Visualizing Recursion Using Code and Stack Animation*

Y. Daniel Liang, Jireh Bethely, Gursimran Singh Walia
Department of Computer Science
Georgia Southern University
Statesboro, GA 30458
{yliang,jb24628,gwalia}@georgiasouthern.edu

Abstract

Recursion is an important programming technique, but it is difficult to learn. Students often struggle to understand how recursion works and why a tail recursion is efficient. We created a code and stack animation to demonstrate how a recursive method is executed and another code and stack animation to demonstrate how a tail recursive method is executed. Our animations show the call stacks and illustrate interactively how the call stacks grow and shrink during the execution of a recursive method and how an activation frame is shared for a tail recursive method. These code animations enable students to trace the code step-by-step visually and help students understand the recursion and tail recursion. This paper presents these two code animations.

1 Introduction

Code animation is a visual tool for tracing the execution of the code. We have developed more than 240 code animations for Java, C++, and Python. The code animation have been integrated in the Pearson's Revel interactive ebooks [8, 5, 6], which have received positive reviews [1, 2]. In a study [1] conducted at Central Michigan University in 2015, *"62 percent of students agreed or strongly agreed that the animations in Revel that stepped line by line through code, showing what was happening in the program, helped them better understand how to replicate the coding on their own."* In another study [2] conducted

at the University of Louisiana in 2016, *"87 percent of students who used the animations strongly agreed or agreed that they helped them better understand how to replicate coding on their own."* The second study was conducted after we added more code animations in the book.

Code animation is effective to help instructors and students to teach and learn programming. In [7], we demonstrated how the code animation can help students to learn variables, selection statements, loops, methods, pass by value, and arrays effectively. Code animation helps students comprehend the code. More importantly, it illustrates important programming concepts using animation. Frequently, we have been asked by instructors to develop code animation for recursion. We listened, responded, and created two code animations for recursion: one for non-tail recursion and the other for tail recursion.

Recursion is an important programming technique. It is ideal for solving inherently recursive problems that would be otherwise difficult to solve without using recursion. Recursion is difficult to teach because students have no idea how recursion is executed behind the scenes. Many researches have been conducted on teaching recursion [3, 11, 10]. The various approaches for teaching recursion are summarized by Dann, Cooper, and Pausch [3]. Three main approaches are algorithm animation, program visualization, and using special software tools. An example of algorithm animation was the GAIGS (Generalized Algorithm Illustration through Graphical Software) package developed by Naps [9]. Using this system, students can run some prepared animations. However, this system is not Web-based and the animation does not show the call stacks. The program visualization is similar to our code animation that directly shows the execution of the code in the program. However, it does not show the change of the stack as the method is called or returned. The special software tools for teaching recursion use a software such as Alice. These special software tools do not show the call stacks. All the existing approaches are not Web-based. Our code animation is developed using HTML, CSS, and JavaScript. It is platform independent and can be viewed on any device.

It is worth to mention that Guo [4] has created code animation for several programming languages including Java, Python, and C++. However, his tool does not read input for Java and C++, does not show the explanation for each statement being executed, and does not show how a tail recursive method is called. So, we created the code animation to visualize recursion and tail recursion.

In the following sections, we will present a code animation to demonstrate how recursion works and another code animation to demonstrate how a tail recursion works. We will then present a class test report. The report shows that these animations help students better understand recursion and tail recursion.

2 Recursion Animation

To show how recursion works, we created an animation for computing factorial using a recursive method. The factorial problem is not a good candidate for recursion, because a non-recursive solution for this problem is more efficient than using recursion. Nevertheless, it is a simple and intuitive problem for introducing the concept of recursion and for demonstrating the techniques of writing a recursive method. Because of this reason, many textbooks use the factorial problem for introducing students to recursion. You can access the code animation for this recursive program from `https://liveexample.pearsoncmg.com/codeanimation/ComputeFactorial.html`, as shown in Figure 1.

