Chapter 2. Methods

Answers to Chapter 2 Review Questions

1. Define the following terms:

   camelCase
   CamelCase is the practice of writing compound names without using blank spaces, but capitalizing the first letter of each name that forms the compound name, like the name CamelCase itself. The very first letter in a CamelCase name may or may not be capitalized, such as CheshireCat or cheshireCat. CamelCase is used with most programming languages and is recommended by many software developers, such as Microsoft.

   encapsulation
   Encapsulation is the concealment of the details of a method (or an object) from the user. Programmers may use an encapsulated method, but cannot see its inner workings. Alice primitive methods, such as move and turn, are examples of encapsulated methods.

   integration test
   An integration test is a test of a software method that checks to see if the method works when it is placed into a larger program in combination with other methods.

   method header
   A method header includes the lines of text at the top of the method that provide information about how the method works, such as the full name of the method and the name and type of any parameters or variables used in the method.

   modular development
   Modular development is the process by which the parts, or modules, that make up a complete solution to a problem are developed individually and then combined to form that complete solution. Modular development is often used together with top-down design in which one concept or big idea, is broken down into several parts, and then those parts are further broken down. The end result is a collection of small solutions, called modules, which collectively contain the overall solution to the original problem.

   off-camera
   Off-camera means that an object in an Alice world cannot be seen on the screen with the camera in its current position.

   parameter
A parameter is a variable whose value is passed from one method to another. Parameters have data types and they are stored in the memory of the computer, just like properties.

**primitive methods**

Primitive methods are built-in, predefined methods that are part of each Alice object. They provide simple basic behaviors, such as move, turn, and roll, and cannot be edited like user-defined methods can be.

**reusable code**

Reusable code includes software modules that are solution to smaller problems that can be applied elsewhere. Most of the software on a computer system is filled with layers of reusable code because computer programs contain many small tasks that need to be repeated, such as getting a user’s name and password. Object-oriented programming and modular development encourage the use of reusable code.

**software development cycle**

A software development cycle is an engineering process in which software developers (programmers and software engineers) design code, test and debug new software.

**test for correctness**

A test for correctness tests to see whether or not a program performs correctly according to its original specifications. It is not concerned with the program’s time or space efficiency, but simply its correctness.

**testing shell**

A testing shell, also known as a testing hull, is a short method created to simulate the environment in which a newly developed method will be run. This is especially important if the method being tested will be receiving values from other methods, or passing values on to other methods. The testing shell can be written to pass known values into the method being tested, and then the output can be captured and examined for correctness.

**top-down design**

Top-down-design is a process in which a single concept or big idea representing a problem to be solved is broken down into smaller parts. If those parts can be further broken down, then the process continues. The end result is a collection of small solutions, called modules, which collectively contain the overall solution to the original problem. This process of decomposition is also known as a “divide and conquer” approach to problem solving.

**unit test**

A unit test is a test of a single software method that checks to see if the method works as expected all by itself. Sometimes a testing shell is used in a unit test to provide the
correct environment for testing a method, especially if the method has input or output parameters.

**User-defined methods**

User-defined methods are written by people who use Alice, and can be edited. That is, they are not encapsulated. Some of the objects in the Alice object galleries, such as the Penguin class of objects, have user-defined methods that were written by the people who created the object, in addition to the object’s standard set of primitive methods.

**variable**

A variable is a memory location that temporarily stores a value while a method is running. Variables have names, data types and values, just like properties do. However, properties are associated with an object, and their values are maintained as long as the object exists, whereas variables are associated with a particular method, and their values exist only inside the method. Once a method stops running, the values of its variables are gone, unless, of course, you had saved them somewhere else while the method was running.
2. Describe the processes known as top-down design and modular development, and how they are used together.

Top-down design is a process in which a single concept or big idea representing a problem to be solved is broken down into smaller parts. If those parts can be further broken down, then the process continues. The process of top-down design leads to modular development, in which the parts, or modules, that make up a complete solution to a problem are developed individually and then combined to form the complete solution.

