Development of a System to Analyze Students' Keystroke Sequence in Programming Education

Tatsuyuki Takano, Kanto Gakuin University, Japan
Osamu Miyakawa, Tokyo Denki University, Japan
Takashi Kohama, Tokyo Denki University, Japan

The Asian Conference on Society, Education & Technology 2014
Official Conference Proceedings

Abstract
In programming education, the procedure used to input a program and its relation to degree of comprehension of the program are seldom studied. In this paper, we report on a study in which one input procedure was taught to a student and an algorithm that judges whether the procedure is inputted correctly was developed and installed in the system. Using the data inputted by the student the algorithm was able to correctly determine whether the procedure used to input the program was correct. Further, using the data acquired, an objective determination of whether a coding procedure is right or wrong can be made from a graph.

Keywords: programming education, software engineering, degree of comprehension

iafor
The International Academic Forum
www.iafor.org

Introduction

When training software engineers, programming introduction education is very important. In programming education, students submit source codes with a subject; however, the submitted source code is only the resulting input, information on how it was inputted is not included. The method used to write a program is important as it facilitates understanding of the program; for example, an indent is effective in enabling understanding of the contents of a program (Oman & Cook, 1988; Miara et al. 1983). A novice programmer may make many errors involving brackets or braces (Jackson et al. 2005). Such errors can be reduced by devising an input procedure in which corresponding parentheses containing a block of code are inputted prior to the block of code. We teach methods such as this to students in programming education. Further, to ascertain whether following this input procedure has an influence on compile errors or mounting capability, we created an algorithm that judges objectively whether the input procedure was followed.

This paper discusses the algorithm implemented in the system. The implemented algorithm compares the input procedure for the source code of the correct answer to that of the source code that the student inputs.

Keystroke Sequence

To aid in the explanation of typing procedure, let us look at the example source code, written in Java, displayed in Figure 1. The procedure is to input the characters used as pairs, such as curly braces and round brackets prior to other elements. Then, the inner elements are inputted. Consequently, the input procedure for the source code displayed in Figure 1 is as shown in stages 1 to 7 in Figure 2.

```
public class Student {
    private String name;
    public Student(String name){
    }
    public String getName(){
        return null;
    }
}
```

Figure 1: Sample Java source code

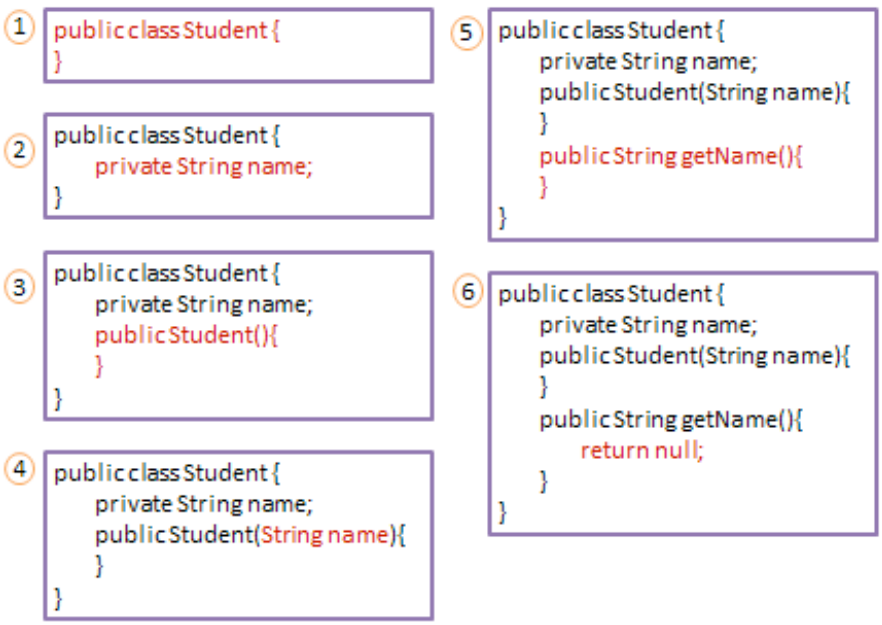

Figure 2: Stages in the coding procedure for the sample source code

Algorithms

The outline of our proposed algorithm is shown in Figure 3.

