CodeMem: A Trace-Based Evaluation Tool

ABSTRACT

Trace table is a technique used to demonstrate a program's state by specifying line of code to be executed and values of the variables in the code. However, students and teachers are required to draw trace tables by hands, which is a tedious and time consuming activity. It also requires much effort by the teachers to check solution provided by each student, when used for students' assessment. In this paper, we present a tool, Code-to-Memory (CodeMem), which is a replacement to manual trace-based evaluation. CodeMem is a web-based user friendly tool developed for evaluating basic programming concepts using trace table approach. The tool allows teachers to upload C++ code snippet as assignments and automatically generates (trace table) solution of the code. Students solve the assignment(s) by filling a trace table to reflect changes made in the memory (if any) in response to each executable line in the given code. CodeMem also auto-grades the students' solutions and provide feedback to both teachers and students.

KEYWORDS

Trace table, code dry-run, programming, auto-grade students' solution.

1 INTRODUCTION

Learning how-to-code and mastering programming is an essential skill required by Computer Scientists. Programming skills are not always limited to writing code from scratch and it may also involve reading, understanding and analyzing already written code. However, it has been seen that many students find difficulty in understating programming concepts at earlier stages which results in higher student failure and course drop out ratio in preliminary programming courses at various universities [1,4]. Building conceptual models of pre-written code is considered as a good exercise to learn programming. It is essential to create concrete models of complex programming concepts in preliminary programming courses. Study in [1] showed that building conceptual models at early stages tend to be helpful in course retention and better student performances.

Methods used to build viable mental models are commonly referred as: code visualization, trace tables and memory diagrams. These approaches are used as part of various programming courses and proved to be effective in students' learning, such as in [1,3]. Code visualization and memory diagrams are useful to bridge the gap between textual representation of source code and its representation during program execution. Trace table is used to demonstrate a program's state by specifying line of code to be executed and values of the variables in the code. A sample code and its trace table from [4] is given in Figure 1 below to understand the basic idea.

Code segment

```
int x=0;
int i=1;
while(i<5){
    x=x+i;
    i++;
}
```

Trace Table

i	x
1	1
2	3
3	6
4	10

Figure 1*:* Example dry-run exercise and trace table taken from [4]

This is a simple code snippet containing only two variables: *i*, *x* and a *while* loop; values of the variables are recorded in the table to show changes in each iteration of the loop. The table contains a separate column for each variable and each row contains values against one loop iteration. This is a simplest trace table which is usually used however we can modify the table as per our needs i.e. first column can be used to show line number executed from the given code snippet [14] (assumingly line numbers are already specified in the code).

These trace-driven approaches are not only useful for teaching but also to evaluate students' understanding of programming concepts [6]. Trace-based or code dry-run exercises proves to be very effective as it helps novice programmers to better understand the errors [14] and flow of a program [4]. Code dry-run exercises can also very effectively improve the code reading and analyzing skills [7]. However, students are required to draw memory diagrams and/or fill trace tables by hands, as described in [3,4,5,6]. At the same time, teachers are required to design solutions and evaluate several set of exercises manually, which is tedious and time consuming activity. The need to develop a tool for trace-based evaluation becomes essential for introductory programming courses offered to Computer Science undergraduate students where students enrolled into these courses every semester in a large number.

In this paper, we present a web-based tool, Code-to-Memory (CodeMem), developed to assist teachers and students in trace-based C++ coding exercises. Followings are the salient features of CodeMem, details are provided in Section 3.

- Automatically generates solution trace table of error-free C++ code snippet uploaded by the instructor.
- Allows students to solve assignment(s) by filling the trace table to reflect changes made in memory (if any) in response to each executable line of code.
- It also auto-grades the submissions made by the students and provides feedback to both teachers and students.

We provide a brief background of this tool in Section 2, before providing CodeMem details (Section 3). Related work is

discussed in Section 4 which is followed by a summary in Section 5. Lastly in Section 6, we have mentioned our future work.

2 BACKGROUND

CodeMem aims to support the instructors teaching basic programming course using C++ to the first-year undergraduate students. The tool is developed as part of a Final-Year Project at XYZ University. Current implementation supports the following programming constructs taught in the very first programming course *(CS 101-Introduction to Computing)*: variables declaration and initialization, operators (arithmetic, logical and relational), selection statements, repetitive statements (while loop), one-dimensional array and user-defined functions (parameter pass-by-value, including function overloading).

