Project Number: CS-HCL-GRDR

CS2303 Autograder MQP

A Major Qualifying Project

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

By

Chang Liu

Tim Whitworth

March 21st, 2018

Project Advisor

Professor Hugh C. Lauer, Advisor

Abstract

During the teaching of the course CS2303, System Programming Concepts at Worcester Polytechnic Institute, teachings assistants and Professors spend significant time grading students' programming assignments. The CS2303 Autograder Major Qualifying Project, a graduation requirement for students to research and solve a problem related to their major, aimed to develop a series of programming assignments that are autogradable to help alleviate the burden of grading. The autograder also informs students how well they perform in their programming assignments before submitting them.

Table of Contents:

1 Introduction

The class CS2303 at WPI, according to the undergraduate catalog, “covers memory management, pointers, the machine stack, and input/output mechanisms” in the C and C++ languages for coding. A typical CS2303 class has more than 80 students, though more than 150 students could be taking this course at the same time. The size of the class is a heavy burden to the teaching assistants who are grading the students’ programming assignments. We note that some parts of the grading job are repetitive and tedious and hence automation would save time. We achieve this goal by developing an autograder along with autogradable programming assignments. This could save a lot of time for the TAs so they could focus on the parts that can only be graded by a person, such as style and commenting. Our autograder also helps students check their scores before they submit their assignments. If a student’s program does not compile or a file is missing, the autograder will warn the student before he or she submits. Therefore, our autograder facilitates the grading process for both the TAs and the students.

2 Background

2.1 Basics of Autograding Scripts:

One method of simplifying the grading of programming assignments is to develop an autograding script. An autograding system consists of both a server side program and a client side program. Students begin by downloading code templates for the programming assignments. After each programming assignment is completed, students run the client side program to submit their assignments and get a score. Students can make changes to their assignments and submit again to improve their score. Thus, students receive real-time feedback while they are doing the programming assignments, since they can submit their assignments multiple times and know their scores before the deadline.

There are two approaches to develop the client side program. One is to compile the student programs locally and upload the compiled programs to the server. The other approach is to compile the student programs locally and test the student programs locally. Only the test results will be sent to the server to compare with the correct answers. In terms of server safety, compiling locally is much safer for the server. Malicious software can be uploaded to the server by the client side program, and running software from unknown sources is dangerous. However, with local compilation, the autograding system can easily be hacked if the test cases are not sophisticated enough. Therefore, the test cases must not be built into the client side program and must change each time the assignments are submitted.

The server side program must be able to handle multiple connections simultaneously. Also, the server side program must identify which student or group is submitting the assignments and record the scores for each student and group. In the example of Carnegie Mellon University's Bomblab [1], the number of bomb explosions is recorded. If a student or group submits the same assignment multiple times, the highest score will be kept. If the second approach above is used, the server must also be able to generate different test cases for each submission and send them to the client side program.

The autograding script should be as generic as possible so that different programming assignments can share the same autograder. The only changes that should be made between different

assignments would be test cases and correct answers. Once an autograding script is developed, pre-existing assignments can be modified to make them suitable for the autograder.

A simpler and less robust method of easing assignment grading is creating assignments that are graded on completeness alone, such that a script is not required. Students may or may not be given a template to code in. Students are given specific requirements on program output or test cases. While this is an easier method of creating programs to implement, it has significant downsides. Any assignment turned in incomplete would need to be manually graded. It is also necessary to ensure that students cannot output correct looking results with shortcuts that allow them to avoid the assignment itself.

Another method of teaching basic computer science skills is an in-browser coding window, found on a few online courses such as GeeksforGeeks [2] and Ideone [3]. These coding windows are a small area to write or edit code, with a 'run' or 'test' button that both compiles and runs the code, outputting it to a nearby window. This availability of easily compiling and running code while learning allows those taking the course to visualize and practice concepts step by step as they progress through the course. In some cases the window will fill with starter code relevant to the current material, or require a small coding exercise before the student moves to the next concept. This method is most often used for teaching the basics of a language, likely as it would be less effective or practical for larger examples.

2.2 Further Applications of Autograding

Many challenges separate a functional autograding script from a useful script. In the following section we look at research done on increasing the usefulness of the concept of autograding, specifically scalability when integrated with an LMS (Learning Management System), and providing meaningful feedback. This section will be particularly relevant for any continuation done on the work in this paper. Scalability is incredibly important in large courses, where there exists the potential for hundreds of submissions nearing the deadline. A report by Danutama and Liem [4] looks at the viability of grading approximately 400 submissions through an LMS. They found that the process of grading submissions could be reduced from 3 minutes to 10 seconds per submission with the introduction of an autograder,

allowing for a significant decrease in required time of course staff. This also allows for more immediate feedback to students. They tested using a known LMS, Moodle, along with a written autograder and a dispatcher to handle scheduling and queues as three separate modules. Their testing proved the viability of grading in under a minute with over 100 submissions per minute. For the purpose of coursework at WPI, this is far beyond the necessary scale.