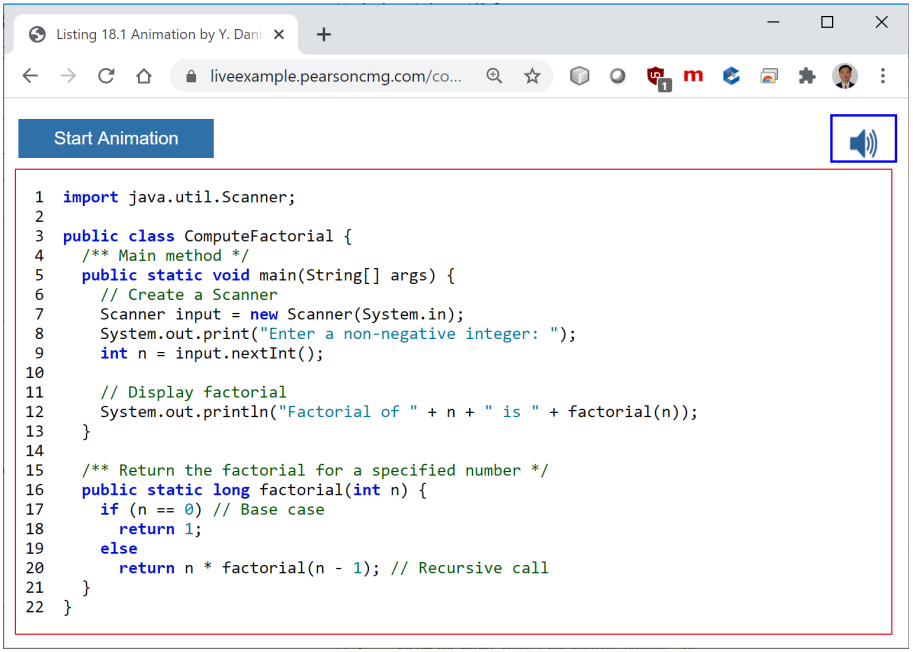

Figure 1: Code animation for recursion

The animation shows how a recursive method is executed with a stack step-by-step visually. When the program executes, the main method is invoked (line 5). The animation shows that an activation record is created and pushed to the stack for the main method. After declaring and reading n, the animation shows the value of n is stored in the activation record. When calling factorial(n) in line 12, the animation shows that a new activation record is created. To compute factorial(3), factorial(2) is recursively invoked in line 20. The animation shows that the activation record for factorial(2) is pushed to the stack. To compute

factorial(2), factorial(1) is recursively invoked in line 20. The animation shows that the activation record for factorial(1) is pushed to the stack. To compute factorial(1), factorial(0) is recursively invoked in line 20. The animation shows that the activation record for factorial(0) is pushed to the stack, as shown in Figure 2. When n is 0, factorial(0) returns 1 in line 18. The animation shows that the activation record for factorial(0) is removed from the stack. Now factorial(1) is on the top of the stack, it returns 1 * factorial(0). The activation record for factorial(1) is removed from the stack. Now the animation shows that factorial(2) is on the top of the stack, it returns 2* factorial(1). The activation record for factorial(2) is removed from the stack. Now the animation shows that factorial(3) is on the top of the stack, it returns 3* factorial(2). The activation record for factorial(3) is removed from the stack.

