Online Programming System for Code Analysis and Activity Tracking

Tian Qiu, Purdue University

Tian Qiu is a senior undergraduate in Computer Engineering and Mathematics-Computer Science.

Mr. Mengshi Feng, Purdue University

Mengshi Feng is a senior student at Purdue University. He is one of the team member in ACCESS project supervised by Yung-Hsiang Lu.

Mr. Sitian Lu, Purdue University

Sitian Lu is a junior studying Computer Engineering in Purdue University. He has been working on the online programming system (ACCESS) since later 2015. Sitian Lu has been studying in Purdue University for 3 years. He is also the vice president of Purdue Billiards Club since 2015.

Mr. Zhuofan Li
Mr. Yudi Wu
Dr. Carla B. Zoltowski, Purdue University

Carla B. Zoltowski is an assistant professor of engineering practice in the Schools of Electrical and Computer Engineering and (by courtesy) Engineering Education at Purdue University. She holds a B.S.E.E., M.S.E.E., and Ph.D. in Engineering Education, all from Purdue. Prior to this she was Co-Director of the EPICS Program at Purdue where she was responsible for developing curriculum and assessment tools and overseeing the research efforts within EPICS. Her academic and research interests include the professional formation of engineers, diversity and inclusion in engineering, human-centered design, engineering ethics, leadership, service-learning, and accessibility and assistive-technology.

Dr. Yung-Hsiang Lu, Purdue University

Yung-Hsiang Lu is an associate professor in the School of Electrical and Computer Engineering and (by courtesy) the Department of Computer Science of Purdue University. He is an ACM distinguished scientist and ACM distinguished speaker. He is a member in the organizing committee of the IEEE Rebooting Computing Initiative. He is the lead organizer of the first Low-Power Image Recognition Challenge in 2015, the chair (2014-2016) of the Multimedia Communication Systems Interest Group in IEEE Multimedia Communications Technical Committee. He obtained the Ph.D. from the Department of Electrical Engineering at Stanford University.

Online Programming System for Code Analysis and Activity Tracking

Abstract
Many tools have been developed to assist programmers develop high-quality code. However, installing, updating, configuring, learning, and running these tools can be unnecessary burden on students. Moreover, instructors do not have detailed knowledge about students' learning experience before programming assignments are submitted. This paper presents an online system that can automatically analyze students' programs and provides insightful information about their code. This system records every syntax and run-time error so that an instructor can obtain real-time view of students' activities and progress. Hence, the instructor can identify common misconceptions before an assignment is due. This system is evaluated in an A-B test of a sophomore C programming class of 42 students. The results suggest that this system has positive effect on helping students learn.

Keywords
Online Education, Web-Based Technology

1. Introduction

Imagine that a student must learn how to install, maintain, operate, and repair an engine before learning how to be a professional pilot. Now, imagine that a student must learn how to install, maintain, configure, and execute programming tools before learning how to be a professional programmer. These would be unreasonable requirements for the students. Yet, the latter is commonly expected by instructors. Even though pilots do not need to know how to install an engine, pilots need to know how to interpret the information from engine sensors (such as overheating) and take appropriate actions. Even though programmers do not need to know how to configure tools, programmers need to know how to correct their programs in response to the information reported by the tools (such as memory leak and test coverage). Therefore, it is necessary to provide a system integrated with useful tools for students learning to program.

Software development is a complex process; many tools (such as performance profiling, test coverage, and memory leak detector) have been created to help software developers. These tools can provide valuable information to help understanding, improving, and debugging software. Learning how to use these tools can occupy precious lecture time. Teaching students to write programs without these tools is like teaching how to fly an airplane without instruments. Airplane manufactures and ground crew are responsible for installing and maintaining the instruments. Following the approach to the aviation industry, we created an online programming environment that provides valuable information (like an airplane cockpit) so that students can learn to program (like pilots) and do not have to worry about how the information is generated.

This online system automatically generates information about the runtime behavior of students' programs, including test coverage, performance profiling, and memory usage (the amount of heap memory allocated, whether any invalid address is accessed, and memory leak). Students are able to replace their unfinished or incorrect functions by the correct functions written by the instructors; thus, students can perform unit tests more easily. Function replacement also makes it easy to give students partial credits. If a student's program terminates abnormally (such as a segmentation fault), the system automatically invokes a debugger and shows the call stack. Students are also able to interactively debug their programs. Even though this system sounds like an IDE (integrated development environment) on the surface, it is much more than an IDE. The online system records every syntax and runtime error in the students' programs and reports the statistics to the instructor. Such information allows the instructor to identify common misconceptions and provide additional help to the students.

