing the likelihood that some event will occur. Students will develop probability models and test those models against frequencies generated by physical simulations. Students will develop an understanding of sample space and begin exploring compound events as some subset of outcomes contained within an appropriate sample space that describes all outcomes. Therefore, students develop an understanding that the probability of a compound event will be the fractional ratio of desired outcomes to the size of the sample space.

However, in the 8th grade standards, once again the Common Core minimizes the curricular emphasis on probability in the statistics standards and focuses instead on exploring “patterns of association in bivariate data” ((National Governors, 2010, p. 56)
through creating and analyzing data in scatter plots and developing linear equations that model data trends.

High school Common Core State Standards in mathematics further outline expectations for the advancement of students’ understanding of probability, however, standards are no longer delineated by grade level and are instead characterized only as high school level expectations. In high school, students are expected to develop an understanding of the vital role that randomization plays in making statistical inferences and the ability to make conclusions about an entire population through random sampling. In addition, students should understand that probability models allow for mathematical description of processes that are generated through randomization.

High school Common Core State Standards of probability expect mathematics curricula to develop in students an understanding of independent/dependent events and conditional probability, both conceptually as well as symbolically. For example, students should understand that the probability of an event A can be impacted by whether or not some other event B occurs and, if so, describe this conditional probability of A given B symbolically as \( P(A \text{ and } B)/P(B) \). Students are also expected to understand the addition and multiplication rules; \( P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B) \) and \( P(A \text{ and } B) = P(A) \cdot P(B|A) = P(B) \cdot P(A|B) \), respectively. Students should be able to recognize situations in which these rules will aide in computing probabilities of compound events and be able to apply them. Students are also expected to be exposed to combinations and permutations, understand how these concepts are used in probability problem solving, and be able to apply their formulas in computation. Lastly, students are expected to learn
how to use probability to aide in real-world decision making processes and how to perform basic experimental research using probability and statistics.

Common Core Standards in Mathematics have admirably created expectations for probability and statistics education to be included in middle school and high school mathematics curricula. Common Core Standards in Mathematics also note the importance of using technology in solving probability and statistics problems, saying:

Technology plays an important role in statistics and probability by making it possible to generate plots, regression functions, and correlation coefficients, and to simulate many possible outcomes in a short amount of time (National Governors, 2010, p.79).

However, despite acknowledging the importance of technology in statistics and probability, not to mention its potential importance in other mathematical domains, the Common Core State Standards disappointingly fail to include computer science and computer programming education anywhere in the standards.

Ignoring the role of computer programming in probability and statistics problem solving may lead one to question if Common Core State Standards do enough to prepare K-12 students for a future in science and mathematics. Considering the enormous, synergistic roles that computer programming, probability, and statistics play in the real-world study and advancement of science and applied mathematics, at the college and beyond, the omission of any computer science education in Common Core State Standards seem extremely shortsighted.
Summary

It has been well established that simulation plays an important role in solving probability problems. The power of the computer to perform simulations on a large scale makes it too important of a tool to ignore in any probabilistic endeavor in the mathematics classroom.

The major contributions of technology to the teaching and learning of probability concern simulation and drawing on the capacity of technological environments to simulate random events in various ways. Simulation allows students to experience random events first hand and to facilitate understanding of key probability concepts (Kissane & Kemp, 2010, p. 4).

What is simply not clear at this time is what kind of technology tools best engage students in simulation practices that are conducive to learning probability concepts. As with all tools, the manner in which you use them will impact their effectiveness. The research results from the studies discussed in this report represent some of the best literature in this field, but do not provide a means for directly comparing modeling and simulation technologies to one another nor for directly comparing the effectiveness of differing pedagogical approaches that use computer programming to teach probability. As Kissane and Kemp point out, research on the effects of technology on probability education is still in its infancy:

The relationships between technology of various kinds and probability education have been surprisingly neglected in the research literature. In particular, a search of recent ACM conferences revealed little work at all in this
area, so that at this stage it is premature to summarize empirical evaluations of the
effects of using technology for probability instruction. (Kissane & Kemp, 2010, p. 1)
Chapter 3: Design Description

This study investigated the effectiveness of a three-week summer workshop course, offered at a large, Pacific Northwest public university, in preparing teachers to integrate computer programming into their own mathematics courses. This study evaluated the workshop’s effectiveness in teacher preparation by examining the workshop’s impact on teachers’ problem solving ability in probability and their confidence and motivation levels for implementing computer programming in their own mathematics courses. To evaluate the effectiveness of the workshop, this study explores the workshop’s impact on the teachers through three research questions:

1. How does the workshop’s computer programming experience impact the teachers’ content knowledge?
2. How does the workshop experience impact the teachers’ self-efficacy to teach both computer programming and probability?
3. How does the workshop experience impact the teachers’ motivation to integrate computer programming into mathematics curricula?

The workshop’s impact was measured through the use of surveys, tests, and journal entries of volunteer subject teachers participating in the summer 2015 workshop. This chapter describes (1) the teachers who participated in the workshop and were the
subjects of the study; (2) the workshop’s curriculum, pedagogy, environment, typical session, and instructors, as well as the rationale for its format; and (3) the measurement tools, data collection methods and analysis plans along with the rationale for use of these methods.

Subjects

The target population for this study was secondary school mathematics teachers. As is common in many educational studies, this study was conducted in a classroom setting of teachers, most of who were enrolled and received three graduate mathematics credits for the 30-hour workshop/class. This meant that randomly sampling the population for participants was not possible. Furthermore, not all participants perfectly fit the target population. Although most were in-service secondary mathematics teachers, some were pre-service teachers, one was a middle school mathematics teacher, and a few were community college or university instructors teaching lower division mathematics courses, similar in content to that taught at the secondary level.

On the first day of the workshop, the teachers were recruited to participate in the study and informed that, should they volunteer to take part, they would be asked to participate in pre and post surveys, pre and post probability tasks/exam questions, and to allow researchers to use transcriptions of their workshop journal entries. Teachers were directly told that participating was voluntary and that they could opt-out at any time. All teachers agreed to participate in the study and were informed that the purpose of the workshop was to help train teachers to introduce computer programming into their mathematics classrooms.
<table>
<thead>
<tr>
<th></th>
<th>Affirmative Responses</th>
<th>Total Number of Respondents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participated in a high school that offered computer programming</td>
<td>4</td>
<td>16</td>
<td>25%</td>
</tr>
<tr>
<td>In-Service Teacher</td>
<td>7</td>
<td>16</td>
<td>44%</td>
</tr>
<tr>
<td>Graduate Teaching Assistant (teacher)</td>
<td>7</td>
<td>16</td>
<td>44%</td>
</tr>
<tr>
<td>Pre-Service Teacher</td>
<td>1</td>
<td>16</td>
<td>6%</td>
</tr>
<tr>
<td>UG, Interested in teaching someday</td>
<td>1</td>
<td>16</td>
<td>6%</td>
</tr>
<tr>
<td>Currently uses programming in teaching</td>
<td>0</td>
<td>15</td>
<td>0%</td>
</tr>
<tr>
<td>Experienced in teaching probability</td>
<td>9</td>
<td>15</td>
<td>60%</td>
</tr>
<tr>
<td>Has taken a course in probability</td>
<td>12</td>
<td>17</td>
<td>71%</td>
</tr>
<tr>
<td>Has taken computer programming before</td>
<td>14</td>
<td>17</td>
<td>82%</td>
</tr>
</tbody>
</table>

Table 1: Teachers’ Backgrounds

Workshop Description

The workshop is a three-week course designed for mathematics teachers to learn how to program a computer and experience how clever use of a minimal number of programming language constructs is sufficient for creating a multitude of computer programs that aide in mathematical problem solving activities. Workshop participants are pre-service or in-service mathematics teachers and therefore will commonly be referred to henceforth as “teachers.” Those who teach and lead the workshop will only be referred to as “instructors” in order to avoid confusion.

The workshop is designed to not only to help teachers learn programming skills and develop mathematical problem solving strategies, but also to show them that computer programming is fun and engaging and could be a useful tool to support the mathematical problem solving skills of their own students. The instructor hopes to give
teachers confidence in their capability to integrate programming into their own classes and motivation to do so.

The instructor introduces programming using the C programming language, which is a very popular, general-purpose programming language that is commonly used by computer scientists and in college undergraduate programming classes. Workshop teachers are not expected to have any prior computer programming experience, although often times some do. However, all workshop teachers are expected to have strong mathematics skills, as course prerequisites included two upper-division undergraduate mathematics proof courses. While the curriculum creators theorize that basic algebra skills can, and perhaps should, be developed in conjunction with learning to program, the three-week duration of the workshop is not enough time to cover all of the material that the workshop presents for participants without an appropriate background in algebra.

Theoretical framework

The workshop’s pedagogy is grounded in the theory of constructivism. The teachers are considered to be active participants in the construction of their own knowledge rather than information receptacles. To promote this concept, lecture is kept to a minimum and teachers are never expected to understand new content simply from having it told to them.

Carefully sequenced programming tasks are introduced as a precursor to any new content and the instructor often facilitates a group conversation as to which tools are needed to accomplish the task. This conversation is designed to help the teachers reach the conclusion that their current knowledge of programming is insufficient for solving the
task and some new tool is necessary. New content typically takes the form of some programming construct that can be viewed as the missing tool. This new construct is indeed then explained through brief lecture, but the teachers always have a computer in front of them during class and right after being introduced to the new content they begin to use it to solve the programming task with which they have been presented. By using this approach, teachers can immediately begin to develop meaning-making for the new programming construct and the programming activity helps to give them context to grow their understanding of computer programming to include knowledge of the new construct. Workshop instructors then go around and interact with the teachers while the teachers work on writing their program. During this time instructors are available to answer questions teachers have, but, just as importantly, to also try and understand how the teachers use the new construct in conjunction with the rest of their programming skills so that the instructor can help teachers eventually develop a more expert understanding of the construct’s use.

Consider the programming task example, *bigger.c*, in Appendix A, which is a simple task used early on in the workshop that asks the teachers to write a program that allows a user to type in two numbers and then outputs the larger number or the word “equal” if the numbers are equivalent. At this point in the workshop, teachers have only been exposed to I/O (input / output) and basic arithmetic programming constructs. The input construct allows for a user to supply a program with input values through the keyboard. The output construct allows the values associated with variables or words to be shown on the monitor. Basic arithmetic constructs include the ability to add, subtract,
multiply, and divide constants and/or variables and then to be able to store the result in another variable. A group discussion of the programming task is led by the instructor and is designed to let the teachers conclude that comparing the value of two or more variables cannot be achieved through any combination of the I/O and arithmetic tools that currently encompass their world-view of computer programming. A hypothetical, new “comparison” tool is then proposed. At this point the instructor reveals a conditional statement construct (known as the If statement in the C programming language) and briefly lectures on its basic syntax. The teachers are now ready to develop a program that uses the new conditional statement construct. How the teachers use the construct reveals their understanding of it and their ability to use it in a problem-solving context. Later programming tasks in the sequence are then given, which require more complex usage of the conditional statement, in hopes of systematically developing the teachers from novice to more expert users of the conditional statement.

*Environment, length, and typical day*

Over the course of three consecutive weeks, the workshop convenes four days per week for a total of 12 class sessions. Each session lasts for three hours with built in breaks. No outside work is assigned as all work is expected to take place during workshop sessions, although most teachers stayed later or worked on programs at home. Each teacher had their own computer, and workshop instructors discouraged more than two teachers to a single computer so that each teacher can have ample time physically typing in code. This is designed to help each teacher get comfortable with the programming syntax through repeated implementation. However, collaboration is always
encouraged in the workshop and teachers are welcome to use each other as resources in developing solutions to programming tasks. There are enough computers in the room that all of the teachers can instead work alone, if they choose to, and some teachers will mix and match.

Teachers are told on the first day that their grades are only dependent on attendance and participation. Teachers’ grades are not impacted in any way by the amount of course material that they master or by their participation in the study. This workshop design feature lets the teachers focus on learning for simply the sake of learning and is consistent with the workshop philosophy that a participant’s success is about personal growth rather than meeting standards and benchmarks.

A typical day three-hour session involved less than 30 minutes of instructor lecture. Lecture is used to introduce new programming tasks and new programming constructs as necessary. Lecture in the workshop is often a highly interactive process between the teachers and the instructor, where teachers’ are encouraged to ask questions and offer ideas. In-fact, teacher dialogue will often shape the direction of the lecture as the instructor attempts to introduce new content by connecting it to how the instructor guesses teachers may understand foundational knowledge that the new content is built upon. This CGI-inspired approach can only work if lecture engages teachers in discussion and elicits their thought process (see Chapter 2, p.36 for a discussion of CGI). In the workshop, lecture is not simply the recitation of information in a one-way flow from instructor to teachers. The term *lecture* is used here simply because this is the only
time when the workshop brings the entire group of teachers together for large-group discussion.

While typically less than 30 minutes per session are spent on lecture, teachers spend the remaining 150 minutes developing programs and mathematical ideas through programming tasks. In the first few sessions of the workshop, two to four programming tasks are often done in a single session, while during the last few sessions programming tasks are more complex both mathematically and programmatically and a single task might take more than an entire single session.

*Instructional programming tasks*

The sequential activities/programming tasks for this workshop are described in Appendix A and include full computer programming solutions in the C programming language. The number of instructional programming tasks covered by the workshop is up to the discretion of the instructor, as the workshop is designed to progress at whatever pace the instructor believes best matches the needs of the teachers. Since not all teachers will progress at the same rate, challenge problems that extend the complexity and richness of certain programming tasks are often given to more advanced teachers. In the summer 2015 workshop 20 unique tasks were given over the course of 12 sessions/days, with an additional 6 extension tasks that offered an extra challenge for teachers that finished the main tasks more quickly and easily. A list of the summer 2015 workshop programming tasks by sequential order, name, and brief description of each are in Table 2 (note that tasks with a number followed by the letter ‘b’ are extension tasks that were encouraged that the teachers try, but were not required). A more detailed description of
the task and a sample of coding for each task is presented in Appendix A: Programming Tasks and Coding.

Although the workshop is designed to develop programming skills in conjunction with exploring mathematical problem solving in novel ways, the sequence chosen for instructional programming tasks is selected primarily for the benefit of learning programming skills. This design choice best suits the learning needs of workshop participants who have much stronger backgrounds in mathematics than computer programming.

The instructional programming tasks (see Table 2) are sequenced to develop an understanding of how to utilize five different categories of computer programming constructs; namely, I/O, mathematics constructs, conditional statements, loops and arrays.
<table>
<thead>
<tr>
<th>Lab</th>
<th>Description/Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hello.c</td>
</tr>
<tr>
<td>2</td>
<td>age.c</td>
</tr>
<tr>
<td>3</td>
<td>ftoc.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>bigger.c</td>
</tr>
<tr>
<td>5</td>
<td>tens.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>6a</td>
<td>change.c</td>
</tr>
<tr>
<td>6b</td>
<td>change_ext.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>jack.c</td>
</tr>
<tr>
<td>8a</td>
<td>mulitples.c</td>
</tr>
<tr>
<td>8b</td>
<td>primes.c</td>
</tr>
<tr>
<td>9</td>
<td>uniformity.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>10a</td>
<td>craps.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>10b</td>
<td>craps_ext.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>first_match.c</td>
</tr>
<tr>
<td>12</td>
<td>last_match.c</td>
</tr>
<tr>
<td>13a</td>
<td>rooks.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>13b</td>
<td>bishops.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>birthday.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>15a</td>
<td>area.c</td>
</tr>
<tr>
<td>15b</td>
<td>sphercube.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>volume.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>17a</td>
<td>frog.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>17b</td>
<td>bee.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>dice.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>chips.c</td>
</tr>
<tr>
<td>20</td>
<td>defective.c</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Sequential Programming Tasks
**I/O.** I/O (input/output) is always the first construct introduced in the sequence and teachers will use I/O commands in their first programming task on the first day of the workshop. It is necessary to put I/O first in the sequence, because programs are largely useless if the user cannot interact with them on some level and that is the purpose of I/O. Using I/O constructs is straightforward in comparison to using other types of constructs with regard to all of the programming tasks given in the workshop. Perhaps ironically however, the syntax associated with I/O commands in the C programming language is likely more challenging to learn than the syntax for any other programming construct commands that are used in the workshop. Workshop instructors anecdotally report that they see teachers make more syntax errors with regard to I/O than any other construct. It may be beneficial then that the construct with the most difficult syntax is associated with the conceptually easiest, and most basic, early-on programming tasks. While I/O constructs are the featured learning target for only the first two programming tasks of the sequence, hello.c and age.c, teachers will get practice using I/O constructs throughout the duration of the workshop since I/O is a necessary feature in all workshop programming tasks.

**Mathematics constructs.** Mathematics constructs encompass all of the programming constructs introduced in the workshop for specific mathematical computation and variable representation. The third programming task of the sequence, ftoc.c, is a program that takes a temperature in Fahrenheit as input, and then outputs the temperature in Celsius. This task is designed to introduce the teachers to simple arithmetic operation constructs such as addition, subtraction, multiplication and division.
The use of variables to store numerical values is also introduced. The teachers’ mathematical backgrounds make it very easy for them to learn the syntax for the commands of these constructs, since the syntax for algebraic expressions in the C programming language looks almost exactly the same way as a mathematician would write the expressions on a piece of paper.

As the workshop moves through the programming task sequence, the programming tasks begin to explore mathematics problems of greater complexity. These new problems generate a need for more mathematical programming constructs. Throughout the course of the workshop, constructs are introduced for computing the mathematical floor of a real number, the quotient and remainder of a division of two numbers, the square root of a number, the absolute value of a number, and the generation of a random number value (technically speaking pseudo-random, as computer algorithms are incapable of generating truly random values), and typically nothing more.

*Conditional statements.* The conditional statement construct makes it possible to test if a certain condition is true and then branch to one course of action or another based on the results of that test. This is an incredibly valuable tool in a programmer’s arsenal and there are very few interesting mathematics problems that a program can model without this construct. In the C programming language the command for a conditional statement is known as the “if” statement. Workshop teachers have mentioned before that they are amazed at how many mathematics problems can be solved with ample, yet judicious use of the “if” statement.
Loops. Loops could be considered the construct that unlocks the power of the computer. The loop construct is similar to the conditional statement except that whereas the conditional statement takes a certain course of action if testable condition evaluates to true, a loop will repeatedly take that course of action over and over again as long as, after each completion of the course of action, the condition being tested continues to be true.

Loops allow segments of code to be repeated many times over as quickly as the computer can perform them. This is very valuable in the workshop for modeling probability problems using a Monte Carlo experimental approach, because it allows for millions of trials to be simulated in a second or less, thanks to the speed of modern day computers and the code generated by C compilers. Without the ability to perform Monte Carlo experiments so quickly, the pedagogical approach of model generation and refinement, as a means for developing expertise in Monte Carlo based probability problem solving, would be impractically slow or unconvincingly inaccurate.

Consider the programming task, rooks.c, which asks the teachers to create a programming model, using the Monte Carlo method, to determine the probability that two rooks, randomly placed on a chess board, are in attacking position of each other (See Appendix A). The theoretical probability of 2/9 was solved by the teachers first, so that they would have a way to verify the quality of their model by comparing the experimental probability approximation that their program outputs to the theoretical probability. The random number generator programming tool that the teachers use has enough pseudo-random precision to be accurate to the thousandths place if a sufficiently
large enough number of trials are simulated. Experience indicates that roughly one
million trials will yield experimental probabilities that will be accurate to the thousandths
place.

Teachers will often write code that creates a programming model that will yield
an experimental probability of about 0.234 over one million trials. Initially an
experimental probability of 0.234 might seem impressively accurate and good enough for
the teacher to conclude that their model is correct. However, this result is only accurate
to the tenths place and workshop instructors have the experience to realize that the law of
large numbers should have guaranteed a better result with the quality of programming
tools at the teachers’ disposal. Teachers who wrote code that gave this 0.234 result had
programmed a faulty model that allowed for two rooks to randomly be placed on the
same square. This was a mistake that would never be made by a physical simulator, who
had a chess board and two rooks placed in front of them, but it would be impractical for a
physical simulator to run enough trials to achieve a result that is likely to be indicative of
2/9. While this incorrect programming model is a mistake that is commonly made, it is a
mistake that is welcomed in the workshop, because it evokes discussion on a clear
misrepresentation of the sample space. Since the faulty model allows two rooks to be on
the same board space, the simulation over-represents the outcomes that exist in the
sample space. Another way to say this is, that simulation of this model creates a sample
space with a sample size that is too large. This programming model also over-counts the
number of trials that result in an attacking event.
Probabilities obtained experimentally can never be guaranteed to exactly match those obtained theoretically, but the law of large numbers does guarantee that with a sufficient number of simulation trials, the gap between the two results should get smaller and smaller. Without running millions of simulation trials, one cannot determine if the gap between the experimental probability and the theoretical probability is due to an incorrectly programmed model or the imprecision inherent in experimentally computed probability. This rooks.c programming task illustrates why the computer is a necessary tool for developing expertise in Monte Carlo based probability problem solving.

**Arrays.** Arrays are the final programming construct to be introduced in the workshop. In computer programming, an array is a data structure comprised of a series of data sub-structures where each sub-structure is of the same type and size. Sometimes arrays are called lists and they can also be thought of as a chain of variables where each variable can be accessed without having to be uniquely named. For example, imagine a program needs to save 1,000 unique numbers. A programmer could create 1,000 different variable data structures, all with different names, but that would take 1,000 or more different characters of code, would be difficult to read, and be time consuming. By using an array, a programmer essentially makes one data structure that has 1,000 different accessible compartments to hold each of the 1,000 unique numbers. This is done with minimal syntax and time.

The workshop introduces arrays because certain programming tasks are impractical to do without them. Consider the following two example programming tasks first_match.c and last_match.c (See Appendix A). The first programming task,
*first_match.c*, asks the teachers to input a sequence of numbers from the keyboard and have the program output if the first number inputted is ever repeated in the sequence. This program requires the first number to be saved in a variable data structure (for the sake of argument, this variable will be called \( V1 \)), so that it can be compared against all of the other numbers in the sequence that are yet to come. After the first number is saved, the second number needs to be saved in a second variable data structure, call it \( V2 \) and then \( V1 \) is compared with \( V2 \) to see if they are numerically equivalent. If they are not, then at this point the program allows the user to type in the third number of the sequence, which needs to be compared to the first. This can be accomplished by overwriting the value of \( V2 \) with the third number in the sequence, because the value of the second number can now be forgotten. This means that this programming task can be accomplished with essentially only two variables, \( V1 \) and \( V2 \).

The second programming task, *last_match.c*, similarly asks the teachers to input a sequence of numbers from the keyboard, but differs from the first task by having the program output if the last number inputted is ever repeated in the sequence. This task cannot be accomplished with only two variables, because the comparison number (the last number) is not saved into a variable until all of the other numbers have been typed in first by the user. As these preceding numbers are typed in they need to be saved into their own distinct variables or those numbers will be lost before any comparisons can ever be made. Assume for a moment that the last number of the sequence, the comparison number, is still saved in variable \( V1 \). Just two variables, \( V1 \) and \( V2 \) are now no longer sufficient, because overwriting a number in \( V2 \) with the next number in the
sequence is only a viable option if comparison between \( V_1 \) and \( V_2 \) has already taken place and this can no longer happen between the user inputting numbers since \( V_1 \) cannot be established until after all of the numbers in the sequence have been inputted.

These tasks, when presented in succession, help the teachers see a need for a new kind of programming construct that can hold a series of data elements and be syntactically efficient and it is in this context that arrays are typically first introduced. They are then used in models for probability programming tasks that require them, such as the classical “birthday” problem, used in the programming task *birthday.c* (See Appendix A)

**Measurement Tools, Data Collection Methods, and Analysis**

Using the workshop model previously described, this study employed tests, surveys, and journal entries to help evaluate the effectiveness of the workshop. These measurement tools were developed to measure the workshops’ effective impact on the teachers and to help answer the research questions:

1. How does the workshop’s computer programming experience impact the teachers’ *content knowledge*?
2. How does the workshop experience impact the teachers’ *self-efficacy* to teach both computer programming and probability?
3. How does the workshop experience impact the teachers’ *motivation to integrate* computer programming into mathematics curricula?

The research conducted in this study is described as an *Efficacy Research* by the *Common Guidelines for Education Research and Development* (2013). Efficacy
Research “…allows for testing of a strategy or intervention under “ideal” circumstances … [where one] may choose to limit the investigation to a single population of interest” (Common Guidelines, 2013, p.9). The strategy or intervention of this research is the summer workshop that has been conducted for several years, and the investigation is focused on the impact of the workshop on the single population of interest, secondary mathematics teachers, to teach and motivate them to use programming in mathematical problem solving.

Tests

To examine the impact of the workshop on the teachers’ ability to use programming as a tool for improving content knowledge, a pretest and posttest on probability problems was administered. Recall that this study is not directly focused on teachers’ understanding of probability. What is of interest is the ability of the teachers to develop Monte Carlo models that can solve probability problems through simulation, as well as the confidence and motivation that teachers report in using programming to solve probability problems, as measured prior to the workshop and at its conclusion.

The pretest and posttest are provided in Appendix B. The probability problems for the Pretest were chosen from classic elementary probability problems encountered in secondary school statistics courses. None required knowing probability distribution functions, although some teachers may have chosen to identify the distribution to answer the problem. The three probability problems are listed in Figure 1.
1. What is the probability of rolling a sum of “eight” at least twice in three rolls using two six-sided dice?

2. What is the probability of randomly drawing, without replacement, a total of 2 red chips and 3 blue chips, in any order, from a bowl of 60 chips where 10 are white, 20 are red, and 30 are blue.

3. Factory A produces cars where 5% have major defects, and Factory B produces cars where 10% have major defects. A dealer received 30 cars, 12 from Factory A and 18 from Factory B. A customer randomly bought one of the 30 cars and found it to be defective. What is the probability that it came from Factory A?

Figure 1. Pretest Questions

With each of the above problems the teachers were also asked the following questions listed in Figure 2:

How confident are you with your answer? Circle one
   a. Sure of my answer
   b. Fairly confident
   c. Not very confident
   d. I have no idea how to solve it

Check all of the following that apply:
   ___My answer is complete.
   ___My answer is incomplete because I ran out of time.
   ___My answer is incomplete because I do not know how to proceed further.

Comment on the problem solving strategy/strategies that you used for this problem

Figure 2. Supplemental questions to pretest
The teachers were not given a time limit, but it was expected that they would spend no more than 60 minutes on the Pretest. The companion questionnaire that accompanied each problem was included to help determine, along with the surveys, which strategies the teachers employed in solving the problem and how confident they were in their problem solving endeavors.

The teachers were allowed the use of manipulatives such as dice, colored chips with containers, and calculators. They were also allowed to use their computers to use any statistical software package where Excel, Minitab and Tinkerplots were all available. The teachers were informed that they could find either the experimental or theoretical probability to each problem and could leave their answers in either fractional or decimal form. As the pretest occurred on the first day of the workshop, it was not expected that any teachers would use programming to model the problems and then solve the problems through simulation. However, the teachers did have access to the necessary technology to implement such a programming solution, should they have the expertise and inclination to do so.

The Pretest was not graded during the duration of the workshop, nor were the tests returned to the teachers. At the conclusion of the workshop, three weeks later, the Posttest asked the same three questions with the added requirement that they use computer programming to model the problems and run simulations to determine the probability experimentally. In addition to explaining how they attacked the problem
using the C programming language, the teachers also submitted the programming code for their models.

For those teachers taking the workshop for university credit, none of the instruments discussed contributed or influenced the teachers’ grade in the course. Their grade was completely determined by attendance and participation in programming tasks that were not included in the study.

**Surveys**

Two surveys were conducted during the 3-week workshop. The *Pre-Survey* was conducted on the first day of the workshop and the *Post-Survey* on the last day. The Pre-Survey’s purpose was to establish the teachers’ 1) teaching background and description of current use of programming at their school, 2) background in computer programming and probability, 3) comfort level in using technology and using, or teaching, computer programming and 4) motivational level for integrating programming into mathematics both personally and in their teaching. Both open response and Likert scale levels were used. The Pre-Survey and Post-surveys are located in Appendix C. Both surveys have been de-identified so information regarding the course number, institution, instructor’s name, etc. has been marked out.

The design of the Pre-Survey and Post-Survey followed the practice of first doing some interviews as suggested by Desimone & Le Floch (2004). Although informal, this summer workshop model has been conducted and improved upon for several years. Improvements and adjustments to curriculum and pedagogy were based on teacher evaluations and informal questioning of teachers. The survey questions evolved from
these past workshops. A Likert-style five-point, unipolar response scale is often recommended when a Confidence construct is being assessed (Artino, La Rochelle, Dezee, & Gehlbach 2014) and so served this study by quantitatively comparing the pre-survey and post-survey results to measure the workshop’s impact on teachers’ comfort and motivation levels. For example the teachers were queried both on the first and last day of the workshop: “If the technology was available, how comfortable would you be using programming in a mathematics course?” Followed by the response asking them to circle and comment:

<table>
<thead>
<tr>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very</td>
<td>Not at all</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Likert responses for survey

A copy of the Post-Survey can be found in Appendix C. It was conducted at the termination of the workshop, repeated many of the questions in the Pre-Survey, and included some new questions exploring the comfort level of the teachers regarding the use of programing as listed in Figure 4:
• “While doing a probability problem/lab where the experimental (programming) result differed from the theoretical results, which result did you trust?”

• “After completing this class, how comfortable would you be using programming in a math course you teach? If not, what would it take for you to become more comfortable?”

Figure 4: Comfort level questions on Post-Survey

In order to help evaluate the effectiveness of the workshop, the post-survey also asked the teachers to evaluate the workshop’s curriculum and pedagogy for preparing mathematics teachers to incorporate programming in their courses. The Post-Survey urged the teachers to suggest improvements or changes. An added component was to address the question of support needed to begin teaching with programming, such as computer equipment availability, IT support at their school, more problems to use with students, a mentor, another workshop, or administrative support. This question was added in response to the Computer Science Teachers Association’s (CSTA) list of obstacles preventing teachers from teaching computer science that went beyond just the obstacles of content and pedagogical content knowledge (CSTA K-12 Computer Science Standards, 2011).

Journals/portfolios

During the course of the twelve workshop sessions each teacher kept a journal as a part of their overall portfolio, with entries that responded to general verbal prompts given by the instructor. The instructions and prompts are listed in Figure 5.
Portfolio

Your portfolio needs to be in a 3-ring binder. There will be about 20 activities. Order the activities in chronological order. Your binder will be collected, scanned and returned on the last day of class.

For the “write-up” of each activity or task include the following:

1. Each activity/task needs to include your name and the date it was completed.
2. A full description of the problem and how you approached problem.
3. Printout of the program including
   a. Comments so a fellow student can follow reasoning
   b. Highlights of any new syntax learned.
4. Printout results

Daily Journal E-mail Entry

At the conclusion of each day, send an e-mail to ________ with the following:

1. Which activities did you attempt that day? Which did you complete?
2. What did you learn from each activity?
3. What was the most interesting thing? What could have been omitted or emphasized less?
4. How well did you do with the activities? Are you catching on to programming and the problems?
5. Did anything cause confusion?
6. What do you need help with?
7. Any other comments or insights?

Figure 5. Portfolio and journal instructions and prompts

Journal entries were e-mailed to the course instructor, by the teachers, after most days with a few exceptions. Near the end of the workshop, the teachers integrated their journal submissions into their final portfolio product. The portfolio was e-mailed at the conclusion of the workshop and was then de-identified before being made available for
this study. Frequent journal emails to the instructor allowed the teachers themselves to assist directly in the development of the workshop, an approach espoused by Etherington (2004).