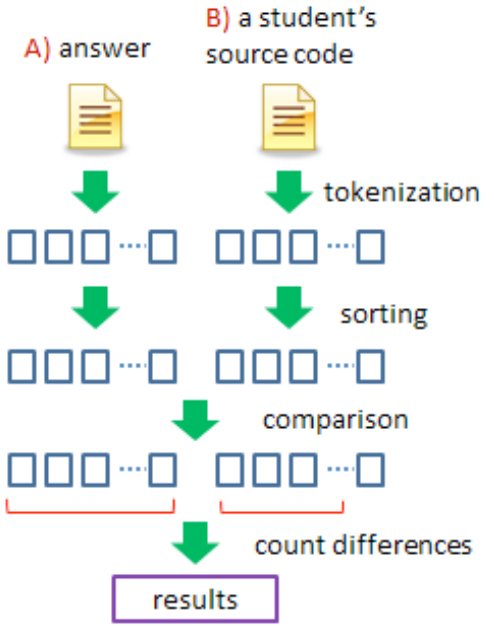

Figure 3: Outline of the proposed algorithm

Keystroke sequence data are created from the source code of an answer and the student's source code, and a result is obtained by comparing both of them. The keystroke sequence data creation method from the source code and input procedure in Figure 4 is illustrated in Figure 5.

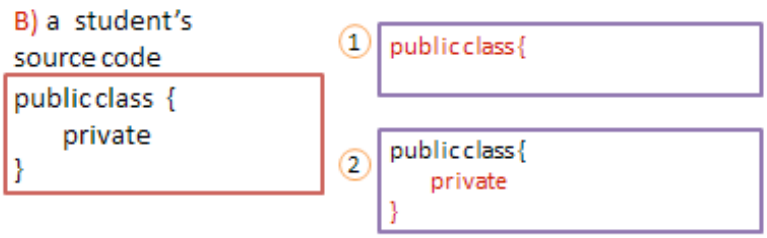

Figure 4: Sample source code inputted by student

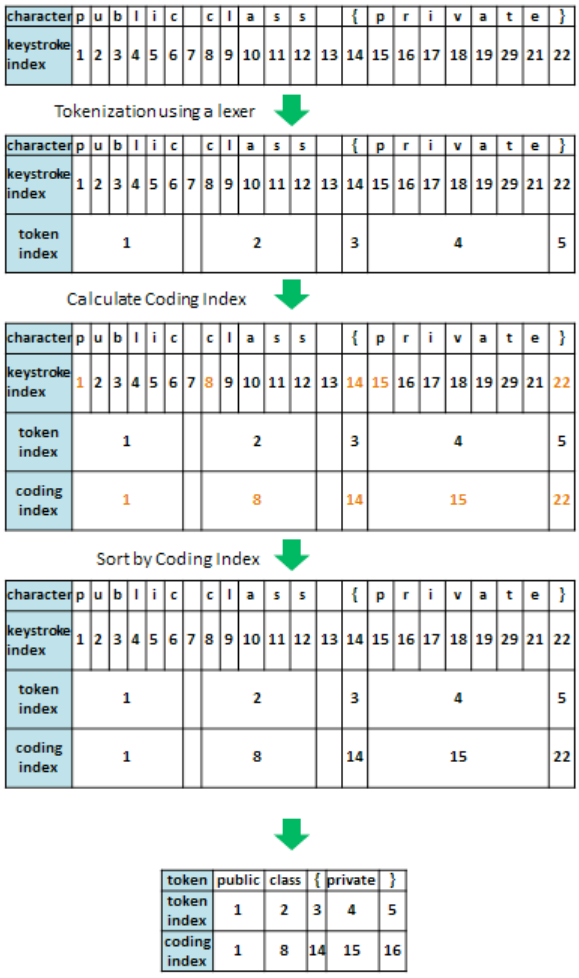

Figure 5: Keystroke sequence data of student's source code

Figure 5 shows how the keystroke sequence data is created from a student's source code. First, the student's source code is put in order as a character string, and a number given to each character by setting each entry sequenced to a keystroke index. The input is then sorted by keystroke index. The turn and keystroke index of a character are the same at this time. The lexical of a character string is then analyzed, the character string made a token sequence, and the minimum keystroke index in the token set up as the coding index. Next, an ascending sort is carried out according to the coding index, and the student's keystroke sequence data created.