This system is evaluated in a sophomore C programming class of 42 students. The students took a pretest to evaluate their understanding about the materials to be covered in the class. Then, based on test results, students were divided into two groups. Both groups could use this system in the first programming assignment. The first group could use the system for assignments 2 and 4. The second group could use the system for assignments 3 and 5. All students took posttests to evaluate their understanding of the materials. The evaluation results suggest that this system is helpful to both students and instructors.

2. Related Work

Recognizing the importance of software development, computer science has become an integral part of general education in USA [1]. Our system integrates many useful tools and helps students learn. Figure 1 shows the architecture of the system. The main components include a web server, a file system, a database, an execution container, and analysis engines. The analysis tools are integrated into the execution container. This system used Docker for the container.

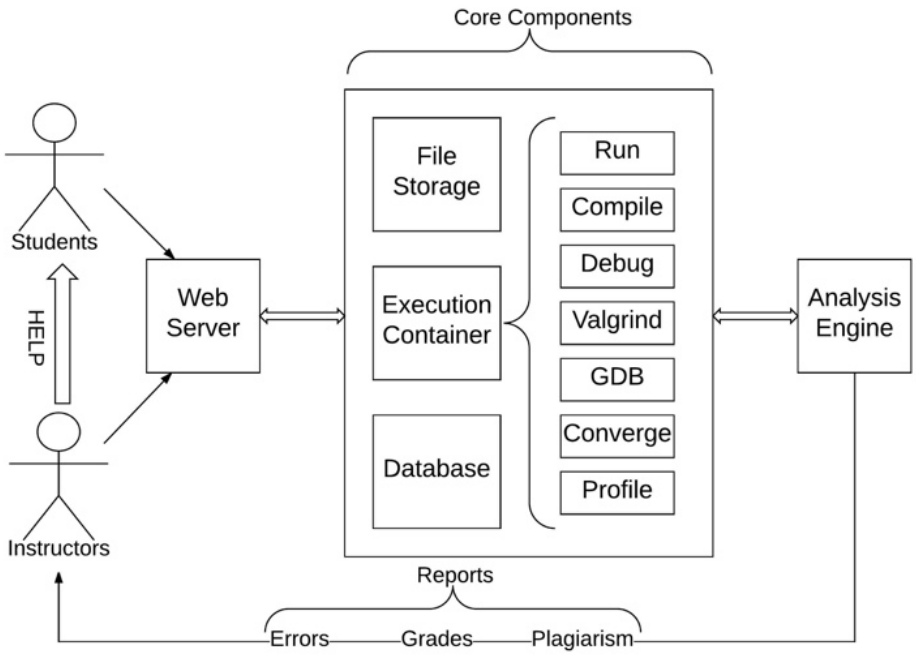

Figure 1 System Architecture

2.1 Web Integrated Development Environment

Web integrated development environment (WIDE), such as Cloud9 [2] and Codeanywhere [3], provides similar features as desktop IDEs, e.g., syntax and compiling [4]. WIDEs store programs on cloud, thus, eliminating the needs for backup. WIDEs may still be too complex for beginning programmers to learn. Even though the process of setting-up (such as installing plugins for language support in Desktop IDEs) is reduced for WIDEs, the required steps to run different tools can still be overwhelming to beginners. Also, existing WIDEs provides no information on instructors about students' learning progress and experience. Our system keeps the advantages of WIDEs and automates many tools to help students. The system provides a text editor that automatically saves multiple versions and requires no configuration (students do not have to configure version control). Students can obtain insightful information about their programs without learning how to configure and invoke these tools. The tools that are already integrated into our system include debugger, test coverage, performance profiling, and, a memory error detector (called `valgrind`). The system eliminates the needs for students to manage the analysis tools. This approach is analogous to obtain an airplane cockpit: pilots obtain crucial information about the plane from the instruments.