To help the teachers in self-reflecting and in writing meaningful and constructive entries for their journals, the teachers were paired and verbally discussed the tasks with their partner before writing in the journals so they could gather their thoughts and provide more thoughtful responses. It was conjectured that the teachers would be more candid with each other than with an outside researcher or workshop leader and after verbalizing their thoughts could write a more reflective and accurate journal entry.

The workshop instructor stated on the first day that the instructor would act as a guide. The teachers were treated as peers, not subordinates. Teachers were asked to not just see themselves as workshop participants, but to help contribute to the improvement of the workshop by self-reflecting and sharing their observations on the impact the workshop was having on them. Did one activity or teaching approach help them? Could the workshop be fashioned better for introducing probability? What did they think of a certain activity? The entries were read by the instructor daily and often ideas developed in a journal entry became the point of focused discussion by the whole group the next day. In many cases the workshop was adjusted to address the teachers’ concerns or suggestions.

Data analysis plans

A mixed methods approach of both quantitative and qualitative data analysis was used for this study. Due to the expected small number of workshop participants (between
ten and twenty expected), the inability to select participants from a random sampling of the population, and the lack of a control group to compare against the workshop treatment, a purely quantitative approach would be limited, as is often the case in educational research. A qualitative approach would be limited as well, since the ability of the researcher to generate large amounts of rich, experiential data would be hindered by the workshop’s short, 3-week duration and the inability to conduct interviews of the participants, mostly because of the large time demands on workshop participants both during and after workshop hours.

Despite limiting factors to both quantitative and qualitative approaches for this study, the projected limitations were viewed as likely dichotomous, if not mutually exclusive. A mixed methods approach was chosen then in an effort to shore up the limitations inherent in a singular approach. Furthermore, analysis of a mixed methods design was theorized to more effectively appeal to the target audience for this research that includes both mathematics educators and computer scientists. While mathematics education research has helped pioneer qualitative research approaches and illuminated their value for social research, computer science research is steeped in a tradition of quantitative approaches. If this research is to be relevant and impactful, it is vitally important to attempt to entice both communities via methods that are both familiar and accepted.

Before data analysis plans are clearly explained for the data collected by each measurement tool employed in this study (tests, surveys, and journals/portfolios), a
Theoretical framework will first be introduced for critically analyzing the qualitative data collected in this study.

Theoretical framework for qualitative data analysis. The theoretical framework chosen to qualitatively analyze workshop data is derived from a philosophy known as phenomenology. This section will briefly define and describe phenomenology and discuss why a phenomenological framework for the qualitative data analyses was deemed a strong fit for this study.

Phenomenology. Phenomenology can be viewed as both a philosophical theory as well as a methodology for qualitative data analysis and can be considered a “part of the constructivist/interpretivist paradigm” (Fridlund & Hildingh, 2000, p.49).

Phenomenology is a school of thought that was developed by the German philosopher, Edmund Husserl, around the time of World War I, suggesting that the world does not exist independently of the human mind. From a phenomenological perspective, the only certainty people can ever have about objects in the world is the certainty of how they consciously perceive them. Therefore, the only meaning of consequence for an object is the meaning derived from the sum total of human experience with the object, thus making reality a collection of “pure ‘phenomena’” (Groenewald, 2004, p. 43).

The phenomenological researcher leverages this world-view philosophy to guide him/her in the qualitative analysis of data collected on some phenomenon. Through a process known as phenomenological reduction, the researcher withholds predisposed judgment of some phenomenon of study and instead seeks to understand the phenomenon
through an analytical description of human experiences with the phenomenon (Groenewald, 2004).

**Why phenomenology for workshop analysis.** The philosophy that guides the development of the workshop and its approach to teaching and learning is the *theory of constructivism*. A constructivist approach to teaching and learning, which borrows heavily from Cognitively Guided Instruction, influences the scaffolding of programming tasks and interactions between the teachers, instructors and the programming tasks themselves.

With constructivism firmly at the core of workshop practices and design, phenomenology promised a research methodology that would yield a qualitative data analysis appropriate for evaluating the workshop within a constructivist paradigm of learning and understanding. Constructivism and phenomenology share very similar philosophies on the role that individuals play in the construction of truth and knowledge. In a constructivist theory of learning, instructional tasks must be grounded in learners actively participating, exploring, and interacting in order to create opportunities that allow them to put new experiences in context with previous experiences. Constructivism theorizes that this is how people understand the world at hand and generate knowledge.

Phenomenological research methodology is therefore extremely well suited for studying people that are participating in a workshop designed with constructivist principles, because in both constructivism and phenomenology, truth is experiential; that is to say that the nature of all things are understood by people and the context they give to things as they experience the world.
Data collection constraints, imposed by the participating institution where the workshop was held, also factored into the consideration of using phenomenology. These constraints kept the researcher from accessing or reviewing any collected data until the workshop was concluded. This meant that qualitative research methodologies such as Ethnography and Grounded Theory using the constant comparative method were not the best choices, as these methodologies often involve interweaving data collection with data analysis. In phenomenology, analysis of a phenomenon ideally only occurs after all experiential data has been collected and considered. Therefore phenomenology offered a framework that did not expect any data analysis to occur concurrently with the data collection process.

Limited researcher access to busy workshop participants further constrained data collection options and meant that there would be minimal, if any, opportunity for any interview data to be gathered. However, because the workshop was grounded in CGI based practices, the teachers were required to keep journals to aid them in reflection. These journals, embedded in their portfolios, were made available for analysis at the conclusion of the workshop and offered rich, experiential, and descriptive text based data. This type of data is a necessity for phenomenological research.

To briefly summarize, not only do phenomenology and constructivism align philosophically, but phenomenological research methodology is also very well suited for analyzing the journal/portfolio-based qualitative data collected for this study. With these advantages in mind, phenomenology was chosen as the theoretical framework for guiding qualitative data analysis in this study. The workshop itself would be the phenomenon of
study and the teachers’ journals/portfolios would provide the descriptive experiences with the phenomenon.

*Analysis plans for each measurement tool.* The three different types of measurements employed in this study are designed to provide independent analyses that collectively explore the research questions in depth. This section provides analysis plans for each measure.

**Analyzing Tests.** Non-parametric statistics were used to compare the pretest and posttest results since, with such a small expected sample, normality cannot be assumed, making standard paired t-tests invalid. *McNemar’s Test* was used to test the null hypothesis that the proportion of teachers that achieved correct solutions was the same on the pretest as the posttest. Descriptive statistics were reported and inter-rater reliability was measured using Cohen’s Kappa. Content validity was established through expert content review of the tests.

**Analyzing Surveys – Likert Scale items.** Two sets of questions were used in the Pre- and Post-Surveys which can be found in Appendix C. The Likert Scale questions were asked on both surveys and used to compare the teachers’ beliefs, motivation, and confidence levels before and after the workshop course. The open-ended questions were used to capture teachers’ backgrounds in the pre-survey and their evaluation of the workshop in the post-survey. This descriptive data was used to conduct an exploratory factor analysis to analyze whether differences may be due to teachers’ characteristics. For example, were in-service teachers more motivated to adopt programming into their mathematics courses than pre-service teachers?
For the Likert Scale comparison questions asked on both the pre-survey and post-survey, a quantitative comparison was explored using descriptive and non-parametric statistics. The expected sample size was small, thus parametric statistics with assumptions of normality were not appropriate. The teachers’ responses were independent of each other though, which is an important assumption for non-parametric tests. The differences in Likert Scale survey items was analyzed using the *Wilcoxon Signed-Rank Test* to help determine if the workshop was likely to significantly impact teachers in the way that each paired Likert Scale survey items described (primarily items relating to teachers self-efficacy to teach programming for mathematical problem solving as well as items relating to their motivation to do so).

Internal consistency between related survey questions was used to establish survey reliability and was measured using Cronbach’s alpha reliability coefficient.

Content validity was established through expert content review of the surveys.

Analyzing journals/portfolios and open-ended survey responses. To qualitatively analyze the journal/portfolio data through the lens of a phenomenological framework, a four-step analytical process was used. This process was adapted from a similar procedure used in Groenewald’s (2004) phenomenological study and attempts to apply *phenomenological reduction* throughout the process.

1. Thoroughly read/familiarize oneself with the raw, experiential data in order to obtain a sense of the data as a whole. This step initiates the researcher in the phenomenological reduction process.
2. Read through the data again and code it in order to create different units of meaning. Categorize each unit as either relevant or non-relevant by considering how often the unit appears in the data and how much weight the context of the text gives to the unit.

3. Organize and group the units into themes that are integral for describing the workshop (i.e. the phenomenon).

4. Thoroughly **describe** the workshop through a summary analysis of the themes. Determine if the workshop description contains themes that indicate if and how the workshop impacts teachers’ self-efficacy and motivation for both using and teaching computer programming to support mathematical problem solving.

Report on these impactful themes.
Chapter 4: Results

Comparison of Pretests and Posttests

Quantitative analysis of the pretests and posttests was used to determine if measured changes between the paired results were statistically significant. Statistical significance could indicate that the Workshop treatment could be expected to affect mathematics teachers’ ability to solve basic probability problems by simulating with Monte Carlo models.

There are three test problems that were administered in the Workshop. A brief summary of the three problems is as follows. These problems were discussed in greater detail in Chapter 3:

1. **Dice**: What is the probability of rolling a sum of “eight” at least twice in three rolls using two six-sided dice?

2. **Chips**: What is the probability of randomly drawing, without replacement, a total of 2 red chips and 3 blue chips, in any order, from a bowl of 60 chips where 10 are white, 20 are red, and 30 are blue?

3. **Defective**: Factory A produces cars where 5% have major defects, and Factory B produces cars where 10% have major defects. A dealer received 30 cars, 12 from Factory A and 18 from Factory B. A customer randomly bought one of the 30
cars and found it to be defective. What is the probability that it came from Factory A?

These problems were used, as written, in both the pretest and the posttest, with the difference being that in the pretest, the teachers’ were instructed to solve the probability problems using any manipulatives or technology that they wanted to use. On the post test, the teachers’ were instructed to use computer programming to create a model that would determine the probabilities experimentally via simulation.

**McNemar Test results**

A non-parametric, one-tailed McNemar’s Significance Test was used to compare paired pretest and posttest results. The McNemar Test was appropriate for this study, because a normally distributed sample could not be assumed due to only seventeen study participants (n=17) and the inability to recruit from the population using random sampling. Furthermore, each question on the pretest and posttest was graded with either a score of ‘0’ or ‘1’, with a score of ‘0’ indicating that the solution to the problem was judged incorrect and a score of ‘1’ indicating that the solution was correct. McNemar’s Test is appropriate for this sort of dichotomous data.

Using this dichotomous, ordinal grading scheme forced the raters to determine if a partially correct solution would be considered either correct or incorrect, thus disregarding any qualification of how correct a solution is. This scheme was adopted in order to attempt to establish a strong inter-rater reliability score for the pretest and posttest. It was believed that scoring problems as either correct or incorrect would result
in greater agreement between raters because of the wide variability of techniques the teachers may have employed to answer the problems.

The null hypothesis for a McNemar’s test is that the proportion of subjects that exhibit some trait is the same before the treatment as it is after the treatment. For this research the trait was correctly solving a probability test problem and the treatment was the workshop. The fundamental level of significance chosen for this analysis was $\alpha = 0.05$, which is a widely accepted significance level for scientific research. For each of the three problems, a continuity corrected chi-squared distribution was used to create a test statistic. The continuity corrected chi-squared distribution is $\frac{(b-c-1)^2}{b+c}$ where $b$ and $c$ are the upper-right and lower-left cells of a 2x2 contingency table. These cells are often referred to as the discordant cells. At an $\alpha = 0.05$ level, the McNemar test has a critical value of 3.84. This meant that in order to reject the null hypothesis, the computed test statistic had to be greater than or equal to 3.84.

Analysis of pretest and posttest question sets showed a significant improvement, from the pretest to the posttest, in the teachers’ ability to solve the first two problems, Dice and Chips. Analysis of the teachers’ posttest solutions to the third problem, Defective, ultimately resulted in the third problem being considered invalid and so all significance testing results that include the Defective problem, while still presented in this paper, were ultimately discarded as erroneous results. Analysis of the Defective posttest solutions did however provide insight on ways to improve the Workshop in the future. In the following sub-sections, scoring and significance-testing results will be given for all three questions independently and then for the first two questions combined and then all
three questions combined. The reasons for invalidating the third question, Defective, and the inferences drawn from its analysis, will also be presented.

<table>
<thead>
<tr>
<th></th>
<th>Pretest</th>
<th>Posttest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dice Problem Rater</td>
<td>Chips Problem Rater</td>
</tr>
<tr>
<td>Alice</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Barry</td>
<td>1 1 0 0</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td>Cory</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Denise</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Esther</td>
<td>1 1 1 1</td>
<td>1 1 3 3</td>
</tr>
<tr>
<td>Greg</td>
<td>1 1 0 0</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td>Kate</td>
<td>1 1 1 1</td>
<td>1 1 3 3</td>
</tr>
<tr>
<td>Lester</td>
<td>0 0 1 1</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td>Lisa</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Melinda</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Molly</td>
<td>1 1 0 0</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td>Mort</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Muriel</td>
<td>0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Shawn</td>
<td>1 1 0 0</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td>Victor</td>
<td>1 1 1 1</td>
<td>0 0 2 2</td>
</tr>
<tr>
<td>Wanda</td>
<td>1 1 0 0</td>
<td>1 1 2 2</td>
</tr>
<tr>
<td>Wilma</td>
<td>1 1 1 0</td>
<td>1 1 2 3</td>
</tr>
<tr>
<td>% correct</td>
<td>53 53 29 29 47 53 43 45</td>
<td>94 100 88 82 41 59 75 80</td>
</tr>
</tbody>
</table>

Table 3. Rater I & II Scores of Pretest & Posttest Questions 1=correct, 0= incorrect
Again as a reminder as to the three pretest and posttest questions, they are listed below:

1. *Dice*: What is the probability of rolling a sum of “eight” at least twice in three rolls using two six-sided dice?

2. *Chips*: What is the probability of randomly drawing, without replacement, a total of 2 red chips and 3 blue chips, in any order, from a bowl of 60 chips where 10 are white, 20 are red, and 30 are blue?

3. *Defective*: Factory A produces cars where 5% have major defects, and Factory B produces cars where 10% have major defects. A dealer received 30 cars, 12 from Factory A and 18 from Factory B. A customer randomly bought one of the 30 cars and found it to be defective. What is the probability that it came from Factory A?

For all three problems all 17 teachers responded to both the pretest and posttests and thus \( n = 17 \) in each case. For all three pretest problems, the teachers were not constrained to using any particular problem solving method to find the probability, while in the posttest they were instructed to use computer programming to find an experimental probability.

Table 4 shows the results using the McNemar test at \( \alpha=0.05 \) where the test statistic was the continuity corrected chi-squared distribution with one degree of freedom. The null hypothesis was that the proportion of teachers that achieved correct solutions was the same on the pretest as the posttest.
<table>
<thead>
<tr>
<th>Dice Problem</th>
<th>Pretest Correct</th>
<th>Posttest Correct</th>
<th>Posttest Incorrect</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pretest</td>
<td>Rater I: 9</td>
<td>Rater I: 0</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>Correct</td>
<td>Rater II: 9</td>
<td>Rater II: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pretest</td>
<td>Rater I: 7</td>
<td>Rater I: 1</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Incorrect</td>
<td>Rater II: 8</td>
<td>Rater II: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>Rater I: 16</td>
<td>Rater I: 1</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Rater II: 17</td>
<td>Rater II: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

McNemar Test Result *Dice* REJECT NULL
Rater I: Difference = 0.4118  Test Statistic = 5.14  p =0.007813 (one-tail)
Rater II: Difference = 0.4706  Test Statistic = 6.12  p =0.003906 (one-tail)

<table>
<thead>
<tr>
<th>Chips Problem</th>
<th>Pretest Correct</th>
<th>Posttest Correct</th>
<th>Posttest Incorrect</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pretest</td>
<td>Rater I: 5</td>
<td>Rater I: 0</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Correct</td>
<td>Rater II: 5</td>
<td>Rater II: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pretest</td>
<td>Rater I: 10</td>
<td>Rater I: 2</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>Incorrect</td>
<td>Rater II: 9</td>
<td>Rater II: 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>Rater I: 16</td>
<td>Rater I: 1</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Rater II: 17</td>
<td>Rater II: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

McNemar Test Result *Chips* REJECT NULL
Rater I: Difference = 0.5883  Test Statistic = 8.10  p =0.000977(one-tail)
Rater II: Difference = 0.5294  Test Statistic = 7.11  p =0.001953(one-tail)

<table>
<thead>
<tr>
<th>Defective Problem</th>
<th>Pretest Correct</th>
<th>Posttest Correct</th>
<th>Posttest Incorrect</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pretest</td>
<td>Rater I: 4</td>
<td>Rater I: 4</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Correct</td>
<td>Rater II: 6</td>
<td>Rater II: 3</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>Pretest</td>
<td>Rater I: 3</td>
<td>Rater I: 6</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>Incorrect</td>
<td>Rater II: 4</td>
<td>Rater II: 4</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>Rater I: 7</td>
<td>Rater I: 10</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>Rater II: 10</td>
<td>Rater II: 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

McNemar Test Result *Defective* DO NOT REJECT NULL
Rater I: Difference = 0.0588  Test Statistic = 0.00  p =0.50 (one-tail)
Rater II: Difference = 0.0588  Test Statistic = 0.00  p =0.50 (one-tail)

Table 4. McNemar Test on pretest and posttest scores for *Dice, Chips* and *Defective*
The results from both Rater 1 and Rater 2 suggested that a mathematics teacher’s ability to solve the *Dice* probability problem can be expected to significantly increase from the workshop treatment. Likewise the results from both Rater 1 and Rater 2 suggested that a mathematics teacher’s ability to solve the *Chips* probability problem can be expected to significantly increase from the workshop treatment. And finally, the McNemar results from both Rater 1 and Rater 2 suggested that a mathematics teacher’s ability to solve the *Defective* probability problem cannot be expected to significantly increase from the workshop treatment.

While grading the *Defective* posttest problem the raters encountered a dilemma that forced the researcher to conclude that the *Defective* question was invalid and therefore may have offered erroneous significance testing results. What the raters found while grading the *Defective* problem was that several teachers developed a computer program that would arrive at the correct probability of $p = 0.25$, but were judged to be solutions deserving a score of ‘0’. Solutions with correct results were never intended to receive scores of ‘0’, however these questionable solutions were all rejected, not because they failed to produce correct answers, but because they were programming solutions that did not determine probabilities experimentally. Both raters noted the relative difficulty in judging solutions of this type in comparison to judging any other types of solutions on any of the pretests or posttests. The difficulty the raters encountered in judging solutions of this type may have contributed to the low inter-rater reliability observed for this problem. Low reliability also suggests low validity.
When the three problems were first developed, expert reviews determined that the pretests and posttests had content validity. What the experts failed to consider upon review was that correct programming models need not only yield correct probabilities upon simulation, but also must be models where simulation produces an experimental approximation of the theoretical probability. If the computer program does not model an experiment, but does offer a correct result, then the model must be a theoretical model simply wrapped inside of a computer program. This means the program is not a Monte Carlo simulation, is not leveraging the power of the computer to repeat experimental conditions many times over, and therefore the program is not really a necessary component of the solution.

Consider Figure 6 of code from a teacher’s Defective posttest (Esther) that, despite giving a correct probability of 0.25 at runtime, was graded by both raters as a ‘0’ or as imperfect:


/*let’s use 300 cars instead of 30 cars*

\[da=0.05*(fa)*10.0; \] //5% of A’s fleet is defective
\[db=0.10*(fb)*10.0; \] //10% of B’s fleet is defective
\[d=da+db; \] //# of cars defective, out of cars*10

\[success=0;\]
\[k=0;\]

\[
while(k<n) \\
\{
    \text{//number of trials}
    \text{pick=floor(d*drand48())+1;} //this picks a random number 1,2,3, …, 24
    \text{if(pick<=da)} \{ \text{//from factory A}
    \text{success=success+1;} \\
    \text{//counting up the number of times the defective car was from factory A}
    \}
    \text{k=k+1; //on to the next trial}
\}

\[prob=success/n;\]

\text{printf("\nThe probability that the defective car is from factory A is \%lf\n\n",prob);}

Figure 6. Sample of imperfect simulation code for defective probability problem

In this teacher’s solution, shown in Figure 6, the variable ‘fa’ represents the number of cars on the dealer’s lot that are from factory A and similarly ‘fb’ represents the number of cars on the dealer’s lot that are from factory B. For purposes of this problem, fa is chosen to be 12 at run time and fb is chosen to be 18. With fa = 12 and fb = 18, it follows that da = 6 and db = 10. The teacher creates da and db to be the number of expected defective cars on the lot from factories A and B respectively based on their knowledge that 5% of cars from factory A and 10% of cars from factory B, should be defective on average. On a dealer lot of 30 cars, as is stated in the original problem, da = 0.6 and db = 1.8, but the teacher has chosen to multiply everything by 10, making the
dealer lot a size of 300, so that da and db can be integers, which is helpful because the
teachers are used to workshop activities where they randomly sample from a range of
integer values. The line of code ‘pick=floor(d*drand48())+1;’ takes a random real
number from [0,1) and transforms it by a scale factor of ‘d’, which is the total number of
defective cars on the lot, and then takes the floor to transform the real number into an
integer [1,2,3,4,5,6,7,8,9,10,…24].

By allowing for the change in dealership size from 30 to 300, (a trivial change
that should only trivially affect the model) already this solution is deviating from a truly
experimental model of the problem by working on the assumption that when the dealer’s
lot is full it has exactly 6 defective cars from factory A and 18 defective cars from factory
B. A truly experimental model would seed the dealership with a fresh set of 300 cars for
each trial where all 120 cars on the lot that came from factory A and all 180 cars on the
lot that came from factory B, would be based on a random sampling of cars produced at
factory A and cars produced from factory B at a defective distribution rate of 0.05 and 0.1
respectively. This random sampling would mean that for some trials there will likely be
6 defective cars from factory A and 18 from factory B, but for other trials there will not
and the law of large numbers will be relied upon to give an overall accurate probability
by simulating a very large number of experimental trials.

The solution this teacher provided does rely on a large number of successive trials
of randomly sampled data, but the only thing being experimentally determined in the
solution is that over a sufficiently large number of trials, 25% of the time a positive
integer of 6 or less will be randomly selected from a range of integers [1,24]. This is a
solution with experimentally determined elements, but it does not come close to experimentally modeling the entirety of the problem.

*Reliability and validity of tests*

In order to establish reliability for the pretest and posttest measurement tools, two raters were used to score the pretest and posttests and their scores were compared to establish Inter-Rater Reliability (IRR). The two raters will be referred to in this paper as Rater 1 and Rater 2.

Rater 1 is the paper’s author and has taught mathematics at both the high school and collegiate level, has taught computer programming at the collegiate level, and has taught courses on integrating computer programming into mathematical problem solving.

Rater 2 is an Associate Professor in Mathematical Sciences at Lewis & Clark College. Rater 2 has a Ph.D. in Computer Science and master’s degrees in Computer Science and Mathematics and has extensively taught both computer science and mathematics courses for over 20 years.
| Inter-Rater Reliability (IRR) | **Agreement** | **K** | **Prevalence Index** | **PABAK** | \( K >0.75 \) or \( PABAK > |0.8| \) Reliability |
|-----------------------------|--------------|-------|----------------------|-----------|---------------------------------------------|
| **Dice**                    |              |       |                      |           |                                             |
| Pretest                     | 100%         | 1.00  | 0.06                 | 0.88      | Reliable                                    |
| Posttest                    | 94%          | 0.00  | 0.94                 |           | Reliable                                    |
| **Chips**                   |              |       |                      |           |                                             |
| Pretest                     | 100%         | 1.00  | -0.41                |           | Reliable                                    |
| Posttest                    | 94%          | 0.77  | 0.77                 |           | Reliable                                    |
| **Defective**               |              |       |                      |           |                                             |
| Pretest                     | 94%          | 0.88  | 0.00                 |           | Reliable                                    |
| Posttest                    | 82%          | 0.66  | 0.00                 |           | NOT Reliable                                |

Table 5. Inter-Rater Reliability on pretests and posttests

Percentage agreements, as well as Cohen’s Kappa reliability scores, were calculated for both the pretest and posttest results of the *Dice*, *Chips*, and *Defective* problems in order to establish IRR for each problem of the testing tool (see Table 5). Percentage agreement is the ratio of the number of item agreements between raters to the total number of items rated. Percentage agreement is often used in IRR because it is easy to compute and analyze, however it is widely considered to be extremely limited due to its inability to account for agreements that may occur due to chance and its lack of a cut-off level to indicate acceptable IRR. Kohen’s Kappa addresses the limitations of percentage agreement, because chance agreement is accounted for and community guidelines for minimally acceptable Kappa scores do exist. In general, Kappa is
considered a more conservative statistic than percentage agreement and so more scientifically acceptable. In this analysis, percentage agreement was computed and presented as a descriptive statistic, but Cohen’s Kappa was a weightier factor in the consideration of IRR.

While there is no exact community consensus on a minimally acceptable kappa value to establish IRR, this analysis used a conservative value of $K = 0.75$ as a cut-off value. Therefore, any reported kappa greater than or equal to 0.75 was interpreted as a sufficiently high level of agreement between raters and therefore statistically reliable.

It is important to note that there are conditions upon which Kappa can be very low and yet inter-rater agreement can still be considered to be very high, thus making Kappa unreliable. One of these conditions is known as prevalence and occurs when there is a sufficiently large difference between the number of correct solutions that the raters agreed upon and the number of incorrect solutions that the raters agreed upon. In situations where Kappa was low, but percentage agreement was very high, high levels of prevalence were suspected to be negatively impacting the Kappa statistic.

To detect issues of prevalence, a prevalence index was calculated to determine which kappa values could be unreliable. Prevalence values range from -1.0 to 1.0 with values close to either extreme indicating high levels of prevalence. Much like Cohen’s Kappa, there is no exact community consensus as to cut-off values that determine if prevalence is making the Kappa statistic unreliable, so a conservative prevalence index score was used in this study. For any pretest or posttest question problem where a
prevalence index score was greater than or equal to 0.8, or less than or equal to -0.8, Kappa was considered to potentially be unreliable due to prevalence.

Kappa can be adjusted to compensate for prevalence by computing a Kappa alternative called PABAK (Prevalence-Adjusted Bias-Adjusted Kappa). PABAK scores that adjusted Kappa were reported in addition to non-adjusted Kappa scores for pre – or posttest problems where prevalence index > |0.8|.

IRR results in Table 5 suggested that all six pretest and posttest questions were sufficiently reliable except for the Defective posttest question. The unreliability of the Defective posttest problem is likely due to the difficulties the raters had in judging solutions that had correct solutions but may have not been derived experimentally.

Content validity of the pre- and post-test was established through expert content review by a professor at Lewis & Clark College with a Ph.D. in Computer Science and a Master’s Degree in Mathematics.

Survey Results

Questions on the pre- and post-surveys were used to study the teachers’ perceptions as to how the workshop affected their abilities, confidence, and motivation to integrate programming into mathematics. In almost every quantitative comparison it was found that the Workshop had a significant effect on teachers’ perceived abilities, comfort levels, and motivational levels to program, teach programming, integrate programming into mathematics and to problem solve. This section will detail the analysis of the collected survey data (see Table 6).
Table 6. Raw Data of Surveys

Two surveys were administered in the Workshop, with a pre-survey being given at the beginning of the first day and a post-survey being given near the end on the last day. The pre-survey contained nine, five-point Likert scale questions and the post-survey...
contained seven. The five-point scale ranged from 1 to 5, discreetly, with a 1 meaning “Not at all” and a 5 meaning “Very”, “Definitely” or “A lot”. The pre-survey and post-survey questions were sorted into one of four categories; paired comparison, sample population characteristics, perceptions of future programming and problem solving improvement, and characteristic and background factor survey questions.

**Paired Comparison Survey Questions**

The paired-comparison survey questions were equivalent pre- and post-survey questions that were the most important for understanding whether or not the workshop experience made a significant impact on the teachers’ comfort with computer programming, belief in its relevance to do and teach mathematics, and motivation to continue to involve computer programming in the mathematics they do and/or teach. The post- and pre-survey differences (post – pre) were examined for statistical significance using the Wilcoxon Signed Rank test. The Likert-scale questions grouped in the Paired Comparison Survey Questions category are as follows:

1. How comfortable are you in writing a computer program?
2. How comfortable are you using computer programming as a tool to do and/or teach a mathematics course? How comfortable are you teaching a course or unit on computer programming?
3. How strongly do you believe that computer programming should be an essential part of mathematics education?
4. How motivated are you to include computer programming in any mathematics course you teach?
All of the above paired-median responses were found to have significantly increased over the duration of the workshop. The paired-comparison survey responses that were asked on both the pre- and post-surveys were analyzed for statistically significant median differences using a two-tailed Wilcoxon Signed Rank test; see Table 7. The Wilcoxon Signed Rank test was an appropriate statistical test for this study as it can measure whether or not the median differences between paired pre- and post-survey Likert scores (which are ordinal by nature) are significantly different. Furthermore, the test is also a non-parametric test, which is important since our sample population was not randomly sampled and therefore normal distributions cannot be assumed. The null hypothesis was that the median difference of all teachers’ scores was zero, where half of all pre- and post-scores should be below zero and half should be above. Significance was established at the 0.05 level.

The last column on Table 7 did not compare pre-survey to post-survey responses, but instead examined the two post-survey responses comparing the comfort level of teachers teaching mathematics with programming versus teaching programming alone. With a significantly higher median score of 4 versus 3, the Wilcoxon test indicated that teachers are more comfortable teaching mathematics with programming than teaching programming alone.
Paired Comparisons of Pre-Survey and Post-Survey Responses

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PRE</td>
<td>POST</td>
<td>PRE</td>
<td>POST</td>
<td>PRE</td>
<td>POST</td>
<td>PRE</td>
</tr>
<tr>
<td>Level 5 - Very</td>
<td>0%</td>
<td>41%</td>
<td>0%</td>
<td>29%</td>
<td>0%</td>
<td>6%</td>
</tr>
<tr>
<td>Level 4</td>
<td>6%</td>
<td>47%</td>
<td>18%</td>
<td>41%</td>
<td>12%</td>
<td>35%</td>
</tr>
<tr>
<td>Level 3</td>
<td>18%</td>
<td>24%</td>
<td>18%</td>
<td>41%</td>
<td>6%</td>
<td>35%</td>
</tr>
<tr>
<td>Level 2</td>
<td>59%</td>
<td>0%</td>
<td>35%</td>
<td>0%</td>
<td>41%</td>
<td>18%</td>
</tr>
<tr>
<td>Level 1 - Not at all</td>
<td>18%</td>
<td>0%</td>
<td>35%</td>
<td>0%</td>
<td>53%</td>
<td>12%</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>2.0</td>
<td>4.0</td>
<td>2.0</td>
<td>3.0</td>
<td>3.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

Table 7. Paired Comparisons of Pre-Survey and Post-Survey Responses

1. “Comfort writing a program” analysis. The first pre- and post-paired survey question asked; How comfortable are you in writing a computer program? Since all teachers answered this question, n = 17 and was then adjusted for computing the Wilcoxon statistic to n_{reduced} = 16 due to a single pair having equivalent pre- and post-survey scores (i.e. a tie). The measured median difference was 2.0. The Wilcoxon statistic was w = 136 and the estimated median of difference = 2.00, which is an estimate of the median of the differences accounting for statistical measurement error. This test resulted in a p-value of p = 0.000, which is less than the significance level α = 0.05. These results suggested that the workshop experience can increase a mathematics teacher’s comfort in writing a computer program by a statistically significant amount.
2. “Comfort using a program to do/teach mathematics” analysis. The second pre- and post-paired survey question asked; How comfortable are you using computer programming as a tool to do and/or teach a mathematics course? For this test, \( n = 17 \) and \( n_{\text{reduced}} = 16 \) due to ties. The measured median difference was 2.0. The Wilcoxon statistic was \( w = 136 \) with an estimated median difference of difference = 1.75. The p-value was \( p = 0.000 \) which is less than the significance \( \alpha = 0.05 \). These results suggested that the workshop experience increased the teachers’ comfort in using computer programming as a tool to do and/or teach a mathematics course by a statistically significant amount.