Providing meaningful feedback is a more nuanced challenge. While grading the strict correctness of a program in a binary fashion is common, researchers from Rutgers University [5] looked at providing feedback including specific hints for common classes of errors.

> "We propose a methodology for extending autograders to provide meaningful feedback for incorrect programs. Our methodology starts with the instructor identifying the concepts and skills important to each programming assignment, designing the assignment, and designing a comprehensive test suite. Tests are then applied to code submissions to learn classes of common errors and produce classifiers to automatically categorize errors in future submissions. The instructor maps the errors to concepts and skills and writes hints to help students find their misconceptions and mistakes"

The researchers found that the hints they developed were able to be correctly applied 91.5% of the time to the first assignment submission, and 87.1% of the time for the second assignment submission, as classified by student researchers who had previously taken the course and looked through the testing done. A further 5.1% and 8.5% respectively of the automatic responses were classified as applied partially correctly. This shows an extremely high success rate of instant feedback for students without further input required from professors or other grading staff.

3 Methodology

3.0 Introduction: What types of assignments are autogradable?

Through our work on these course assignments, we have learned what types of assignments can be easily autograded, and what different methods can be used to autograde assignments. Before delving into the methods we used, we provide insight on our reasoning.

Firstly, the assignment must have a clearly defined input format. The input can be command line arguments, stdin or a file, but the input format must be specified. If the input is a number, it must be specified whether the number should be an integer or a float. If the input is a file, the internal structure of the file must be specified. Thus, the autograder can generate test cases according to the input format. Ideally, the autograder should generate all test cases randomly. However, some assignments have a finite number of possible inputs, or it is necessary to make sure that certain inputs are selected for testing. In these cases we would choose test cases semi-randomly from a predefined set of inputs.

Secondly, if grading the output of the program, the assignment must have a clearly defined output format. There must be no ambiguity in how the output could be formatted. In most cases, our autograder should look for an exact match between the student output and the correct output. Characters often not considered as important to visual formatting, such as spaces or newline characters, will affect the comparison. If the output is to be stored in a file, whether the output should be written in strings or byte sequences should be specified.

Thirdly, our autograder could be adapted and used with assignments that produce non-deterministic output, but the degree of randomness of the output is not tested. Our autograder can only test whether the student output is a possible output from the input or the previous output. Our autograder can also test whether the student output changes when the student program is run with the same input multiple times. However, our autograder cannot test how random the student output is. Our autograder is not capable of determining whether the student output follows a normal distribution or binomial distribution.

3.1 Creation of an Autograder Template

We started with the creation of a script that can compile the student program. This is not a final product; it is a stepping stone template that can be easily modified. The autograding template runs ‘make’ in shell mode to invoke ‘gcc’ to do the compilation. In order to do this, a new process is spawned. We used the Python built-in ‘subprocess’ module to run ‘make’. Our script runs commands in shell mode to enable collecting output from the shell. This is especially useful for generating proper error messages if an error occurs. The autograding template runs the compiled executable with inputs that we specify. While calling ‘make’, we can specify a string which later can be piped into stdin of the new process. Our autograding template also collects student output from stdout as well as stderr. We save that output to strings. Then, the autograding template runs our standard solution with the same inputs. This time it collects its output as the standard output, which later can be compared with the student outputs.

3.2 Creation of a Minimally Functional Local Client

We next added the ability to compare the student output with the standard output. By default, stdout is treated as a long string. Therefore, the autograder uses string comparison functions to match the two outputs. If a mismatch occurs, the autograder prints the differences between the student output and the standard output. This is achieved by importing the ‘diff’ module in Python. The local client only prints the parts of the student output that are different from the standard output. Any parts that are the same are not highlighted. The ability to show differences between the student output and the expected output is critical to our project because this allows the student to know which parts of the output are wrong; this helps the students debug their programs. If the two outputs are the same, the local client will record that the student has passed one test case. After all the test cases are run, the minimally functional local client will print to the screen how many test cases the student has passed and how many test cases the student has failed.