3. How do organizational charts help with top-down design and modular development?

Organizational charts are most commonly seen describing hierarchies of organizations, such as government agencies. They show how a single unit at the top of the chart can be broken down into several parts at the lower levels of the chart, and how the parts at the lower levels of the chart fit together to form the individual units at the upper levels. In computer programming and software engineering, organizational charts are used to keep track of how modules fit together as a problem is broken down into parts during top-down design and as the modules are put back together to form the overall solution to the original problem.

4. List and describe the advantages of using modular development.

Modular development of computer software has the following advantages, especially for larger projects:

- Modular development makes a large project more manageable. Smaller and less complex tasks are easier to understand than larger ones and are less demanding of resources.

- Modular development is faster for large projects. You can have different people work on different modules, and then put their work together. This means that different modules can be developed at the same time, which speeds up the overall project.

- Modular development leads to a higher quality product. Programmers with knowledge and skills in a specific area, such as graphics, accounting, or data communications, can be assigned to the parts of the project that require those skills.

- Modular development makes it easier to find and correct errors in computer programs. Often, the hardest part of correcting an error in computer software is finding out exactly what is causing the error. Modular development makes it easier to isolate the part of the software that is causing trouble.

- Most importantly, modular development increases the reusability of your solutions. Solutions to smaller problems are more likely to be useful elsewhere than solutions to bigger problems.
5. **How does the practice of object-oriented programming encourage the development of reusable code?**

Object-oriented programming, by its very nature, encourages the development of reusable code. Methods to perform simple tasks can be reused within an object to form more complicated methods, and entire classes of objects created for one project can be saved and reused as needed for other projects.

6. **List and describe the steps in a simple software development cycle.**

In a software development cycle, programmers design, code, test and debug computer software.

- In the design phase, programmers begin to design the methods needed to meet the specifications for the software. Many different techniques are used to design methods.

- The coding phase of software development includes translating a software design into a particular language, and then entering that language on a computer as a collection of software methods.

- During the testing phase, programmers follow a testing plan that to see if the new methods perform according to their original specifications, both by themselves and when combined with other methods. More sophisticated testing checks to see if the amount of time and storage space used by new methods is reasonably efficient.

- In the debugging phase, the causes of any problems that were revealed in the testing phase are isolated and repaired.

7. **What is the difference between a unit test and an integration test? Why are they both used in software development?**

A unit test is a test of a single software method that checks to see if the method works as expected all by itself. An integration test is a test of a software method that checks to see if the method works when it is placed into a larger program in combination with other methods. They are both used because methods need to function correctly by themselves, be reasonable efficient, and work in conjunction with other methods without undesirable side effects. Unit tests can determine if a method performs according to its original specifications in an efficient way, and integration tests can expose undesirable side-effects.

8. **What are the differences between primitive methods and user-defined methods in Alice? Which of these are encapsulated methods and what does that mean?**

Primitive methods are built-in, predefined methods that are part of each Alice object. They provide simple basic behaviors, such as move, turn, and roll, and cannot be edited. User-defined methods are written by people who use Alice, and can be edited. Some of the objects in the Alice object galleries, such as the Penguin class of objects, have user-defined methods that were written by the people who created the object. Encapsulation means that the inner workings or inner details of a method are hidden from the user. Only primitive methods are encapsulated in Alice.
9. Describe two different primitive methods that can be used to find objects that are off-camera. What are the differences between using these methods?

The “camera point at” and “camera get a good look at” primitive methods can both be used to “find” objects that are off camera. The “camera point at” method simply points the camera at an object, but does not move the camera. The “camera get a good look at” method will point the camera at an object and move the camera to get a better view of the object, almost like a close-up of the object.

10. Describe what parameters are, and how they are used in Alice methods.

A parameter is a variable whose value is passed from one method to another. Parameters have data types and they are stored in the memory of the computer, just like properties.

Whenever a method is called the values for the method’s parameters must be defined. For example, when the move method is called for an object, the values for the method’s direction and amount must be specified. Some parameters, such as the style and duration parameters for the move method, have default values that will be passed to the method unless the user specifies a different value for them.