3. “Comfort teaching programming” analysis. The third pre- and post-paired survey question asked; How comfortable are you teaching a course or unit on computer programming? For this test, \( n = 17 \) and \( n_{\text{reduced}} = 13 \) due to ties. The measured median difference was 1.5. The Wilcoxon statistic was \( w = 91 \) with an estimated median difference of difference = 1.50. The p-value was \( p = 0.002 \) which is less than the significance \( \alpha = 0.05 \). These results suggested that the workshop experience increased the teachers’ comfort in teaching a course or unit on computer programming by a statistically significant amount.

4. “Belief programming should be essential for mathematics education” analysis. The fourth pre- and post-paired survey question asked; How strongly do you believe that computer programming should be an essential part of mathematics education? For this test, \( n = 15 \) due to one teacher choosing not to answer on the pre-survey and another teacher on the post-survey and \( n_{\text{reduced}} = 11 \) due to ties. The measured median difference
was 1.0. The Wilcoxon statistic was $w = 56$ with an estimated median difference of $\text{difference} = 0.50$. The $p$-value was $p = 0.045$ which less than the significance $\alpha = 0.05$. These results suggested that the workshop experience increased the teachers’ beliefs that computer programming should be an essential part of mathematics education by a statistically significant amount.

It is worth noting however that this paired question showed the least amount of change between the pre-test and post-test of all of the paired survey questions, suggesting that of all of the paired topics studied, the workshop experience had the least impact on teachers’ belief that programming should be an essential part of mathematics education.

5. “Motivation to integrate programming into teaching mathematics” analysis.

This pre- and post-paired survey question asked; *How motivated are you to include computer programming in any mathematics course you teach?* For this test, $n = 16$ and $n_{\text{reduced}} = 11$ due to ties. The measured median difference was 1.0. The Wilcoxon statistic was $w = 62$ with an estimated median difference $= 1.00$. The $p$-value was $p = 0.011$ which is less than the significance $\alpha = 0.05$. These results suggested that the workshop experience increased the teachers’ motivation to include computer programming in mathematics courses they teach by a statistically significant amount.

**Characteristics and beliefs of workshop participants**

Pre-survey only questions were asked to aide in understanding the shape of the sample population for this study. This data was analyzed using descriptive statistics and a non-parametric, two-tailed, one-sample, signed-rank test to determine if the characteristics described by the survey questions were significantly present or absent in
the sample population. Since each question was asked on a 5-point Likert scale, the null hypothesis for the signed-rank test was that the median score was 3, neutral response.

1. How comfortable are you in using computer software such as word processors, e-mail, spreadsheets?

2. How strongly do you believe that calculators should be used in the secondary mathematics classroom?

3. How strongly do you believe that computer packages like Geometer’s Sketchpad, Geogebra, Fathom, Tinkerplots, Minitab, should be used in the secondary mathematics classroom?

4. How strongly do you believe that manipulatives should be used in the secondary mathematics classroom?

The results of these tests (see Table 8) suggested that the common teacher entering the workshop is very comfortable using common, general purpose computer software; does not have strong beliefs about using or not using calculators in the classroom; believes that common mathematics applications for the computer, such as Geometer’s Sketchpad, Geogebra, Fathom, Tinkerplots, and Minitab should be used in mathematics teaching at the secondary level; and believes that manipulatives should be used in a secondary level mathematics class. This teacher can be generally classified as computer savvy and in favor of the use of computer software and manipulatives in the mathematics classroom.
<table>
<thead>
<tr>
<th>Percent of Teachers responding to comfort levels and beliefs</th>
<th>1. Comfort level using general computer software</th>
<th>2. Calculators should be used</th>
<th>3. Computer packages should be used</th>
<th>4. Manipulates should be used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 5 Very high 71% 24% 35% 69%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 4 High 29% 18% 24% 31%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 3 Neutral 0% 53% 35% 0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 2 Low 0% 0% 6% 0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1 Very low 0% 6% 0% 0%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N= 17 17 17 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two-tailed, Signed-Rank test, Null: mu=3 (neutral), alpha = .05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
</tr>
<tr>
<td># below 3</td>
</tr>
<tr>
<td># equal 3</td>
</tr>
<tr>
<td># above 3</td>
</tr>
<tr>
<td>p-value</td>
</tr>
<tr>
<td>Result -significant Yes/No</td>
</tr>
</tbody>
</table>

Table 8. Teachers’ Beliefs on Use of Technology on Pre-Survey

Perceptions of future programming and problem solving improvement. The teachers were surveyed about whether or not they expect to program in the future and if they believe that the workshop experience improved their problem solving skills. These
Likert-scaled questions were deemed relevant only after the workshop experience and therefore were not asked on the pre-survey, but only on the post-survey. The non-parametric Signed-Rank test was used to compare the median with the neutral response of 3 (see Table 9).

<table>
<thead>
<tr>
<th>Level</th>
<th>1. Expect to program in the future</th>
<th>2. Believed problem solving improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 5</td>
<td>A lot</td>
<td>29%</td>
</tr>
<tr>
<td>Level 4</td>
<td></td>
<td>53%</td>
</tr>
<tr>
<td>Level 3</td>
<td></td>
<td>18%</td>
</tr>
<tr>
<td>Level 2</td>
<td></td>
<td>0%</td>
</tr>
<tr>
<td>Level 1</td>
<td>Not at all</td>
<td>0%</td>
</tr>
<tr>
<td>N=</td>
<td></td>
<td>17</td>
</tr>
</tbody>
</table>

Two-tailed, Signed-Rank test, Null: mu=3, alpha = .05

<table>
<thead>
<tr>
<th></th>
<th>Arithmetic Mean</th>
<th>Std Dev</th>
<th>Median</th>
<th># below 3</th>
<th># equal 3</th>
<th># above 3</th>
<th>p-value</th>
<th>Result -significant Y/N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.12</td>
<td>0.70</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>14</td>
<td>0.0001</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>4.00</td>
<td>0.94</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>12</td>
<td>0.0034</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 9. Expectation to Program in Future and Belief that Problem Solving Improved
1. Do you ever see yourself programming again for your own purposes or for another class?

2. How much do you believe your problem solving skills of the mathematics covered in this workshop, improved over the duration of the workshop?

Results of the two-tailed, Signed-Rank test (see Table 9) suggested that the teachers did expect to program computers in the future. Likewise teachers felt that their mathematical problem-solving skills did improve due to their workshop experience.

*Factors based on Characteristic and Background Questions.* The pre-survey also included a handful of background questions to determine if there might be certain key personal characteristics or backgrounds of the sample population that were statistically significant factors in certain workshop outcomes. These were “yes” or “no” questions about teachers’ backgrounds and personal characteristics that were used in conjunction with *paired comparison survey questions* to conduct basic factor analyses. For example, consider the background question; *Are you an in-service teacher or a Graduate Teaching Associate?* The pre- and post-survey differences on the paired comparison question; *How comfortable are you in writing a computer program?* can be analyzed to see if teaching experience is a significant factor in the workshops’ ability to increase teachers’ comfort in writing a computer program.

Significance testing was performed using a Mann-Whitney U test, which is a non-parametric test between two non-paired samples that tests the null hypothesis that the medians from both samples are equal. For this basic factor analysis, the two samples, call them sample A and sample B, were taken from the pre- and post-survey score differences
of each of the paired pre- and post-survey questions. Sample A included all differences from teachers who answered “yes” to a certain characteristics question and sample B included all differences from teachers who answered “no.” Results from the Mann-Whitney U test that were significant at the alpha = 0.05 level, suggested that the factor in question made a significant impact on that particular paired pre- and post-survey question.

For illustrative purposes, consider the paired pre- and post-survey question *How comfortable are you writing a computer program?* Significance testing showed that teachers’ Likert-scores improved significantly from the pre-survey to the post-survey. One of the characteristic questions on the pre-survey asks teachers; *Have you ever taken a course in probability?* For this characteristic question, 71% of teachers responded “yes” and 29% responded “no”. For those teachers that responded “yes”, the difference in their pre- and post-survey scores about their comfort writing a computer program were grouped together to form one sample. Similarly, for those teachers that responded “no”, their difference scores were grouped together in a second sample. A Mann-Whitney U test was then used on these two samples to see if the median differences between the samples were significantly different. This result suggested an answer to the question; *Is being a teacher a significant factor in the workshop’s ability to impact participants’ comfort in writing a computer program?*

Six questions listed below about the teachers’ backgrounds/characteristics were asked on the pre-survey and used in basic factor analysis with the five pre-and post-survey questions:
1. Does/did your high school (either the one you teach at or the one you attended) offer a programming course?

2. Are you an in-service teacher or graduate teaching assistant?

3. Do you currently use programming in any course you teach?

4. Have you ever taught a probability unit or course at your school?

5. Have you ever taken a probability course?

6. Have you ever taken a computer programming course?

Six factor-analyses for each of the five paired pre- and post-survey questions yielded 30 different factor analyses. In order to present the detailed results of significance testing in an organized and readable fashion, they are presented here in the tables below.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Difference of Post-Pre</td>
<td>YES NO</td>
<td>YES NO</td>
<td>YES NO</td>
<td>YES NO</td>
<td>YES NO</td>
<td>YES NO</td>
</tr>
<tr>
<td>N</td>
<td>4 12</td>
<td>15 0</td>
<td>10 10</td>
<td>9 6</td>
<td>12 5</td>
<td>14 3</td>
</tr>
<tr>
<td>Test</td>
<td>Mann-Whitney U Med-Diff Equal</td>
<td>NOT TESTABLE</td>
<td>NOT TESTABLE</td>
<td>Mann-Whitney U Med Diff Equal</td>
<td>Mann-Whitney U Med Diff Equal</td>
<td>Sign Test*</td>
</tr>
<tr>
<td>Test Stat</td>
<td>W=31.5</td>
<td>W=77</td>
<td>W=93</td>
<td>W=31.5</td>
<td>W=31.5</td>
<td>W=93</td>
</tr>
<tr>
<td>p-value</td>
<td>0.8038</td>
<td>0.5864</td>
<td>0.1182</td>
<td>0.0034</td>
<td>0.0034</td>
<td>0.0034</td>
</tr>
<tr>
<td>Significance at .05 level? Y/N</td>
<td>NO NA</td>
<td>NA NA</td>
<td>NA NA</td>
<td>NO NA</td>
<td>NO NA</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 10. Factors Affecting Teachers’ Increased Comfort Levels in Writing Programs

103
### Table 11. Factors Affecting Teachers’ Increased Comfort Level in Using Programming to Do or Teach Mathematics

<table>
<thead>
<tr>
<th>Factors affecting comfort level in using programming to do or teach mathematics</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Difference Post - Pre</td>
<td>2.5</td>
<td>1.5</td>
<td>2.0</td>
<td>1.0</td>
<td>NA</td>
<td>2.0</td>
<td>2.0</td>
<td>1.5</td>
<td>2.0</td>
<td>1.0</td>
<td>1.5</td>
<td>2.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>4</td>
<td>12</td>
<td>15</td>
<td>1</td>
<td>0</td>
<td>15</td>
<td>9</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>14</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>Mann-Whitney U</td>
<td>NOT TESTABLE</td>
<td>NOT TESTABLE</td>
<td>Mann-Whitney U</td>
<td>Mann-Whitney U</td>
<td>Mann-Whitney U</td>
<td>Mann-Whitney U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null Hypothesis</td>
<td>Median Diff = 0</td>
<td>Median Diff = 0</td>
<td>Median Diff = 0</td>
<td>Median Diff = 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Stat</td>
<td>W=40.5</td>
<td>W=108</td>
<td>W=85.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p-value</td>
<td>0.47</td>
<td>0.377</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Significance at .05 level?</td>
<td>NO</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 12. Factors Affecting Teachers’ Increased Comfort Level in Teaching a Programming Class or Unit

| Factors affecting comfort level in teaching a programming class or unit | YES | NO | YES | NO | YES | NO | YES | NO | YES | NO | YES | NO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Median Difference Post - Pre | 2.0 | 1.3 | 1.5 | 0.0 | NA | 1.5 | 1.5 | 1.5 | 1.5 | 1.0 | 2.0 |
| N | 4 | 12 | 15 | 1 | 0 | 15 | 9 | 6 | 12 | 5 | 14 | 3 |
| Test | Mann-Whitney U | NOT TESTABLE | NOT TESTABLE | Mann-Whitney U | Mann-Whitney U | Mann-Whitney U | Mann-Whitney U |
| Null Hypothesis | Median Diff = 0 | Median Diff = 0 | Median Diff = 0 | Median Diff =0 |
| Test Stat | W=39.0 | W=75.5 | W=107 |
| p-value | 0.47 | 0.717 |
| Significance at .05 level? | NO | NO |

### Table 13. Factors Affecting Teachers’ Increased Positive Beliefs that Programming is Essential in Mathematics Education

<table>
<thead>
<tr>
<th>Factors affecting belief that programming is essential in Mathematics Education</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Difference Post - Pre</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>NA</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>3</td>
<td>11</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td>9</td>
<td>5</td>
<td>11</td>
<td>4</td>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>Mann-Whitney U</td>
<td>NOT TESTABLE</td>
<td>NOT TESTABLE</td>
<td>Mann-Whitney U</td>
<td>Mann-Whitney U</td>
<td>Mann-Whitney U</td>
<td>Mann-Whitney U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null Hypothesis</td>
<td>Median Diff = 0</td>
<td>Median Diff = 0</td>
<td>Median Diff = 0</td>
<td>Median Diff =0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Stat</td>
<td>W=20</td>
<td>W=61.5</td>
<td>W=81.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p-value</td>
<td>0.727</td>
<td>0.386</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Significance at .05 level?</td>
<td>NO</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 14. Factors Affecting Teachers’ Increased Motivation to Add Programming to Mathematics Classes Taught

Despite the large amount of significance testing data presented in the tables above, the results are easily summarized. Of the 30 basic factor analyses conducted (5 sets of median differences compared with the six factors), only one yielded statistically significant results. The only statistically significant factor for any of the paired pre- and post-survey questions was that having never taken a programming course before was a significant factor on the impact that the workshop made on teachers’ comfort in writing computer programs. Teachers who had never taken a programming course before improved their comfort in writing a computer program by a median average of 3, while teachers who had some previous programming experience only improved their comfort by a median average of 2. Those teachers who had never taken a previous programming course improved their median comfort level from 1 to 4, which meant the workshop effect raised their comfort level from uncomfortable to very comfortable, and the difference was statistically significantly more than those who had taken a previous
programming course who only raised their comfort level by two levels from
uncomfortable to comfortable.

Interestingly, those without previous programming experience became more
comfortable with programming during the workshop than those entering with
programming experience. It should be noted that this factor analysis was the only one of
the 30 that used a Sign Test rather than a Mann-Whitney U test, which is due to the fact
that the standard deviation of the results of those teachers’ that had no previous
programming experience was zero and a standard deviation of zero violates the
assumptions of the Mann-Whitney U test. This situation was only an issue for this one
factor of the 30 and for all other analyses a Mann-Whitney U test was used.

This factor analysis also proved to be problematic for 10 of the 30 analyses. All
analyses based on the characteristics, Are you an in-service teacher or GTA, and Do you
use computer programming in any classes you teach could not be conducted with any
significant meaning, because of the large homogeneity that existed amongst the group.
Of those that responded to the; Are you an in-service or GTA question, 94% responded
“yes”. Of those that responded to the; Do you use programming in any classes you teach
question, 100% responded “no”.

Reliability and validity

In order to establish reliability for the pre- and post-surveys, internal consistency
between related survey questions was measured using the Cronbach’s alpha reliability
coefficient. The alpha coefficient ranges between 0 and 1 with a higher value indicating a
higher internal consistency between survey questions. An alpha score greater than or
equal to 0.70 is widely considered to denote sufficient internal consistency in social
science research (Cronbach’s Alpha, 2016).

Cronbach’s alpha coefficients were calculated for three different sets of survey
question results; pre-survey and post-survey questions that categorically survey teachers’
general comfort with computer programming and pre-survey questions that categorically
survey teachers’ beliefs in using technology in the teaching and learning of mathematics.
These were the only germane categories with a large enough number of survey questions
in a set to allow for meaningful measurements of internal consistency.

The pre-survey and post-survey general comfort with computer programming sets
were comprised of the following three pre- and post-survey questions:

1. How comfortable are you in writing a computer program?
2. How comfortable are you using computer programming as a tool to do and/or
teach a mathematics course?
3. How comfortable are you teaching a course or unit on computer programming?

The belief in using technology in the teaching and learning of mathematics set
was comprised of the following three pre-survey questions:

1. How strongly do you believe that calculators should be used in the secondary
mathematics classroom?
2. How strongly do you believe that computer packages like Geometer’s Sketchpad,
Geogebra, Fathom, Tinkerplots, Minitab, should be used in the secondary
mathematics classroom?
3. How strongly do you believe that manipulatives should be used in the secondary mathematics classroom?

For the pre-survey set of questions categorized by teachers’ general comfort with computer programming, alpha = 0.8 and for the categorically equivalent post-survey set, alpha = 0.7. For the pre-survey set of questions categorized by teachers’ belief in using technology in the teaching and learning of mathematics, alpha = 1. These alpha coefficients suggested that internal consistency between conceptually similar survey questions was sufficiently reliable.

Of note is that the extremely high alpha coefficient of 1 for teachers’ general comfort with computer programming (post-survey) may indicate that the questions in this category were not uniquely interpreted by the teachers and rather were seen as simply a singular question being rephrased. However, since the same questions on the pre-survey yielded alpha = 0.8, it may be likely that the questions are not in-fact redundant but rather that the workshop experience clarifies the teachers’ positions on what they believe they can comfortably accomplish through computer programming and its application on the teaching and learning of mathematics. This hypothesis needs further and more direct study.

The dissertation committee established content validity of the pre- and post-surveys through expert content review.
Journals Analysis

Qualitative analysis of the teachers’ journals was used to describe the workshop through the meaningful experiences that their journals describe. In order to most accurately capture the essence of the workshop, the journals were thoroughly read and descriptive themes were extracted. These themes were then organized in a hierarchical structure and the journals were then re-read and coded to find quotes that supported and/or characterized the nature of the themes. Themes that contained several similar descriptions from teachers or had supporting quotes of strong impact, relevancy, and conviction, were kept as strong thematic representations of the workshop. Themes that were not supported strongly by the journals were discarded.

The combining of all journals into a single, master-journal resulted in a master-journal that was 47 pages long, with 22,688 words and 2,138 lines of text. Thematic extraction of the journals revealed a description of the workshop that was categorized into three major themes; workshop structure, learning processes, and personal outcomes. Each of these categorical themes contained various sub-themes, or sub-topics, that were well supported by the journal data. The thematic hierarchical structure, developed from the journal data, is as follows (see Figure 7):

The following journal analysis will elaborate on these themes and explore how the themes helped describe the workshop experience. All teacher quotes can be found in Appendix D where a transcription of the master-journal appears. All quotes reference the teacher alias who is quoted and the line number in the master-journal where the quote begins.
I. Workshop Structure
   A. Comparison to previous coursework
      1. Previous programming coursework
      2. Previous probability coursework
   B. Content structure
      1. Scaffolding/sequencing
      2. Pacing
      3. Challenge problems
   C. Pedagogy
      1. Group work (almost always as pairs)
      2. Primary learning/ constructivism
      3. Instruction
II. Learning Processes
   A. Programming strategies
      1. Debugging: playing computer/tracing vs. trial-and-error
      2. Templates/Skeletons
      3. Internet searching
   B. Probability Strategies
      1. Experimental vs. theoretical
      2. Verification
      3. Difficulties with theoretical probability
III. Personal Outcomes
   A. Emotional reaction
      1. Fun, engagement, and personal satisfaction
      2. Frustration
   B. Progression of comfort
      1. Syntax comfort
      2. Programming structure comfort (e.g. conditional statements, loops, etc.)
   C. Future applications
      1. Teaching implications
      2. Personal problem solving implications

Figure 7. Thematic Hierarchical Structure for Journal Analysis
**Workshop Structure**

*Comparison to previous coursework.* Several teachers compared their workshop experiences to similar courses they had taken previously. These comparisons were primarily with other programming courses the teachers had taken previously or with other mathematics courses that contained probability.

Teachers who had previously taken a programming course were more likely to describe their first couple of days in the workshop as successful or easy. These teachers were often able to draw from their past experiences to more quickly and easily accomplish the early programming tasks in the workshop. These early tasks are more focused on programming syntax, basic constructs (like input/output), and the process of compiling and executing, than mathematical and/or logical challenges.

So far I have been using strategies I recall from learning *Python* before, but basically my strategy so far when writing a program has looked like: create the file, copy the shell in, type in the first prompt/scan/input bit, then compile the program and run it. (Esther, 530)

The workshop was designed with mathematics teachers as the target audience due to the conjecture that mathematics teachers had not only the mathematical background to be successful programmers, but were sufficiently trained as logical thinkers and could meet the highly logical demands of computer programming. Without a strong background in logical reasoning it was believed that workshop participants would struggle to absorb the workshop material at the pace at which the workshop progresses. Lester corroborates this conjecture, saying:
I am finding that it really helps to have taken… Intro to Num Analysis class. Having done so makes things like logic, structure, loops with counters, etc. more intuitive and I catch on more quickly. (Lester, 1148)

Several of the teachers had recently taken courses with a substantial focus on counting theory and probability. However, many teachers found that the workshop challenged their belief that the theoretical probability they had done in the past was sufficient for solving a wide range of basic probability problems. Cory expressed embarrassment that she was soon graduating with a mathematics degree and yet the workshop activities revealed that she knew very little theoretical probability. Greg suggested that the probability course he had taken just the term before was surprisingly insufficient for solving the basic probability problems presented in the workshop:

“I did not feel as confident as I should have on the [probability] pretest, just having taken a probability course in the spring! However, I suppose I completed the problems with no notes or resources. I barely used a calculator. The programming I felt good about… “ (Greg, 807)

*Content structure.* Journal data suggested that the sequencing and pacing in which content was delivered to the teachers was appropriately challenging for most of their needs as mathematically competent, budding programmers. Many teachers commented on the ease of early programming activities and the quick ramp up to much more difficult programming activities by the second week. Molly said that “… the problems really increased in difficulty over the course of the week” (Molly, 1538) and Alice corroborated:
This week [week 2] I feel has brought a lot more of fun challenges than the previous week. Last week we just worked on a lot of small problems that were not much of a challenge sometimes and just took a few minutes. With these problems there requires more problem solving and more critical thinking. (Alice, 125)

Generally, the teachers found that the pace of the workshop fit their learning needs well and helped them with the steep difficulty increase. Molly and Mort found that “The challenge has been at the right level” (Mort, 1583) and that “… the instruction was perfect in pace and content” (Molly, 1498). Muriel noted that “Time efficiency is being spent well and the class has a good atmosphere” (Muriel, 1661).

There was only one teacher, Greg, who disagreed about the appropriateness of the pacing of early workshop activities. Greg said in a conversation, “I did well at first, but fell behind trying to takes notes on what I had already completed to retain for the future.” With the large amounts of syntax teachers are exposed to in the first two days of the workshop, giving them premade notes of programming commands and their general usage may be a positive addition for future workshops.

Frequently the teachers approached new programming activities by building on the success of previous activities. The scaffolding of the activities helped them succeed in the face of rapidly growing challenge and complexity by harvesting code they deemed relevant from previous activities and figuring out how it applies to the new activity. The scaffolding of programming activities helped Alice have great success early on:
… these programs required a bit of direction, but mainly just using previous exercises as well as playing around with the program [helped] to achieve the desired result… by doing this I could see how we can take a simple command and manipulate it enough to do something that I did not [at first] relate to it at all.

(Alice, 40, 52)

When Melinda found it difficult to progress in a new activity that required nested loops, she remarked that it was because “I very much did not feel comfortable working through nested loops without any scaffolding around it” (Melinda, 1445). Melinda’s response suggests that when teachers were not sufficiently able to draw on knowledge from previous and more fundamental activities, content scaffolding breaks down and problem-solving ability suffers.

The frequent usage of challenge activities that extended assigned activities for those teachers who finished their assigned activities more quickly, or wanted to take on additional challenge, was an important structural component for smoothing out pacing in the workshop. Shawn said that, “I enjoy when there are specific extensions to each [activity] so that we can work at our own pace and feel comfortable with our results” (Shawn, 1857). Muriel agreed, saying:

It is kind of fun that [Instructor] gives extra programs for a challenge for some people who like and understand the coding so they can progress further. I was able to do some of the extras but they are quite the brainteasers. The structure of the classes moved fluidly and has great time management of having work time
and class time where we can discuss as a class or individually with our peers.

(Muriel, 1785)

Challenge activities also helped generate interest and excitement in the workshop. Kate specifically “enjoyed the challenges on the change activity” (Kate, 1050). Lester and Victor were inspired by the challenge activities to create their own problems. When Lester extended the activity that randomly samples a square to approximate pi by using Simpson’s Rule to accomplish the same task, his excitement was palpable. He said, “I am now working on making a program which uses simpson’s [sic] rule to approximate pi instead of just throwing darts [i.e. randomly sampling]. It will be epic!” (Lester, 1167). Victor had a similar experience, saying:

Today was a blast yet again… I actually spent a fair amount of time working on creating a factorial function program. This was inspired by the fact that near the beginning of the class we received a challenge project of determining the theoretical probabilities of the birthday problem, and in the process I came up with a formula that involved factorials. (Victor, 1901)

In the workshop, a key aspect of teacher preparation involves teachers identifying how computer programming can be used to solve problems that are beyond the focus of the workshop and germane to the classes they teach. Teachers having these types of content creation moments heavily contribute to a successful workshop experience and the journal data suggests that challenge activities strongly contribute to the occurrence of these moments.
Pedagogy. Journal data richly described the workshop pedagogy as strongly collaborative, with primary learning and constructivist experiences driving teaching and learning. Collaboration primarily took the form of paired or group work on programming activities. Some teachers found that one of their greatest problem solving resources was each other:

… we worked in groups today. This worked very well for me actually. The way we approached each problem was by first each telling each other how we thought we could approach the problem. If both of us had some similar ideas we would start there and begin typing up the code we thought would work. As we typed we continued to problem solve and as ideas came to us we each suggested it and worked off of that. This was very helpful because at times my partner could have an idea that might have not come to me right away and it would have taken me a while to come across it. Also there were times this I knew I wanted to do something but I didn't have the path to get there perfectly planned out yet. With a partner we were able to work through this problem and figure out how to get to the goal. (Alice, 93)

Overall I think working with someone worked very well and it got a lot more work done than I would have by myself. It was also a great opportunity to learn from someone else's ideas and bounce and get an insight into how someone else approaches these problems. (Alice, 116)

Melinda, Denise and Lisa agreed with Alice’s analysis on the learning benefits of paired work. Melinda said that “It was extremely synergistic and helpful working in a
pair... because I feel like I learned more than if I was alone” (Melinda, 1406, 1416).

Denise added that “... discussing some of the strategies used by other classmates helped me understand logic a lot better” (Denise, 470). Lisa noted that “The real benefit [of a partnership] I think was being able to talk out the since [sic] with each other. How is our loop working? What are we asking the program to do? Is that what we want?” (Lisa, 1239).

Working in a partnership was also beneficial for minimizing programming mistakes (often known as bugs). Lester said that “I found it easier to work with a partner than by myself, mainly because having two sets of eyes engaged on the code makes it much less likely that a small typing mistake will cause five minutes of confusion and nose-wrinkling…” (Lester, 1162). Lisa agreed that “Finding errors became easier because there were two pairs of eyes on the program” (Lisa, 1237).

Muriel was the only teacher in the workshop who expressed feeling overwhelmed by the pace, syntax, and programming logic throughout the majority of the 3-weeks. For her, working in a partnership became an essential commodity for progressing through the curriculum.

I just like having someone with me (thankful for [Wilma]) who will talk with me... I feel very fortunate to have [Wilma] with me since she has been a MAJOR help for my work in these programs... Partners is definitely a preference of mine rather than going alone with coding. (Muriel, 1685, 1698, 1741)

Cory offered a very unique perspective on the benefits of working in a partnership:
Due to the nature of working in a partnership, the conversation was brought out into the open where we could think out loud and catch each others' thought-flaws before they were set down, and in general I felt more "invited outward" -- if that makes sense… folks spoke more freely with [Instructor] as a result of working with a partner, and I think that this makes perfect sense: the outward space was already engaged, obstacles toward it already overcome.

Also -- and I think this is interesting -- because math-type thinking can sometimes be a bit slippery within the mind, there is something specific that takes place when it is set outside: something like setting it on solid ground. (Cory, 334)

Cory’s notion that communicating programming and mathematics verbally could help frame the problem at hand, guide in its solution, and facilitate conversation with the instructor, was intriguing and may necessitate future research.

Journal data revealed only a few, minor criticisms of group work. Shawn found that communication could be difficult when working with a partner and that the “… challenge was trying to understand each other’s logic” (Shawn, 1843). Molly was not strongly for or against group work, but did note that some individual internalization of a problem did need occur before group work could be effective; “I need to be able to process through the problem by myself first” (Molly, 1524). Despite Cory’s affinity for group work, she did share in Molly’s opinion. Cory and her partner would consider a new problem individually before they would come together and collaborate.

Daily activity in the workshop was primarily spent by teachers’ creating computer programming solutions to mathematical problems. Very little guidance was given to the
teachers about how to approach a problem until they first spent time internalizing it and reconciling it with previously held knowledge. Denise found the workshop’s hands-on learning structure to be “… very interesting because I got to write the program on my own and debug it too” (Denise, 397). Melinda agreed, saying that “… the format of the class is really great… I really appreciate the focus on primary and hands-on learning within the classroom environment” (Melinda, 1290, 1306).

The interactive nature of computer programming combined with the near-instant feedback of compiling and running a program, lends itself well to a primary-learning approach and this was well suited for the learning needs of some teachers. Esther said, “I definitely learn best by doing, and I find it easiest to think through my code by writing it…” (Esther, 595). The workshop’s approach also can promote engagement through a sense of ownership in the process and product:

I feel invested in my programs, Wednesday we left without our program working properly and it was stressful to leave. I thought about it the entire drive home! (Lisa, 1248)

Not all understanding could be achieved through primary-learning experience alone. Realizing this, the instructor often discussed a solution or two to a problem before moving on to the next problem-driven programming activity. Some problems were even introduced with a class-discussion about the problem conceptually before the teachers would begin programming. In some cases, instructor-guided discussion did aide in teachers’ understanding of workshop concepts. Muriel said she “…love[d] having class discussions to talk through the programs before even [sic] writing the program so we also
understand how it’ll work… thank you [Instructor] for sharing your codes after giving us some time to let us figure it out ourselves!” (Muriel, 1675, 1770).