We can explain how comparison of the keystroke sequence data and the student's keystroke sequence data is carried out using the answer source code displayed in

Figure 6. The lexical of the answer source code is analyzed and arranged in the token sequence created. The coding index looks for a corresponding parenthesis if a number is assigned in order and a parenthesis is found, and gives the following number when a parenthesis is found. Then, the coding index is calculated and it returns to the position of the original parenthesis and numbers. This is the index for the coding index, and an ascending sort is carried out. The algorithm also compares the length, when it is longer in that case than the student's data. The number of tokens compared that have a difference is considered as the result.

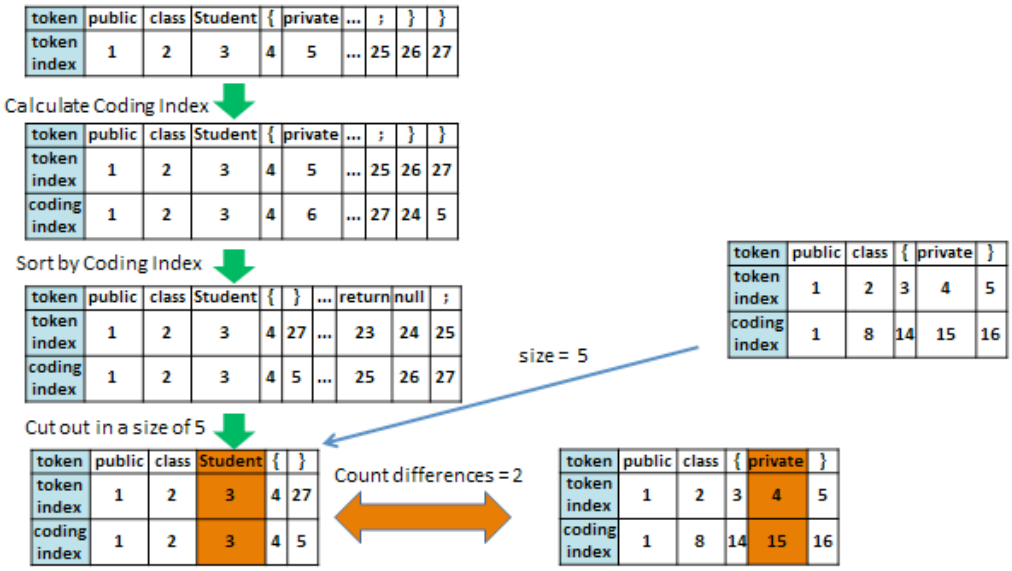

Figure 6: Keystroke sequence data for the answer source code

System Integration

This algorithm was added to a system previously developed by Tatsuyuki et al. (2011) for acquisition of students' input. The system is outlined in Figure 7. A program is created using the editor in the system, and the student creates a Java program in client server form. Further, the inputted data is transmitted to the server side as XML data. The server was mainly developed using Grails.

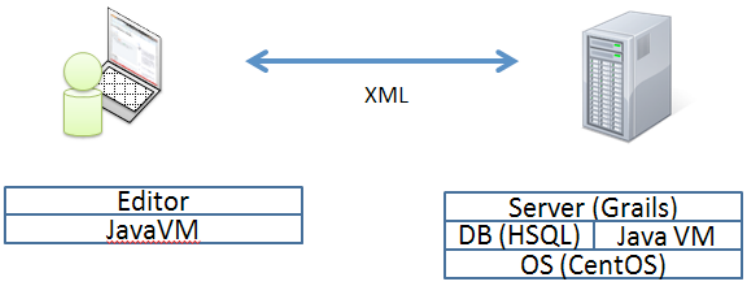

Figure 7: Student input system overview

Results

We evaluated the data acquired by the system before and after the algorithm was used. The data patterns for two different input procedures are depicted in the graphs shown in Figures 8 and 9.