3.3 Editing Given Assignments for Compatibility with Grading Scripts

Here we suggest minor edits to each assignment's wording to reflect the use of a downloadable client. We suggest adding sections to clarify how the scripts would compare code, making students aware of the necessity of exact wording and formatting. The following sections detail recommended changes to the assignments and our analysis of their compatibility with out grading scripts. Note that Professors Michael J. Ciaraldi and Therese M. Smith made changes to the assignment format after the conclusion of our assignment editing. The assignments currently being used for CS2303 are no longer identical to those used for this project.

3.3.1 Assignments 1: Calendar Display

The first assignment [Appendix 7] is autogradable by comparing output without reworking the nature of the assignment. Therefore, no changes needed to be made beyond clarifying grading techniques and defining inputs and outputs. The first assignment as presented to our team by Professor Lauer prior to C term has been edited, see [Appendix 1].

3.3.2 Assignment 2: Game of Life

The second assignment [Appendix 8] is autogradable by comparing output without significant changes to the assignment. The requirement of a pause function could be deleted, as it was only used for grading purposes. We recommend creating five sets of input for each of three categories as follows: early termination by repeat, early termination by oscillation, and indefinite run (interval of repeat >2). Each run of the autograder should randomly select one input from each category.

3.3.3 Assignment 3: Binary Trees in C

The third assignment [Appendix 9] is autogradable by comparing output without significant changes to the assignment. How to deal with certain cases such as words split across two lines would need to be either defined or removed from used inputs. Assignment three could be solved without using

trees, and therefore a human grader will need to view code to ensure it was solved in the correct manner. This could be solved by writing a section of code for students that would output their tree in a parent-left-right order during the execution of their program, but this may give them hints as to how to complete the program, spoiling the assignment.

3.3.4 Assignment 4: Event Driven Simulation

The fourth assignment [Appendix 10] would not be autogradable by comparing output, because random number generation is integral to the development of the program. As such, we break down here both what could be retained while autograding the exact output of the assignment, and what grading method could be used instead to autograde it.

To grade the output using our current method of autograding the use of random number generation could be removed from the assignment, and the program be required to accept an input file of arrival and service timings. This would maintain the use of linked list queues, and would not affect the running of the simulation. The output would then be deterministic based on the input file. A second method of editing the assignment would be providing a seed to be used for the students random number generator. However, this would only be a viable solution if you could define exactly how the student would use the random numbers they generate. Both methods remove the utility of students learning to create and incorporate random numbers. Note that while grading the output after either of these changes, due to comparison of floating point numbers, an acceptable range would have to be used for answers, rather than expecting output to be exactly correct.

Instead of changing the assignment, the current assignment could be autograded by running it many times, and storing the metrics the student outputs. These metrics could then be analyzed to see if they fit a normal distribution around the expected result. Students with appropriately randomized data points would have different outputs each time, and by aggregating a large set of their outputs a grading script could look for expected trends, averages, and deviations. While incorporating these methods into

our autograder would be simple, developing the correct trends to look for may not be. We do not further explore this option in this paper.

3.3.5 Assignment 5: Operator Overloading

The fifth assignment [Appendix 11] is autogradable by comparing output without significant changes to the assignment. Input and output need to be exactly defined. Input files could be created by inserting random numbers into a preformatted file with preset operators.

3.3.6 Assignment 6: Polymorphism

The sixth assignment [Appendix 12] would not be autogradable by comparing output, because random movement is integral to the development of the program. It could instead be autograded by checking if each step of output is a legitimate child of the parent step. Expected formatting of output would have to be defined well enough for the grading program to use it as input, and each ant and doodlebug would have to be represented differently by the student output (A1, A2, A3, D1, D2 etc). Student programs would be required to be runnable in either visual mode where the output is printed to screen as a gameboard, or grading mode where the output is an array of information. A list of rules would need to be developed and checked against each step. Our suggested rules are as follows:

1: Each doodlebug, if adjacent to 1 or more ants, ate an ant, unless said ant was eaten by a different doodlebug

2: Each doodlebug moved unless there was no space

3: Each doodlebug that hasn't eaten in 3 turns was removed

4: Each doodlebug that has eaten for 8 consecutive turns spawned a doodlebug unless there was no space

5: Each ant moved unless there was no space

6: Each ant that has survived 3 turns since breeding spawns an ant unless there was no space

Each ant and doodlebug would need to be tracked by the grading program as an entity, to track its lifespan and time since eating and / or breeding.