Some teachers found that occasionally there was not enough time spent on instructor-led, group discussion. Mort lamented the fact that the birthday probability problem was not discussed as a group in great enough detail and felt that the instructor’s review of the problem before moving on to the next activity was not adequate. Mort said, “… I tend to lose [sic] focus with the teacher [i.e. Instructor] walking through programs, [it] may be nice to have people explain their own programs” (Mort, 158). With limited contact time in the workshop, it may be important to further study the juxtaposition between learning through guided, group discussion and learning through hands-on programming experiences. The journal data suggests both approaches are important to teacher development and further study may reveal the best percentage of time spent on each.

A core pedagogical practice to the workshop is for the instructor to reveal syntax on a need-to-know basis and to introduce new programming constructs (such as conditional statements and loops) through the context of new problems that practically require these tools for any solution. This practice generated quite a bit of puzzled commentary in the teachers’ journals. Cory said, “I wouldn’t mind more emphasis on ‘doubles’ and how/why the computer reserves bits and data… it really is helpful to understand when developing programs” (Cory, 268, 269).

Several students were intrigued by the pseudo-random number generator used in the workshop, called drand48. The instructor shared the fact that the random number
generator was not truly random, but that an algorithm was used that delivered sufficiently random results for precise simulation to the thousandth’s place. Many teachers wanted to know what the 48 meant. Esther noted in a conversation that the “[w]hole random thing is interesting. It makes me wonder how the computer is working behind the scenes.”

Further study may be warranted to determine if learning is positively affected by keeping syntax, language structure, and computing details to a minimum, or if teachers’ learning needs would be better met with more detailed explanation of what occurs behind the scenes. However, the workshop creators believed that keeping students in the dark with regard to certain extraneous programming and computing details would keep the workshop focused on problem solving and minimize a slow pace that could jeopardize coverage of the curriculum.

Learning Processes

Programming strategies. A significant portion of the journal data described programming strategies that teachers found effective in the workshop. A few teachers made reference to playing computer on their code to debug logic errors. Playing computer was a term used by the workshop instructor that is more commonly known in programming circles as tracing. The tracing process involves going through each line of the code, executing each command as the computer would at run-time (including all branching for conditional statements and loops) and keeping track of the changing values of variables. Monitoring how the values of the variables change often reveals patterns that lead to an understanding of why the code is giving different results than expected or intended.
Several students benefited from using this debugging strategy. Denise said, “I… applied the commands and noted down the results after each step. This process helped me see if the sequence of commands I used was correct and if it produced the desired results” (Denise, 433). Victor agreed with Denise that “[t]here are subtle issues which I learned to resolve by trying to think like a computer” (Victor, 1887). Kate noted that playing computer helped her with the programming activity where the teachers are tasked with finding and outputting the smallest number in a series of integers; “I ‘played computer’ using a small example of only three data points. This showed me what I was missing in order to find the smallest value in the list” (Kate, 1091).

Greg and the group that he worked with internalized elements of tracing into a general strategy for program development:

… I learned a useful problem solving skill for thinking through programs: the simple idea of working through your code logically on paper. Make some if then statements yourself for what might happen if you (the user) do this or that and follow the logical steps in the code to explore. I feel that we continued to use this sort of problem solving as we wrote our code in groups for the following activities, and this helps us to catch errors or syntax we were missing at first with this “check the steps” mentality. (Greg, 861)

The instructor introduced the tracing technique to everyone within the first two days of the workshop, but it was often underutilized, or ignored entirely, by the teachers until programs became more complex in the second week. Even then, some teachers never gravitated to a tracing approach to debugging faulty logic. Trial-and-error was a
far more popular programming strategy in the workshop. Lisa used “[m]ostly just trial and error” (Lisa, 1199) to fine-tune the algorithm she used to solve the change problem. Similarly, Wilma characterized programming as a “trial and error process” (Wilma, 2063). Some teachers seemed reluctant to trade the trial-and-error programming strategies that they used so successfully early in the workshop for more logical, algorithmic tracing strategies that tended to be more successful on more sophisticated problems.

This may have been due to the nature of the sequence of activities in the workshop. The first few activities in the workshop were generally easy and uncomplicated. For these early, less-sophisticated programs, trial-and-error often yielded quick and correct fixes and were a natural strategy to reach for when the computer can compile and execute code and give results so quickly.

While trial-and-error is an integral part of exploration and should not be discouraged, it is possible that early activities designed around tracing might be needed to encourage all teachers in the workshop to reach for this strategy. As programming tasks grow in complexity, often trial-and-error leads only to unfruitful changes to the code and frustration. Not all teachers consider using tracing to help them fix their error. Being able to resolve logical issues in their own code is an important aspect of preparing the teachers to be independent programmers.

Another programming strategy, suggested by the instructor, was to use previous programs as templates or skeletons for new programs that would minimize the need to take additional notes on syntax and programming constructs. This became an integral,
early programming strategy for many teachers in the face of the fast pace at which the workshop initially progressed. The teachers may not have fully understood the reasons for a main function and the inclusion of header files, but they knew these components were necessary for their programs to compile. As they found similarities from one program to the next they would copy and paste large sections of foundational code from a previous activity into the new activity. Greg’s strategy was to “begin with a program template, copied and pasted from a word document holding my previous programs” (Greg, 752). Several other teachers began each program with the same approach as Greg.

This approach became less necessary as they gained more comfort with the C programming language. Some teachers realized that as programming activities grew in complexity, less pieces could be copied and pasted from previous code and still be relevant without significant modification. Shawn said, “I found that it’s still much easier to start from scratch and reconstruct the program by copying and pasting pieces of code into it [when relevant]. If I’m just starting from a previous program, there are too many specifics that only work for the specific context and [are] not useful for the new program” (Shawn, 1835). That some teachers found that copy and pasting became less useful as the workshop progressed was actually desirable.

Copying and pasting and using skeletons/templates were useful in the beginning to keep the pace of the workshop moving and minimize frustration with syntax, but were not strategies that the workshop creators ever intended to become habit-forming. The creators did not want these techniques to become a crutch to lean on or to replace critical
thinking. Fortunately, journal data showed no evidence that these strategies ever negatively impacted teacher development.

A couple of teachers mentioned that they used the Internet to search for pre-made functions that could be used in their programs to accomplish specific programming tasks. Wilma specifically used Internet searching as one of her general approaches for solving each activity; “If I do not know the code that I need then I google it…” (Wilma, 2062). Kate turned to the internet too when she recognized a need to sort a list of numbers and thought that there might be a function that could do it for her:

I googled [sic] to find out if there is a sort function in Linux [C programming language], and it looks like there is. However, it looks complicated, and I have never used it before. Therefore, I am trying to write my program using only while loops and if statements, but I have not succeeded yet. I am wondering if it would be simpler to use the sort function or if I am on the right track with the while loops and if statements. (Kate, 1082)

Reaching for the internet to find pre-made functions (a.k.a. components) showed both ingenuity and self-sufficiency, which are both important characteristics to have if the teachers are to grow beyond the workshop and implement computer programming into the courses they teach. It is also quite common for computer programming classes to promote this type of functionalized, componentry-programming for software development. However, internet searching is not a strategy introduced or promoted in the workshop, because it violates the principle of keeping syntax to a minimum and opposes the philosophical message that with only a few, basic programming
structures/components (e.g. loops, conditional statements, etc.), one can create extremely
diverse and powerful programs. In other words, the workshop intends to display the
power and versatility of the basic, building-block fundamentals of computer
programming.

The journal data did not suggest that internet searching, as a programming
strategy, provided enough benefit to begin promoting its use in the workshop, but specific
study on this topic may be appropriate.

*Probability Strategies.* An extremely important realization that some teachers had
during their workshop experience was that theoretical probability and experimental
probability (through Monte Carlo modeling and simulation) require very different
strategies and thought processes for solving a probability problem. Wilma was one of the
teachers who had recently taken a theoretical probability course the term before the
workshop and contrasted the two approaches to probability problem solving thusly:

I appreciate the link [from the workshop] back to last quarter’s probability work.
I think that it is very powerful to revisit similar [probability problems as]
programs but explain and solve them differently. I appreciate that although we
are using simulations to solve problems we are still being pushed to explain using
theoretical probability. I realize that I have to think much differently to solve a
problem by theory VS computer simulation. One thing that I have worked on this
week is the ability to switch back and forth between these two ways of thinking.
(Wilma, 2182).
Melinda seemed to have had an expectation that learning to program Monte Carlo techniques for solving probability problems would also help her solve problems theoretically. Her workshop experience quelled that expectation:

In terms of working on the theoretical probability of the situation [the craps problem], I am not sure if coding it truly helped to think of what the probability would be. I believe that doing the coding helped me think through the physical process of playing this game of craps. In order to calculate the probability, I thought of doing 6*6 for the rollings [sic] of the two dice and then counting up the different ways in which you could get a 7 or an 11. For me there was a pretty big disconnect between thinking through the programming of the game and counting the probability of the chosen situation. Mentally, it felt like I had to take my head out of thinking in code and programming in order to solve the problem of the probability. In order to do this, I moved out of working on the computer and began writing the number combinations out on paper and thinking of the sample space. (Melinda, 1364)

It is unclear from the journal data if Melinda’s expectation that Monte Carlo problem solving expertise would translate to improved theoretical probability problem solving ability, was shared by other teachers. All the teachers were ever told by the instructor was that they would get the chance to experience how computer programming could impact mathematical thinking through probability problem solving. Did Melinda’s expectation arise from the belief that programming Monte Carlo simulation models was
not the same as doing mathematics and do other teachers share this belief? This is an important question that needs direct study in the future.

Although Melinda may have been disappointed that her probability programs did not improve her theoretical probability problem solving skills, she did make it very clear that she saw a potential application for using computer programming and Monte Carlo simulations to develop probability sense in her students and for convincing them of probabilistic results that may be counter-intuitive:

I really, really liked that [Instructor] mentioned about why it might be nice to use a computer program like this in a middle school classroom or something of that nature in order to prove your results. I have students simulate different probabilities in my classroom and it is always very hard to convince them of it. It is also something that I struggle with when I think of probability, because even though it can show a pattern, it isn’t going to guarantee what is going to happen on your next roll. Every year I have some students who are really able to think abstractly, can understand the mathematics behind the ideas, and accept the fact that the probability will model the situation. There are other students who are like, “Yeah, but that number means nothing to me because when we all did the tests, even though 1/6 of the time I was supposed to roll a 2, I got a 3 or 4 every time.” It is very difficult to engage those students in the rest of the material because it diminishes a little bit of trust between the teacher and the student. I find that if the student doesn’t trust the things that we are learning, they have a very hard time with the material and often fail to meet the objectives set forth in
the lesson, simply because they can’t get over the notion of probability being imperfect. I wonder if a computer program could be a valuable tool to help students overcome this disbelief. (Melinda, 1382)

Melinda’s observation lends support to the notion that the workshop can convince mathematics teachers that computer programming may be leveraged to strongly impact mathematical learning.

Several teachers found that experimental, Monte Carlo approaches to solving probability problems synergized with theoretical approaches so that they could offer a method for verifying the other approach. To trust in her experimental result after programming the craps problem, Molly said she “… verified my [experimental] result by finding the theoretical probably [sic] using the sample space of rolling 2 die” (Molly, 1517). Greg and Lisa had similar comments about verification. Greg said that, “[b]eing able to move back and forth between the theoretical and simulation probability has been helpful for problem solving approaches when coding, or simply just check to see results” (Greg, 953). With regard to the craps program, Lisa said, “[my program] gave me a probability of 0.222, which matches my theoretical probability of 2/9 or 0.222, so I believe!” (Lisa, 1229).

For some teachers, the workshop’s Monte Carlo simulation activities led to self-realization that probability problems could be more difficult to solve theoretically than their mathematical abilities allowed for. Compared to a statistics course emphasizing probability over the course of an entire term, the limited duration of the workshop involved teachers only minimally in Monte Carlo programming techniques. Despite this
limited exposure, most teachers who struggled to solve workshop activities theoretically had great success solving the problems experimentally. Cory noted that:

… I’m beginning [week3] to realize how uncomfortable I am with theoretical probability, and how uncomfortable that fact makes me given that I’m supposed to hold a math degree soon! I’m glad that I am catching this blind spot now… [Monte Carlo programming] is providing a good start and a really nice way of approaching [probability] more confidently. (Cory, 365)

Similarly, Greg said:

I did not feel as confident as I should have on the [theoretical probability] pretest, just having taken a probability course in the spring! However, I supposed I completed the problems with no notes or resources. I barely used a calculator. The programming [today] I felt good about… (Greg, 807)

**Personal Outcomes**

*Emotional reaction.* Showing teachers that computer programming can be beneficial for mathematical thinking and problem solving is not the only way the workshop attempts to convince teachers to integrate computer programming into their mathematics curriculum. The workshop is also designed to show teachers that computer programming can help make mathematics fun and engaging, therefore motivating them to include it in the mathematics they teach. Fortunately, the journal data strongly suggests that the teachers had a lot of fun, were very engaged, and found programming personally satisfying. With regard to having fun and being engaged in the workshop, the teachers said:
This is fun! (Cory, 266)

The pi problem was fantastic. (Barry, 248)

I am enjoying writing the programs, since it is like solving logic puzzles… I am keeping up well with the programming and having fun solving problems! (Esther, 601, 610)

[day 4] Still fun so far! [day 6] The conditional probability application was particularly fun to think about! (Greg, 819, 962)

It has been a fun experience and a lot more fun than when I was completing this course in my undergrad. Application is KEY!! (Lisa, 1251)

I really enjoyed myself today! ... I greatly enjoyed the class. (Molly, 1484, 1556)

[The Instructor] does a wonderful job making the programs relate-able [sic] for real life and being fun… It is kind of fun that [Instructor] gives extra programs for a challenge for some people who like and understand the coding so they can progress further. (Muriel, 1662, 1786)

I enjoy the programming. I find all of it interesting. (Wanda, 1962)

Today was a blast yet again. (Victor, 1901)

Several teachers expressed how exciting and personally satisfying it could be to complete a program and often times that satisfying sense of accomplishment was linked to being challenged:

I felt good about this week, and there were some challenge problems that I felt particularly proud of myself for finishing and making a successful program. (Esther, 600)
I was excited about finishing the first challenge… I was able to apply an if statement! … [Week 3’s] programs are tough and make you feel very successful when you complete the program successfully. (Lisa, 1202, 1247)

So I have working code. WOOO HOO!! (Wilma, 2176)

… when it worked [primes program] I felt like a computer wizard. (Barry, 223)

Unfortunately, frustration could be a fairly common emotion in the workshop. Debugging logic problems could be time consuming and a small programming error sometimes led to very confusing results during runtime. Molly said that she … really messed up the Birthday problem and had to leave [for the day] with it unfinished which was frustrating… [It] took me the longest to write. I had not thought of the probability correctly” (Molly, 1544, 1549). Muriel remarked that “[i]t becomes frustrating to be on the brink of getting the code entirely right but there are just simply minor errors that need to be noticed and fixed for everything to work the way that it should” (Muriel, 1738).

Frustration was likely an unavoidable consequence of writing computer programs, however the journal data suggested that for most teachers frustrations were manageable and overall the workshop experience was positive. Muriel found that working with a partner could help alleviate frustrations: “Talking out your thoughts can make a difference rather than being frustrated with just yourself and the computer” (Muriel, 1758). Is it possible that overcoming manageable frustrations was a key component in the high incidence of personal satisfaction that teachers experienced in the workshop? This is a question that could merit further study.
Progression of comfort. A common criticism of using computer programming and especially the C programming language, as a tool for mathematical problem solving is that notion that getting comfortable with the syntax of the language and the logical construction of programming takes too long. Since each of the teachers’ journal entries corresponded to a date, the journal data was able to offer a description of how teachers programming comfort may have progressed.

Most teachers that reported on their comfort with C programming described feeling comfortable with syntax before the end of the first week and the logical use of programming constructs (e.g. conditional statements and loops) during the second week. The first week ranged from June 22 through June 25, the second week from June 29 through July 2, and the third week from July 6 through July 9.

By the end of June 24, the third day of the workshop, Denise and Muriel both reported increased comfort with programming language syntax. On June 22, the first day of the workshop, Denise struggled with syntax. She said, “I learned that I have to be very careful about the syntax… I fell a little behind a couple of times but eventually caught up with [help from the instructor]” (Denise, 396, 399). By the end of June 24, Denise reported that “I am feeling more confident about writing the programs now than three days ago and it feels really good! … Now I can spot the mistakes in syntax much faster than before” (Denise, 419, 423).

On the same day Denise reported increased comfort with C programming syntax, Muriel similarly reported increased comfort with syntax while also revealing that the logical development of a program was still a struggle:
I am having some difficulties of trying to think in a programming way because it
doesn’t seem to fit within my brain too well… [but] I feel like I am growing to be
a better programmer than I was a couple days ago [because] [t]he problems that
are caught by program [sic] so it can run are easily noticeable and I can fix them
and go about what I need to do.  (Muriel, 1643, 1648)

By the end of June 30, the second day of the second week of the workshop,
Muriel began to feel that with some assistance and reassurance from her partner, she was
also beginning to gain some comfort with programming logic.  She said, “I feel like I am
starting to get the hang of getting the programs to work how I want them to… plus I have
[Wilma] next to me who has been very helpful…” (Muriel, 1767, 1777).

The journal data suggested that many teachers had gained comfort with
programming logic by the middle of the second week.  On June 29, Victor attributed his
increased comfort with programming to the aide of his peers, the instructor, and playing
computer:

   I feel like a learned a great deal about loops, if functions and else functions,
finally seeing how to use them so they actually work. There are subtle issues,
which I learned to resolve by trying to think like a computer, and syntax issues,
which my peers and [Instructor] helped a great deal to resolve.  (Victor, 1885)

On June 30 Denise worked in a partnership with Victor.  By the end of class, she
expressed comfort with programming, saying; “I think working alone in the first week
helped me learn the basics better. Once I was comfortable with basics, working in [a]
group was more helpful as we could discuss different strategies…” (Denise, 459). Also
on June 30, Esther commended that; “[Cory] and I are both pretty independent workers and both feel very comfortable with the programming aspect of the class” (Esther, 585).

Future applications. Ultimately the success of the workshop will be largely measured by whether or not it influences teachers to use computer programming in the mathematics that they do and teach. A future, longitudinal study will be needed to accurately measure this influence. However, the journal data did suggest that the workshop influenced some teachers to consider how they might apply computer programming in their future personal and teaching endeavors. The journal data implied that the large majority of the teachers had a very positive workshop experience that, for some, generated motivation to make programming a part of their futures. For one teacher though, it was clear that the workshop had failed her.

Melinda was the only teacher in the workshop whose journal data expressed an overall negative impression of the workshop experience. Although Melinda had said early in the first week that things were going well, by the end of the second week her journal entries mostly spoke of feeling overwhelmed by frustration with programming, the conceptual difficulty of the problems, and the perceived lack of guidance from the instructor. Melinda did not leave the workshop intending to use programming in the future:

I… worked on [the birthday] problem, but I felt like I was wayyy [*sic*] out of my league. I very much did not feel comfortable… and I felt like I didn’t have anyone to help me. I really feel like I was floundering by the end of the week. I felt like I hadn’t been very productive and that I might not be able to accomplish
any future problems… I also feel like I am not learning a whole lot of tangible things that I could take into my classroom. I understand the big-picture ideas behind why bringing more programming into schools can be a really positive thing, but I don’t really feel empowered to make it happen. I would appreciate a little bit more information on this. (Melinda, 1444, 1454, 1466)

Unlike Melinda, Barry and Shawn both seemed excited to try to incorporate computer programming into their teaching. Barry said:

Already I am thinking about the possibilities for creating formulas and programs. Especially since formulas are entered much like you would on a TI calculator… I am interested in how this might look in a high school. There are some students who have expressed interest in software/game design already and I’d love to show them basic programming. (Barry, 141, 146)

Shawn’s future plans were to teach in another country and he left the workshop eager to expand the use of computer programming beyond Monte Carlo modeling and simulation:

This class was a great introduction to programming and I intend to use it in my future classes (in [foreign country]) and studies. More than just opening the door to analyzing empirical probabilities through computer-based simulations, this class (teachers, atmosphere, students [i.e. teachers], all of it was inspiring and pushed me to consider the power of computing in the various fields of math, such as topology, series, and probability… I look forward to your [Instructor’s] guidance during the fall as I prepare an application. (Shawn, 1862, 1874)
While Melinda’s negative experience with the workshop was a statistical outlier and not representative of the experiences that the other teachers had, her difficulties and frustrations might be able to be leveraged into improvements for the workshop in the future. Melinda’s journal data suggested that her negative experiences stemmed primarily from the content being too difficult, the pacing too fast, and direct instruction being too minimal. However, the workshop’s content, pacing, and primary-learning philosophy were listed as positive aspects in the journals of so many of the other teachers. It is therefore recommended that any changes to the workshop based on Melinda’s experience be thoroughly scrutinized and perhaps limitedly piloted in future workshop iterations, before being fully committed.
Chapter 5: Conclusions

This chapter concludes the study by discussing how one teacher utilized computer programming after the workshop, conclusions inferred from data analysis, factors that limited study procedures and results, and the direction of future research.

Post-Workshop Implementation: An Interview with Terry

Approximately one year after the workshop concluded a public middle school teacher who had participated in the workshop agreed to a brief meeting to describe how he had been using computer programming since his workshop experience. For the purposes of preserving anonymity, he will be called Terry. Between the end of the workshop in early July of 2015 and the beginning of the 2015/2016 academic school year in late August of 2015, Terry developed a computer programming curriculum and implemented it in an elective programming course that he offered at his school for students at grade levels 6-8. This programming class was the first programming class to be offered at the school. The class was conducted over two separate quarters and each quarter had 8 to 10 students enroll. There was no academic requirement to join the class, but about fifty percent of the students were identified as academically gifted. The focus of the course was to offer students an introductory computer programming experience
that covered the basic programming concepts of input/output, variables, basic mathematics functions, conditional statements, loops, and exposure to lists (i.e. arrays).

*Python* was the programming language Terry used in the class. He chose the language because he believed it was a robust and widely used programming language but one with minimal and easy to grasp syntax. Students developed their programs through an online system using *Chromebook* computers. To create his programming curriculum, Terry borrowed heavily from the workshop design and learning philosophies. Terry said that he learned from his workshop experience that problem-based instruction makes learning more interesting, so he focused his class on minimizing lecture and maximizing hands-on learning.

Terry used several of the same activities in his class that were used in the workshop, including the *change* problem to develop mastery of conditional statement usage. However, due to his target audience being middle schoolers, the pacing of his course had to be much slower than the workshop and several more advanced programs that were used in the final week of the workshop were not appropriate for his students. Terry also had to develop several of his own activities that slowly built up his students’ programming knowledge and served as a bridge between the gaps in the activities he used that were taken from the workshop.

Terry noted that the general student reaction to the class was very positive and that engagement level was very high compared to one of his typical mathematics classes. He said that only one student was continually frustrated over their programming experience but that the student was typically eager to work after class, with Terry’s help,
to push past that frustration. Terry characterized the class as a pretty good time for everyone.

Terry said that he would like to see the nature of the programming class change next year. In the first year, the class did not focus as much on mathematical problem solving as he would have preferred. He would like to teach the students Monte Carlo techniques for solving experimental probability problems, because he now believes that computer programming can be used to help kids’ assess their knowledge of mathematics and to highlight their mathematical misconceptions.

Eventually Terry would like to design lessons for his mathematics classes that involved students learning mathematics through programming, but said that this would be difficult to implement because in public school kids do not have a programming foundation coming in to math class. Terry did create some of his own probability simulations that he used in his mathematics class to demonstrate probability to his students. He also wrote some simulations to verify for himself that he had created correct solutions for certain probability problems he had designed for his mathematics class. Terry said that he is more comfortable creating Monte Carlo simulations now than he is solving probability problems theoretically, which interestingly enough, was not how he felt a year ago.

When asked to reflect back on his time in the workshop and consider if there were any experiences he did not have that he now wished that he did, he said that he wished the workshop had set aside time for teachers to sit down together and develop grade-level appropriate lesson plans that they could take with them if they chose to implement
programming into their teaching. He also wished that the workshop better prepared the teachers to set up the infrastructure necessary to create a computer programming class at their respective schools.

Interest in Terry’s programming class is growing. A second year for the course has been approved by his school and before the first year was over, he already had over 25 students who had signed up for next quarter’s class. Terry’s efforts have been recognized by his local high school and he has been asked to help them design a computer programming course at the high school too.

Study Conclusions

The purpose of this study was to investigate how the workshop effectively prepares mathematics teachers to integrate computer programming into their mathematics classes. Research analysis proceeded from the workshop premise that preparedness is a function of content knowledge, teaching self-efficacy, and motivation to integrate computer programming into mathematics curricula.

To evaluate the “effectiveness” of the workshop on teacher preparedness, this study explored three research questions:

1. How does the workshop’s computer programming experience impact the teachers’ content knowledge?

2. How does the workshop experience impact the teachers’ self-efficacy to teach both computer programming and probability?

3. How does the workshop experience impact the teachers’ motivation to integrate computer programming into mathematics curricula?
Testing data revealed that teachers’ probability-based problem solving knowledge greatly and statistically significantly increased over the duration of the workshop. For the Dice and Chips test problems combined, 17 out of 34 (according to both raters) teachers’ solutions received a different correctness score on their pre-test versus their post-test. In all seventeen of these differences the post-test solution was correct while the pre-test solution was incorrect. Cory and Greg both noted in their journals that the workshop experience had revealed to them that their ability to solve probability problems theoretically was not very strong. Examination of their pre-test and post-test scores on the Dice and Chips problem showed that Cory could not solve either problem theoretically and Greg could only solve the Dice problem theoretically. Both were able to solve each problem using Monte Carlo simulation techniques by the end of the workshop.

The large percentage of test programs that successfully compiled, ran, and achieved correct results showed that teachers had a sufficient mastery of computer programming by the workshop’s end. Correct solutions for the Dice and Chips test questions both required non-trivial application of conditional statements, loops, the random number generator, and in many cases for the Chips test question, arrays.

The aforementioned results combined to strongly suggest that Monte Carlo modeling and simulation can be assimilated in a very short amount of time (roughly ten days or less) by a mathematically inclined individual and implemented effectively to solve classic probability problems.
Based on results from the faulty *Defective* test problem, it is recommended that the workshop’s curriculum be revised in the future to train teachers to recognize when a probability programming solution truly models an experiment versus when it only partially models an experiment, or does not model an experiment at all. Journal data from Melinda’s experience in the workshop corroborates a need to explore the differences between experimentally derived solutions and theoretically derived solutions to help teachers understand that Monte Carlo modeling offers an alternative method to solve probability problems that can often be successful when theoretical methods are too difficult or complicated and does not necessarily purport to improve theoretical problem solving strategies. Without an understanding of whether or not a programming solution is experimental, it becomes problematic to confirm that the probability result derived by the program can independently verify the probability result derived theoretically.

Survey analysis showed that over the duration of the workshop experience, teachers’ self-efficacy to teach computer programming showed statistically significant improvement. Teachers became more comfortable in their ability to write a computer program, more comfortable in their ability to use computer programming to do or teach mathematics, and more comfortable in their ability to teach a unit or class focused on computer programming. Furthermore, the data showed that teachers believed their problem solving skills had significantly improved over the duration of the workshop, with 70% of all teachers characterizing the amount of problem solving improvement at level 4 or above on a 5-point Likert-scale where 1 meant *not at all* and 5 meant *a lot.*
Journal data showed that most teachers felt confident in their programming skills by the end of the workshop experience. Terry believed enough in his programming skills and ability to teach computer programming that he created and then taught his own programming curriculum to middle school students. Terry’s experience combined with the teachers’ journal data, offer a strong endorsement of the workshop’s ability to raise teachers’ self-efficacy to teach computer programming.

Since the workshop is designed for a target audience of mathematics teachers, it is recommended that in the future the curriculum make room for opportunities for teachers to co-create curriculum that is specific and age appropriate for the needs of the students that they teach. It is also recommended that a solution be developed for teachers to feel prepared to integrate programming tools of the workshop (namely the programming language) into the technological infrastructure that their schools provide. Both Terry and Melinda expressed a need for the workshop to focus on these things in order to best prepare them to use programming in the classes they teach and generating a belief that the some facsimile of the workshop is viable in their own classroom is important for developing their self-efficacy to teach programming.

Survey and journal data both showed that the workshop was a very fun, engaging, relevant, and empowering experience, which were all important factors that contribute to high levels of motivation. Journal data suggested that the most common emotional responses that teachers experienced in the workshop were having fun, feeling engaged, feeling challenged, feeling frustrated, and feeling proud and accomplished. Teachers often expressed a strong feeling of ownership over their programs, perhaps due to the
numerous solution paths that a computer programming solution often offers and how their choice reflects their sense of identity. Frustration was often a by-product of challenge. While too much frustration could suppress motivation, most teachers found their frustrations were manageable and when they overcame their challenge with a working program that they conceived, developed, and coded, they felt a great sense of personal satisfaction. Some teachers remarked about how fascinated they were with all of the probability problems that they could explore by leveraging computer programming and were intrigued with how they could use their programming skills to explore other branches of mathematics.

Pre- and post-survey data corroborated these journal descriptions. Survey data showed that the workshop experience significantly increased teachers’ beliefs that programming should be an essential part of a mathematics curriculum and that the experience increased their motivation to add programming to the mathematics that they taught. In both cases, teachers gave a median post-survey score of 4 out of 5. When asked on the post-survey how likely they thought they were to use programming again in the future, 82% gave a Likert score of 4 or higher, with 1 meaning not at all and 5 meaning definitely.

This study concludes that the workshop is a highly successful experience for training mathematics teachers to program a computer, solve classical probability problems and provides impetus for teachers to want to teach computer programming and use it as a mathematical problem solving tool.
Study Limitations

A significant limitation of this study was that it occurred as a part of a functioning university classroom for university credit. As is typical with a university course, workshop participants were self-selected and reasons for enrolling in the workshop were varied. Some teacher-participants enrolled in the workshop to fulfill elective credit requirements or to receive professional development credit, while others were simply interested in learning how to program a computer. This self-selection meant that the study’s sample may not have strongly represented the target population, namely K-12 mathematics teachers, and therefore any study outcomes were limited in their ability to draw conclusions back to the population.

Due to the difficulty in integrating a research study into a working, classroom setting, this study was limited by its inability to generate a control group to be compared to the treatment group of the workshop. Without a control group and without the ability to enroll participants via random sampling of the target population, a true experimental research design was not possible. The study implemented both pre- and post-tests and pre- and post-surveys which allowed for analysis of how the teachers’ changed and grew over the course of the workshop. Without a control group however, test and survey analyses cannot rule out potential biases (such as history, maturation, and testing bias) that undermine the ability of the study to draw cause-and-effect conclusions.

By conducting the workshop as a university course the study was also limited in the type of data that could be collected. The large time demands placed on the teachers by the workshop and other university courses they were taking meant that conducting
clinical interviews was not a viable option. As a result, journal narratives were often unfocused with numerous comments and passages therefore being discarded as non-descriptive data. This lessened the amount of rich descriptive data available for qualitative analysis.