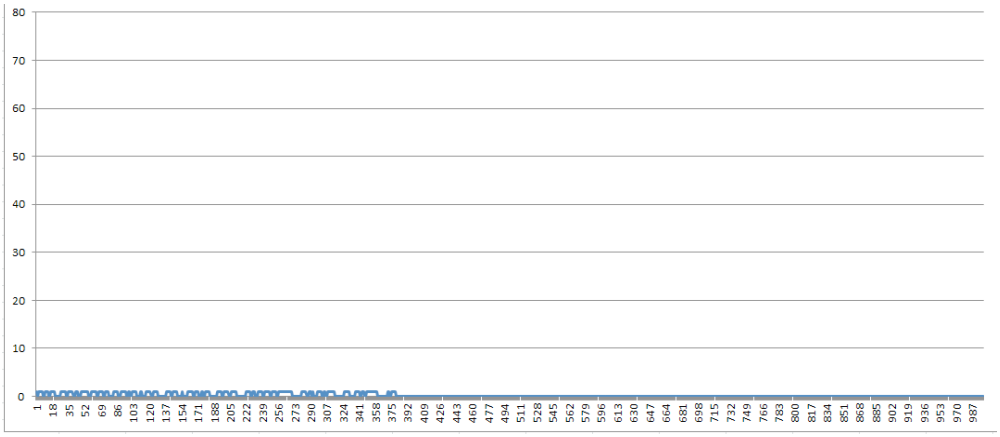

Figure 8: Good pattern graph

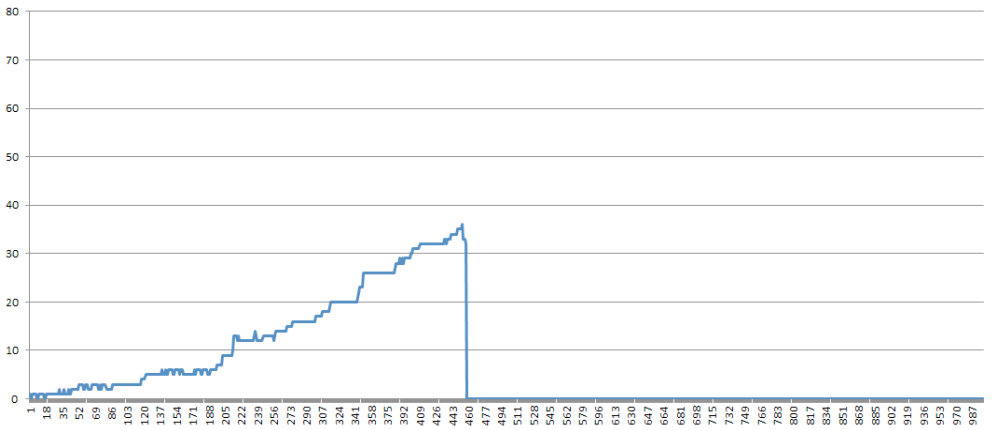

Figure 9: Bad pattern graph

The vertical axis of each graph is the number of keystrokes, while the horizontal axis shows the score of the algorithm. Figure 8 shows the graph with repeated 1 and 0. When an input of a token is set to one and the token is inputted correctly, it will return to zero. Figure 9 shows the procedure that summarized the contents of the method at the end and inputted them. It is clear that the score is increasingly larger.

Discussion and Conclusion

As a result of using acquired input data, an objective determination of whether a coding procedure is right or wrong can be made from a graph. We plan to improve the system presented in this paper and use it in actual lessons to analyze incoming data and investigate its relevance to the students' results in future work.

References

Jackson, J., Cobb, M., & Carver, C. (2005). Identifying top Java errors for novice programmers. Proceedings of the 35th Annual Conference on Frontiers in Education, 2005. FIE '05, pp. T4C24-T4C27.

Oman, P. W., & Cook, C. R. (1988). A paradigm for programming style research. ACM Sigplan Notices, 23(12), 69-78.

Miara, R. J., Musselman, J. A., Navarro, J. A., & Shneiderman, B. (1983). Program indentation and comprehensibility. Communications of the ACM, 26(11), 861-867.

Tatsuyuki T., Takashi K., & Osamu M. (2011). Development of the coding process analysis system for programming education. Proceedings of the Asian Conference on Education, pp. 517-527.