The method header that can be seen at the top of user-defined methods in the editor area shows the name and data type of parameters that are needed for that method. Parameters can be created for a method by clicking the “create new parameter” button while editing the method.
Chapter 2 End of Chapter Exercises

Exercise 2-1 A.
Sample answers for items 1A and 1B are included below. Similar Answers to 1C, 1D, and 1E vary widely.


Carrot Cake

Dry Ingredients (Combine and set aside):
- 1 1/3 cups flour
- 1/2 tsp. salt
- 1 1/3 tsp. baking powder
- 1 1/3 tsp. baking soda
- 1 1/3 tsp. cinnamon
- 1/2 tsp. cloves
- 1/2 tsp. ginger

Combine:
- 1 cup sugar
- 1 cup cooking oil
- 3 eggs (added separately)

Preheat oven to 300 degrees Fahrenheit.

Add the dry ingredients to the wet mixture and stir well.
Fold in two cups of grated carrots and 1 cup of chopped walnuts (optional).
Pour into 9x13" non-stick pan and bake for 50-60 minutes or until done. (Make sure it is cooked)

Icing:
- 8 oz. package of cream cheese
- 1/2 cup butter (or less)
- 1 1/2 cup of icing sugar (I just add icing sugar 'till the mixture tastes right)
Exercise 2-1 B.

First, students need to find the directions; then they can organize them and diagram the parts. Here is a typical answer. I picked Troy, NY as the starting location. Most major cities have flights to London. Regional airports have flights to the hub cities. There are several alternatives upon arriving in London. Students can look up maps or travel information quickly on the internet. A Yahoo search turned up this page: http://answers.yahoo.com/question/index?qid=20061023102226AAC8gR1

The St. James and Green Park Underground stops are a little closer to the Palace, and there are also bus and Taxi alternatives.

The important thing here is to see the journey as a collection of major steps that can be broken down into smaller steps.

You could also ask your students “How do you get to Carnegie Hall?” The same way that you become a good programmer … practice, practice, practice.
Exercise 2-2.
The Alice world in the file *Ex2-2.a2w* contains a solution for this exercise.

Exercise 2-3.
The Alice world in the file *Ex2-2 bunny hop.a2w* contains a solution for the first part of the exercise, creating a bunny hop method. The styte and duration parameters have been set in the bunny.hop method to make the hop look better, but it the method is correct without these parameters being changed.

The Alice world in the file *Ex2-2 generic hop.a2w* contains a solution for the second part of the exercise, and this method could be used to make any object jump. This is what is meant by a generinc method – onethat is a world level method, which works with any object.

Exercise 2-4.
The Alice world in the file *Ex2-4 Simon says.a2w* contains a solution for this exercise. This is a more difficult exercise than many of the others for this chapter because students will need to figure out a few things on their own.

A subpart of an object can be programmed by clicking the plus sign next to an object, then selecting the appropriate subpart. Methods for the subpart will now appear on the *Methods tab* in the *Details area*. Although this is shown in Chapter 1, it is not used in the tutorials in this chapter.

An instruction to change the value of a property, such as the *opacity* property, is created by dragging the property tile into the editor area.

The images below should be helpful in explaining this to students.
Exercise 2-5.
The name of the instance of the Chicken starts with a capital letter. For most Alice objects, such as the horse and cow, instance names start with lowercase letters and class names start with capital letters. The person who created the Chicken class of objects did it this way instead.

Exercise 2-6.
The name “world.my first method” should be “world.myFirstMethod” in camelCase. The penguin class tile is shown below, with the names of the penguin’s built-in user-created methods. In camelCase, “Wing_flap” should be “wingFlap”, “turn_head_right” should be turnHeadRight, and “turn_head_left” should be turnHeadLeft.