2.2 Interactive Tutorial-Oriented Platform

Codecademy [5] presents an interactive and collaborative platform that offers free coding lessons, and also offers features such as feedback, personal rating. Codecademy has some "hints" containing mandatory directions, and occasionally does not give a reason why a student fails a test [7]. Both tools lack debugging support. CS circles [6] integrates university curriculums, embeds thousands of course-related auto-gradable practices assignments [7] and

also provides console and visualizer to execute and debug program [8]. Brusilovsky and Sosnovsky [9] developed another online tool called QuizPACK, which receives parameterized questions and exercises and generates different questions. The quiz generator and KnowledgeTree [10] may improve students' understanding, by doing quiz-like questions. Our system addresses the deficiencies and embeds analysis tools. These tools may have dozens or hundreds of options. Learning these tools can occupy precious lecture time. Also, our system generates statistics of students' mistakes while they are doing homework assignments. Such information can help instructors know the students' progress.

2.3 Automatic Grading

Edwards [11] proposes test-driven development using an automatic grading tool called Web-CAT [12] and encourages students to submit their own test cases. ASSYST [13] provides compile/test/result-comparison for students and accepts student-provided test cases. However, students are not rewarded when providing their own test cases [11]. Anabela Gomes, and António José Mendes [14] pointed out that some instructor's methodologies may not sufficiently consider students' learning styles. Instructors often evaluate students' understanding by only assignment scores, i.e., after the assignments are submitted. Three professors at University of British Columbia [15] use Mylyn Monitor framework (a plug-in for Eclipse IDE) to trace students' coding activities, e.g., timestamps. Instead of checking correctness only, our system considers the software development process and provides useful information on both students and instructors. It collects different types of data, e.g., how much time a student spends on an assignment and which part of the assignment. It also records compilation errors and run-time errors, how students navigate through the learning materials. The system provides the information to instructors so that they can answer common questions before assignment deadlines. Table 1 summarizes the difference between our system and other existing work.

Approaches	Designed for program beginners	Debugging Tool	University Assignment-Embeddable	Practice Problems	Assignment-Gradable	Code quality check	Data feedback to instructor
WIDEs		✓				✓	
Codecademy	✓			✓	✓		
quizPACK	✓			✓			
CS Circles	✓		✓	✓	✓		
Web-Cat/ASSYST	✓				✓		
Proposed-System	✓	✓	✓		✓	✓	✓

Table 1 Comparison of different approaches that facilitate the learning process. Our system is designed to help both students and instructors.

3. Integrated Tools and Methods

The system is designed for scalability: it uses two separate servers. One is a web server that hosts the web interface and student database. The other is a backend server that executes and analyzes students' code. The web server uses the Django framework. The backend server

uses Docker containers. Docker provides the ability to isolate each student's code into an individual container so that even if student's code misbehaves, it will not crash the backend server. This design makes the system easier to scale up: when more students use this system, more machines can be added to the backend and host more containers.

The communication flow between the two servers is shown in Figure 2 below. When a student clicks the "run" button at the web site, the web server's JavaScript sends an Ajax request. Then the request synchronizes the student's code to backend server. The backend compiles and runs student's code. Then the backend server sends the result (or error messages) to the web server. The database in the web server records the results and displays them to the student.

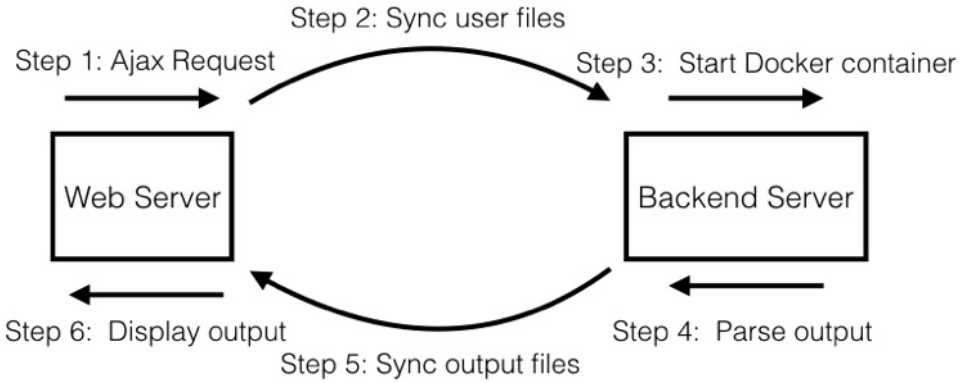

Figure 2 Communication between web server and backend server.