Quantitative analysis was limited by the low number of teachers that participated in the workshop and the inability to draw participants from a random sampling of the target population, which made it impossible to assume that the sample was normally distributed. Parametric tests, which are often the gold standard for quantitative analyses, had to be abandoned for non-parametric alternatives that did not need to assume a normal distribution. Low numbers of participants rendered basic factor analysis moot for certain highly-homogenous factors among the teachers, therefore lessening the amount of factor analysis that was originally intended in the design of survey analysis procedures.

The purpose of this study was to evaluate the effectiveness of the workshop to effectively prepare mathematics teachers to integrate computer programming into their mathematics classes. While this study was able to accomplish this goal, because the workshop was led by a single instructor, the study was limited in its ability to determine if the workshops’ effectiveness was more heavily attributable to the instructor, the curriculum, or some combination of both. Would this workshop experience be effective with a different instructor at the helm? This study was not equipped to answer this question.
Future Research

Research on how to effectively train teachers to integrate computer programming and mathematics into their teaching is still in its infancy. This study is perhaps best treated as an in-depth pilot-study that can be used to shape the direction, and motivate the funding, of future teacher preparation research.

Future study should explore the juxtaposition between learning through guided, group discussion and learning through hands-on programming experiences. The journal data suggests both approaches are important to teacher development and further study may reveal the best percentage of time spent on each.

This study suggests that a link may exist between overcoming manageable workshop frustrations and the high incidence of personal satisfaction that teachers experienced in the workshop. This is a question that could merit further study.

Future study also may be warranted to determine if learning is positively affected by keeping programming syntax, language structure, and computing details to a minimum, or if teachers’ learning needs would be better met with more detailed explanation of what occurs behind the scenes. Can computing details be revealed without sacrificing the coverage of problem solving activities and the pacing of the curriculum?

While this study was able to conclude that the workshop effectively trains teachers to program and integrate programming into probability problem solving, it is unknown how the workshop affected anyone’s teaching, other than a surface level understanding of how it impacted Terry to design and implement a computer programming class for middle schoolers.
The most pressing need at this time is to develop a longitudinal study that follows workshop teachers for a year or two after they have completed the workshop course and studies how they leverage the workshop curriculum to enhance the academic experiences of their own students.
References


152


http://www.csta.acm.org/Communications/sub/DocsPresentationFiles/TCEAPres07.pdf


APPENDIX A: Programming Tasks and Coding
The following pages provide the detailed description of the tasks along with the associated coding in C. The last three tasks are the Posttest questions.

<table>
<thead>
<tr>
<th><strong>Lab</strong></th>
<th><strong>Description/Purpose</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hello.c Getting started</td>
</tr>
<tr>
<td>2</td>
<td>age.c Input/output</td>
</tr>
<tr>
<td>3</td>
<td>floc.c Fahrenheit to Celsius – simple arithmetic computation, variables, assignment</td>
</tr>
<tr>
<td>4</td>
<td>bigger.c if statements</td>
</tr>
<tr>
<td>5</td>
<td>tens.c Extracting out the 10’s digit of a number – remainder and floor functions</td>
</tr>
<tr>
<td>6a</td>
<td>change.c Converting a number of pennies to dollars, quarters, dimes, nickels</td>
</tr>
<tr>
<td>6b</td>
<td>change_ext.c Challenge: Determine if coin is plural/singular, and omit if no coin. Use the word <em>and</em> before the last denomination</td>
</tr>
<tr>
<td>7</td>
<td>jack.c while loops: “all work and no play makes Jack a dull boy” repeated</td>
</tr>
<tr>
<td>8a</td>
<td>multilites.c Creates multiples of a number (practice with loops)</td>
</tr>
<tr>
<td>8b</td>
<td>primes.c Challenge: enter a positive integer and output whether or not prime</td>
</tr>
<tr>
<td>9</td>
<td>uniformity.c Generate a random number (done as a class), using <em>drand48()</em>, and seeding the generator. Roll a six sided die a million times and list the number of 1s, 2s, … 6s.</td>
</tr>
<tr>
<td>10a</td>
<td>craps.c Determine the experimental probability of rolling 7 or 11 with two dice/compare to theoretical probability.</td>
</tr>
<tr>
<td>10b</td>
<td>craps_ext.c Challenge: Find probability of rolling 7 or 11 with three dice and compare to theoretical probability</td>
</tr>
<tr>
<td>11</td>
<td>first_match.c <strong>No arrays needed.</strong> Does the first number of a sequence occur later?</td>
</tr>
<tr>
<td>12</td>
<td>last_match.c <strong>Arrays needed.</strong> Does the last number of a sequence occur earlier?</td>
</tr>
<tr>
<td>13a</td>
<td>rooks.c Find the probability of two rooks on a chess board attacking each other and compare to the theoretical probability</td>
</tr>
<tr>
<td>13b</td>
<td>bishops.c Find the probability of two bishops on a chess board attacking each other and compare to the theoretical probability</td>
</tr>
<tr>
<td>14</td>
<td>birthday.c Find the probability of at least two people having the same birthday in a group of <em>n</em> people (no not use formula), compare to formula</td>
</tr>
<tr>
<td>15a</td>
<td>area.c Estimate <em>π</em> (pi) by using Monte Carlo integration</td>
</tr>
<tr>
<td>15b</td>
<td>spherecube.c Challenge: Estimate the volume of a unit sphere by using Monte Carlo integration.</td>
</tr>
<tr>
<td>16</td>
<td>volume.c Compare the volume of a cone to the circumscribed cylinder using Monte Carlo integration.</td>
</tr>
<tr>
<td>17a</td>
<td>frog.c Find the probability that after 3 jumps of one unit in any direction that the frog ends up inside the original circle.</td>
</tr>
<tr>
<td>17b</td>
<td>bee.c Challenge: Find the probability that after 3 jumps of unit each, the bee ends up inside the original 3D sphere.</td>
</tr>
<tr>
<td>18</td>
<td>dice.c Posttest problem #1: Rolling 2 dice three times and the probability of getting at least two 8s</td>
</tr>
<tr>
<td>19</td>
<td>chips.c Posttest problem #2: Sampling WIHTOUT replacement</td>
</tr>
<tr>
<td>20</td>
<td>defective.c Posttest problem #3: Probability of being from Factory A given it is defective.</td>
</tr>
</tbody>
</table>

Table 15: Sequential Programming Tasks (repeated)
1. **hello.c**

   // Program Name: hello.c

   // Program Task:
   /* Create a program that outputs to the screen “hello world!” and then a new line. */

   #include <stdio.h>
   #include <math.h>
   #include <stdlib.h>

   int main()
   {
     printf("Hello world!\n ");
   }
2. **age.c**  Print your age

    // Program Name: age.c

    // Program Task:
    /* Create a program that asks the user to input their age and then outputs "you are
    <age> years old." where <age> is the value of the user input.
    */

    #include <stdio.h>
    #include <math.h>
    #include <stdlib.h>

    int main()
    {
        double age ;

        printf("Please input your age in years: ");
        scanf("%lf", &age);
        printf("nYou are %lf years old.\n", age);
    }

3. **fot.c  Fahrenheit to Celsius**

// Program Name:  fotoc.c

// Program Task:
/* Water boils at 212 degrees Fahrenheit which is 100 degrees Celsius.  Water freezes at 32 degrees Fahrenheit which is 0 degrees Celsius.  Find a formula for converting Fahrenheit to Celsius and then create a program that allows the user to enter a temperature in Fahrenheit and then outputs that temperature in Celsius. */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double fahrenheit, celsius ;

printf("Enter a temperature in Fahrenheit: ") ;
scanf("%lf", &fahrenheit) ;

celsius = (fahrenheit-32.0) * (5.0/9.0) ;

printf("\n%lf degrees fahrenheit is %lf degrees celsius.\n", fahrenheit, celsius) ;
}

4. bigger.c

// Program Name: bigger.c

// Program Task:
/* Create a program that allows the user to input two numbers and output whether the first is greater than the second, less than the second, or equal to the second. */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double num1, num2;

printf("Please input a first number: ");
scanf("%lf", &num1);

printf("nPlease input a second number: ");
scanf("%lf", &num2);

if(num1 > num2) { printf("nYour first number %lf is greater than your second number %lf n", num1, num2); }
if(num1 < num2) { printf("nYour second number %lf is greater than your first number %lf n", num2, num1); }
if(num1 == num2) { printf("nYour numbers are equal\n"); }

}
5. tens.c

// Program Name: tens.c

// Program Task:
/* Create a program that allows the user to input a real number, as a decimal, and
outputs the digit in the tens place.
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double num, tnum, onum ;

printf("Please input a real number in decimal form: ");
scanf("%lf", &num) ;

onum = num ;
num = fabs(num) ;
num = floor(num) ;
num = num/10.0 ;
num = floor(num) ;
num = num/10.0 ;
tnum = floor(num) ;
um = num - tnum ;
um = num*10.0 ;

printf("The tens digit of the number %lf is %lf\n", onum, num) ;

}
6a. `change.c`

// Program Name: change.c

// Program Task:
/* You take a large jar of pennies to the bank and ask the teller for exact change. The
teller can give you only single dollar bills, quarters, dimes, nickels, and pennies back.
Create a program that allows the user to input a number of pennies and outputs exact,
least change, so that you can check that the bank teller gives you the proper change.
Example: For input 521, acceptable output is "5 dollars, 0 quarters, 2 dimes, 0
nickels, and 1 pennies."
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double jar, ojar, dollars, quarters, dimes, nickels, pennies;

printf("Enter the number of pennies that you wish to receive change for: ");
scanf("%lf", &jar);
printf("\n");

ojar = jar;
dollars = floor(jar/100.0);
jar = fmod(jar,100.0);

quarters = floor(jar/25.0);
jar = fmod(jar,25.0);

dimes = floor(jar/10.0);
jar = fmod(jar,10.0);

nickels = floor(jar/5.0);
pennies = fmod(jar,5.0);

printf("Exact, least change for %lf pennies is:\n%lf dollars, %lf quarters, %lf dimes,
%lf nickels, %lf pennies.\n\n", ojar, dollars, quarters, dimes, nickels, pennies);
}

162
6b. change_ext.c

// Program Name: change_ext.c

// Program Task:
/* You take a large jar of pennies to the bank and ask the teller for exact change. The
 teller can give you only single dollar bills, quarters, dimes, nickles, and pennies back.
 Create a program that allows the user to input a number of pennies and outputs exact,
 least change, so that you can check that the bank teller gives you the proper change.

 For this extension to the change.c program, make sure to modify the output to omit
denominations of zero, correctly make each demonination singular or plural, and use
the "and" conjunction only with the last denomination.

 Example: For input 3530, acceptable output is "35 dollars, 1 quarter, and 1 nickle."
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double jar, ojar, dollars, quarters, dimes, nickels, pennies;
double pado, paq, padi, pan, flag;
/*
 pado means pennies after dollars
 paq  means pennies after quarters
 padi means pennies after dimes
 pan  means pennies after nickles
*/
  pado = 1; paq = 1; padi = 1; pan = 1;

  printf("Enter the number of pennies that you wish to receive change for: ");
  scanf("%lf", &jar);
  printf("\n");

  ojar = jar;
  dollars = floor(jar/100.0);
  jar = fmod(jar,100.0);
  if(jar == 0) { pado = 0 ; }

  quarters = floor(jar/25.0);
  jar = fmod(jar,25.0);
if(jar == 0) { paq = 0 ; }

dimes = floor(jar/10.0) ;
jar = fmod(jar,10.0) ;
if(jar == 0) { padi = 0 ; }

nickels = floor(jar/5.0) ;
jar = fmod(jar,5.0) ;
if(jar == 0) { pan = 0 ; }
pennies = jar ;

printf("Exact, least change for %.0lf pennies is: n", ojar) ;

flag = 0 ;
//dollars
if(dollars > 0) {
    if(pado == 1) {
        if(dollars == 1) { printf("%.0lf dollar, ", dollars) ; }
        else { printf("%.0lf dollars, ", dollars) ; }
    }
    else {
        if(dollars == 1) { printf("%.0lf dollar.", dollars) ; }
        else { printf("%.0lf dollars.", dollars) ; }
    }
    flag = 1 ;
}

//quarters
if(quarters > 0) {
    if(paq == 1) {
        if(quarters == 1) { printf("%.0lf quarter, ", quarters) ; }
        else { printf("%.0lf quarters, ", quarters) ; }
    }
    else {
        if(flag == 1) { printf("and ") ; }
        if(quarters == 1) { printf("%.0lf quarter.", quarters) ; }
        else { printf("%.0lf quarters.", quarters) ; }
    }
    flag = 1 ;
}
//dimes
if(dimes > 0) {
    if(padi == 1) {

if(dimes == 1) { printf("%.0lf dime, ", dimes) ; }
else { printf("%.0lf dimes, ", dimes) ; }
}
else {
  if(flag == 1) { printf("and ") ; }
  if(dimes == 1) { printf("%.0lf dime.", dimes) ; }
  else { printf("%.0lf dimes.", dimes) ; }
}
flag = 1 ;

//nickels
if(nickels > 0) {
  if(pan == 1) {
    if(nickels == 1) { printf("%.0lf nickel, ", nickels) ; }
    else { printf("%.0lf nickels, ", nickels) ; }
  }
  else {
    if(flag == 1) { printf("and ") ; }
    if(nickels == 1) { printf("%.0lf nickel.", nickels) ; }
    else { printf("%.0lf nickels.", nickels) ; }
  }
  flag = 1 ;
}

//pennies
if(pennies > 0) {
  if(flag == 1) { printf("and ") ; }
  if(pennies == 1) { printf("%.0lf penny.", pennies) ; }
  else { printf("%.0lf pennies.", pennies) ; }
}
printf("\n") ;
}
7. **jack.c**

    // Program Name:  jack.c

    // Program Task:
    /* Sometimes it is necessary to perform a certain task many times over and this is a
     job that a computer is well suited for. For this task, use a loop to create a program
     that allows a user to input a non-negative integer and then outputs the phrase "All
     work and no play makes Jack a dull boy" that number of times. After all it would be
     truly "crazy" to print that phrase out using numerous print statements, on after
     another.
    */

    #include <stdio.h>
    #include <math.h>
    #include <stdlib.h>

    int main()
    {
        double input, counter;

        printf("Enter a non-negative integer: ");
        scanf("%lf", &input);

        printf("\n");
        counter = 0;
        while(counter < input) {
            printf("All work and no play makes Jack a dull boy.\n") ;
            counter = counter + 1;
        }

        printf("\n");
    }
8a. multiples.c

// Program Name: multiples.c

// Program Task:
/* Create a program that outputs the first 20 multiples of a user inputted non-negative integer. */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double input, output, c ;

printf("Please enter a non-negative integer: ");
scanf("%lf", &input);

output = input ;
c = 0 ;
while(c < 20) {
    printf("%.0lf\n", output) ;
    output = output + input ;
    c = c + 1 ;
}
}
// Program Name: primes.c

// Program Task:
/* Create a program that allows the user to input a positive integer and outputs
whether or not the integer is prime. */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double input, flag, c ;

printf("Please enter a positive integer: ");
scanf("%lf", &input);

flag = 0 ;
c = 2.0 ;
while((c < input) && (flag == 0)) {
  if(fmod(input,c) == 0) {
    printf("%.0lf is not prime
", input) ;
    flag = 1 ;
  }
  c = c + 1 ;
}

if(input == 1) { printf("%.0lf is not prime\n", input) ; }
else if(input == 2) { printf("%.0lf is prime\n", input) ; }
else if(flag == 0) { printf("%.0lf is prime\n", input) ; }
}
9. uniformity.c

// Program Name: uniformity.c

// Program Task:
/* Create a program that models a six-sided die being rolled and simulate with one
million trials. Then output the number of 1's, 2's, 3's, 4's, 5's, and 6's that were rolled.
Do results from the pseudo-random number generator, drand48(), appear to be
uniformly distributed? */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double x, k, one, two, three, four, five, six;

srand48(time(0));

one = 0; two = 0; three = 0; four = 0; five = 0; six = 0;
k = 0;
while(k<1000000) {

x = floor(6*drand48()) + 1;
if(x==1) { one = one + 1; }
if(x==2) { two = two + 1; }
if(x==3) { three = three + 1; }
if(x==4) { four = four + 1; }
if(x==5) { five = five + 1; }
if(x==6) { six = six + 1; }

k = k+1;
}
printf("number of times one occurred = %.0lf\n", one);
printf("number of times two occurred = %.0lf\n", two);
printf("number of times three occurred = %.0lf\n", three);
printf("number of times four occurred = %.0lf\n", four);
printf("number of times five occurred = %.0lf\n", five);
printf("number of times six occurred = %.0lf\n", six);
}
10a.  craps.c

    // Program Name:  craps.c

    // Program Task:
    /* Create a program that models the game of craps, where two six-sided dice are
    rolled and a sum of 7 or 11 means you have hit "craps".  Simulate to find the
    probability that a roll of the dice will result in craps.
    */

    #include <stdio.h>
    #include <math.h>
    #include <stdlib.h>

    int main()
    {
        double k, die_1, die_2, sum, craps ;

        craps = 0 ;
        k = 0 ;
        while(k < 1000000) {
            die_1 = floor(6*drand48()) + 1 ;
            die_2 = floor(6*drand48()) + 1 ;
            sum = die_1 + die_2 ;

            if(sum == 7 || sum == 11) {
                craps = craps + 1 ;
            }

            k = k + 1 ;
        }

        printf("The probability of rolling craps is %lf\n", craps/1000000.0) ;
    }
10b. **craps_ext.c**

// Program Name: craps_ext.c

// Program Task:
/* Create a program that models the game of craps, but instead of two six-sided dice,
three six-sided dice are rolled and a sum of 7 or 11 means you have hit "craps".
Simulate to find the probability that a roll of three dice will result in craps.
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{
    double k, die_1, die_2, die_3, sum, craps;
    craps = 0;
    k = 0;
    while(k < 1000000) {
        die_1 = floor(6*drand48()) + 1;
        die_2 = floor(6*drand48()) + 1;
        die_3 = floor(6*drand48()) + 1;
        sum = die_1 + die_2 + die_3;
        if(sum == 7 || sum == 11) {
            craps = craps + 1;
        }
        k = k + 1;
    }
    printf("The probability of rolling craps is %lf\n", craps/1000000.0);
}

---

---

171
11. first_match.c

    // Program Name:  first_match.c

    // Program Task:
    /* Create a program that prompts the user to input a sequence of non-negative
    integers, one after another, until the user types in a negative integer.  The program
    outputs whether or not the first number of the sequence is repeated in the sequence.
    NOTE: Do not include the negative integer as a number in the sequence
    */

    #include <stdio.h>
    #include <math.h>
    #include <stdlib.h>

    int main()
    {

        double num_1, num_2, flag ;

        flag = 0 ;
        scanf("%lf", &num_1) ;

        if(num_1 >= 0) {
            num_2 = 2 ;
            while(num_2 >= 0) {
                scanf("%lf", &num_2) ;
                if(num_1 == num_2) { flag = 1 ; }
            }
        }

        if(num_1 >=0) {
            if(flag == 1) { printf("%.0lf is repeated in the sequence\n", num_1) ; }
            else { printf("%.0lf is not repeated in the sequence\n", num_1) ; }
        }
    }
12. last_match.c
   // Program Name: last_match.c

   // Program Task:
   /* Create a program that prompts the user to input a sequence of non-negative integers, one after another, until the user types in a negative integer. The program outputs whether or not the last number of the sequence is repeated in the sequence. NOTE: Do not include the negative integer as a number in the sequence and you can assume that the sequence will have less than 1000 numbers */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double input, num_1, num_2, flag ;
double sequence[1000] ;
int c, k ;
flag = 0 ;

input = 2 ;
c = 0 ;
while(input >= 0) {
    scanf("%lf", &input) ;
    sequence[c] = input ;
c = c + 1 ;
}
c = c - 2 ;
um_1 = sequence[c] ;
k = 0 ;
while(k < c) {
    if(sequence[k] == num_1) { flag = 1 ;}
k = k + 1 ;
}
if(c != -1) {
    if(flag == 1) { printf("%.0lf is repeated in the sequence\n", num_1) ; }
    else { printf("%.0lf is not repeated in the sequence\n", num_1) ; }
}
}
13a.  **rooks.c**

    // Program Name: rooks.c

    // Program Task:
    /* What is the probability that two rooks, randomly placed on a chessboard, will be in attacking position? Program a model and simulate to find the probability. */

    #include <stdio.h>
    #include <math.h>
    #include <stdlib.h>

    int main()
    {
        double row_1, col_1, row_2, col_2;
        double attack, trial;

        srand48(time(0));
        attack = 0;
        trial = 0;
        while(trial < 10000000) {

            row_1 = floor(8*drand48()) + 1;
            col_1 = floor(8*drand48()) + 1;
            row_2 = floor(8*drand48()) + 1;
            col_2 = floor(8*drand48()) + 1;

            if((row_1 == row_2) && (col_1 == col_2)) { /* do nothing, bad trial */ }
            else {
                if((row_1 == row_2) || (col_1 == col_2)) {
                    attack = attack + 1;
                }
                trial = trial + 1;
            }
        }

        printf("nThe probability that two randomly placed rooks are in attacking position is: %lf\n\n", attack/trial);
    }

_______________________________________________________________________
13b. **bishops.c**

// Program Name: bishops.c

// Program Task:
/* What is the probability that two bishops, randomly placed on a chessboard, will be
in attacking position? Program a model and simulate to find the probability.
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double row_1, col_1, row_2, col_2;
double attack, trial;

srand48(time(0));

attack = 0;
trial = 0;
while(trial < 10000000) {

    row_1 = floor(8*drand48()) + 1;
    col_1 = floor(8*drand48()) + 1;
    row_2 = floor(8*drand48()) + 1;
    col_2 = floor(8*drand48()) + 1;

    if((row_1 == row_2) && (col_1 == col_2)) {/* do nothing, bad trial */}
    else {
        if(fabs(col_2 - col_1) == fabs(row_2 - row_1)) {
            attack = attack + 1;
        }
        trial = trial + 1;
    }
}

printf("The probability that two randomly placed bishops are in attacking position
is: %lf\n", attack/trial);

}
14. birthday.c

// Program Name: birthday.c

// Program Task:
/* What is the probability that in a classroom of 'n' number of people, at least two people will be born on the same day of the year? Program a model where the user inputs 'n' and simulate to find the probability. NOTE: You may assume that there are always 365 days in a year.
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double histogram[365], n, success ;
int temp, k, c, flag ;

printf("Please input the number of people: ");
scanf("%lf", &n) ;

srand48(time(0)) ;
success = 0 ;

c = 0 ;
while(c < 1000000) {

//initialize histogram
k = 0 ;
while(k < 365) {
    histogram[k] = 0 ;
    k = k + 1 ;
}

//get birthdays and update histogram
k = 0 ;
while(k < n) {
    temp = floor(365*drand48()) ;
    histogram[temp] = histogram[temp] + 1 ;
    k = k + 1 ;
}

...
// check histogram for at least 2 people with the same birthday
flag = 0;
k = 0;
while((k < 365) && (flag == 0)) {
    if(histogram[k] >= 2) {
        success = success + 1;
        flag = 1;
    }
    k = k + 1;
}
c = c + 1;
}

printf("The probability that at least two people will have the same birthday in a classroom of size %.0lf, is: %lf\n", n, success/c);
}
// Program Name: area.c
// Program Task:
/* Imagine a circle with radius 1 inscribed in a square. A large, random sampling of points inside the square will result in some points also being inside the circle. Create a model that randomly generates a point inside of the square and then determines if the point is inside of the circle. Simulate to determine the probability that a randomly generated point inside of the square will also be inside of the circle. This ratio should closely approximate the area of the circle divided by the area of the square. The ratio multiplied by 4 should be Pi, because the circle has a radius of 1, and therefore the area of the circle is Pi, and the square has an area of 4. This technique for finding the ratio of the areas is called Monte Carlo integration. */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main()
{
    double px, py, trial, circle;

    srand48(time(0));

    circle = 0;
    trial = 0;
    while(trial < 10000000) {
        px = 2*drand48()-1;
        py = 2*drand48()-1;

        if(sqrt(px*px + py*py) <= 1) {
            circle = circle + 1;
        }
        trial = trial + 1;
    }
    printf("The ratio of points in the circle to points in the square is: \%lf\n", circle/trial);

    printf("The ratio times 4 yields an approximation of Pi which is: \%lf\n", 4.0*circle/trial);
}
15b.  **spherecube.c**

    // Program Name:  spherecube.c

    // Program Task:
    /* Use Monte Carlo integration to determine the volume of a unit sphere by randomly
    sampling the cube that circumscribes it.  NOTE: The volume of the cube is simply
    length * width * height = 2 * 2 * 2 = 8
    */

    #include <stdio.h>
    #include <math.h>
    #include <stdlib.h>

    int main()
    {
        double px, py, pz, trial, sphere;

        srand48(time(0));

        sphere = 0;
        trial = 0;
        while(trial < 10000000) {
            px = 2*drand48()-1;
            py = 2*drand48()-1;
            pz = 2*drand48()-1;

            if(sqrt(px*px + py*py + pz*pz) <= 1) {
                sphere = sphere + 1;
            }
            trial = trial + 1;
        }

        printf("The experimental volume of the unit sphere is: \%lf\n\n", 8.0*sphere/trial);
        printf("The volume is theoretically (4/3)*Pi which is: \%lf\n", (4.0/3.0)*M_PI);
    }
16. **volume.c**

   // Program Name: volume.c

   // Program Task:
   /* Use Monte Carlo integration to determine the ratio of the volume of a cone to the
   volume of the cylinder that circumscribes it.
   */
   
   #include <stdio.h>
   #include <math.h>
   #include <stdlib.h>

   int main()
   {

   double px, py, pz, trial, cone, l;

   srand48(time(0));

   cone = 0;  
   trial = 0;  
   while(trial < 10000000) {

   px = 2*drand48()-1;
   py = 2*drand48()-1;
   pz = 2*drand48();

   l = sqrt(px*px + py*py);
   if(l <= 1) {
      //point is in the cylinder
      if(pz/l >= 2.0) {
         //point is in cylinder and cone
         cone = cone + 1;
      }
      trial = trial + 1;
   }
   }
   printf("The volume of the cone is %lf the volume of the circumscribed cylinder.\n\n", cone/trial);

   }
// Program Name: frog.c

// Program Task:
/* Imagine a frog that always hops a distance of exactly 1 meter, but can do so in any direction. Create a program that determines the probability that the frog will be within 1 meter of its starting position after hopping in 'n' different random directions, where 'n' is a non-negative integer inputted by the user. */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double angle, x, y, trial, success;
double n, k;
srand48(time(0));

printf("Please input the number of hops the frog will take: ") ;
scanf("%lf", &n) ;

success = 0 ;
trial = 0 ;
while(trial < 10000000) {

    x = 0 ;
    y = 0 ;
    k = 0 ;
    while(k < n) {

        //M_PI is C code for a close approximation of pi
        angle = 2*M_PI*drand48() ;
        x = cos(angle) + x;
        y = sin(angle) + y;

        k = k + 1 ;
    }

    if(sqrt(x*x + y*y) <= 1) {

181
success = success + 1;
}

trial = trial + 1;
}

printf("The probability that the frog, after %.0lf hops, is within one meter of its starting position is: %.lf\n\n", n, success/trial);
}
17b.  bee.c

// Program Name: bee.c

// Program Task:
/* This problem extends frog.c to 3-dimensions. Imagine a bee that, when it moves,
always flies a distance of exactly 1 meter, but can do so in any direction. Create a
program that determines the probability that the bee will be within 1 meter of its
starting position after moving in 'n' different random directions, where 'n' is a non-
negative integer inputted by the user.
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{

double x, y, z, tx, ty, tz, trial, success;
double n, k, len;

srand48(time(0));

printf("Please input the number of flights the bee will take: ");
scanf("%lf", &n);

success = 0;
trial = 0;
while(trial < 1000000) {

x = 0;
y = 0;
z = 0;
k = 0;
while(k < n) {

    tx = 2*drand48() - 1;
    ty = 2*drand48() - 1;
    tz = 2*drand48() - 1;

    //decide if the bee's position tx, ty, tz was made from a legal flight
    len = sqrt(tx*tx + ty*ty + tz*tz);

    //...
if(len <= 1) {
  tx = tx/len ; ty = ty/len ; tz = tz/len ;
  x = tx + x ; y = ty + y ; z = tz +z ;
  k = k + 1 ;
}

if(sqrt(x*x + y*y + z*z) <= 1) {
  success = success + 1 ;
}

trial = trial + 1 ;

printf("The probability that the bee after %.0lf flights, is within one meter of its starting position is: %lf
", n, success/trial) ;

}
APPENDIX B: Pretest and Posttest
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
</table>
| 1. What is the probability of rolling a sum of “eight” at least twice in three rolls using two six-sided dice? | **How confident are you with your answer?** Circle one  
- a. Sure of my answer  
- b. Fairly confident  
- c. Not very confident  
- d. I have no idea how to solve it  

Check all of the following that apply:  
- _ My answer is complete.  
- _ My answer is incomplete because I ran out of time.  
- _ My answer is incomplete because I do not know how to proceed further.  

Comment on the problem solving strategy/strategies that you used for this problem:
2. What is the probability of randomly drawing, without replacement, a total of 2 red chips and 3 blue chips, in any order, from a bowl of 60 chips where 10 are white, 20 are red and 30 are blue?

<table>
<thead>
<tr>
<th>How confident are you with your answer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle one</td>
</tr>
<tr>
<td>a. Sure of my answer</td>
</tr>
<tr>
<td>b. Fairly confident</td>
</tr>
<tr>
<td>c. Not very confident</td>
</tr>
<tr>
<td>d. I have no idea how to solve it</td>
</tr>
</tbody>
</table>

Check all of the following that apply:

- My answer is complete.
- My answer is incomplete because I ran out of time.
- My answer is incomplete because I do not know how to proceed further.

Comment on the problem solving strategy/strategies that you used for this problem:
3. Factory A produces cars where 5% have major defects, and Factory B produces cars where 10% have major defects. A dealer received 30 cars, 12 from Factory A and 18 from Factory B. A customer randomly bought one of the 30 cars and found it to be defective. What is the probability that it came from Factory A?

<table>
<thead>
<tr>
<th>How confident are you with your answer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle one</td>
</tr>
<tr>
<td>a. Sure of my answer</td>
</tr>
<tr>
<td>b. Fairly confident</td>
</tr>
<tr>
<td>c. Not very confident</td>
</tr>
<tr>
<td>d. I have no idea how to solve it</td>
</tr>
</tbody>
</table>

Check all of the following that apply:

- My answer is complete.
- My answer is incomplete because I ran out of time.
- My answer is incomplete because I do not know how to proceed further.

Comment on the problem solving strategy/strategies that you used for this problem:
Posttest with Programming:  
Name ______________________________

1. Using programming determine the experimental probability of rolling a sum of “eight” at least twice in three rolls using two six-sided dice?

2. Using programming, what is the probability of randomly drawing, without replacement, a total of 2 red chips and 3 blue chips, in any order, from a bowl of 60 chips where 10 are white, 20 are red and 30 are blue?

3. Factory A produces cars where 5% have major defects, and Factory B produces cars where 10% have major defects. A dealer received 30 cars, 12 from Factory A and 18 from Factory B. A customer randomly bought one of the 30 cars and found it to be defective. Using programming determine the experimental probability that it came from Factory A.

