AC 2008-2323: EXTENDED ACTIVE LEARNING AS A MEANS TO LEARN SYNTAX IN PROGRAMMING LANGUAGES

Steven Hansen, University of St. Thomas

Extended Active Learning As A Means To Learn Syntax in Programming Languages

Abstract

Active learning is an education form that has gained much interest in recent years. Many instructors can come up with schemes that help students better learn algorithm development, program development, project management, and other aspects of needed skills in the computer sciences. In the past decade, I worked on the development of active learning techniques to learn syntax. I find that these techniques allow the students to gain better understanding of both computer language syntax and useful processes to enhance learning. To assist the students in active learning, I write labs that ask them to enter and run a program, and then ask the students (in the lab) questions about the syntax, gently pushing them to think about why various parts of the syntax are there, and what those parts do. In this paper I demonstrate the types of questions I ask, focusing on the degree of difficulty, the intent of each question, and the overall goal in affecting the thinking process the students use to approach the task of writing solutions to problems using programming languages.

Body of the Paper

Active learning is a pedagogical technique that has gained favor in recent years. Bonwell and Eison[1] suggest that to be considered active, learning must involve more than listening on the part of the students, it must also involve reading, writing, discussion, and/or solving problems. Chickering and Gamson[2] extend this definition to state that students "must talk about what they are learning, write about it, relate it to past experiences and apply it to their daily lives. They must make what they learn part of themselves". McConnell[3] applied the concept to computer science realm with his noting that when we prepare to teach, we read, compare what we read to our experiences, synthesize the information into coherent notes, and develop examples to illustrate the concept. He then notes that in doing this we are depriving our students of the very methods we use ourselves to gain an understanding of the material. McConnell further suggest that one can use techniques such as posing questions like:

- What will happen if I change this input?
- What could happen if I don't include this conditional check?
- Why is it important to reduce the number of operations in sorting algorithms?
- Why are all four conditions necessary for deadlock to occur?

Early in my graduate education I had the opportunity to experience the effectiveness of active learning in the collegiate environment. As a graduate teaching assistant in the field of biology, I helped students learn the introductory biology course through the use of stations where they used experimental methods to help them gain first hand the knowledge to which they had been introduced in large lecture classes. Included with the presentation of the information were lab manuals that asked the students questions. Materials to run small experiments were available at lab stations, and the students could

run small experiments to arrive at the information to answer the questions provided in the lab manual. This method worked well and was used for a number of years.

The learning environment for computer science majors at my university

With my arrival at the University of St. Thomas, I had the opportunity to teach programming in several different classes with students who were taking the classes as majors or as classes supporting a major in business or liberal arts. The variety of students allowed me to test techniques to aid them as they worked to develop an understanding of programming.

The majors course at St. Thomas has long been divided into two segments, one consisting of two lecture periods, the other providing two longer time periods in a lab format in which the students would apply the knowledge they had gained in lecture. These labs often were used to provide the students with time to work on programming assignments in an environment in which the instructor was available to answer questions and provide explanations.

Utilization of the active learning technique to computer programming

In addition to some lecture, I began to write a series of labs that would provide reading pages in the book, and would then present code in the language being used in the course. Along with the code, I would provide explanations of the syntax and associated semantics. The students would then be requested to enter the code and execute it, with the goal of observing the output with suggested input (if appropriate). The lab then requested the student to make some changes to the code while predicting what effect the changes would be. This method of learning the meaning of the syntax was usually accomplished by having the students work in pairs to allow sharing of ideas. In addition to asking the students to make set changes to the code, they were asked to provide answers to probing questions and to suggest changes to the code to produce requested results. These labs that asked the students to think about the syntax and semantics helped the students to solidify an understanding of the material.

Helping the students gain an understanding of code structures

The methods suggested above worked well when asking students to learn the expected semantics of statements along with the correct syntax. When the need for learning arises above individual statements however, an added level of complexity is encountered. Authors often handle the presentation of these structures by showing code segments along with discussion of the code segment in the text. While this method has some level of success in helping the student understand the semantics of the code, the complexity of structures with multiple statements can be daunting to some students. To deal with this problem of complexity, I suggest using a version of a system developed for the identification of birds in the first part of the previous century.

The Peterson Identification System

An area of learning that also has to deal with problems of complexity is the task of bird identification. In the latter part of the 19th century and first part of the 19th century, an increase of efficiency in the production of goods in the United States provided many people with leisure time. One activity that resulted from this leisure time was the sport of bird watching. The complexity of the number of plumages of birds due to age, sex, and breeding plumages made identification a very difficult task for some species.