Also note that not all of the names of the sample files and solution files for this text follow the camel case naming convention.
Exercise 2-7.
A. The file *Ex2-7 lakeSkaterCode.HTML* contains the code for the world in a single document that is easier to examine. Although the code does exhibit some modular design, it could be better. Methods like `jump`, `spin`, and `circleAround` are separate modules, but they are not reused as `iceSkater.skate` is. The most obvious improvement would be to break `world.my first method` into separate modules, guided by the comments in the code itself. The chart below is based on this. There are many ways to modularize this program, but this shows one way to start. The module names, except for “First movement”, are based on the comments in the code.

**Exercise 2.8**
There are many different worlds that could be based on the 100 film quotes. The important things are for students to: 1. keep it simple, and 2. use good modular design and 3. plan the world before implementation.

**Exercise 2.9**
The file *Ex2-9 JDE article.doc* contains an article I wrote for the Journal of Developmental Education that discusses Alkhwarizmi’s importance.

**Exercise 2.10**
Even more than with Exercise 2.8, above, this exercise could result in many different worlds, no two alike. The important things are for students to: 1. keep it simple, and 2. use good modular design, and 3. plan the world before implementation. This is a longer exercise that might be suitable for groups of students to complete together. Doing so will encourage them to articulate their ideas as they plan the world.
Alice 2.0: A Computer Programming Project that Promises to Impact Developmental Education

By Charles Herbert

Alice is an exciting new software tool that enables users with little or no computer experience to create three-dimensional, animated virtual worlds, like a modern animated film or video game. It is free software, developed at the University of Virginia and Carnegie Mellon University. In August of 2003 the National Science Foundation awarded a grant to a team of faculty members from several colleges and universities to explore the effectiveness of Alice in teaching computer programming in the community college. Their work promises to have a noticeable impact on developmental education. This article includes a brief introduction to the notion of algorithms, object-oriented programming, and the Alice software, along with a description of the NSF-funded project and a few conclusions. It is intended to be a qualitative introduction to the topic rather than a quantitative analysis of the effectiveness of Alice.

Algorithms, Objects and Alice

Simply put, an algorithm is a step-by-step process. Sometime in the ninth century of the Christian Era, a scholar named Muhammad ibn Mūsā al-Khwārizmī (Arabic: ﻭد ﺪﻤﺤﻣ ﻲﻣزراﻮﺨﻟا ﻲﺳﻮم) came to live and work in the City of Baghdad. Al-Khwārizmī means from Khwarizm (Arabic: ﻲﻣزراﻮځ, Russian: Хорезм, Uzbek: Xorazm), in what is now Uzbekistan, which was his ancestral homeland. He was one of many scholars brought to the Baghdad by the Caliph to form what became one of the world’s first modern universities. Al-Khwarizmi wrote several important books on Mathematics, Geography, and Astronomy, but perhaps his most important work was his second book, *al-Kitab al-mukhtasar fi hisab al-jabr wa'l-muqabala* (اًتْبَاَسْمَاً ﺭَوْصَخْتُمْ ﺑِاًتَكاً ﺑِاًرِجَاءً ﻗِبَاسِمُه), in which he presented a method for solving mathematical problems by balancing equations that came to be known as *algebra*. (Al-jabr = algebra.) This was only one part of his systematic approach to solving math and science problems, known, in various languages as the method of al-Khwarizmi, and from which we get the Latinized term *algorithm*.
According to the French writer and historian Andre Allard, almost all Western ideas about mathematics, and hence almost all of modern science, technology and engineering, were derived in part from seven critical texts written during the 12th Century and widely circulated through Europe in the centuries that followed. All seven of those books quote extensively from al-Khwarizmi’s second book.

A computer program is a step-by-step set of instructions telling a computer how to perform a specific task. As such, every computer program is an algorithm. Students in a good computer programming course are consciously learning about the logical structure of algorithms and techniques for organizing algorithms that can be traced back to al-Khwarizmi more than a thousand years ago.

In order to help manage the growing complexity of computers and their algorithms, computer scientists have introduced the notion of objects and object-oriented programming. An object is anything that can be represented by data in the computer’s memory and manipulated by computer programs. It is possible for modern computers to manipulate many objects at the same time. An object can be something in the physical world or even just an abstract idea. An airplane, for example, is a physical object that can be manipulated by a computer. A student’s grade point average is an example of an object that is not a physical object.