3.1 Web Server

The web server hosts three main web pages: home, workspace, editor. The home page is the introduction before a student logs in. The workspace allows students to manage their courses, projects, and account settings. The editor page is the place where students write code. The ACE editor [20] is embedded into the system. It can automatically format code for better readability. The web interface has multiple buttons (such as run, memory test, coverage, and profile) to invoke these tools:

- `valgrind` - memory error checks
- `gcov` - code coverage
- `gprof` - performance profile
- `git` - version control system
- `gdb` - interactive debugging

Students can click these buttons without any effort to set up the tools. Students do not need to know the commands to invoke these tools. When using version control, students do not have to create repositories. Our system handles all the necessary steps in the database and the backend. When a student clicks one specific operation (such as coverage), the Ajax request takes the student's information and operation to write as an input log. The information and operations send the student's files to the backend container. When the Docker container in the

backend finishes the compilation, execution, profiling, or other operations, the result is synchronized with the web server and displayed in the browser. Figure 3 and 4 show two screenshots of the web interface. Figure 3 shows a run-time error: the program intends to dereference an NULL pointer. When this occurs, the program has a segmentation fault. This system automatically invokes `gdb` and finds the function of the top frame. The line of the top frame is highlighted so that the student can immediately know which line causes the segmentation fault.

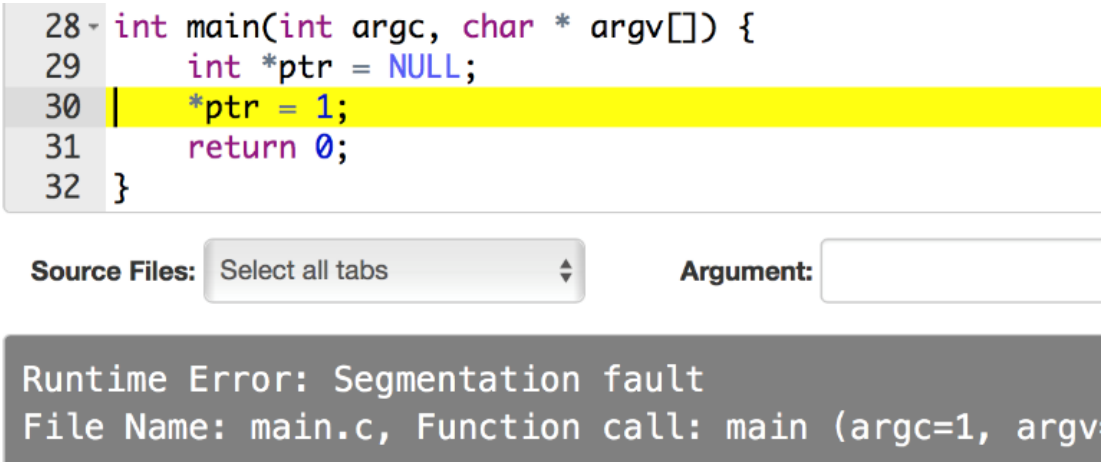

Figure 3 Sample programming scenario in our system

Figure 4 shows the editing environment. The left panel shows the programming assignments. The center is the editor and a menu bar for editing commands (such as undo). The right panel contains buttons of the analysis tools.