Note: The italicized words above were added to the identical questions asked in the Pretest. No other changes were made between Pretest and Posttest.
APPENDIX C: Pre-Survey and Post-Survey
SURVEY

MTH ------: Computing for Math Teachers, Summer 2015
Instructor: -----------------
Date: ------------------

Are you enrolled in MTH --- MTH --- Not enrolled (Circle one)

Does/did your high school (either the one you teach at or the one you attended) offer/ed a programming course?
YES  No (Circle one)  If yes, please list the school and describe the course: ________________________

Who taught the course, and what was their background:

---------------------------------------------------------------

If you are an in-service teacher (or GTA), list school, grades you teach, subjects you teach: __________

Do you currently use programming in any course you teach? Yes  No  (Circle one)

If yes, please elaborate: ________________________________________________________________

If you are a pre-service teacher, list expected date of licensure: _____________________________

Where did you complete your UG work? _________________________________________________

Where did you do your student teaching? _________________________________________________

What is your reason for taking this class: ________________________________________________

Have you ever taught a probability unit or course at your school? Yes  No  (circle one)

If yes, what type of technology did you use, if an? _______________________________________

Have you ever taken a probability course? Yes  No  (Circle one)

If yes, when and where? Please describe the course and what you learned from it.

_____________________________________________________________________________________

_____________________________________________________________________________________

Have you ever taken a computer programming course? Yes  No  (Circle one)

If yes, when and where? Please describe the course and what you learned from it.

_____________________________________________________________________________________

_____________________________________________________________________________________

OVER
How comfortable are you in using computer software such as word processors, e-mail, spreadsheets? Circle comfort level and operating system:

5 4 3 2 1 Windows  MAC  UNIX/Linux
Other(specific)______
Very Not at all
Comment:

If the technology was available, how comfortable are you today in writing a computer program?

5 4 3 2 1 Very Not at all
What language? _________________
What Operating System (windows, etc.)?
Comment:

If the technology was available, how comfortable would you be using programming in a mathematics course?

5 4 3 2 1 Very Not at all
Comment:

How strongly do you currently believe that computer programming should be an essential part of a secondary mathematics curriculum?

5 4 3 2 1 Very Not at all
Comment:

How strongly do you believe that computer packages like Geometer’s Sketchpad, Geogebra, Fathom, Tinkerplots, or Minitab should be used in the secondary mathematics classroom?

5 4 3 2 1 Very Not at all
Comment:

How strongly do you believe that manipulatives should be used in the secondary mathematics classroom?

5 4 3 2 1 Very Not at all
Comment:

How motivated are you to include programming in any of your mathematics classes that you teach?

5 4 3 2 1 Very Not at all
Comment:

192
Post –SURVEY

MTH ------ Computing for Math Teachers: Summer 2015
Instructor: ------
Date: ------ Last day

Name: __________________________(your name is requested to compare with your pre-Survey responses)
What WAS your reason for taking this class:
__________________________________________________.

Did the class meet your expectation?  YES  NO  Please explain:

_______________________________________________________________________________________

How comfortable are you today in writing a computer program?

5  4  3  2  1
Very   Not at all

Comment:

Do you ever see yourself programming again for your own purposes or for another class?

5  4  3  2  1
Definitely   Not at all

After completing this class, how comfortable would you be using programming in a math course you teach?

5  4  3  2  1
Very   Not at all

Comment:
If not, what would it take for you to become more comfortable?

How comfortable would you be teaching a programming class at your school?

5  4  3  2  1
Very   Not at all

Comment:
If not, what would it take for you to become more comfortable?

How strongly do you currently believe that computer programming should be an essential part of a secondary/ community college mathematics curriculum?

5  4  3  2  1
Very   Not at all

Comment:

How motivated are you to include programming in any of your mathematics classes that you teach?

5  4  3  2  1
Very   Not at all

Comment:
While doing a probability problem/lab where the experimental (programming) result differed from the theoretical result, which result did you trust?  Experimental  Theoretical  (Circle one)

Why?

Under what circumstances did you trust the experimental probability result and when did you trust the theoretical probability result?

One purpose of this course was to introduce a curriculum and pedagogy for preparing mathematics teachers to incorporate programming in their courses.

What portions of this course’s curriculum (the problems and how they were sequenced) do you think helped you toward this preparation?

What portions of this course’s curriculum could be improved or changed to help you toward this preparation?

What portions of this course’s pedagogy (how it was taught and how you learned) do you think helped you toward this preparation?

What portions of this course’s pedagogy could be improved or changed to help you toward this preparation?

What added support would you need to include programming in your teaching (i.e. another course, IT support at your school, more problem sets, a mentor, administrative support, other??)

Another purpose of the course was to improve problem solving skills for various mathematical problems.

How much do you believe your problem solving skills of the mathematics covered in this course, improved over the duration of the course?

5  4  3  2  1  
A lot  Not at all

Please elaborate:

What connections do you see between this course and other mathematics courses?
APPENDIX D: Journal data referenced by line number
Alice Journals

6/22/2015
Hello [Instructor]
Today we composed hello.c, ftoc, and age.c programs. Throughout these activities I learned how to output a string of text with the hello.c program. The age program taught how to ask for information to input and then combine that with text to output. Lastly the ftoc taught us how to program basic mathematical operations. I think that the most interesting this was the last program ftoc since it combines everything we learned. I think that a little less explanation would have been great. At times if felt a little elementary but that might have just been since I have some programming background. Since I have practice with programming I was able to catch up pretty well and was able to finish up the activities relatively quickly. The only times I felt a little confused was when I was distracted with my own code and didn't hear the next steps. Overall I think that this was a great first day with a quick and concise overview with not a terrible amount of introduction that felt unnecessary. Have a great day!

Alice

6/24/2015
Hello [Instructor],

For the various problems today there was a lot more problem solving than the ones we have been having. The change.c problem I first approached by thinking that I was going to need a way to get it so that I can separate the original amount in terms of 100s, 25s, 10s, 5s, and 1s. My initial thought for this was modular arithmetic so when I approached this I had to think about what else I would need to get the desired result. This involved me just playing around with numbers and doing the math in my head and seeing how I could get the computer to do the same.

For jack.c we were introduced to a new command the while loop. I had used this command in some of my previous programming classes so I had some familiarity with it. In that previous class I had done a similar problem so all I did was try to remember. I had to make sure I manipulated a variable so that it continuously changed every loop and that it eventually had an end.

For multiples.c I used a very similar approached as to the one for jack.c. I also used the different way that You had approached jack.c with the multiple variables that I didn't have. Using [Instructor’s] example and the my experience I was able to write the program fairly quickly.

In the end, all of these problems required a bit of direction but mainly just using previous exercises as well as playing around with the program to achieve the desired result.

Have a great day!,

196
Hello,

Today in class I worked on uniformity.c as well as craps.c.

For uniformity.c we as a class came up with the command \(1 + \lfloor 6 \times \text{drand48}() \rfloor\) which simulates a die by outputting an integer from 1 to 6. We came up with that command by first starting with the command \(\text{drand48}()\) and seeing how to manipulate it to get our desired result. By doing this I could see how we can take a simple command and manipulate it enough to do something that I did not relate to it at all. With this uniformity.c we also wanted to check the randomness of the command, in other words we wanted to see if the command was uniformly random. Which was easy to do with if statements.

The craps.c command wanted us simulate the game craps by rolling two dice and seeing if we won or lost depending if we rolled a 7, 11 or not. We also wanted to find the theoretical probably of winning as well as the probability based on our results with the program. I approached this problem by first looking at the uniformity.c program and using that as a base. I took the dice simulator command and wrote it again so that I had two dice rolling one through six. Labeling each of those dice as a value I then wrote a command to add the sum of each of the resulting rolls. I knew that I wanted to check if that was a win or a lose, I then turned to if statements to fix this problem. With the if statements I set it up to check if the roll was a 7 or 11. If that was the case then count it as a win. If not then count it as a lose. I knew to find the probability of a win with my results I would need to roll A LOT of times hence I knew that I would want a loop. I put all of my commands inside a while loop so that it would operate as many times as I told it too. Inside the if statements I then added a counter so that it could keep track of how many times i lost or won.

To find the probability from my results I, outside of the while loop, took the count of wins that had been tallied and divided it by the amount of times that I told the program to roll. This gave me the desired result. I was able to check if this result was indeed correct by first finding the theoretical probability of this game. I found that by using my previous knowledge of probability. First I found all the ways to make seven and eleven with two dice. I noticed that there were eight ways when counting double of combinations, aka 5 and 2 and 2 and 5. I knew that there are 36 different combinations of results hence I could calculate that the theoretical probability was going to be 8 out of 36. This happened to be VERY similar to what the computer program was telling me.

Have a great day!

Alice

6/30/2015

Hello!

Today in class I worked on the two programs first_match.c as well as last_match.c. The former problem required us to write a program that would input
a series of numbers and then check if the first digit appeared again anywhere else in the sequence. The latter problem did a similar thing but backwards. This problem also inputted a series of numbers but then checked if the last digit had appeared anywhere else in the sequence prior to it.

With these problems we worked in groups today. This worked very well for me actually. The way we approached each problem was by first each telling each other how we thought we could approach the problem. If both of us had some similar ideas we would start there and begin typing up the code we thought would work. As we typed we continued to problem solve and as ideas came to us we each suggested it and worked off of that. This was very helpful because at times my partner could have an idea that might have not come to me right away and it would have taken me a while to come across it. Also there were times this I knew I wanted to do something but I didn't have the path to get there perfectly planned out yet. With a partner we were able to work through this problem and figure out how to get to the goal.

For these two problems I let my partner [Greg] do all the typing. We were both communicating a lot and telling each other what we were thinking as were were doing so I didn't feel like it hindered my understanding of the program. [Greg] was very good at telling me what he was typing as he was doing it so I was constantly in the loop. If we came across something that I didn't understand I would ask [Greg] and he would explain. When it came to code that I came up with [Greg] was very willing to type it up and was always willing to cooperate with me. I didn't type up the code not because I didn't want to or feel comfertable doing so, it was just because of ease and he already had the program ready. I don't think that it would have been better if we would have jumped around typing in the one program since I hink it would have broken the pace and groove of the collaboration. With one person typing we could move more quickly.

Overall I think working with someone worked very well and it got a lot more work done than I would have by myself. It was also a great opportunity to learn from someone else's ideas and bounce and get an insight into how someone else approaches these problems.

Have a great day,

Alice

Hello,

This week I feel has brought a lot more of fun challenges than the previous week. Last week we just worked on a lot of small problems that were not much of a challenge sometimes and just took a few minutes. With these problems there requires more problem solving and more critical thinking. The fact that we had to work with others also gave this week and improvement. I think that really helped a lot.

Have a great weekend!

Alice
Barry Journals

Monday June 22 class:

1. We worked through the intro to C language in Linux. We created a Hello message, then learned how to input a number, and finally were creating a formula for F to C degrees conversion. This last program is unfinished as of yet.

2. We learned basic commands for printing to the monitor and inputting a number value for a variable.

3. Already I am thinking about the possibilities for creating formulas and programs. Especially since formulas are entered much like you would on a TI calculator.

4. I followed along with the class pretty well - I think a few others may have missed a few things, but when you can work with a partner it helps quite a bit.

5, 6, and 7: No confusion at the moment. I am interested in how this might look in high school. There are some students who have expressed interest in software/game design already and I'd love to show them basic programming.

Barry Journal Entry for Tuesday, June 23, 2015:

We began by completing the Fahrenheit to Celsius conversion program. ftoc.c takes an input value (that I assigned f) and uses the formula c=(f-32)*5/9 to output a Celsius value. Here, once double precision variables are established, formulas need to be carefully written so that no integer values pop up to when they should have given a double precision: for example: c=5/9*(f-32) The 5/9 would evaluate to 0 because the program reads 5 and 9 as integers and 5/9 in integer world equals 0.

The next task was an if-then scenario with less than/greater than decisions. bigger.c asks for two input values and compares them. It will tell you which is greater or if they are equal.

The final task was to create a program (I called tens.c) that returns the tens-place digit of any real number. Given at the end of class, a number of pennies entered should return the fewest amount of change in dollars, quarters, dimes, nickels, and pennies. Example: input of 467 would return "4 dollars, 2 quarters, 1 dime, 1 nickel, and 2 pennies" Challenge: remove any zeros. Make it say "and" before the last returned coin.

Barry Journal #3: Wednesday June 24, 2015:

We began class by working on the change.c program that takes the amount of change and converts it to lowest change. I used fmod several times to achieve the amount of dollars, quarters, dimes, nickels and pennies. The initial program was done relatively easily, but the challenges of outputting singular vs. multiple/removing zeros/switching the "and" before the last coin were tedious to get all at once.

Then we created a program to repeat a line of text as many times as the user wanted. This used the while loop function to repeat printing while the variable stored was greater than 0. I had the stored variable subtract 1 after every print so
Finally we created a program that lists the first 20 multiples of a given integer. This program is not yet complete because I have a small bug to work out regarding the type of variable being used.

**STRATEGY -**

I think about the order of how the program runs. What it should do first, then next, etc... Then I go to scratch paper and figure out the math involved - how I can do the math steps as so on. Afterwards, I go back to the computer to try to fit the computer steps to the math steps and start typing.

6/25/2015 Thursday

I completed the challenges for the change.c program by brute force: I listed out just about every possibility in the form of a ton of if statements. I was copying and pasting and changing the new paste to save time. Tedium, but I think I got it (there may be some typos in the dialogue printed).

Then I tried to work on the program to find primes. I barely got started when the class began working on the random generator.

We were introduced to the drand48() function. Then worked through how to use it to get random numbers between whatever bound we'd need. (For use with rolling a die, for example).

6/29/2015 Moday

Today we completed uniformity.c (rolling a 6-sided die) and craps.c (rolling 2 dice and looking at their sum). I also competed primes.c (determining is if a user input number is prime).

**QUESTION POSED IN CLASS ABOUT HOW I CREATED CRAPS.C:**

I worked on rolling a six sided die over the weekend.

I used the tips at the end of class for using drand48() to create a random integer between 1 and 6 by essentially applying function multiplication(by 6) and translation(add 1) to drand48.

I created variables to use as counting bins for ones, twos, threes, etc.

In order to play many opening rolls in craps, I adjusted my uniformity.c code by including another drand48 to roll a second die. I figured it would be easier to start with the basics for rolling one die and simply do it again. I let x= first die and y= second die. Then I added them together, z=x+y. I increased my bins to go to twelve and counted the number of sevens+elevens for the total wins. I reported win amount out of n rolls and percentage. (I also counted losses on 2,3 or 12).

We compared this to simply using drand48 only once to roll 12-sided die to see the pitfall in adjusting drand48 to model something incorrectly.

<btw, the probability of winning by rolling 3 dice is 42 out of 216.>

I completed primes by using fmod function to find the remainder when a user input number is divided by some n. I set the n at 2 and looped it to increase by 1 until the remainder hits zero. Then if n equals the number at that point, it would be prime. I included cases when the input number was 0, 1 or 2. And then when it worked I felt like a computer wizard.
Today we worked on finding matching numbers from a user input list. The first program looked for matches of the first against any subsequent number. The second looked for matches of the last number and any previous number. This program required an array to store numbers. Mort and I worked together on these problems. We both took notes as [Instructor] talked and then we both needed think time to start. We created a few shells in our program - sketched out a while loop or an if statement. Then we talked to each other about our initial ideas. We looked at each other's work and agreed on a skeleton to start by thinking through the logic together. We ended up using my computer for first_match.c and her computer for last_match.c. Once code was written to complete a program, we went through it on paper by creating bins for the variables and changing them with whatever input we liked. Then we ran the program and looked for bugs and such. We went back through the computer logic until we figured out what was wrong (which took some time). In the end, we arrived at two working programs and learned about some pitfalls.

7/2/2015
This week: We really got into the math in programming. Probability. Wow. I could find the experimental probability for two bishops attacking WAY easier than trying to get the theoretical. The pi problem was fantastic.

7/6/2015
Using the pi problem as a basis for our coding, we created a program to find the ratio of the volume of a cone to the volume of the cylinder that circumscribes it. Then the frog program!

Cory Journals

Daily Journal/Reflection – 6/22/15 – day 1

(1) Activities attempted: Terminal commands, emacs use, compilation via terminal. Three programs saved: a first hello, age output and input, celsius/fahrenheit conversion.

(2) What did you learn from each activity: I've done some programming in C++, so I'm familiar with the general format. Each of the C commands is different, though, and I've also never worked through the terminal before. I'm starting to recall that sense of how the machine loads and saves data, and I'm learning the command line from scratch. (This is fun!)

(3) Emphasized more/less: I felt like everything was necessary to the subject. I wouldn't mind more emphasis on "doubles" and how/why the computer reserves
bits and data -- I've forgotten most of that, and it really is helpful to understand when developing programs.

(4) How well did you do...? Everything's coming along well. I feel good about it.

(5) Confusion? The storage via &lf was unclear to me, but Instructor gave me a pretty good idea upon asking.

(6) Help with anything? Nothing for now.

(7) Comments? Thanks for organizing this -- I think it's a wonderful idea!

Journal for 6-24 – day 3

Work strategies: My general approach is to first think the program through in terms of structure. If it's something I'll need to compose in multiple pieces, then I'll first jot out a little web before beginning to type in emacs; if it is relatively straightforward or all of a piece, then I'll begin composing my program straight away.

I will first try to do the computational portion, and initially use only the most rudimentary input/output to determine whether my calculations are working correctly. Once they are, I'll begin refining my output statements and methods. For instance, in the challenges portion of the dollar-change conversion, I first used the most basic "__ dollars __ quarters, etc" output statements to check on the conversions, then gradually broke these guys down into if statements, and finally created further embedded ifs and elses to account for correct punctuation and "and" usage.

So far this is the best method I've found, and it's working for me (for now!).

Thanks for reading,

[Cory]

Journal 6-29: Constructing craps.c

I first wanted to give the user an opportunity to choose to roll; to do this, I created a variable (roll), and asked them to enter 1 to play/roll. If they did, 1 was stored into roll and we entered an if statement in which everything else was to be embedded. (Nothing needed to be outside the if statement, because nothing needed to happen if they didn't want to play!)

I approached the overall construction very similarly to previous while loops that I've written, since I feel like I have this approach down. I set a variable n = 0, and then ran the while loop as long as n < 1,000,000 (at the end of each loop, I incremented n up by 1). This is my standard approach.

Within the while loop I thought it would be good to account for two rolls of the die. I stored one random result into x1 and the other into x2, then saved the sum in a variable called "sum." I hadn't thought through the sample space yet, but I didn't want to take any risks by trying to condense the counts prematurely (into, say, 2-12).

I then set if/else loops: if the sum turned out to be 7 or 11, I'd increment up my bin called "win" by 1; if not, I'd increment up the bin called "lose" by 1.
Outside of the while loop (that is, once the sum was tested for win vs. loss 1,000,000 times), I stored the percentage of wins in a variable called percent. (I'd calculate this by dividing wins by the total of wins + losses). I then printed the number of wins, the number of losses, and the overall percentage of wins.

This worked quite smoothly! I don't think there were any bugs (except maybe my forgetting to close my int main bracket!). Again, my strategy was to deal with each level of the program separately: to run the calculations, to house them in the while loop, and then to house this in our big ("do you want to roll?") conditional statement.

Thanks for reading!

Daily Journal: Working with a partner!

I really enjoyed working with Esther, but I have to confess that I especially liked that her approach was similar to mine: rather than talking everything through from start to finish, we gave each other time to mull things over before comparing notes. In general, if I feel pressed to address something before I've had at least a moment to frame it in my own mind, privately, I'll be a little scattered from the start.

This was perfect: the conversation was brought out into the open where we could think out loud and catch each others' thought-flaws before they were set down, and in general I felt more "invited outward" -- if that makes sense. You noted that folks spoke more freely with Instructor as a result of working with a partner, and I think that this makes perfect sense: the outward space was already engaged, obstacles toward it already overcome.

Also -- and I think this is interesting -- because math-type thinking can sometimes be a bit slippery within the mind, there is something specific that takes place when it is set outside: something like setting it on solid ground.

Esther and I both had roughly the same approach to the programs, and in the second one we wound up making the same error (to do with comparing the two different sets of array counters and keeping track of these). There was a kind of parallel timing that arose in our thinking, I'd say, and we even jumped to the same conclusions (independently!) at right around the same moment.

Thanks for reading,

[Cory]

Journal: The week in review!

As far as the programming itself goes, I feel like I am on par and still really enjoying all the new tools coming our way. Working in the classroom has gotten a little bit tougher for me, though (yesterday in particular, which then threw me a bit today as well). As problems become a little bit more complicated, I really have
to descend into myself, so to speak. My partners lately have been pretty different
from me stylistically -- a little chattier throughout, maybe? -- and although I've
really enjoyed them as people, I have to confess that it's much harder for me to
keep picking up where I left off. As I think I've mentioned before, I really have to
clarify the heart of the matter within my own mind before I can fiddle with it
externally. Once I have a clear vision, then I'm game!

Also, I'm beginning to realize how uncomfortable I am with theoretical
probability, and how uncomfortable that fact makes me given that I'm supposed to
hold a math degree soon! I'm glad that I'm catching this blind spot now, at least.
I'll have to dig into that throughout and after this class, I think, and this is
providing a good start and a really nice way of approaching it more confidently.

Thanks for reading,

[Cory]

7/9/15

Instructor,

I just wanted to give you a heads up and apologize in advance if I don't make it in
for the last day of class tomorrow. I've been feeling pretty queasy and having
some trouble staying asleep -- hopefully it'll clear off, because I'm enjoying our
class so much (if it's not obvious!) and I've been looking forward to it every day.

If I don't make it, do you think you could send the survey (and maybe any
journal prompts that come up) my way? I have dice.c complete and defective.c
more or less planned out, so I would just have to pop by the lab before Sunday to
type that one up.

If I don't make it, thanks so much for a wonderful class! I'd love to come back for
the graphics version.

Denise Journals

[6/22 -- day 1] Hello,

We attempted writing our first program independently using basic input and
output commands. We also learned about basic Linux commands like creating a
file, changing directory and compiling a program.

I learned that I have to be very careful about the syntax.

This class was very interesting because I got to write the program on my own and
debug it too. [Instructor] did a great job of explaining every aspect of the
command so it was easier to understand, although I fell a little behind a couple of
times but eventually caught up with your help.

I finished writing first two programs and I think I get it.

I am very excited for this class and looking forward to learn more complex stuff
but a little worried too!
Thanks,

[Denise]

[6/23 – day 2]

This is the first program [age.c] I worked on independently. It helped me learn and understand basic C and Linux commands, syntax and utility of each as well as how to sequence the commands logically to get desired output. This program [ftoc.c] requires the user to input temperature in Fahrenheit and converts it into Celsius. In this assignment we learned to assign the result of a mathematical formula to the variables, types of variables (double, int ), and order of operations. Before typing the program, I made a flowchart on the paper and wrote down all the steps involved in the program with the examples, and it helped me greatly. I decided to continue using this strategy to help me write more complex programs later on.

[6/24 – day 3]

I am feeling more confident about writing the programs now than three days ago and it feels really good!

Debugging the program on my own or with the help of my partner has helped me a lot. Now I can spot the mistakes in syntax much faster than before.

Working out the solution on the paper before writing the program had helped me a lot. For example in change.c, I first took a number to work with and then began writing commands in sequence and the expected result next to it. Also writing out the variables and its description also helped me find the point of confusion fairly quickly. This program [tens.c] extracts the digit in tens place from the number entered by the user. I used three new functions for this program: fabs( x ) – finds the absolute value of a number x , floor(###.### ) – returns the closest integer from a number and fmod (x, y ) – returns the mod or remainder when x is divided by y . I used a number for example and applied the commands and noted down the results after each step. This process helped me see if the sequence of commands I used was correct and if it produced the desired results.

This program [change.c] displays the dollars, quarters, dimes, nickels and pennies for the value of cents entered by the user. This was a little complex program to write compared to what we had done so far in the class. I had to write the instructions on paper a couple of times before I felt confident about typing the code. But once again writing commands with examples and results as follows helped me write the program with fewer numbers of errors than I expected.

[6/29 – day 5]

When I began to write this program [CRAPS] I really did not think about the theoretical probability but visualized myself as playing games with two dice and noting down the result on the paper, recording the number of total trials and
in the end dividing the number of winning combination by the total number of trials. I simply translated these steps into c commands using while loop.

This program simulates the game of rolling a die multiple times and finding the probability of each number being rolled experimentally. We can then compare the experimental probability of rolling each number with the theoretical probability.

As we increase the number of trial we can observe that the experimental probability gets closer and closer to the theoretical probability. For the purpose of simulating the roll of dies the random number generator command drand48() is used.

[6/30 – day 6]
I think working alone in the first week helped me learn the basics better. Once I was comfortable with basics, working in group was more helpful as we could discuss different strategies or commands to achieve results.

By working with [Victor] today I learned about few debugging techniques like running part of programs to check if we were getting the results that we were intending to.

I think I liked it better when I was typing the code, because that made me pay more attention to what I was doing rather than just listening to my partner and nodding.

[Week 2 journal]
Working in groups was very helpful. I discussed various strategies of solving rooks.c and birthdays.c problems with my partner. Also discussing some of the strategies used by other classmates helped me understand logic a lot better.

[7/9 – last day]
In this program [defective.c] I tried to find the probability of getting a defective car out of 30 cars from two companies: Factory A (12 cars) and Factory B (18 cars). Factory A has 5% of cars that are defective and Factory B has 10% of cars that are defective. This program proved to be very challenging to write. It took me a long time to figure out how to simulate the defective cars from both the factories. Unfortunately I could not finish the program within given time limit.

Esther Journals

[6/22 day 1]
1. Which activities did you attempt that day? Which did you complete?
   I completed all activities.

2. What did you learn from each activity?
   The basic syntax and how to make sure I have all the skeleton components for a program in C.

3. What was the most interesting thing? What could have been omitted or emphasized less?
The most interesting thing to me is the specific codes that are different in C than in Python, which I learned the basics of a couple years ago.

4. How well did you do with the activities? Are you catching on to programming and the problems?
   I was able to complete the activities with no issues, and I am catching on fairly quickly to C.

5. Did anything cause confusion?
   There were a couple interface issues where windows were disappearing, but I now know how to access them again.

6. What do you need help with?
   Right now I'm doing fine, I just need to practice the basics.

7. Any other comments or insights?
   Fun so far! Thanks for the help!

Day 3: 6/24
1. Which activities did you attempt that day? Which did you complete?
   I finished the change.c program and finished challenge 1. I did not get to challenges 2 and 3. I finished the jack.c program.

2. What did you learn from each activity?
   I learned how to use "%_.lf" - I've been wondering how to do this.

3. What was the most interesting thing? What could have been omitted or emphasized less?
   The most interesting thing to me right now is troubleshooting how to make my code shorter and more efficient.

4. How well did you do with the activities? Are you catching on to programming and the problems?
   I did well with the activities, and am catching on fine to the world of C.

5. Did anything cause confusion?
   I lose & signs here and there sometimes. Just because I'm not paying close enough attention.

6. What do you need help with?
   Everything is fine so far.

7. Any other comments or insights?
   Not that I can think of.

8. What are the strategies you have been using?
   So far I have been using strategies I recall from learning Python before, but basically my strategy so far when writing a program has looked like: create the file, copy the shell in, type in the first prompt/scan/input bit, then compile the program and run it. If there are any issues with the basic things, I fix them before moving on and adding any of things that are specific to the program. I compile and run the program regularly so I can check and fix any bugs as I go, so they
don’t get lost too deep in the code. I am also keeping an up-to-date google doc of all the programs and commands as I go.

Day 4: 6/25
1. Which activities did you attempt that day? Which did you complete?
I finished challenges 2 and 3 from change.c, started (and finished) random.c
2. What did you learn from each activity?
I learned what drand() does, which is pretty cool.
3. What was the most interesting thing? What could have been omitted or emphasized less?
The whole random distribution thing is interesting. It makes me wonder how the computer is working behind the scenes.
4. How well did you do with the activities? Are you catching on to programming and the problems?
So far the activities are going well. I am doing fine with C.
5. Did anything cause confusion?
I was confused about why I was getting the same numbers every time when using drand, until it was explained a little more.
6. What do you need help with?
I can not think of anything at the moment.
7. Any other comments or insights?
I would be interested in checking out [Instructor]’s code using the “flag” thing in the loop. I have an issue with an extra comma at the end of my program, but I can work with that by adding in an “enjoy!” or something at the end, so it looks less sketchy.
8. What are the strategies you have been using?
Still following the process of compiling and running often to check for bugs. The program codes are not long enough yet to need this, but I am also a fan of “commenting out” anything that is giving me issues and letting it back in the program a little bit at a time to see what is good and what is bad.

Day 5: 6/29
How did you think about and develop the craps.c program?
I knew I wanted to roll 2 six-sided dice, so I started with that piece of code. I made my loop to do it a certain number of times (based on user input) and store each result in a bin for its value. Once I got that far, I needed to add some more variables to my program so I could view the percentages to make sure they were correct (by comparing to the theoretical probabilities).
I had the program print a table of how many times we got each sum and a table of what the percentage was for each sum. Once I was satisfied, I added the percentages of the sums 7 and 11, and had the program print that, with a statement that the probability of the player winning after n rolls is that percent.
I keep running into bugs where I have my variables defined wrong (double vs. integer) or I am missing a % or an &. but since the error messages are pretty
specific, it is usually pretty simple to troubleshoot. I compile and run my
programs on a regular basis as I add things to them.

Day 6: 6/30
1. How was working with pairs different than working individually? Did the
strategy change?
[Cory] and I are both pretty independent workers and both feel very comfortable
with the programming aspect of the class, so I feel like it was not very different
with us working together than separately. We both learn better by writing our own
code, so we still both wrote our own programs. We had a few little differences in
syntax, but the programs were basically the same.
I like to check in with the people I sit next to anyway, whether it’s because I have
a question or because they look like they might be stuck, so there is still
communication and problem-solving together, even if we are not specifically
working in “groups.” The strategy did not change, because we would have still
talked about issues, had they come up.
I definitely learn best by doing, and I find it easiest to think through my code by
writing it, and I think [Cory] probably feels the same way, so we both started out
by just jumping in and writing our programs, then talking as we came upon issues.

Day 8: 7/02
1. I felt good about this week, and there were some challenge problems that I
felt particularly proud of myself for finishing and making a successful program. I
am enjoying writing the programs, since it is like solving logic puzzles, which are
always really fun for me. I installed virtualbox with ubuntu on my laptop at home
so I can write and test programs at home also. It is nice to be able to check if
something will work even when I’m not in the lab. When I finish programs in
class, I also like to check in with [Kate] to see what she is doing, since she is
working on different things and it’s fun to see what she’s doing. Today I helped
her come up with a more efficient way to write a program than she had been
thinking about, so we still have teamwork even though we’re writing different
programs. I am keeping up well with the programming and having fun solving
problems.

Last day, 7/9
Task 3: green sheet - Factory A produces cars where 5% have major defects, and
Factory B produces cars where 10% have major defects. A dealer received 30
cars: 12 from A and 18 from B. A customer randomly bought one of the 30 cars
and found it to be defective. What is the probability that it came from A?
NOTE: my solution is not as robust as [Instructor’s]. S/He made the assumption
that 12 random cars were chosen from factory A, where 5% of the cars were
defective, and 18 random cars were chosen from factory B, where 10% of the cars
were defective. In this way, the number of defective cars on the lot from A can
vary (probably 0 or 1) and the number of defective cars on the lot from B can vary
(probably 1 or 2 or 3). This is a little more random than what I have. I did not
randomize the selection on the lot, but assumed there were 300 cars instead of 30.
This gives me a constant value of 5% of the cars on the lot from factory A, and
10% of the cars on the lot from factory B. I then randomly selected a defective car
from this set of cars and checked if it was from A or B.
Additionally, I realize that not all inputs of fa and fb will work. It definitely works
for the numbers in the problem, and most other values as well, but there are some
that will not work as well. However, the probability will still be close for these
values anyway.