In the 1930s, a method of identification was developed by Roger Tory Peterson[4] that broke down bird identification to a listing of identifiable patterns, and he created the concept of field guides containing paintings of each species with a listing of field identification marks to look for when identifying the species. This method has been widely copied, and is now used for many organisms and other natural entities in addition to birds. An example of the use of this system for program code is provided in Figure 1.

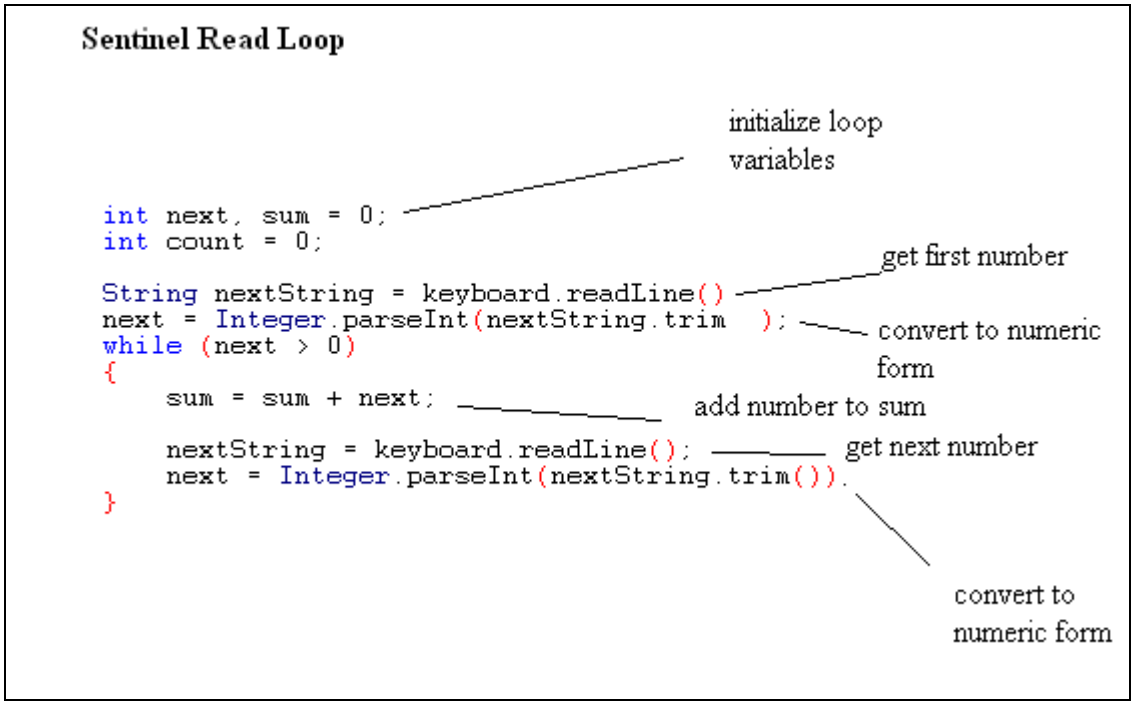

Figure 1. Example use of Peterson Identification System for program code

Applying the Peterson Identification System to program code

Although this concept has its uses when trying to identify sections of program code as separate entities, another method of birding that is similar to what I use is the identification of code by gestalt, which is a method often used by expert birders who have seen birds many times, allowing them "to just know" a bird when they see it.

There are many corollaries between birding and process of learning programming. In birding, people find a fascination with the ability to name (in other words to recognize)

an aspect of their environment. Gestalt identification is almost always the means used. A person sees a relatively small red bird, and "knows" it is a Cardinal. The name Northern Cardinal comes later, as does the ability to recognize a female cardinal. Similarly, the understanding of a loop as a structure in programming comes with the observation of the products of a loop, and thus a loop that creates output is the type of loop that is easiest to understand. The numbers that appear as output on a page of paper or a web page have some connection to the structure of the loop. This need to make the connection is why instructors invariably request that the statements "inside" the loop be indented, with the intention of aiding the student (and later the reader of the program) in identifying those statements with the looping structure.