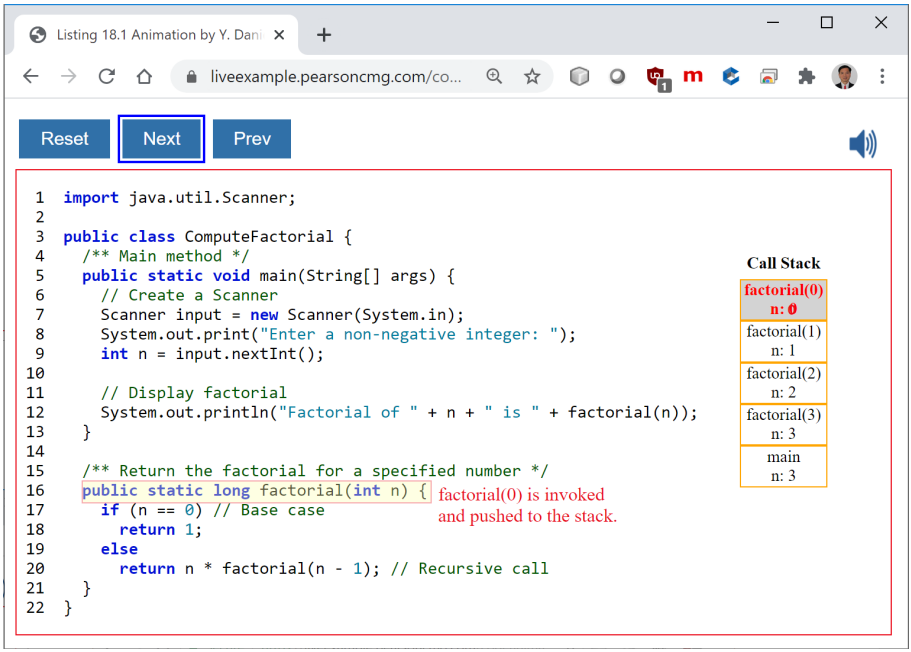

Figure 2: The activation record for factorial(0) is pushed to stack

Finally, when the main method exits, the animation shows that the call stack is empty. With the help of this animation, students can see how a recursive method is called and tracked using activation records in the call stack.

3 Tail Recursion Animation

To show how a tail recursion works, we created an animation for computing factorial using a tail recursive method. You can access the code animation for tail recursion from `https://liveexample.pearsoncmg.com/codeanimation/ComputeFactorialTailRecursion.html`, as shown in Figure 3. factorial(n, result) in lines 10-15 is a tail recursive method. The main method invokes factorial(n) in line 24. Suppose that n is entered as 3, the animation shows that factorial(n) invokes factorial(n, 1) in line 6. factorial(n, 1) is a tail recursive method. The animation shows that it invokes factorial(n – 1, n * result) in line 14. factorial(3, 1) invokes factorial(2, 3). factorial(2, 3) invokes factorial(1, 6). factorial(1, 6) invokes factorial(0, 6). Since factorial(n – 1, n * result) is a tail recursive, when a new factorial(2, 3) is invoked, the compiler does not create a new activation record. It will simply reuse the current activation record on the top of a stack for a tail recursive method. When the tail recursive method returns result in line 12, the animation shows that the activation record for the tail recursive method on the stack is removed, as shown in Figure 4.