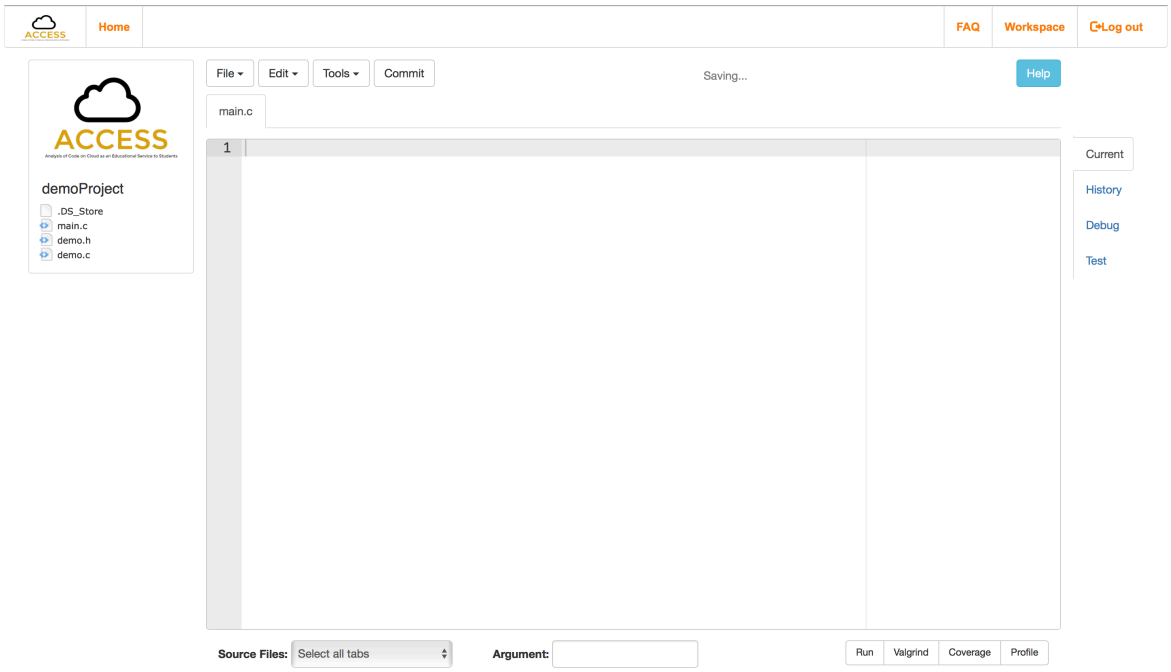

Figure 4 Overview of coding environment

3.2 Backend Server

The backend server compiles, executes, and analyzes students' code in Docker containers. Docker can package application for a standardized unit for software development [18]. A docker image is created to invoke a docker container containing all the integrated tools including `gdb`, `valgrind`, and `gcc`. Thus, all students work in identical environment and eliminate the confusion from different versions of tools. After a Docker container starts, the student's code is sent into the container. Then, a python script invokes the desired operation based on student's request. Another advantage using a container is to separate and control the working environments for better protection. It is also possible, if requested by the instructor, to set resource limits, such as the amounts of heap memory, stack memory, or time a student's program can use. These limits can be helpful when an instructor wishes to teach resource management or efficiency.

The tools for the backend generate results of different formats. Hence, post-processing of these tools' outputs is needed to present consistent information on students. Moreover, some tools may generate many lines of information. Our system extracts the outputs from these tools and provides succinct summaries to the students. Invoking these tools has three stages: verification, execution, fetching. Before starting any operation, the backend server checks whether it receives all the necessary information on the expected format. Before execution, the backend checks if all the files are valid ".c" files. The execution stage uses Python's subprocess [19] to simulate terminal commands and to invoke the corresponding operation. For example, if a student's program has run-time errors, the backend invokes "`gdb`" and prints the call stack. The last stage fetches and filters critical information on students. For example, a `valgrind` report is usually very long. Post-processing keeps the memory error types, the locations, and the line numbers. The system also removes duplicate information. For example, if the same line of code causes multiple memory errors, `valgrind` prints multiple error messages and our system keeps only one. Discarding some information helps students focus on correcting their programs without knowing all details (such as the size of leaked memory and how many times the same error occurs). Our observations suggest that students could correct their mistakes faster because they do not have to read the lengthy error reports.

4. Evaluation Results

During summer 2016, this system was utilized by a class of 42 sophomore students taking a C programming class. This is the second programming class the students took. The prerequisite is a programming class that teaches C and MATLAB.

4.1 Pretest

In the first day of class, the students took a pretest of multiple-choice questions. This pretest contains three types of questions: prerequisite, questions to specific assignments, and questions that requires knowledge about multiple assignments. The first type of questions is used

to understand students' background. The other two types of questions evaluate whether the students already know the materials. For all three types of questions, students could indicate that they did not know the answer so to reduce the effect of guessing at the results. Students could obtain bonus points by taking the pretest. The pretest results indicate that the students had sufficient understanding of the prerequisite and most did not know the materials to be covered in this course. Based on the pretests, the students were divided into two groups so that the average scores were almost equal. Each group had 21 students. Both groups could use this system for the first programming assignment. Group A could use this system for assignments 2 and 4. Group B could use this system for assignments 3 and 5. In this paper, the group that uses this system is called the experimental group. The other group is called the comparison group.

4.2 Experiment Results

	HW02	HW03	HW04	HW05
Experimental	7.6	4.1	7.3	4.6
Comparison	6.7	2.8	6.1	7.6

Table 2 Homework assignment average scores. The full score of each assignment is 10.