3 CODEMEM

Code-to-Memory (CodeMem) is a web-based user friendly tool for evaluating basic programming concepts of Computer Science students using trace table approach. The tool allows teachers to upload C++ code dry-run exercises and automatically generates solution trace table of the code. Students can solve the assignment(s) by filling the trace table to reflect changes made in the memory (if any) in response to each executable line in the given code. CodeMem also auto-grades students' solutions and provide results to both teachers and students.

3.1 Instructor Module

Figure 2 below, shows the instructor module with *Upload Assignment* tab opened where an instructor can create an assignment for the students by writing C++ code or uploading a source file (.cpp). Once the error-free code is uploaded, the system automatically generates its solution trace table by parsing the code into tokens and recording each new identifier and respective attributes in a data structure, similar to how it is being done in compilers [8]. CodeMem allows a teacher to perform some basic operations by using the links available on the left pane such as: *Manage Classes*, *Manage Students* and *View Grades* and few more (see Figure 2).

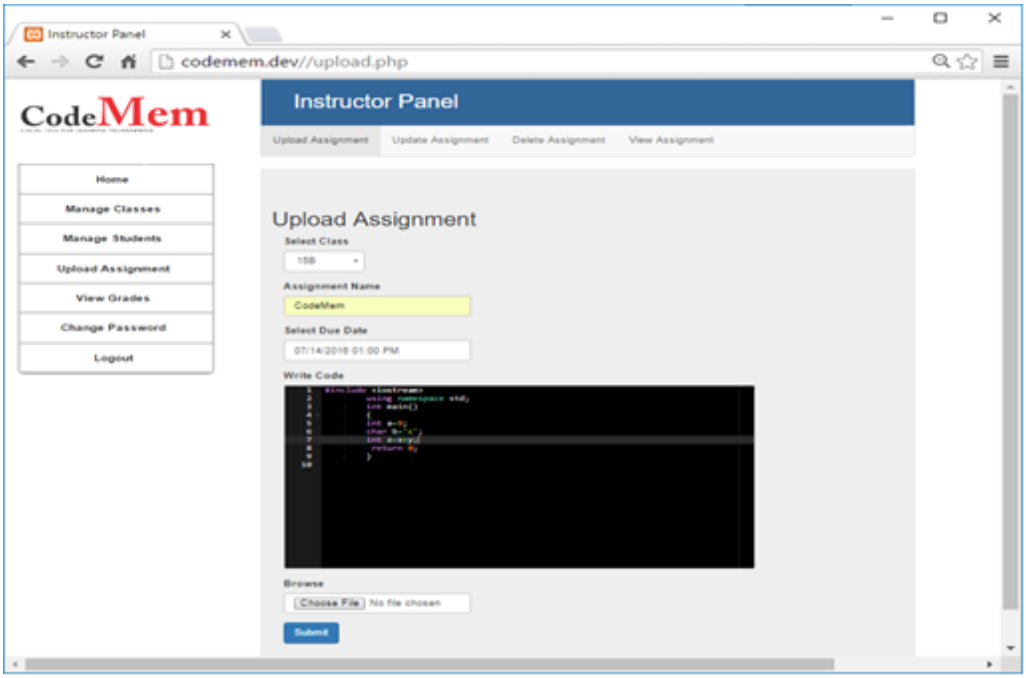

Figure 2: Instructor module interface (with Upload Assignment tab opened)

3.2 Student Module

Figure 3 shows the student interface with an assignment opened. The assignment window is divided into two sections: code shown on the left side and trace table to be filled (three rows already added to the table in this example) on the right side. A student may add/delete rows to the trace table as required using the buttons provided above the trace table. The student's solution is then compared with an auto-generated solution and results are provided to both teachers and students.