The data that represents the object are organized into a set of properties. Each property describes the object in some way. The programs that manipulate the properties of an object are known as the object’s methods. We can think of an object as a collection of properties and the methods that are used to manipulate those properties. This modern approach to computer programming is known as object-oriented programming, or OOP for short.

Alice is an object-oriented system of programming. The objects in Alice exist in a three-dimensional virtual world, much like a modern video game. In some ways, Alice is just like other modern object-oriented programming systems that use languages such as Java, C++, or Visual Basic, and in some ways it is different. The Alice language has a grammar and syntax like other programming languages, but it is constructed so that create methods by assembling tiles containing instructions and references to objects. This allows students to concentrate on
learning about the ideas of computer programming and algorithm development without having to worry about the details of a strange new language as they do so.

There is some evidence to show that one-third or more of all students who take an introductory computer course fail because of difficulties with programming languages. Alice helps to bypass this problem. The available evidence also indicates that once students learn about the ideas of programming and algorithm development, then language becomes much less of a hurdle.

The virtual world of Alice is one that students can see. Like the real world, it has three-dimensional space (and time), and the objects have properties just like physical objects, such as color, size, position, what direction an object is facing, and so on. Alice has a “camera” that allows us to see its virtual world on a computer screen, just as we might view a movie or a video game. In a sense, the abstract ideas that students are manipulating through their programs are made more concrete in Alice because they can see the effects of their programs just as we see the world around us.

This ability to see what happens to objects in a virtual world also provides students with rapid feedback. If we try to program a white rabbit to run around in a circle in our virtual world, and instead it simply stays in one spot and spins around in a circle, we can see that happening on the screen. We can get instant feedback from viewing the way Alice runs the programs we have created. Not every programming system is so easy to use. Often it is necessary to go through a process known as “compiling” before we can run a program.

So, there are three that make it easier to learn programming with Alice than most other systems of programming. First, Alice is constructed in such a way that students do not need to learn the grammar and syntax of a strange new language and can instead focus on the concepts of programming and algorithm development; second, Alice provides visual feedback that allows students to see the effects of new programs; and third, Alice, provides rapid feedback shortening the creative cycle of conceptualization, implementation, and results.

Alice is also fun and interesting to use, which never hurts when one is trying to learn something new. In fact, it may be critical to effective and efficient learning. The educational psychologist Jerome Brunner has shown that complex bodies of knowledge, such as the ability to develop
algorithms to solve technical problems, are best taught if they are introduced gradually, in a playful way, leading into a long and engaging spiral of learning\textsuperscript{7}. The source of the name “Alice” is itself somewhat related to this concept. The name Alice comes from Lewis Carroll’s two Alice books about Alice in Wonderland\textsuperscript{8}. It was chosen because the creators of the software wanted a name that was easy to remember, early in the alphabet, and conveyed a sense of the fun and playfulness associated with virtual worlds. The second page of the original manuscript for Alice’s Adventures in Wonderland (now in the British Museum), describes how Alice began her adventures: “Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it . . .”

Howard Gardner, in describing Jerome Brunner for the anthology *Fifty Modern Thinkers on Education*\textsuperscript{9}, wrote: “Jerome Bruner is not merely one of the foremost educational thinkers of the era; he is also an inspired learner and teacher. His infectious curiosity inspires all who are not completely jaded.” This sense of playfulness and curiosity, whether it is the “infectious curiosity” of Brunner or the “burning curiosity” of Alice chasing the White Rabbit, seems to be an important part of learning and effective teaching that can easily be neglected in technical courses. The Alice software can bring us back to that in computer science education.