**Greg Journal**

[6/22 – day 1]

1. What activities did you attempt today? Which did you complete?
   1. We created a program in C-commands in linux that we titled hello.
   2. We created another called age that prompted the user’s age and displays a
      sentence saying “[Greg] is (inserted value) years old”.
   3. We talked about a program to convert from Fahrenheit to Celcius.

2. What did you learn from each activity?
   I learned a lot of commands! How to display things, how to create variables and
   prompt for their input, how to re display those variables (“doubles” and
   “integers”.)

3. What was the most interesting thing? What could have been omitted or
   emphasized less?
   Being able to create a program that took an input, stored it, and displayed it in a
   new context was the most useful or interesting. I think that we could have omitted
   the final activity.

4. How well did you do with the activities? Are you catching on to
   programming and the problems?
   I did well at first, but fell behind trying to take notes on what I had already
   completed to retain for the future.

5. Did anything cause confusion?
   I started to fall behind and still didn’t quite catch what certain commands stood
   for or where they came from. There seems to be more specificity with commands
   I have learned that I do not fully understand yet.

6. What do you need help with?
   Getting my age program to work still.

7. Any other comments or insights?
Fun first day, I just need to brush up on my typing and other “technology” skills on the computers.

[6/23 – day 2]

1. What activities did you attempt today? Which did you complete?

We created a program in C-commands in linux that we titled hello.
We created another called age that prompted the user’s age and displays a sentence saying “[Greg] is (inserted value) years old”.
We talked about a program to convert from Fahrenheit to Celcius.

2. What did you learn from each activity?

I learned a lot of commands! How to display things, how to create variables and prompt for their input, how to re display those variables (“doubles” and “integers”).

3. What was the most interesting thing? What could have been omitted or emphasized less?

Being able to create a program that took an input, stored it, and displayed it in a new context was the most useful or interesting. I think that we could have omitted the final activity.

4. How well did you do with the activities? Are you catching on to programming and the problems?

I did well at first, but fell behind trying to take notes on what I had already completed to retain for the future.

5. Did anything cause confusion?

I started to fall behind and still didn’t quite catch what certain commands stood for or where they came from. There seems to be more specificity with commands I have learned that I do not fully understand yet.

6. What do you need help with?

Getting my age program to work still.

7. Any other comments or insights?

Fun first day, I just need to brush up on my typing and other “technology” skills on the computers.

[Greg] (Day 3) Describing working on change.c

I’m starting by taking initial conditions from Word and set up my program
Then, I made sure I understood question: I’m taking a few hundred pennies and converting them to the min number of dollars, quarters, dimes, nickels, and pennies.
Hmm…
Looks like I will need 6 variables, but maybe more. I was confused a bit about extra storage variables.

Now, how do I name an integer (referred back to Word document where he had written up commands). Can I us $ as variable, thought it was too risky and changed it to “dol”

Used printf command to prompt for the input, wanted to add a new line but couldn’t recall if \n or /n and determined it was \n since / is used for division.

Now I’m going to look at the dollars. I need ‘modf’ or what was it…oh, it’s ‘fmod’.

dol=fmod(x,100);

Now how will I get my ‘quarters’?
q=fmod(x,25);

Wait, this won’t work. It will give me the remainder after I pull out quarters and not the number of quarters. [Greg] then did an example and discovered, “I need to divide by 25 and use the floor function.
Oops, dol is wrong, too! “I should have done an example to start with”
After trying to compile I had several errors, although all the same type, I used %lf in the printf commands where I should have used %d. Then it worked beautifully.

It took about 45 minutes to get this program to work.
06/24

Today’s journal we were asked to reflect on our strategies for problem solving while completing the activities, and specifically what we did step by step while writing the programs to follow up on or modify these strategies.
The activities we worked on today were:

change.c, converting an integer number of cents / pennies to the least amount of coins / change.
Challenges for change.c: singular place values, omit zeroes. Insert “and” before final place value.
jack.c, prints a phrase from The Shining a user specific number of times using a loop.

Strategies:
- Begin with program template, copied and pasted from a word document holding my previous programs and commands.
- Make sure template includes closed braces in the int main part of program.
- Define variables to be used and start thinking about how they will show up in the commands.
Looking back over these, I am glad I took the time to look back at my word document each time to help me remember the commands so that I would be more
confident later in the period writing new programs. However, I should have been quicker to think of what would happen to my “variable” if I actually choose my own example integer and start doing the operations I am talking about on paper before I begin thinking a step ahead in the code. Sometimes getting your shell of an idea in the command lines can be helpful, but in this case I actually lost some time because I wanted to use a command that mathematically I would have recognized as a mistake much sooner. It is interesting learning new notation for operations that I already fully understand, and noticing myself make silly mistakes because of this extra step in translation. I should use my mathematical problem skills first where I can before I start trying to “fit the puzzle pieces together.”

When I went to first compile the program, I was also able to develop a new strategy of looking for the error messages and interpreting them to solve the problem in my commands as opposed to brute force

[6/25 – day 3]

1. What activities did you attempt today? Which did you complete?
   1. For the course pretest on probability. I completed all three problems, but did not feel strongly about at least one of them.
   2. I took another look at my code for change.c, and cleaned it up using the fmod command. I began working on some of the challenges for change.c.
   3. I began working on the prime.c but kept getting distracted by other programs! So prime.c is still a work in progress, as are my challenges for change.c.
   4. We then began working on an activity to introduce a new command drand58(), called genrandom.c. This generated random numbers between 0 and 1 a user specific number of times. We were then introduced to uniformity.c to determine whether or not these random numbers were randomly distributed, or that the digits’ outcomes were uniformly distributed.

2. What did you learn from each activity?
   1. From the pretest, I learned that I learned a good problem solving approach last term in my probability for teachers course to look at any general probability problem. I did not immediately recall formulas, but generally as I started creating the sample space, I would remember more theory as it would make sense in the activity of modeling the question.
   2. I learned how to clean up some of my code. I also got some work on while, if statements, and more practice using loops. I learned a little bit about the random number generator, but certainly want to play with it more!

3. What was the most interesting thing? What could have been omitted or emphasized less?
   1. The most interesting was the random number generator and beginning to talk about the program to test if it is truly “random.” However, I felt rushed trying to grasp the code discussion beyond just the mathematics involved.
4. How well did you do with the activities? Are you catching on to programming and the problems?
I did not feel as confident as I should have on the pretest, just having taken a probability course in the spring! However, I suppose I completed the problems with no notes or resources. I barely used a calculator. The programming I felt good about, but did not have time to finish much today as I felt like I was jumping around.

5. Did anything cause confusion?
The final activity I did not follow the code fully as I was working on this journal!

6. hat do you need help with?
Starting the uniformity.c next class period!

7. Any other comments or insights?
Still fun so far!

[6/29 – day 5 of 12]
How did you write the craps.c program? How did you verify it worked?
I started by copying my main program shell from a word document, defining doubles (not worried about how many yet), and closing the braces.
Next I sat back and thought about the process. I knew I would need to loop the game a certain number of times. I knew I would want to come back and make a user input number of games, but decided to just run it a set one million times first.
I knew I would roll two dice, so I would need two variables to store the rolls each time. I would also need a variable to store the sum of the dice, and bins to store the results of a sum of seven and a sum of eleven. Now I input the actual doubles, seven, eleven, x, y (the rolls), sum, wins, n. I wrote the while loop with n<1000000, and stored results into the bins.
I realized later writing my program that I may not even need two bins for seven and eleven, but just one for the wins. However, I decided to leave this in case I felt like later modifying the code to display information about the number of sevens or elevens each, respectively.
I had some troubleshooting issues:
When I tried to compile for the first time, there was an error message “In function ‘main’:
Craps.c:18:10: error: lvalue required as left operand of assignment”
I found my error: I had x+y=sum. CORRECTION: sum=x+y. This is so that it will actually store the sum properly, and the loop can work. I changed this, recompiled, and my program worked. I then worked on a first modification for a user input number of games. I then added a print of simulated probability outcome at end as well. Then I completed craps.2, the 11 sided (or “mistake”) program, as well as the rolling of 3 die.

[6/30]
What activities did you attempt today? Which did you complete?
We first tried to figure out what a given program will do, given the program code, whatdoido.c. We completed this, and wrote the program to play with it. We then were to write a program first_match.c, that was to take a user given sequence, and output one of two things: The first number does appear later in the sequence, OR The first number does not appear later in the sequence. We completed this as a group as well. Next, we began a similar activity: Want to take a user given sequence that truncates upon entering a negative, and compares the last number prior to the negative to the rest of the sequence, stating whether or not the last number occurred earlier in the given sequence. We completed this in our pair as well.

What did you learn from each activity?
In deciphering the whatdoido.c code, I learned a useful problem solving skill for thinking through programs: the simple idea of working through your code logically on paper. Make some if then statements yourself for what might happen if you (the user) do this or that and follow the logical steps in the code to explore. I feel that we continued to use this sort of problem solving as we wrote our code in groups for the following activities, and this helped us to catch errors or syntax we were missing at first with this “check the steps” mentality. Are these braces closed? Did we make our step variable increase by one? In the first match program, we worked quickly to a solution by brainstorming the most simple ways to store the first entry, comparing each time in the loop. I learned about some more loop ideas by playing with this activity, as well as the following that required we look at comparison to the last entry. This activity was more challenging, as it required the use of a new command for variables arrays. I really enjoyed this final activity, and once we talked through the program first, we were able to write rather quickly to get to compiling and testing to find the bugs in our step variable “if” and “while” commands. I learned about thinking through multiple loops and step or flag variables, and in this began to make // comments much more often inline in the program so that coming back to look at it, I will understand what each part does.

What was the most interesting thing? What could have been omitted or emphasized less?
The challenge problem that compares a user given 3 digit number and compares to the previously given user input sequence as in the other activities, truncating with a negative.

How well did you do with the activities? Are you catching on to programming and the problems?
I feel that my groupmates and I did extremely well with the activities, and got to testing the code and talking about other possibilities by the end of each activity.

Did anything cause confusion?
When we were working on the last_match.c program (that compares the final entry to the rest of the entries) we ran into some bugs with our integer step variables, and how far to run the comparison while loop to ensure we compared to all variables in the array except the last non negative integer.

What do you need help with?

Nothing after today it seems!!!

Any other comments or insights?

Very fun day today! Great exploratory activities / forcing us to think through first.

GROUP WORK:

Did the process change working in pairs?

How did you strategize?

I would say that the process for me certainly changed in pairs. I normally work at a slower pace getting started on problems then others do. I was typing, so this may have been helpful for me to be able to hit the ground running with the group’s idea, and I can support up front with still asking the meticulous questions about what we are starting with. I found that my partner thought similarly to me at the start as well, by wanting to copy the shell of the previous program when we had already created a lot of the necessary commands for the loops to give us the user input sequences.

We strategized well by talking through the layout of the program first, what steps we would need, and how we can most efficiently use the loops to reduce lines of code. We also would go through quickly to check that things we closed, etc. Anytime we added a variable in to the program we made sure in was the proper type for its use, integer or double, as well as making sure we set it to zero to start where necessary, and add one to step variables. We also worked well together checking bugs in the program with suggestions for what to look at. When we checked for bugs, as well as up front prior to writing the program code for the while loops, we worked on paper to ensure we understood which inputs in the arrays we were looking at (it is always interesting to see the counting numbers force even math minds into confusion if you do not look at some examples of how your logic is “stepping” through the loop).

[7/2]

This week we did more in depth probability applications. We were able to begin generating random numbers, and learned about the use of pseudo random number generators, and seeding to ensure “more randomness”. I thought the idea of being
able to check results from random number generators against other’s code was quite interesting / useful. We used loops in much more depth to be able to run several trials of the same “game” or experiment. Being able to move back and forth between the theoretical and simulation probability has been helpful for problem solving approaches when coding, or simply just checking to see results. Finally, arrays have been extremely helpful for storing information from these loops, and simplifying data sets for checking conditions or comparisons within subsequent loops. I feel quite strong about my new c programming from only 2 weeks of the course so far.

[7/6]

What activities did you attempt today? Which did you complete?
We wrote a program to approximate the ratio of the volume of a cone to the volume of the cylinder circumscribing it. We completed this quickly! We then moved to frog.c, a program that simulates a frog jumping 3 times, and then evaluates whether or not the frog is within 1 meter of its start point, then using this to approximate the probability of this event. Next [Barry] and I came up with a challenge for ourselves after hearing a classmate talk about a user input number of frog leaps!

What did you learn from each activity?
The volume program was quite exploratory as [Barry] and I brainstormed on our process, so we got a good review of our process and got to discuss at length the struggles with the setting up of the geometry, and the possible applications for students in geometry, calculus, or probability. The conditional probability application was particularly fun to think about! In the next activity I learned some new commands to be able to use trigonometry operations, and an approximation of pi.

What was the most interesting thing? What could have been omitted or emphasized less?
The most interesting thing was looking the different ways students approached the nested conditional statements in these activities.

How well did you do with the activities? Are you catching on to programming and the problems?
I did well! I think I’m catching on! I am making many less mistakes, and often able to compile the first time with no errors, or simple fixes that I can locate much more quickly now.

Did anything cause confusion?
We struggled for a few minutes setting up the first geometry problem. Once we checked some points and had more coffee, we were able to tackle the rest of the geometry in the later problems with more ease.

What do you need help with?
ny other comments or insights?

[7/9 – last day]

What activities did you attempt today? Which did you complete?

Today I completed problem #1 from the Post-test, dice.c, a problem from the pre-
test, but as a simulation program. Yesterday I completed problem #3, defective.c.

We then looked quickly at coding ground on
tutorialspoint.com/codingground.htm

What did you learn from each activity?

I learned that I can program these kinds of probability simulations quite quickly at
this point! I used an array for the defective.c problem yesterday, but used some
notion of probability theory to take the simulation down another level, using the
idea of conditional probability, or baye’s law.

What was the most interesting thing? What could have been omitted or
emphasized less?

I was most interested in the solution we were given for the defective problem. I
really like the use of drand48() to generate a percentage that represents the actual
given probability that a car from factory A has a .05 chance of being defective.
This allows the simulation to seem a lot more like the real world problem that is
being simulated, rather than scale the number of cars to be integer values.

How well did you do with the activities? Are you catching on to programming
and the problems?

I moved quickly, and had few compiling errors. I felt confident enough to not use
any scratch paper, and moved directly to writing the program. My method was to
re-write the problem in program comments, and then start the int main code. I
would then re-write a concise version of what I wish to simulate in comments as
well. This would allow me to start creating a few variables necessary to the
problem, and my first loop variable. Once I began, I would simply write a new
comment line whenever I began a new section of code, and again to complete
large loops, explaining the steps and new variables. Any time I introduce a new
variable, I make sure to return to create it at the top of the program, as well as set
it to an initial value in the correct line of code. This is a method I have developed
to help keep me from having compiling errors later, with programs that entail
several embedded conditional statements as these have.

Did anything cause confusion?

I was slightly confused looking at scratch because it was such a quick endeavor! I
might like to see that again!

What do you need help with?
Any other comments or insights?

**Kate Journal:** Second time with class

Day 1, June 22

1. I attempted the hello.c, age.c, and ftoc.c. I completed all three of the activities.
2. I relearned the command codes to print, scan, assign variables, put text on new lines, compile code, and run code.
3. The general computer information could have been emphasized less. For example, minimizing windows. The most interesting thing was ftoc.c because we actually assigned an equation to a variable.
4. The activities were fairly easy for me because I took this class last term. I had forgotten most of the command codes, but I remembered that they existed so I am able to look them up and/or ask for them.
5. Nothing was confusing today.
6. At the moment, I do not need help with anything.
7. I was able to have my age.c ask for both the person's name and age, and print both components. I did this by looking up how to input strings.

Day 3, June 24, 2015

1. I attempted the change activity, jack activity, and multiples activity. I have completed all of them.
2. I re-learned how to program a loop.
3. I enjoyed the challenges on the change activity.
4. I think that I did well on the activities. Programming is coming back to me quickly.
5. I forgot an &, which messed up my program for a while.
6. I do not need any help currently.
7. I did ten if statements, which took care of two of the challenges for the change problem at once. It took care of both singular and zero values.
8. The two main strategies I used to help me today were to write some of my ideas out on paper and to use print statement as a way to test my program as I went. For the change problem, I used an example number on paper to quickly see if the commands I planned to use (mod and floor) made sense before I began programming. I also used print statements to check that my program correctly calculated the number of dollars, quarters, dimes, etc... as I went so that I did not get too far into my program only to encounter several errors.

6/29/2015

1. I attempted the uniformity program and the sorted list program (this was instead of doing the craps program with the rest of the class). I have completed the uniformity program, but I am still working on the sorted list program. So far, I have a program that allows the user to input data points, and it can printed the original array. My program can also go through the entire list, find the smallest, and put that at the top of the list. It can also handle the case when two or more
numbers are all the smallest number. I have not reached the point where it sorts
the rest of the list though.
2. The uniformity program used skills that we learned last week. I relearned the
syntax for an array while attempting the sorted list program.
3. The sorted list program is the most interesting thing today because it is
challenging me.
4. The uniformity program was very easy for me. The sorted list program is more
challenging.
5. Nothing is confusing.
6. I googled to find out if there is a sort function in Linux, and it looks like there
is. However, it looks complicated, and I have never used it before. Therefore, I
am trying to write my program using only while loops and if statements, but I
have not succeeded yet. I am wondering if it would be simpler to use the sort
function or if I am on the right track with the while loops and if statements.
Currently my loop checks each data point against every other data point to find
the smallest. It then moves the smallest data point to the first position in a new
array.
7. None.
8. To help me with the sorted list program, I "played computer" using a small
eexample of only three data points. This showed me what I was missing in order to
find the smallest value in the list.
6/30/2015
1. I attempted the sorted list activity and the bag of marbles activity. I completed
the sorted list activity, and I am just beginning the bag of marbles activity.
2. I learned that it is possible to define variables and functions outside of the main
program, but I am still working on how to execute that.
3. The most interesting thing is the thing I have not done yet but am working
towards, which is writing a program to simulate drawing five cards from a
standard deck.
4. The activities are going well for me.
5. Nothing is confusing.
6. I may need help on creating functions outside of the main program since I am
still learning how to do that.
7. None.
8. To create the sorted list program, I initially started on the computer but wrote
things out on paper whenever I got stuck. When I was stuck on the logic, I talked
to [Instructor].
7/2/2015
This week went well. I finished the bag problem, and I have made a lot of
progress on the card problem.
Lester Journals
Day 2 (before class) 6/23/2015  [He was able to join]
My future Ph.D. advisor has recommended that I learn maple and mathematica. Several people including Professor […] have suggested that it might benefit me to learn C or C++ as a part of understanding these programs better.

I e-mailed […] in the computer science department, and she said that it would be fine for me to audit her CS162 course. I think though that maybe she was meaning for me to sign up and pay the tuition charges to audit. I'm not sure, so I e-mailed her back for clarification.

In either case, I'd really like to learn C if it might be of benefit to me. [Esther] mentioned to me that you are teaching a class where she and the other students are practicing using C. Do you think that I might be able to just come and sit in on that? Might that be possible?

Thanks again for your advice and help with everything!

6/24/2015

Thank you so so much for letting me do this! I am very grateful for the opportunity!

I have so far only done the change program. My strategy was to first try to identify the method my own brain uses to solve the question. I have calculated appropriate change for so long that I was no longer cognizant of how I actually do it, so that took some thinking. I had to then try to think about how to use the tools available to me (the commands [Esther] had written out in the shared document) to try to re-create that thought process on a computer. I then had to figure out how to solve the challenge of being able to move a decimal over and separate the first two digits from the problem, using those tools. From there, the rest was basically arithmetic, syntax, and keeping track of variables.

I am finding that it really helps to have taken Doctor […] Intro to Num Analysis class. Having done so makes things like logic, structure, loops with counters, etc. more intuitive and I catch on more quickly.

Again, thank you so much for the opportunity!

6/30/15

Working with a partner, I found that I had to spend far less time compiling and fixing my programs from really simple but annoying bugs. I did the typing, but both of us were engaged and watching the characters appear on the screen. I tend to structure my programs by outlining them in comments on screen, and my partner outlined similar ideas in text on a pad of paper. We then came together and implemented the best of both our ideas.
I found it easier to work with a partner than by myself, mainly because having two sets of eyes engaged on the code makes it much less likely that a small typing mistake will cause five minutes of confusion and nose-wrinkling! ;)

7/2/15
This week made total sense! I am now working on making a program which uses simpson's rule to approximate pi instead of just throwing darts. It will be epic!

Lisa Journals

6/22/2015
We attempted to command the computer to say Hello. We then asked the user to input an age and the program output a statement including the users individualized answers. We began a program to covert Fahrenheit to Celsius. I did not finish.

Today was a refresher on my undergrad classes. I have seen most of this but definitely do not remember it all! The most interesting was being able to have the user input and then enter a command that outputs whatever the user input. I think I did well. I was able to move along quickly and add different commands to make it look pretty without being told. The last program is a little confusing currently but I have not yet had time to work on it yet!

Ready for tomorrow :)  
6/24/2015
Today I completed change.c, jack.c, multiples.c.

My strategy for change.c was more or less trial and error. I thought out a process in my head about how I make change if I had a large amount of pennies. First I would look at the hundreds and that would determine dollars. After I decided on how many dollars I no longer needed my hundreds place.

This is how I approached my code. Divided my users number by 100 to find the dollars then I subtracted that amount from the users number because I did not need it any longer. I just kept printing my remainder to check to see if my code was working properly and I had the appropriate change listed above. I worked my way down to pennies. Mostly just trial and error.

I was excited about finishing the first challenge by making the unit singular if it only needed one of the unit. I was able to apply an if statement!

jack.c my neighbor helped me out. I could not think of how to make the loop stop. I was able to apply what my neighbor helped me with on the multiples problem to
complete it with ease! Once I figured out that I can multiply my number by which line I was one so it would find all of the portfolios.

6/29/15

This one definitely took some thought. I first started by trying to develop the game. I need to get the program to roll the dice twice. I ran into an issue of the program printing two different number generated dice values. However it kept giving me the same two numbers. So we then decided that it needed to be in a loop and I had it run through one million times before it finally spit out two different face values. I then made w stand for the sum of die one and die two. I then went into three if statements. One that says the sum is 7 you win! Another that says the sum is 11, you win! And finally an else statement that said your sum was not 7 or 11 so you lose.

Next we to develop the probability using the law of large numbers. I created a loop that rolled to dice like above a million times. Similar to our uniformity program, every time I rolled a sum of 7 I increased my sum7 bin by 1. Every time I rolled a sum of 11 I increased my sum11 bin by 1.

When the loop was finished I made it print how many times I rolled a sum of 7 and how many times I rolled a sum of 11. I then added the totals together and divided by one million since that is how many times I rolled my dice.

This gave me a probability of 0.222, which matches my theoretical probability of 2/9 or 0.222, so I believe!

7/1/15

I am so sorry that I forgot to email yesterday. I totally spaced once I got home!

I loved working with a partner. We were able to complete the program very efficiently because we were able to check each others work. Plus be able to talk it out and negotiate with each other on the best plan. Finding errors became easier because there were two pairs of eyes on the program. The real benefit I think was being able to talk out the since with each other. How is our loop working? What are we asking the program to do? Is that what we want?

It was very helpful!!!

Again sorry for the delay!

7/2/15

This week feel like it has totally ramped up our skills! last week felt like we were really just learning the code. While this week has felt more like application. Working with others has been so awesome because you can think things out. This weeks programs are tough and make you feel very successful when you complete the program successfully. I feel invested in my programs, Wednesday we left without our program working properly and it was stressful to leave. I thought about it the entire drive home!
It has been a fun experience and a lot more fun than when I was completing this course in my undergrad. Application is KEY!! Have a great weekend.

7/9/15

This code has a customer buying a car from a dealership. The dealership buys cars from two different factories. A has 5% of their cars as defective and B has 10% of their cars as defective. If the dealer buys 12 cars from A and 18 cars from B.

What is the probability that the customer bought a defective car from factory A. I set up two arrays for the factories. Factory A had 20 cars which means that one is defective, so I set slot 1 as a one representing a defective car. Factory B has 20 cars which means two of them are defective so I set slots one and two as a two, they represent defective. I then created an array for the dealer. I randomly filled the first 12 slots with random cars from factory A, then filled the remained 18 slots with random cars from factory B. Then I randomly selected one car from the dealer and if it was a one I knew it was defective from Factory A so I increased my factory A defective bucket. If it was a two I knew it was from Factory B so I increased my Factory B bucket.

After one million trials I calculated the probability from being defective from Factory A and Factory B. Then I divided the probability of Factory A by the sum of the probabilities! I feel like this is slightly cheating! HA! but resulted in the correct answer.

Melinda Journals

6/22/2015

1. Which activities did you attempt today? Which did you complete?

--> I completed the hello.c, age.c, and began working on the ftoc.c files. I played around with the age.c program and was able to create a sort-of Fate program where it asks for your age and favorite number, then adds the two together to determine the age at which you will die. It was fun

2. What did you learn from each activity?

--> I was refreshed on programming language, syntax and formatting and was introduced to working with prompts in C, which is a language I have never worked in before.

3. What was the most interesting thing? What could have been omitted or emphasized less?

--> Just the coding information in general. I don't have any feedback on anything that could be emphasized or omitted yet, but I do think that the format of the class is really great. [Instructor] is very helpful and knowledgeable and it is nice to have [Instructor] to help troubleshoot, along with supportive classmates.

4. How well did you do with these activities? Are you catching on to programming and the problems?
The activities today were do-able but tricky and fun, simply because it's been a while since I have done any programming.

5. Did anything cause confusion?
-- Not today

6. What do you need help with?
-- TBD, I will let you know ;)

7. Any other comments or insights?
-- I really appreciate the focus on primary and hands-on learning within the classroom environment. It's also very nice to not have out-of-class work.

6/24/15
Today I worked on the making change problem makechange.c as well as the loop problems: jack.c and multiples.c. I was able to complete all of theses activities and I attempted but did not complete any of the challenge problems with makechange.c.

In order to solve some of the computational problems I have been working them out on paper and playing around with numbers to see if my results match my desired results. I have also been working a little consulting my neighbor, [Molly], who has been helpful when I get stuck. I also try all of my options before I call for help from [Instructor]. Usually when I run into an issue it is something where I know and understand the issue, but cannot figure out what is missing or wrong within my code. Many times its something silly like a capitalized letter or a misspelled command such as "printf."

The class is going well. I feel like I am learning a lot and challenging myself appropriately. I really enjoy the aspect of troubleshooting and find it the proper amount of frustrating.

Tomorrow I would like a little bit of time to work on the programs that I have done so far in order to organize them into word files and type up descriptions for when I put them into my portfolio. I feel like I would need maybe 30 minute to work on this and get things together.

6/29/2015
Today I worked on the problem for simulating a game of craps. The purpose of the program was to simulate rolling two different die and finding the probability of rolling either a 7 or an 11.

In order to piece this program together, I started by thinking about the physical properties involved in simulating this game of craps. I knew that in order for it to work, you would have to roll to dice. So what I did was I created two variables to represent the dice which I called x and y. In order to get the code for that, I just
copied over the line from my uniformity.c program which simulated the rolling die.

From there, I knew I would need to set up a loop with a nested "if" statement. I set up my loop first and said that if the number of rolls left is greater than 0, do the if statement, and then when the if statement is done, decrease the number of rolls left by 1. One pitfall or bug that I had in my program was that for a long time, I had my line of code for decreasing the number of rolls by 1 was inside of my "if" statement. That resulted in only decreasing my number of rolls if I had rolled a 7 or an 11. So I would always get a result of winning every single time (10 out of 10) because I essentially only counted rolls that won. (If I didn't win, my code just said, "Oh that one didn't count" and re-rolled again until it got a win.)

One other thing that was a little bit tricky was remembering the code for the situation that I wanted to choose for as my win. I knew conceptually that winning meant rolling a 7 or an 11, but it was a little bit tricky to remember how to translate that into c language---> (x+y==7.0 || x+y==11.0). The other thing that may or may not have caused issues in my code was making sure that things were coded correctly for double vs. an integer. This has kind of been an ongoing/tricky thing because sometimes I will have errors and I go back and look and see "well does it matter if its 7 vs. 7.0?" Sometimes it makes a difference and sometimes it doesn't and that can be a little frustrating.

In terms of working on the theoretical probability of the situation, I am not sure if coding it truly helped to think of what the probability would be. I believe that doing the coding helped me think through the physical process of playing this game of craps. In order to calculate the probability, I thought of doing 6*6 for the rollings of the two dice and then counting up the different ways in which you could get a 7 or an 11. For me, there was a pretty big disconnect between thinking through the programming of the game and counting the probability of the chosen situation. Mentally, it felt like I had to take my head out of thinking in code and programming in order to solve the problem of the probability. In order to do this, I moved out of working on the computer and began writing the number combinations out on paper and thinking of the sample space.

I am very surprised that you mentioned that lots of students thought of the problem as the chance of rolling a 7 or an 11 out of 12 rolls because I very much did not think of it that way. I am a fairly literal thinker and it makes much more sense to me to think of the problem as broken into two groups of six (Roll 1) and (Roll 2), rather than out of the two rolls together (out of 12).

I really, really liked that [Instructor] mentioned about why it might be nice to use a computer program like this in a middle school classroom or something of that nature in order to prove your results. I have students simulate different probabilities in my classroom and it is always very hard to convince them of it.
is also something that I struggle with when I think of probability, because even
though it can show a pattern, it isn't going to guarantee what is going to happen on
your next roll. Every year I have some students who are really able to think
abstractly, can understand the mathematics behind the ideas, and accept that fact
that the probability will model the situation. There are other students who are like,"Yeah but that number means nothing to me because when we all did the tests,
even though 1/6 of the time I was supposed to roll a 2, I got a 3 or a 4 every time."
It is very difficult to engage those students in the rest of the material because it
diminishes a little bit of trust between the teacher and the student. I find that if the
student doesn't trust the things that we are learning, they have a very hard time
with the material and often fail to meet the objectives set forth in the lesson,
simply because they can't get over the notion of probability being imperfect. I
wonder if a computer program could be a valuable tool to help students overcome
this disbelief. It would definitely be another piece of support that proves that the
probabilities are accurate, but I could also see students saying,"Well yeah that's a
computer so it's super smart and can just make up the probabilities really
quickly." It would definitely be something to play around with and test out.

6/30/2015

It was extremely synergistic and helpful working in a pair today. I worked with
[Lisa] and she and I have a similar way of working on problems: we have a
similar initial approach to problems so when it came to setting up the framework
of the problem we worked really well together. However, when we ran into
issues, we both have a kind of different approach to debugging which was really
helpful. The nice this also is that we were concrete in our approach so before we
tested our program out on the computer, we tested it out on paper first to see if
we were thinking through the problem correctly. This is different than what I
would do if i was working independently.

I found it very helpful to work in a partnership because I feel like I learned more
than if I was alone. I think that I would have given up much more quickly and
would have asked for help if I didn't have a partner to bounce ideas off of. I was
primarily typing while we were working so it's possible that my partner had a
slight disadvantage because she did not get as hands-on of an approach to the
problem.