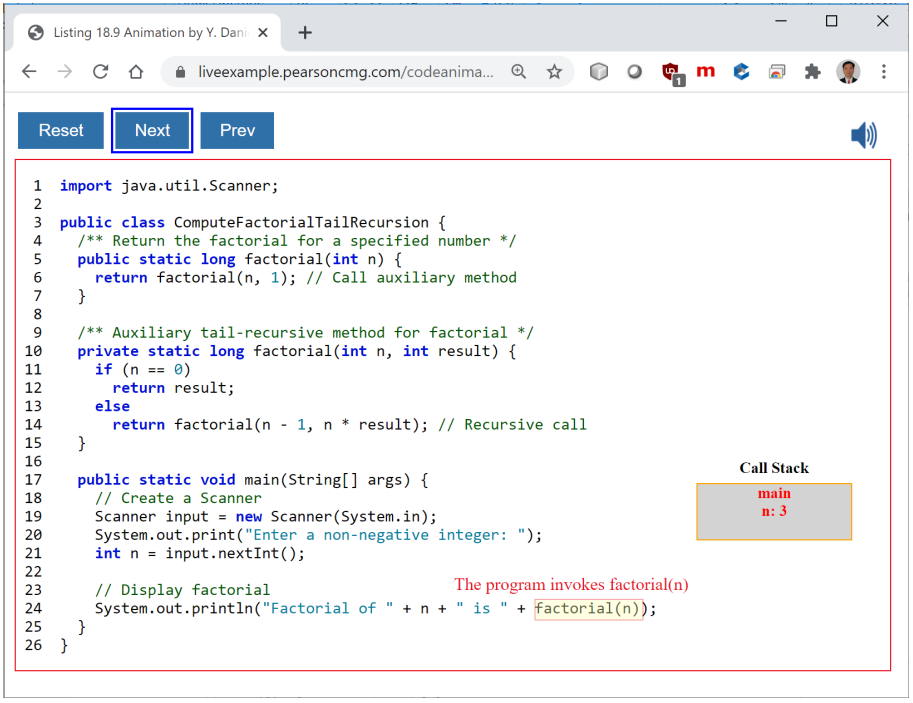

Figure 3: Code animation for tail recursion

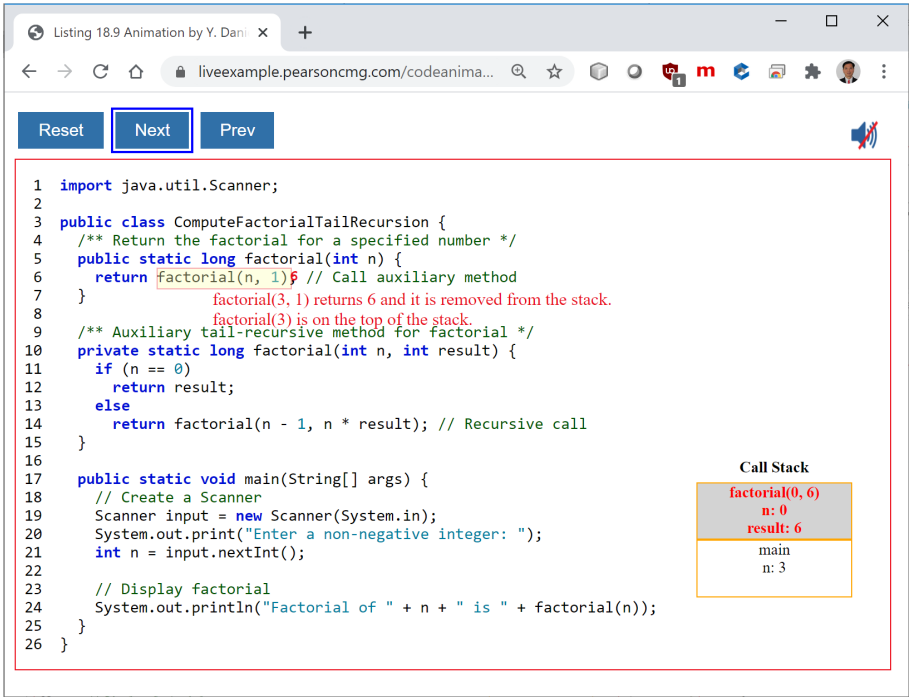

Figure 4: The activation record for the tail recursive method is removed when the method is returned

With the help of this animation, students can see why a tail recursive method is efficient in space and time, because the activation record is created only once for a tail recursive method.

4 Evaluation

The idea of creating code animations for recursion was suggested by an instructor who has used our code animation for non-recursive programs. We have created the code animation for recursion and tail recursion in the summer of 2018 for Java, C++, and Python. In the Fall 2018, we surveyed the students in a class of 9 students. We used a scale of 1 to 10 for answers, where 1 is poor and 10 is excellent. The result is as follows:

1. Does the ComputeFactorial.java animation help you understand how a recursive method is called using activation records? 8.66

2. Does the ComputeFactorialTailRecursion.java animation help you understand how a tail recursive method is called using activation records? 8.44
3. Do the animations for ComputeFactorialTailRecursion.java and ComputeFactorial.java help you understand why a tail recursive method is more efficient than a non-tail recursive method? 8.44

The survey shows that students agree strongly that the animations for recursion and tail recursion help them understand recursion and tail recursion.

We conducted further studies in the Spring 2021 in two classes. Right after we introduced recursion using ComputeFactorial and ComputeFactorialTailRecursion, we tested students in two classes with the following six questions:

> Q1: To compute factorial(3), how many times is the factorial method invoked? Count invoking factorial(3) as the first time. (1 point. Answer: 4 times).
> Q2; To compute factorial(3), how many times is the tail recursive factorial(n, result) method invoked? Count invoking factorial(3, 1) as the first time. (1 point. Answer: 4 times).
> Q3: What is the maximum number of the activation records when running ComputeFactorial with input 3? Count invoking the main method as 1 activation record. (1 point. Answer: 5).
> Q4: What is the maximum number of the activation records when running ComputeFactorial with input 4? Count invoking the main method as 1 activation record. (1 point. Answer: 6).
> Q5: What is the maximum number of the activation records when running ComputeFactorialTailRecursion with input 3? Count invoking the main method as 1 activation record. (1 point. Answer: 3).
> Q6: What is the maximum number of the activation records when running ComputeFactorialTailRecursion with input 4? Count invoking the main method as 1 activation record. (1 point. Answer: 3).