Users of Alice do not create programs; they build virtual worlds by populating the world with objects and creating methods that manipulate those objects. Some of the virtual worlds are narrative worlds, in which a story unfolds on the screen with little or no user interaction. Others are interactive worlds in which the user controls objects and events, more like a modern video game. Hundreds of built-in objects are contained in a gallery that is part of the software, while many more are available in Carnegie Mellon’s free online gallery, including animals, people, buildings, vehicles, plants, and environmental elements, such as mountains, a frozen lake, and a desert island.
The NSF-ATE Alice Project

The NSF-ATE funded Alice project started on September 1st, 2003 and was originally scheduled to continue through August 31st, 2006. It has since been extended for an additional year. There were five goals for the project:

1. to improve computing FITness, as described by the National Research Council in their 1999 report through the introduction of a short programming module in general computer literacy and application courses.
2. to decrease attrition in introductory programming courses.
3. to improve student achievement in introductory programming courses.
4. to increase the attractiveness of computer courses and computing-related majors to the general student population.
5. to train faculty in the use of innovative instructional technology.

A team of faculty members from the Community College of Philadelphia in Pennsylvania, Camden County College in New Jersey, and Tompkins Cortland County College in New York are working under the direction of Stephen Cooper from St. Joseph’s University, Wanda Dann from Ithaca College and Randy Pausch from Carnegie Mellon University, who had developed curricular materials to teach introductory programming using Alice. Their materials have since been published as the textbook *Learning to Program with Alice*. Two other textbooks are also based on the work of the project: *An Introduction to Programming with Alice* by Charles Herbert, published in March, 2006, and a third book *Alice 2.0: a Brief Introduction* by Gary Shelly, Tom Cashman and Charles Herbert, due to be released this Fall.

The three community colleges participating in the project were selected because they represent the variety of community colleges in the United States: Community College of Philadelphia with roughly 40,000 credit and noncredit students in an urban environment; Camden County College with a little over 12,000 students in a largely suburban environment in Southern New Jersey, and Tompkins Cortland Community College, serving 3,000 students in a rural area of central New York State.

Activities included the development of additional curricular materials, training faculty at the community colleges in the use of Alice, introducing Alice into the curriculum, and measuring its
effectiveness. At Community College of Philadelphia, for example, faculty have been using Alice in introductory programming courses and an Alice module has been tested for computer literacy and applications courses. A new course, *Computer Information Systems 102 – Programming and Problem Solving with Alice*, which will target students who place below the college level in math and English, is under development. Bill Taylor at Camden County College has already developed a similar course, which he and his colleagues have been teaching for the past four semesters. Sharon Wavle at Tompkins Cortland Community College is teaching online sections of introductory computer courses using Alice.

Wanda Dann and Steve Cooper have been strong advocates of the storytelling approach to Alice; their textbook focuses on this approach to learning computer programming. Their students implement short stories or narratives using Alice, although their text, the seminal work in the field, includes the creation of games as well. Caitlin Kelleher is another researcher working with the storytelling aspect of Alice. She created the tutorials included with the Alice software, and is just finishing her doctoral work in Alice at Carnegie Mellon University, where she has spent the last several years researching the impact Alice can have on adolescent girls. Her work is based on the thesis that female students tend to make career decisions much earlier than male students, and that the use of Alice can attract more women to computer programming and related disciplines. Her work also shows that girls are more interested in the story-telling aspect of Alice than in creating simulations or video games, which boys seem to prefer.

Chuck Herbert’s textbook focuses on an engineering approach to Alice, in which students are taught to design a solution to a problem as a collection of modules that solve smaller, more manageable parts of the overall problem. They can then implement and test their modules as individual methods for Alice objects, and assemble the parts they have created into software that solves the original complex problem. Piaget has shown that student should be able to learn problem solving techniques by the time they reach adolescence, from 13 to 18 years of age, but many secondary and post-secondary curricula do include such instruction. The use of Alice may be a good way to change that.

The story telling and engineering approaches to Alice are not mutually exclusive; both introduce students to concepts of problem solving and programming in a casual, almost playful way,
gradually adding more sophisticated concepts as the semester progresses, much in line with Bruner’s ideas about the spiral curriculum.