Table 2 indicates students' scores for each assignment. The experimental group outperformed in the second to the fourth assignments. The last assignment reveals the limitation of the system. This system aims to play the role of scaffolding: helping students understand the valuable information provided by many programming tools and eventually use the tools of this system. For this purpose, the fifth assignment required that students design their own test cases but many students failed to do so. As a result, their programs were not adequately tested before submission. The last assignment suggests that some students became overly dependent on this system. When this system is used in future courses, it would be necessary to design better transitions.

4.3 Error Report

One advantage of this system is the ability to record every error message the students encounter. Table 3 shows the most common compilation errors and their percentage. Some rows (such as "pointer error") are aggregates of multiple error messages. Some students did not understand the meanings of the error messages. By knowing the common errors, the instructor could explain to students how to correct their programs before the assignments were due. The top three errors or warnings are "Expected identifier or expression not found", "Undeclared Variables", and "Pointer error". The first two errors can be summarized as "students writing codes without a plan" [16]. The third error is the gap between understanding pointers and their types. It is observed that some students tried to cast types of understanding the implications. Their programs could pass compilation but had run-time failures.

Compilation Error	Percentage of total errors
Expected identifier or expression not found	23.5%
Undeclared Variables	20.8%
Pointer error	17.2%
Not used Variables	13.3%
Implicit declaration	11.2%
Control reaches end of non-void function	8.3%
Shadow declarations	2.3%
Invalid type or operands	1.8%
Uninitialized Variables or functions	1.4%
No such file or directory	0.3%

Table 3 Compilation errors and their percentages.

The most common memory errors are listed in Table 4. As we can see that most students' errors are invalid memory read and uninitiated conditions (i.e., a condition depends on a variable that has not been initialized yet).

Memory Error	Percentage of memory errors
Invalid Memory Read	21.4%
Uninitiated Condition	18.5%
Invalid Memory Write	11.3%
Invalid Memory Free	8.3%
Memory Leak	6.9%
Uninitiated Value	1.3%
System call Parameter	0.6%

Table 4 Memory errors and their percentages.

4.4 Posttest

A posttest of multiple-choice questions was given at the very end of the semester. The questions were again categorized into three types. The first type had four questions and was not specific to any homework assignment. The second type contained another 4 questions and they required knowledge from multiple assignments. The third types had 12 questions specific to assignments 2 to 5. The first two types of questions were used to evaluate the students understanding of the entire class. Among the 12 questions on the third type, the group that used the system were better in 8 questions. The average advantage was 7%. Among the other four questions, one was discarded due to ambiguity in the question. The average score of the pretest (including four prerequisite questions) was 10.6. The average scores of the posttest (without the prerequisite questions) was 13.9 and 13.95 of the two groups. The results suggest that the students have learned the course materials. In addition to the pretest and posttest

comparison, some students told the instructor that the system was very helpful in their learning. The evaluation showed promising results and we are extending this system to a class of data structures. The following paragraphs provide more details of the insight we obtained from the evaluation.

4.4.1 Memory Leak

This system is particularly helpful for students to correct mistakes that might not be obvious. Among the twelve type-3 questions in the posttest, all students in the experimental group were correct in one particular question. In contrast, only 70% students in the comparison group answered this question correctly. This question was about memory leak and the order of releasing memory for a multi-dimensional array. Program will not crash immediately due to memory leak. Thus, it is often overlooked by students. Our system makes checking memory leak easy, by clicking a single button to invoke `valgrind`. Moreover, the results were presented succinctly. Thus, the experimental group was able to determine whether their programs leaked memory. According to our collected data, on average one student checks memory leak five times for each assignment before submission.

4.4.2 File Operations

We also discover that the experimental group outperformed in the question about file I/O. This question requires understanding of `stdin` and `stdout`. Our system separates students' `stdin` from the command line in terminals. Instead of typing in one command line with argument and executable file name together, students who used our system typed in arguments separately on our website. By doing so, students understood the concept of `stdin` better and were able to distinguish `stdin` from command-line arguments. Among the two posttest questions related to file I/O, the experimental group outperformed by 12% (65% vs. 53%) and 28% (60% vs. 32%).

4.4.3 String

In C programs, a string is an array of characters with a special ending character '\0'. This special character is a frequent source of misunderstanding and errors. It is common to forget the memory space needed for this special character. It is also common to forget ending a string with this special character. Three questions in the posttest were related to strings. The experimental group showed slight advantage over the comparison group with only 5% (95% vs. 90) and 5% (64% vs.58%). Since each group has only 21 students, this difference is caused by only one student.