The process I use to help the student develop an understanding of a loop is this:

1. Give a program that contains only the loop code (which makes recognition of the loop as a structure a default)
2. Have the students enter the code into a compiler or interpreter
3. Fix any syntax errors (this usually involves the instructor or aide walking around and answering questions).
4. Have the students run the code and observe the output
5. Ask questions of the student (or students) that attempt to draw a connection between the output and the loop code. This process is usually done via a written lab that requires answers concerning the output and its connection to the code.
6. Based on the understanding of the code-output connection, have the student(s) suggest and make a change to the code with a prediction of how it will change the output.
7. Have the student re-run the code to determine whether their predicted change in the output is observed.
8. Repeat steps 6 and 7 for other aspects of the code to aid the student in developing a model of the structure and behavior of a loop within the model of computer programming that they are developing in their mind.

Once the students have developed a gestalt understanding of the loop structure, it is possible to move on an explanation of the loop structure using identification techniques developed by Peterson.

The result of the techniques outlined above is that the students are asked to start their understanding of the material several levels up in taxonomy suggested by Bloom and the rest of the cohort from which the taxonomy of learning was suggested in their 1956 publication. Immediately, the student is asked to act at the level of applying (putting the code into the compiler, and causing it to execute) and analyzing (looking at the output and the code, and working to make a correlation, and to act upon that that correlation to suggest a change to the code, and to make that change and to observe whether the predicted outcome based on the change does indeed result.

In addition, these techniques also have results in the affective domain, also suggested in the 1956 publication by the committee of which Bloom was a part. The affective domain

includes emotional responses, and these responses are ones that have been of concern by the computer science education community as a whole since the turn of the century, when the interest of students in the computer science field took a nose dive.

By demonstrating confidence in the ability of the students to make associations between the code and the output (in other words the ability of the student to understand the meaning of the program segment), one demonstrates a belief in the integrity of the student's intelligence and capability to work with the program. The effectiveness of this method of working with the students has been demonstrated by a consistent rating in semester evaluation with a high rating of me for "demonstrating respect for the student".

Additional means of helping the student take ownership of the material

In my programming classes after the entry level, I also ask students to try testing questions that don't directly apply to the use of syntax in computer programs, but are rather larger questions that arise from the manner in which syntax could be used. These questions often open up an excitement in the learning process as the students realize that there are many questions begging to be asked, and that while the answers may not be directly applicable to programming, they do give one a better understanding of the language being used.

Several examples of questions of this type include:

In the Java programming language, is there a limit to the length of variable size?

> When I helped a student realize that this was even a question, the student built programs with longer and longer variables until he had confirmed that a variable that was 2,000,000 characters long was still legal.

What is the shortest legal Java program?

- The first sub-question generated here is what is the definition of a legal program?
- Does String args[] have to be included in the main method?
- Can a Java program have no methods?
- With time, students can come up with additional questions that need to be be answered before the main question can be answered with reliability.

What happens if one creates a for loop with only semicolons in the parentheses?

> Ex: for (; ;)
> This creates an infinite loop. Why?

The above examples have the effect of generating more student interest as the students realize that there are many more such questions waiting to be asked.

Assessment

While a formal assessment of these learning methods has not yet been done, the courses at our university are assessed by student response forms supplied to the students in the last two weeks of the semester. Ratings are done on a scale of 1 to 5, with 5 representing the most positive rating. Students are asked to rate the instructor as well as the course. Rankings in introductory Java courses tended to be in the 4s and the 5s for instructor. Most students indicated that most of the material in the course was new material.

Summary

In this paper I review the usefulness of active learning as a method to help students learn syntax in programming languages. Of particular importance is the incorporation of methods that graphically point out the key identifiers for programming structures in the code such as *read loops* or decision statements. The students develop a gestalt ability to identify code structures, and to pull an understanding of these structures out of the background of seemingly endless and similar appearing statements.
Finally, students learn to use the scientific method to increase their understanding of the semantics represented by the syntax. This allows them to develop an ownership of the learning process. Thus, they develop an excitement about the learning process by applying the scientific method by using programming statements in unexpected ways.

Bibliography

[1] Bonwell, C. C. & Eison, J. (1991). “Active learning: Creating excitement in the classroom.” Washington D.C.: ASHE-ERIC Higher Education Report No. 1
[2] Chickering, Arthur W. and Gamson, Z. (1987). "Seven principles of good practice in Undergraduate Education." AAHE Bulletin, 39, pp. 3-7.
[3] McConnel, Jeffrey J. Active Learning and its use in Computer Science. ITiCSE '96: Proceedings of the 1st conference on Integrating technology into computer science education. June 1996.
[4] Peterson, Roger Tory (1934) “A Field Guide to the Birds