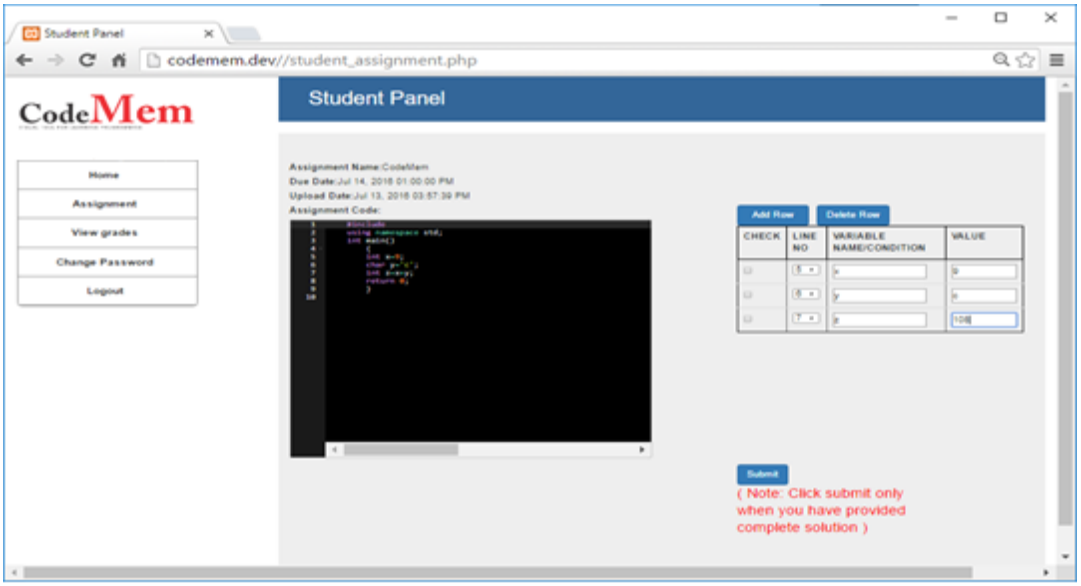

Figure 3: Student module (a student's solution to an uploaded assignment)

3.3 Trace Table Design

The trace table is very simple as shown in Figure 3, and much similar to the one used in the classrooms typically. Students are required to fill the last three columns of the trace table (first column is used to select a row(s) to be deleted) for the given code snippet displayed on the left side. Initially the table is empty and a student may add rows as many as required against each executable line of code which is expected to change the value of some identifier(s) or flow of the program (i.e. in case of a conditional statement). The snapshot given in Figure 3 shows the sequential code example along with its solution table filled by a student.

3.4 Implementation Details

CodeMem is a web-based tool with a backend implemented using Java and PHP, and frontend is designed using HTML, CSS and JavaScript.

3.4.1 Backend (Java, PHP)

The backend of CodeMem involves taking C++ source code as input, parsing and processing of the code and generating its solution trace table. Java is used for processing C++ code uploaded and all other backend operations are performed in PHP.

When an instructor uploads an assignment, the C++ code is checked for errors and exceptions using *Sphere Engine API* [9] as it is not safe to run untrusted C++ code on the server. *Sphere Engine API* also prevents the execution of infinite loops and other dangerous constructs such as eval, exec and File I/O. Once checked and verified, the C++ code is processed by an executable *jar* file which parses the code, creates a symbol table like the one created by a compiler [8], extracts the information into a specified form of trace table and stores it as a solution table.

3.4.2 Frontend (HTML, CSS, JavaScript)

The frontend provides user-friendly logins, navigation side bars and attractive menus to both teachers and students. Teachers can write code on the editor, which is an embeddable code editor written in JavaScript, known as *Ace Syntax Highlighter* [10] (as show in Figure 2, above). *Ace* is also used to display C++ code in the tool, mostly in read-only mode. Students fill the trace table in a dynamically created table developed using JavaScript, allowing students to add/delete rows as required. On clicking the submit button the trace table is converted into a two-dimensional array which is used to initialize a JSON object and send it to the backend for processing.

4 RELATED WORK

There are several existing systems developed for teaching programming using the techniques discussed in the Introduction (see Section xxx), for example: Online Python Tutor [2], Code-memory diagram animation software [11], CoffeeDregs [12], JVALL [13]; providing visualization at varying levels of abstraction. Online Python Tutor [2] makes use of debugging tool to generate the solution trace of the given code in Python and run passive animations at the front-end accordingly. The tool offers interactive GUI showing memory diagrams for animation which differentiates it from a simple debugger. Tool presented in [11] also generates a solution initially using Microsoft Visual BASIC and uses less attractive way to visualize changes made in the variables. CoffeeDregs [12] and JVALL [13] adopt similar approaches for code visualization and are useful for teaching Object Oriented and linked list concepts respectively by animating block diagram to show changes in the memory. JVALL also allows students to debug their own code and better understand errors in their code using visualization.