4.4.4 Structure

Two questions in the posttest were related to structures. The experimental group outperformed significantly in one question (74% vs. 45%). This question is about retrieving an attribute through a pointer to a structure. This question asked students two different ways: Suppose `ptr` is a pointer and `ptr -> x` is an integer. What is the type of `(*ptr).x`? It was

not expected that the experimental group would outperform the comparison group significantly. We interviewed a few students and they suggested that this system was easy to use and thus encouraged the students to try different solutions. Thus, they understood different ways to solve the same problem.

4.5 System Usage

The system uses Google analytics [17] to track students' usage. We added several tags to track behavior flow, acquisition, operations, etc. The behavior flow indicates each step that a student goes from link to link. Also, it shows where the drop-offs happen. Acquisition helps understand the different traffic sources to our site [17]. The system can report to instructors who is using the system right now. It is also possible to know when more students write assignments and this information may help the instructor schedule office hours. The program activities show during the experiment period the intensity that students used system. This information helps instructors understand how much time students spend on the assignments. If the instructors discover that students spend much more time than expected, the instructors may provide more help during lectures. If one particular student spends much more time, the instructor could communicate with this student and provide personalized help.

5 Conclusion

This paper presents an online programming system to help students learn and to help instructors understand their students' progress and experience. The system adopts the approach to an airplane cockpit: information about students' programs is presented to the students but the students do not have to take much effort generating the information. Such information can offer deeper understanding of their programs. Instructors can know whether most students (or a specific student) need help before an assignment is due. The posttest of our experiment shows that the experimental group (the group that used our system for a specific area of knowledge) significantly outperformed the comparison group in questions related to memory leak, memory operation and structure. This promising result suggests that this system is helpful.

REFERENCES

1. Ormond, J. 2015. *ACM Hails New "Every Student Succeeds"* https://www.acm.org/media-center/2015/december/essa-epc
2. https://codeanywhere.com/
3. https://c9.io/
4. Wu, L., Liang, G., Kui, S., & Wang, Q. 2011. CEclipse: An online IDE for programing in the cloud. *In 2011 IEEE World Congress on Services, IEEE*, pp. 45-52.
5. https://www.codecademy.com/
6. http://cscircles.cemc.uwaterloo.ca/
7. Pritchard, D. and Vasiga, T. 2013. CS circles: an inbrowser python course for beginners. In *ACM Technical Symposium on Computer Science Education*, pp. 591-596.
8. Guo, P. 2012. Online Python Tutor: Embeddable web-based program visualization for CS education. http://pythontutor.com/.

9. Brusilovsky, P. and Sosnovsky, S. 2005. Individualized exercises for self-assessment of programming knowledge: *An Evaluation of QuizPACK. ACM Journal of Educational Resources in Computing,* 5(3), pp.6
10. Brusilovsky, P. 2004. A component-based distributed architecture for adaptive Web-based education, in: Hoppe, U., Vardejo, F. and Kay, J. (eds.) Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies, pp.386-388.
11. Edwards, S. H. 2003. Improving Student Performance by Evaluating How Well Students Test Their Own Programs. *Journal of Educational Resources in Computing*
12. http://webcat.org/
13. Jackson, D. and Usher, M. 1997. Grading Student Programs Using ASSYST. *In Proc. 28th SIGCSE Technical Symp. Computer Science Education, ACM Press*, pp.335-339.
14. Gomes, A. and Mendes, A. J. 2007. An environment to improve programming education. *In Proceedings of the 2007 international conference on Computer systems and technologies*, pp.88.
15. Murphy, G. C., Viriyakattiyaporn, P., & Shepherd, D. 2009. Using activity traces to characterize programming behavior beyond the lab. *InProgram Comprehension, 2009. ICPC'09. IEEE 17th International Conference on IEEE*, pp. 90-94.
16. Eick, S.G., Loader, C. R., Long, M. D., Votta, L. G. and Wiel, S. V. 1992. Estimating Software Fault Content Before Coding. In proceeding of *ICSE '92 Proceedings of the 14th international conference on Software engineering, ACM*, pp. 59-65.
17. http://www.google.com/analytics/
18. https://www.docker.com/what-docker
19. https://docs.python.org/2/library/subprocess.html
20. https://ace.c9.io