In Class 1, 17 students studied the materials using the code animation. In Class 2, 13 students studied the materials without using the code animation. Table 1 shows the result from the two classes. The table shows that the average on the preceding six questions is 0.696 for the class using code animation and 0.5 for the class without using code animation.

The test was given immediately after the lectures on recursion and tail recursion. So, most of the scores are low, because students were just introduced to recursion. The score for question Q5 is higher than the rest for Class 1, because we used the question as the example in the code animation for Class 1. Students have fresh memories of that question. Q1, Q3, and Q4 are on non-tail recursion in Table 1. Q2, Q5, and Q6 are on the tail recursion in bold in Table 1. Class 1 has a slightly better result than Class 2 on the non-tail

Table 1: Test Question Result from Two Classes (tail recursion in bold)

	Q1	**Q2**	Q3	Q4	**Q5**	**Q6**	Average
Class 1 (using animation)	0.765	**0.587**	0.647	0.647	**0.882**	**0.647**	0.696
Class 2 (without using animation)	0.538	**0.615**	0.692	0.615	**0.307**	**0.231**	0.5

recursion (Q1, Q3, and Q4) and has a much better result than Class 2 on tail recursion (Q5, Q6).

The test result shows that the performance for Class 1 is better than Class 2 on average. In particular, Class 1's performance is much better than Class 2's performance on tail recursion. The animation for recursion and tail recursion was used for Class 1. The animation helps students learn recursion and tail recursion.

5 Conclusions

This paper presented the code animations for recursion and tail recursion. These are the effective tools for helping students to learn how recursion and tail recursion work behind the scenes. Our future work is to establish a framework for creating code animation for recursive programs and create more code animation for recursive programs.

References

[1] Revel educator study assesses quiz, exam, and final course grades at central michigan university, Fall 2015. `http://www.pearsoned.com/results/revel-educator-study-assesses-quiz-exam-final-course-grades-central-michigan-university/`.

[2] Revel™ educator study observes homework and exam grades at university of louisiana, Spring 2016. `http://www.pearsoned.com/results/revel-educator-study-observes-homework-exam-grades-university-louisiana/`.

[3] Wanda Dann, Stephen Cooper, and Randy Pausch. Using visualization to teach novices recursion. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 109–112, 2001.

[4] Philip J Guo. Online Python tutor: embeddable web-based program visualization for CS education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 579–584, 2013.

[5] Y Daniel Liang. *REVEL™ for Introduction C++ Programming and Data Structures. 4e. ISBN-13: 978-0134669854.* Pearson Education, 2018.

[6] Y Daniel Liang. *REVEL™ for Introduction Python Programming and Data Structures. 2e. ISBN-13: 978-0135187753.* Pearson Education, 2018.

[7] Y Daniel Liang. Teaching and learning programming using code animation. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, pages 24–30, 2018.

[8] Y Daniel Liang. *REVEL™ for Introduction Java Programming and Data Structures. 12e. ISBN-13: 978-0135945476.* Pearson Education, 2020.

[9] Thomas L Naps and Brian Swander. An object-oriented approach to algorithm visualization—easy, extensible, and dynamic. In *Proceedings of the twenty-fifth SIGCSE symposium on Computer science education*, pages 46–50, 1994.

[10] E. Roberts. *Thinking Recursively with Java ISBN-13: 978-0471701460.* John Wiley & Sons, Inc., 2005.

[11] Cheng-Chih Wu, Nell B Dale, and Lowell J Bethel. Conceptual models and cognitive learning styles in teaching recursion. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 292–296, 1998.