Both approaches also foster the kind of socially-based learning in a structured environment suggested by Lev Vygotski and others. Vygotski\(^{16}\) maintained that “Psychological development and learning are socially based.” He argued that many complex skills cannot be acquired by individuals alone, but through interaction with teachers and peers slightly ahead of them on the learning curve. He advocated the study of social relations as a way to better understand learning and thus become better educators. Proponents of activity theory have developed his ideas further, often focusing on the way in which the tools one uses in various roles in society shape both the society and the individual\(^{17}\). University of Pennsylvania anthropologist Fred Erickson\(^{18}\) has advocated that teachers become more aware of the social processes within the classroom, and use them to foster more collaborative learning, such as through the use of group projects. Alice is a tool that educators can use to improve instruction in programming and problem solving by invigorating the social environment of the classroom under the teacher’s control.

So far, research on the use of Alice shows that it attracts new students to what the National Science Foundation calls the STEM disciplines – Science, Technology, Engineering and Mathematics, particularly female and minority students; increases student success in the course and in follow-up programming courses; and increases student retention in computer-related curricula. Perhaps most surprisingly, the data shows that students who take a single course using Alice are more likely to stay in school whatever their major, and that they achieve a higher grade point average than students who do not take a course in Alice\(^ {19}\).

Bill Taylor’s work at Camden County College is the first specifically targeted toward developmental students. His preliminary reports to the NSF, cited in Moskal, et. al.\(^ {20}\), are encouraging, as is anecdotal evidence from students and teachers at Camden, but more work is needed to specifically evaluate the lasting effect of Alice on developmental students.

**Conclusions**
Algorithms are important – very important. In fact, they are the foundation of Western science and technology. An awareness of algorithms, their development, and their use in problem solving might help students to be better thinkers, or at least, to be more organized and persistent in the way they think about solving problems.

Brunner\textsuperscript{21} has shown that complex technical ideas, such as algorithm development and problem solving, are best taught if they are introduced gradually, almost playfully, leading into a long and engaging spiral of learning. Vygotsky\textsuperscript{22} and others claim that these skills do in fact need to be taught, and that navigating the “Zone of Proximal Development” is a social process. Piaget\textsuperscript{23} says that our students are ready for learning about such things in adolescence, yet most college and university faculty will tell you that many of their students are not so prepared. This is especially true of our developmental students.

Alice is a new and exciting tool that can attract students to courses about programming and algorithm development, and then provide them with the opportunity to persistently and successfully solve problems of increasing complexity while studying the process of problem solving itself. It can also be used to foster a positive social environment in the classroom. More work needs to be done, but preliminary investigations into the use of Alice with developmental students shows that it seems to have a positive impact on a significant percentage of those students.

A long-standing question in education, in developmental education in particular, centers on whether or not we can teach someone how to think. There are two metaphors for teachers that have been used in describing how computer technology might affect education. Some people posited that a teacher could be “a guide on the side” rather than “a sage on the stage.” Another way to think of a teacher is as a gardener who cultivates the thinking ability of each student. To do so, the teacher must also cultivate a positive learning environment for the student. The successful teacher cultivates a shared curiosity essential to effective education. Alice can be a very powerful tool toward that end.
Notes

1 For more information on Alice, research on the use of Alice, and to download the software, see www.alice.org

2 National Science Foundation Grant # NSF-ATE ATE 0302542, available online via www.nsf.org

3 For more about Al-Khwārizmī and his work see::
   http://www-groups.dcs.st-andrews.ac.uk/history/Mathematicians/Al-Khwarizmi.html


5 Lemut, E, et. al. (1993) Cognitive Models and Intelligent Environments for Learning Programming, Genova, Italy: Springer


8 Alice’s Adventures in Wonderland and Through the Looking Glass and What Alice Found There


14 Caitlin Kelleher’s dissertation may be accessed via www.alice.org upon publication


19 Moskal, et. al. op. cit.

20 described in Moskal, et. al.

21 Bruner op. cit.

22 Vygotski, op. cit.

23 Piaget,