These tools are useful to teach difficult programming concepts by code animation and showing changes made in memory as an effect of line(s) of code executed recently. However no or very limited student interaction is provided and these tools do not support students' assessment at any stage.

5 SUMMARY

CodeMem is developed for students' assessment using trace-based exercises, which is done manually in the classrooms earlier and proves to be an exhausted activity for both teachers and

students. The tool has already been used in CS 101-Introduction to Computing class of Spring 2016 (29 students) at XYZ university; we could conduct only three sessions during the semester as the tool was in its development phase. The instructor uploaded four to five set of trace-based exercises (C++ code snippets) in each session using *Instructor module* and students solved those exercises using *Student module*. Immediate results were available to both instructor and students at the end of each session. Correct (trace table) solution was also available to the students so that each student could compare it with his/her own solution and identify the mistakes made in the student's submitted solution.

Students were able to use the tool easily after understanding the basic operations available in the *Student module* and structure of the trace table (which was slightly modified from what they have been using in the class). Students were observed closely while interacting with the tool, observations were recorded during the sessions and their feedback was also taken at the end of each session. Based on the feedback and recorded observations, appropriate changes were made to the interface in two iterations. CodeMem is a user-friendly tool allowing both students and teachers to perform trace-based code evaluation in an efficient manner.

6 FUTURE WORK

We plan to extend the scope of the tool by adding more programming constructs (i.e. multi-dimensional arrays, for loop, do-while loop, pointers, etc.). The current implementation supports trace-based evaluation; however, we desire to make changes in the instructor panel such that the tool can also be used to teach problem solving during lectures using code visualization. Designing and developing a mobile phone application for CodeMem is also included in our future work directions so that students can use the tool on-the-move.

REFERENCES

[1] Matthew Hertz and Maria Jump. 2013. Trace-based teaching in early programming courses. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13), 06-09 (Mar. 2013), 561–566.

[2] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13), 06-09 (Mar. 2013), 579–584.

[3] Mark A. Holliday and David Luginbuhl. 2003. Using Memory Diagrams When Teaching a Java-Based CS1. Proceedings of the 41st ACM Southeast Conference (ACMSE'03), March 2003.

[4] Leonard J. Mselle. 2014. Using Memory Transfer Language (MTL) as a Tool for Program Dry-running. International Journal of Computer Applications 85(9), (Jan. 2014), 45–51.

[5] Michael Striewe and Michael Goedicke. 2014. Code reading exercises using run time traces. In Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE '14). ACM, New York, NY, USA, 346-346.

[6] Mark A. Holliday and David Luginbuhl. 2004. CS1 assessment using memory diagrams. In Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE '04). ACM, New York, NY, USA, 200-204.

[7] K. K. Zaw and N. Funabiki. 2015. A Concept of Value Trace Problem for Java Code Reading Education. 2015 IIAI 4th International Congress on Advanced Applied Informatics (IIAI-AAI), Okayama (2015), 253-258.

[8] Niklaus Wirth. (1996). Compiler Construction. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA.

[9] Do you need to run code in your app?. Sphere Research Labs, [Online]. Available: http://sphere-engine.com/features

[10] "Ace - THE HIGH PERFOMANCE CODE EDITOR FOR THE WEB," Ace Syntax Highlighter, [Online]. Available: https://ace.c9.io/#nav=about.

[11] Mark Dixon. 2004. Code-memory diagram animation software tool: towards on-line use. In Proceedings of the IASTED International Conference on Web-based education, Innsbruck, Australia (Feb. 2004), 16-18.

[12] Cornelis Huizing, Ruurd Kuiper, Christian Luijten and Vincent Vandalon. 2012. Visualization of Object-Oriented (Java) programs. In Proceedings of the 4th International Conference on Computer Supported Education (CSEDU'12), 65-72.

[13] Herbert L. Dershem, Ryan L. McFall, and Ngozi Uti. 2002. Animation of Java linked lists. In Proceedings of the 33rd SIGCSE technical symposium on Computer science education (SIGCSE '02). ACM, New York, NY, USA, 53-57.

[14] Step-Form algorithms - the simplest form of algorithm and: How to use Trace Tables. (1997). Learning Systems, 1997. Available at: http://users.evtek.fi/~jaanah/IntroC/DBeech/3gl_step.htm#tracetables