One little speed bump that I came across was after we made our program and
tested to make sure that it worked, it was hard for me to go back and add in
comments independently. I think that part of the reason for this was because the
two programs we worked on today were very similar so it was hard to remember
specific pieces and clarify them with comments. I think also that these two
programs had some new syntax that was very significant but very small. So for
example there may have been something that was really really essential to my
code, but because it was a simple line, I did not remember to comment it out to really highlight how it works.

 Impressions of the Week:

This week I had a very positive experience working in a group and a pretty negative experience working in a group. I feel like on Monday and Tuesday, while we were working with arrays and doing the firstplace.c and lastplace.c programs, things went well. I feel like these were problems I could have done independently but it was really nice to have a partner to work with, to bounce ideas around with, and work work a little more efficiently.

When working on the birthday problem on Wednesday and Thursday, I also worked on a problem, but I felt like I was wayyy out of my league. I very much did not feel comfortable working through nested loops without any scaffolding around it. I felt like my partner understood the problem and saw how to piece it together. My partner did a good job of explaining things to me and I understand what was happening; however, it was not the same approach that I would have taken so, although we got a solution, I did not feel like I could take ownership of the solution because it wasn't really my way of thinking of things. I kind of shut down on Wednesday, but worked a little on the problem at home. I started to rework the problem independently on Thursday and I typed up my own code that I thought that it would work. However I went to go test it, it totally didn't work, I didn't understand why and I felt like I didn't have anyone to help me.

I really feel like I was floundering by the end of the week. I felt like I hadn't been very productive and that I might not be able to accomplish any future problems. I also feel a little overwhelmed by having the task of typing up my portfolio hanging over my head. I have been keeping track of programs we have done and ones that I have typed up. It is getting kind of hard because we've been changing seats so much that my programs are all in different places. I've been taking screenshots to show my printout results so now what I have to do is re-create the programs on different computers by copying and pasting.

I also feel like I am not learning a whole lot of tangible things that I could take into my classroom. I understand the big-picture ideas behind why bringing more programming into schools can be a really positive thing, but I don't really feel empowered to make it happen. I would appreciate a little bit more information like this.

Molly Journal

6/22/2015
I wrote 2 complete programs - hello and age. I began the ftoc program. I learned how to access the Linxx operating system, as well as some basic Linxx commands. With the hello program, I learned how to print. With the age program, I learned how to scan. The most interesting thing for me was the similarity to other programming I have done in the past. I thought the instruction was perfect in pace and content. I feel like I caught on very well and finished the first two programs fairly fast. I am not left with any confusion or need for help at this point.

I really enjoyed myself today!

6/24/15

I wrote 3 complete programs today - change, jack and multiples. With the change program, I practiced using the truncate and remainder commands. To approach this problem, I used paper and pencil and worked on an example number to establish the process that was needed. I then switched to the computer and initially started with embedded commands of truncate and remainder, but it got too confusing, so ended up establishing more variables and separating the process more. With the jack program, I learned how to use a loop. I did not use paper and pencil with this. I had trouble placing my equations in the correct place and ended up having to reference [Instructor]'s solution for the correct placement. With the multiples program, I practiced using the loop command. The most interesting thing for me was learning to correctly use the loop command. Again, I thought the instruction was perfect in pace and content. Although these programs were more challenging for me, I am not left with any confusion or need for help at this point.

6/24/15 extra

I used paper and pencil and worked on an example number to establish the process that was needed. I then switched to the computer and initially started with embedded commands of truncate and remainder, but it got too confusing, so ended up establishing more variables and separating the process more.

6/29/2015

To develop the Craps program, I just started right away on the computer and used the Uniformity program as a base and adjusted it. I had to change the variables used, added another random variable equation, used an "or" statement inside of the "if" statement, created an equation to calculate the percent of sums of 7 or 11, then adjusted my print statement. I realized that I had an error of not including outside parentheses around my "or" statement, so added them. I was also printing out the number of counted 7 and 11 sums instead of the probability, so I had to fix the variable used in the statement. I verified my result by finding the theoretical probability using the sample space for rolling 2 die.
Today I worked with a partner instead of individually. We initially started discussing the problem, but realized that we needed to get our ideas in place with some structure, so we started programming. We then discussed/compared as we found common breaking points or were stuck. For me, I need to be able to process through the problem by myself first. I am very visual, so have a hard time with a discussion if I can't visualize it. After discussing, we would change our coding. My partner did most of the typing on the first program, then I did most of the typing on the second program. We then copied what was needed from each other. I found that I do better when I am typing and can discuss as I get stuck or compare when finished. For the second problem, we pulled out pencil and paper to act out the program, but still couldn't find our mistake without help from [Instructor].

The process to solve the problems mainly stayed the same even though I was working with a partner. The difference was that we compared and adjusted our programs throughout the problems.

I found that the problems really increased in difficulty over the course of the week. On Monday, the Uniformity and Craps problems were fairly easy for me. I was able to try one of the Craps challenges. I was working next to you that day, so basically working by myself. On Tuesday, I worked with a partner on the First Match and Last Match problems and they went pretty well - we were able to complete them on our own. On Wednesday, I worked with a different partner on the Rooks problem, which went well. However, we really messed up the Birthday problem and had to leave with it unfinished which was frustrating. Today, I was still not able to complete the Birthday problem (I'm having a problem with the counters), so I plan to work on it over the weekend. I started the Area problem with a different partner and I feel confident that we have a solid start. Yay! This program (Birthday.c) took me the longest to write. I had not thought of the probability correctly.

7/6/15 I have worked with a partner for many of the programs, now. We used pencil and paper first to draw out the scenario and determine what we wanted.

I scanned in my notes with the OS commands and C commands.

I greatly enjoyed the class. Thank you so much! Have a great rest of your summer!

Mort Journal

6/22/15

1. Program asking age and outputting age. Completed
2. input and output commands
3. fewer programming classes now than in the 70's was surprising
4. Did pretty well. Not great at syntax
5. no
6. a class google doc with notes may be useful. Here is mine

6/24/15
First I try to copy all the stuff at the top correctly. Second I try to figure how to do
the first step in the problem. The I decide if that first step is repeatable, to get the
rest of the steps. If not I try to think of a way that would make it repeatable, so I
am not doing new code for each step. Then I type the code in for the steps. Then I
dialog the code. Then I run through some cases to test to see if it works.

6/29/15
I talked to my teammate and we discussed the number of variables we would
need. We decide that we would only need to get two rolls and a sum. Then we
would add the numbers to get a sum. We would need 2 more variables, one to
count the number of rolls, and one to count the number of wins. So we made a
loop that counted wins an then found the portion of the total tries.
I only had problems with syntax and remembering how to write an or statement to
count 7's or 11's. My teammate looked in his notes.

7/2/15
Mostly the week has been very fun. The challenge has been at the right level. I
have found the terminal interface to be a little challenging to deal with.
Also I tend to lose focus with the teacher walking through programs, may be
two nice to have people explain their own programs.
I wish we had some discussion of the problem. Why is the probability of
coincidental birthdays so high in a group of people? Can this be explained while
walking through the program?

Muriel Journal

6/22/15
The activity that we did today was creating our first program that asked us "How
old are you?" waiting for our reply of an answer and then would state how old we
were back to us. Today was just getting some background knowledge of basic
commands that are used within programming. [Wilma] and I were able to make
that program work, we were satisfied with our work.
We learned the basic commands like having the commands to print out words we
tell it to and to wait for our answer, if there was a question.
It is very interesting that how much you put in that will be the equivalent of what
the program will put out. You get back, what you put in. For the first day, I want
all the knowledge we can get, so nothing should have been omitted or lessened in
my opinion.
Since I have some previous knowledge from Computer Science classes from my
undergraduate, I felt comfortable to an extent. I felt like I did well with the age
activity and was able to figure out my issues without help from [Instructor].
Sometimes doing less at first and then building on your work is helpful because
you can fix little problems instead of having multiple to try to fix all at once.
I had a small misinterpretation of [[Instructor]]'s writing but that was easily fixable and not too much of a problem. That was the only thing that caused confusion for today.

No help needed today but probably once I get to the converting degrees program I will come to some stopping points.

Very excited for this class!

6/24/15

The activity today was starting with change.c which is inputting how many pennies you have and the program will separate that number into dollars, quarters,... pennies. The strategy that I usually follow going about my process is where I start with the shell and then go piece by piece and see if my program is still working. This way I notice my mistakes and errors early and don't have to deal with a bunch errors when I think I have finished. So by doing each piece individually I can gradually work my way through knowing if I can get one part to work, I can move forward and get everything to work. Being next to someone where we are doing the same program is definitely helpful since you can bounce ideas off of each other, so I agree with [Researcher] about asking peers and neighbors about questions and ideas before going to [Instructor].

I completed the original change.c program and also made it work for challenge 2, challenge 1 on the other hand I have an idea and I am close but I still haven't gotten it down just right. Challenge 3 hasn't been attempted which I may try out later today. What I learned from these activities is that there are multiple ways to get the same program and everything will still work the same. It makes me think of individual students and how they have their "unique" ways of writing down and interpreting math.

The most interesting thing for today was learning how to get rid of the 0.000000 because it was quite annoying to me and I have been wanting them to go away. I am most excited to not have to deal with them now and have it look better.

Nothing in my opinion needed to be omitted today, everything seems rather important because there are many ways to interpret what is going on. So laughs, opinions, trial and errors, and time is all very helpful and useful!

I am having some difficulties of trying to think in a programming way because it doesn't seem to fit within my brain too well. Sitting next to [Wilma] makes me feel more comfortable though because I can rely ideas and thoughts off of her.

Working together with a partner makes things easier for me, where I can work through and not just get the answer and be done. That's not how this should go about. I feel like I am growing to be a better programmer than I was a couple days ago. The problems that are caught by program so it can run are easily noticeable and I can fix them and go about what I need to do. It's getting the idea in my head and working through it that is difficult for me.

Even though everything is being taught in a linear fashion, I get confused of all of this knowledge that we know and keeping it in order in my head to rely onto the
programs. Even after a couple days, I feel overwhelmed with information but
doing the programs is helpful to use our inputs and outputs in use.

As long as I have a partner to talk with, I feel fairly comfortable and have the help
that I need. I think it would be nice if [Instructor] could come to each student
individually and check how things are going, so we can ask questions right then
and there. Not all the time but every now and then would be nice. I know you all
are busy.

Time efficiency is being spent well and the class has a good atmosphere.

[Instructor] does a wonderful job making the programs relate-able for real life and
being fun. I think having laughter in a classroom is very important.

6/25/15

There were many activities today: multiples, primes, random, and uniformity. I
was able to complete multiples with [Wilma]'s assistance and we have both been
working on primes but have yet to finish. Random was completed with the entire
class and we only got into the discussion of uniformity.

I learned how to terminate an if/while statement so that it wouldn't proceed further
and include all other if/while statements using the command

exit(EXIT_FAILURE). I am nervous that I might be abusing this command, since

I don't fully understand but so far it is doing what I want so I'll go with it for now.

Also from the other activities that we've done, I've learned that each activity is
just going to get more difficult as we go along. A bit overwhelming but is
necessary since we don't have lots of time. I love having class discussions to talk
through the programs before even writing the program so we also understand how
it'll work.

The most interesting thing today is our discussions on talking about our programs
before turning them into code and seeing if those work. We get a better
understanding by knowing the program before the code. Nothing should be
lessened currently.

I have been kinda overwhelmed with all these programs, I'm mostly caught up
and am intrigued to progress but I get lost in the commands and how to progress
forward without asking for help. So I don't feel like I am doing too well, I know I
have lots to learn and much progress to make. I just like having someone with me
(thankful for [Wilma]) who will talk with me. I am usually able to fix all
problems with the help of the program telling me where they are and then with
some time I can get an idea and try out more options.

I think the weekend break will be helpful for me so I can reboot and go over the
codes again. Maybe some osmosis will be good for me.

6/29

The activities today was assessing and completing the uniformity program and
attempting the casino game craps and craps2. The uniformity of a challenge for
me with having this new command of drand48(), since I don't fully understand it
but am using it since we are suppose to know. Plus I am not sure what other way
the program would be completed without drand48(). The casino game was similar
in ways in the structure and build-up as uniformity but has the differences. I feel
very fortunate to have [Wilma] with me since she has been a MAJOR help for my work in these programs.

I learned from the uniformity that it is similar to Minitab in generating numbers but the programs seem more similar to each other in results than being truly random like Minitab. Also with these programs I was able to learn that even if there are problems ahead, you can work your way around them. Shortcuts can be helpful from time to time, if you know how to use them and make them work as well. Good ole brute force and long-drawn out code can do that same thing and get the same result.

The most interesting is how these computer programs can be made by these codes written by us students which can appear very similar to Minitab. It's an interesting comparison. Nothing that I can think of at the moment that needs to be omitted or lessened.

I think I am doing fair with the programs, I don't feel excellent with these programs because it's just not sinking in as much as I want it to so some information is lost in my brain, causing confusion (thankfully I have [Wilma] to help me out!). Truthfully, I would say that I am not catching onto the programs because they aren't computing in my own mind. These codes is like a different like another language which isn't working in my brain. It seems to make sense when I see someone else's code but going about it on my own, I feel lost and behind.

Sometimes I get confused with the writing on the board because it is not organized, so if it could be more sectional and orderly, I would find that helpful. Talking with [Wilma] and [Greg] make me feel better in working through the codes and getting the program to do what we want. I believe I have all the help I need.

If it would be alright, could we see some students from the class solutions, so we could compare each other's codes.

For the casino game craps, I started out by using the same code for the uniformity since they seemed similar enough to work with craps. So I copied all of that over and then modified it for two dice and then obtaining the sum from those dice. Running this in our while loop 1,000,000 times and then storing the amount that occurred into the correct bins. Unfortunately, I wasn't able to obtain the probability yet so I will look more into it tonight and come back tomorrow.

6/30

Before getting started with the class, I was able to finish the challenge code of primes and the required codes of craps and craps2. I was able to do with this with the help of [Wilma] and [Instructor]. It becomes frustrating to be on the brink of getting the code entirely right but there are just simply minor errors that need to be noticed and fixed for everything to work the way that it should.

Partners is definitely a preference of mine rather than going alone with coding. We were able to talk out the codes with each other, [Wilma] wrote them down to have more of a visual to see the code. I feel better when I am able to write the
code into the program and then compile and run it to check out every part that I
write in.
For today's class, we attempted to dissect whatdoido code without running it, we
spoke to each other (my partner was [Wilma]) and discussed what was happening.
[Wilma] and I came to the conclusion that s was equal to zero and we input x so
the statement s=s+x would be x. Now for the next coding challenge we are doing
the first mach program which was giving a set of numbers and seeing if the first
number given was later repeated in the sequence. [Wilma] was ecstatic to get it on
her own. I fell a little behind in getting stuck in the thought process of the code. I
was able to get the code done and it was very helpful to have [Instructor]'s
version of the code to be on the screen which helped me finish mine.
I learned to not to over-complicate the code when sometimes it is a simple matter.
Overthinking can be a hindrance to the code making process. So talking with a
partner and having both of our minds can simplify the code and it's a much shorter
process in making the code. Talking out your thoughts can make a difference
rather than being frustrated with just yourself and the computer.
The most interesting today was talking about the difference of the first match
code in which we wanted the to see if the first number was repeated later in the
sequence and then now look at the last number and see if it was previously stated
in the sequence. Having the first number is a heck of a lot easier than the last
number. It was a well-needed discussion to hear from [Instructor] about the arrays
and clarifying what can be used for the future. Nothing in my opinion needs to be
lessened or omitted.
I feel like I am starting to get the hang of getting the programs to work how I want
them to but I do feel a lot more comfortable when I am able to see other's codes
so I can keep checking if I am on the right path for the particular coding. Thank
you [Instructor] for sharing your codes after giving us some time to let us figure it
out ourselves! The coding is feeling better and better every little bit. Making sure
to remember the & , "" {} () and having words be written correctly.
The flag is a bit confusing because I feel like it has some significance but the flag
could be defined as anything we want it to be, so what is special about saying
flag?
As for today I feel like I am doing alright and am not too worried plus I have
[Wilma] next to me who has been very helpful in working the programs with me.
[Instructor] is doing a great job in time management to let us figure out code
before showing his and also his explanations are clear and punctual.
7/2/2015
This week was particularly a difficult week for me since we are learning so much
code and then left to do the coding with the knowledge of what we received from
[Instructor]. Definitely bringing in partners was a great relief in being able to talk
about your thoughts/ideas with someone else. It is kind of fun that [Instructor]
gives extra programs for a challenge for some people who like and understand the
coding so they can progress further. I was able to do some of the extras but they
are quite the brain teasers. The structure of the classes moved fluidly and has
great time management of having work time and class time where we can discuss
as a class or individually with our peers. It is very helpful when [Instructor] looks
a a piece of code and then goes through it piece by piece and see what is
happening in the code that is written.

A note I would like to make about a sense of "saving your codes":
I opened a word document and have been screen shooting the codes and
terminals. I have attached this below if you would like to look at what I am saying
(if it didn't come across clearly). People can "save" their codes like this so it can
be see in ubuntu inside of typing it into word itself.

**Shawn Journal**

6/22

1. We constructed two computer programs. One modelled writing text and the
other what for entering a value in response to a text. We then previewed a further
challenge, converting a temperature reading from F to C. Both activities were
completed and the third was initiated.

2. OS commands, C commands, general programming rules and quirks, and how
to maintain an inquisitive attitude to further improve programs.

3. Feeling successful programming and overcoming challenges. In terms of the
content, this is all fairly fundamental material so the concepts are more interesting
than the actual applications. Specifically, it's an amazing achievement of humans
to be able to construct a machine that can process.

4. The main challenges are just the windows, resizing, and non-programming
related bugs that come from moving, collapsing, and resizing windows.

5. The instructions were very clear and well aimed to the audience.


7. Good start to programming, looking forward to working on more mathematical
issues ~

June 29th Reflection

For the craps game, I deconstructed my previous program and just took out the
lines of code that determined random numbers 1-6. Then I created two variables
to represent each dice, set if-else parameters when the values of them added
together equaled 7 or 11, and then constructed a loop to repeat the process for a
number of trials.

More fun came when you asked us to determine how the program would change
when three dice were rolled instead. I've always enjoyed this topic for the reason
that it generates the normal distribution curve. Calculating the empirical
probability was simple since you just add another random variable generator and
tack it on to the previous equations for 7 and 11. Calculating the theoretical
probability came down to analyzing all the different possibilities for equaling 7 or
11 with three numbers 1-6.
The main struggle in the whole program was adapting old code for a new program. I found that it's still much easier to start from scratch and reconstruct the program by copying and pasting pieces of code into it. If I'm just starting from a previous program, there are too many specifics that only work for the specific context and not useful for the new program.

6/30

For the two programs today we were asked to work in pairs. This entailed a bit of discussion, critiquing, and shared problem solving. My partner and I collaborated on the first problem together and the only main challenge was trying to understand each other's logic. We elected to build a program each for the second challenge and then use the time to debug each other's program. It turns out to be quite a challenge to read each other's code, as the logic and organization is quite different. It would make sense to standardize the variables that people use so as to expedite the debug and logic critique process.

6/24

Typically we chat a bit before and after class, we we will discuss the chess problem to trouble-shoot it before putting it down in text. I find that paper is generally useful for penciling out sketches and examples, but it's important to have the general structure clear in your head.

Reflection: July 2nd

The problems are getting more challenging and time consuming. I enjoy when there are specific extensions to each so that we can work at our own pace and feel comfortable with our results.

7/9

This class was a great introduction to programming and I intend to use it in my future classes (in [foreign country]) and studies. More than just opening the door to analyzing empirical probabilities through computer-based simulations, this class (teachers, atmosphere, students, all of it) was inspiring and pushed me to consider the power of computing in various fields of math, such as topology, series, and probability.

I intend to keep in touch during the coming academic year. Many professors in the math ed department here at [Workshop Institution] ([Professor A], [Professor B], [Instructor]) have been very encouraging and nurturing to me during these past couple weeks. I look forward to coming back and deepening studies with the faculty. Hopefully we'll be able to stay in touch and I look forward to your guidance during the fall as I prepare an application.

Victor Journal

6/24
I jumped into class today and learned syntax and programming structure together. [Kate] was super helpful in showing me the ropes. We learned some basic structure via the change program and some more complex structure via a "while" command. I need to learn more navigational structure. Look forward to tomorrow!

6/29
This is being super helpful and interesting. I feel like I learned a great deal about loops, if functions and else functions, finally seeing how to use them so they actually work. There are subtle issues, which I learned to resolve by trying to think like a computer, and syntax issues, which my peers and [Instructor] have helped a great deal to resolve.

Today for the gambling program, I started by modeling with two die, calling them x and y, finding their sum and looping that. This didn't work at first for syntactical reasons. Later we used [Instructor]'s method of modeling on one die labeled 2 to 12. This gives different probabilities of winning as the number of ways to roll a is just one on such a die, but there are multiple ways with two die.

I hadn't really had too much time to think through the issues before writing the code, as I was having syntax bugs. These are becoming rare. But glad to hear my initial impulse was right!

Today was a blast yet again. Since [Lester] and I had finished working on the birthday problem and bishops problem last class, I actually spent a fair amount of time working on creating a factorial function program. This was inspired by the fact that near the beginning of class we received a challenge project of determining the theoretical probabilities of the birthday problem, and in the process I came up with a formula that involved factorials. Since I didn't know of a formula for the factorial in C, I decided to try to make one so that I could calculate the theoretical probability and compare it directly to the simulation in the same program. I had a tough time for a while getting the factorial to function, but eventually I feel like I came up with the right code. However, for some reason, it starts giving negative numbers after around 18, which I think may be a problem with the computer's memory size, and not the code, or else it is a manner of displaying the answer I am not used to, in perhaps some sort of exponential form. I would like to learn how to make this program into a function I could use in other programs.

Overall this week has been about integrating my understanding of while loops and creating better ways to use these to get creative. It was a pleasure working with [Lester] yesterday. Today was more solo. I sat next to someone but they said they were a lone wolf. Que sera, sera.

7/7
I have been working today on using character variables to allow for a more expanded and flexible program to test the chips probabilities. I attached the text
of the program. So far everything has lined up with theoretical models. Next I
hope to be able to incorporate a theoretical calculation as part of the program, but
I need to define a factorial function for that I think.

Wanda Journal

6/22
I completed the age problem requiring to ask "How old are you?" input a value
for age, store the value as a variable, and output "You are 'age' years old." I started the ftoc.c file.
I learned a number of commands for the operating system, c commands and some emacs commands. Including "pwd" to show the personal directory I am working in, and cd .. to go back a level in the directory.
Most of what we did today was getting started concepts. I have been considering using linux for a number of years, but never had the time to dig into it. I don't think anything could have been omitted or emphasized less, we seemed to get through just enough.
I didn't have any confusion on material and don't need help with anything.

6/24
Each day as I sit down to start working the challenges from the previous class, I attempt first to remember what I did to get such and such window... before I check my notes. I also try to remember how each program starts before looking. If I get stuck, I try rewriting something in the program and recompiling. If that doesn't work, I check previous programs I have written. If that doesn't work I look over my notes, or check online. If that doesn't work then I will raise my hand to ask a question. So far I've only had about three questions, so the process seems to be working well. I'm also able to remember most of what I need to get started and all of the beginning lines of the program.
I've managed to complete the change.c program and two of the challenges. The last one is giving me some difficulty. I think I have a plan of attack, but keeping track of all the nested if-else statements is difficult. I completed the jack.c program today, and have started the multiple.c program today as well.
I've learned the while statement. How to print a specific number of values past the decimal point I looked up on the first day.
I enjoy the programming. I find all of it interesting. At this point nothing is causing me confusion, and I don't require help with anything.

6/25
I left at about 12:15 today because of an appointment.
Before that I completed the pre-test on probability. It seemed fairly straightforward, though nicely complex. I think I did pretty well on it. We'll see when I get it back.

After the pre-test I worked on change3.c the third challenge for the change.c program, and on the primes.c which was the challenge we were given after the multiples.c program. I didn't finish either, but made some progress on both. I started with the change3.c program, but it was just taking too long and [Instructor] was talking some about who had tried/finished the primes.c challenge so I switched to that. Then before I finished that we started talking about the new function for a random variable. I wrote a program that would print three random variables, though it oddly always gave me the same three values. [Instructor] suggested something like that might happen.

No questions or problems.

Today we completed the craps.c program which simulates rolling two dice, adding them and determining the probability of getting a 7 or 11. For this journal we are supposed to explain our thinking about how we went about the problem. We started by creating a program that would roll one die and determine the probability of each number. So, my first thoughts were about how to adjust that program to work the way I wanted. I needed to roll a second die, and write a sum. That was pretty easy. I also allowed the user to input how many times to roll. This allowed me to try different amounts and see how this affected the experimental probability. I already knew the theoretical probability so when we asked to figure that out, I was already seeing how many rolls were needed to get reasonably close.

Today we worked in groups. I worked with [Lester]. He did the typing of the program. The biggest thing to overcome for me was that he writes programs a little different. I put in a few comments as I go through, and enter things as I think through them. He started differently entering the variables he thought he needed first... More linearly. Entering comments as he went. He also puts the { on if() line, where I put it below... He set up a flag system which he called "flag". I usually use a variable like 'n' or 'k' for my flag. I think it's good to see how others program. I think mine is more organized, but it was challenging to read through his with the differences as we were trying to debug the programs.

This journal is a reflection of the work that we have completed this week. We've been working on some more complicated problems, developing some ways to find the experimental probability of problems that can be found theoretically.
and seeing how many trials are needed to get the probabilities within a reasonable
amount of precision.
The largest problem I have had is in determining how much information I wanted
to allow the user to enter. The first program I created for rooks.c actually just ran
through all of the possibilities, instead of running trials as experiments.
I've found the programs reasonably challenging. For some of them, the birthday
and rooks problems I didn't really know where I was going with the program
before I started it. I've developed a way to create a part of the program that I know
I'll need and have it generate an output to check that that part is working. Then I
build the next part from there. This is the way I'm creating the bishop and the area
problems.

Wilma Journal

6/22
1. Which activities did you attempt today? Which did you complete?
Today, [Muriel] and I worked together since my computer did not have oracle VM.
Age activity: Today, we completed the age activity. This activity required that we
could print "how old are you" then waited for a response. After a response was entered then the program would print "[Wilma] is ___ years old."
Conversion activity: We also started the Conversion from fahrenheit to Celsius activity. We did not finish this activity.
2. What did you learn from the activity?
Today I learned many about emacs (text editor) and many emacs commands.
3. What was the most interesting? What could have been omitted?
I enjoyed the time to work with a partner.
4. How well did you do with the activities?
I feel that the material was very easy to understand.
5. Did anything cause confusion?
The different between C commands and emacs commands. Are they the same?
7. Anything else?
Thank you! I am really looking forward to this class.
6/24
What strategies are you using to approach each activity?
When attempting each activity I follow a similar sequence of steps:
1. First write down each math step needed to achieve the goal.
2. Find the code that will do the math that each step indicates.
3. I write each mathematical piece of code then compile and check.
4. If I do not know the code that I need then I google it:)
5. I have found that writing code is very much a trial and error process.

Example: Change Activity
My thinking process:
1. First I took the example 467 pennies and determined the minimum amount of dollars, quarters, dimes, nickels, and pennies that I could use to represent 467 pennies. Here is where I decided what math steps were needed and in what order they were needed in.
In particular, I needed to take my pennies and divide by 100 to get the dollars.
I needed to take the left over pennies and divide by 25 to get the quarters.
I needed to take the left over pennies and divide by 10 to get the dimes.
I needed to take the left over pennies and divide by 5 to get the nickels.
Then the left over pennies were my pennies.
2. I set up the template by typing the preprocessor information. Here is where I started to write the code for each of the steps before. The tricky step is figuring out how to save the left over pennies from each step. Since I chose to use integers then I needed a different code for the mod function so I googled to find it.
3. I then compiled and tried my program to find some details that I wanted to change. I need to find out how to make spaces between the list of denominations. This was accomplished by trial and error of putting "\n" into many different places.
4. Then I moved onto the challenge problems that ask to not show 0 of something. Here is where I remembered if statements. I know the if statement that i needed so I just tried a few places where I thought that it should logically go until it worked.
Overall, I am still in the beginning stages of code writing and the "trial and error" approach seems to be helpful.

Today, was very frustrating class. I feel like I was able to get started on my programs but would not get exactly what I was looking for. Therefore, I would be trying to debug as the class went on without actually seeing any solution. So, I did not actually finish the Craps programs but I will figure it out tonight.
I started off writing down my big idea of the process.

Generate rolling two fair 6-sided dice by generating two different random variables.

dice1 = 1 + floor(6*drand48());
dice2 = 1 + floor(6*drand48());

Then I knew that I needed to add them together. I wanted to change my variables to integers so that I could use my bin commands that I knew.

d = (int)dice1 + (int)dice2

Then I wanted to tally the number of each output that I received.

bin[d]++;
i = i + 1;

I know that I want to repeat all the above steps so I put it into a loop.

Then I used a print statement to print the number of outcomes in each bin

printf("bin %d = %d/2000 \n", i, bin[i]);

I was able to get the bins to print with the number of outcomes in each bin but I was not able to convert to a percent. Geez, this was frustrating but I will keep working on it.

6/30

Thank you for taking the time to hand out a solution to the craps program. I also appreciated that today we were given time to work but also went over an example code before moving on to the next part of class. I also loved the activity at the beginning of class when we looked at code and tried to figure out what it did.

Today, [Muriel] and I worked together. We completed the craps program before class. I had worked on it last night and was able to get my code to work. As we looked through my code we compared it to the presented solutions. This was very helpful since we were able to start understanding the other methods.

We also worked on the First match & Last match programs. We finished both of these programs. We tended to talked through the code before we started to type. Often one person typed so that we could check our plan & code. This is when we often found mistakes with our code or problems with our plan. After we were able to get the code to work then both of us would type the code.

7/1

Thank you so much for your help today. I enjoyed talking through the logic with you. After my girls went to bed I decided to figure out what went wrong with this code.

Here are my steps:

1. I noticed that I had a semicolon on the end of a while statement so I deleted it.

Then the program compiled and ran.
However, I noticed that I kept getting a probability of 1.00. So I knew something was wrong.

2. I checked to make sure that all my brackets were correctly placed since I tend to forget them often:(. No luck their everything looked good. I also started adding in LOTS of comments to make sure that each block of code was doing what I intended it to.

3. I tried to change all the variables to integers except probability. That would eliminate any mistake that I could have make with the comparison of integers and doubles.

No luck still crazy probability of 1.0.

4. Next, I seeded my random variable generator because the numbers I was getting was causing a lot of matches. That helped a little but still was getting very high probabilities.

5. I added print statements to see what random number were being generated. I also added a print statement that showed the trials had matches.

This was were I noticed a couple errors:

a. I was not clearing my array before each trial. So a bday in one trial was matching in the next trial and counting as a success.

b. I was count the total number of matches for all trials rather than the number of trials that were successful.

6. I correct my logic in my code and tested at 23 people and 70 people for 10000 trials and got approx. 50% and 99% which matches closely to the theoretical probability. So I have working code. WOOO HOO!!

I attached my revised code.

7/2

Prompt: How was the week?

This week went really well. The activities have increased in rigor and become very interesting. I appreciate the link back to last quarter's probability work. I think that it is very powerful to revisit similar programs but explain and solve them differently. I appreciate that although we are using simulations to solve problems we are still being pushed to explain using theoretical probability. I realize that I have to think much differently to solve a problem by theory VS computer simulation. One thing that I have worked on this week is the ability to switch back and forth between these two ways of thinking.