3.4 Adaptation of Local Client for Assignment 1: Calendar

In order for us to test the student program, we need to generate input. The inputs should be generated randomly so that every possible input has a chance to be tested. The random number generator of our local client is seeded with the current system time every time it is run. For Assignment 1, we need to generate random year numbers to test the student's program. Since the Gregorian Calendar was adopted in 1582, no previous years will be tested. Also, no year numbers with five or more digits will be chosen in order to achieve a consistent output format.

According to Gregorian calendar, every year number that is evenly divisible by 4 is a leap year, and hence should be treated differently from other years. The local client chooses 4 year numbers that are not multiples of 4. In addition, according to Gregorian calendar, every year number that is evenly divisible by 100 is not a leap year, even if it is evenly divisible by 4. The local client chooses 3 year numbers that are multiples of 4 but not multiples of 100. Also, according to Gregorian calendar, every year number that is evenly divisible by 400 is a leap year. Therefore, the local client chooses 2 year numbers that are multiples of 100 but not multiples of 400 and 1 year number that is a multiple of 400. In total, the local client generates 10 year numbers which contain both the common years and the leap years and other edge cases. By considering every type of year, we can make sure that the student program is tested thoroughly.

3.5 Testing Local Client for Assignment 1

During the creation process of the local client, the client was run with our standard solution against a modified version of the standard solution that would create incorrect output. This allowed us to test functionality of the scripts, but did not test ease of use of our final product.

During C-term, we worked with Professors Michael J. Ciaraldi and Therese M. Smith -- both of whom were teaching CS2303, providing our client for their students and TAs to use. We presented during their lectures briefly on the setup and use of the autograder. We collected student feedback on ease of use and helpfulness of running the client through a survey using qualtrics [Appendix 13].

4: Results

This section includes a report of the functional code created, the results of testing against student submissions, and a report of incomplete objectives.

4.1 Generic Autograder Client

We successfully created a generic autograder client that can be modified to work for the individual assignments. Our generic autograder client was able to generate test cases, compile and run student programs, collect the students' outputs and compare them with the standard output and give a score. Our autograder was able to signal errors in the student output and show comparisons between them and the correct output.

4.2 Assignment 1 Specific Client

We successfully modified our generic autograder client to make it specific to assignment 1. We were able to generate test cases that were suitable for assignment 1 and write a standard solution for this assignment. We also modified our generic autograder client so that it could detect student errors that were specific to assignment 1 such as not printing "CALENDAR\n". Our assignment 1 specific client autogrades output satisfactorily.

4.3 Assignment 1 Client Testing

The results of a survey given to students who used our autograder can be found in [Appendix 7]. Thirty four responses were collected electronically. As no questions were mandatory, and a few questions only appeared conditionally based on previous answers, not all questions had thirty-four responses. Twenty-three students replied to the prompt "How many times did you utilize our autograder? (Running it repeatedly without changing your code would count as utilizing the autograder once)" with "2-10 times". Twenty-four students agreed "The autograder helped me know my completed project was correct". Twenty students agreed "The autograder helped me notice problems with my program's output". Twenty-

three students responded affirmatively to “Did you have any difficulty using the autograder?”, however sixteen replied “Some, but I resolved it without help from grading staff” and a further two replied “Some, but I resolved it with help from grading staff”. Ten students reported no trouble using the autograder; five students reported “Yes, it was difficult to use”. Two issues were mentioned repeatedly as a response to “What difficulty did you have with the autograder?”; seven students mentioned difficulty deciphering the comparison output, and four students had trouble correctly placing the files to run the autograder. Anecdotally we found that many of the problems resolved by grading staff and peers pertain to placing the file correctly or misnaming their files -- contrary to assignment instructions.

We did not collect meaningful feedback from grading staff.

4.4 Incomplete Objectives

Our main objective for this project was to have a functional autograder for the first assignment in time for testing. Although that was completed, several other objectives were considered that we did not complete.

We did not create a server-side client. At the beginning of the project, we proposed having students submit to a server, running our code on the server, and returning their results. This would have added a layer of obscurity to keep students from figuring out to hardcode responses to our tests. We decided instead to use randomized testing, and include only a compiled version of the correct code for their assignment. By having the software be exclusively client side, we avoided large server loads and the need for integration with an LMS.

Due to our client-side choice, we did not attempt to integrate with InstrustAssist, a Learning Management System created by Professor Craig A. Shue for use at WPI. While this integration could be useful for collecting student submissions, it was deemed to be low utility for several reasons. Primarily, the autograder outputs a number of test cases passed, NOT a final grade. Since the TAs will be involved with grading regardless, the integration would not save as much time as if it stored grades for students

directly. In addition, this was considered a reach goal, not the core of our project. We discarded it fairly early, but it should be a consideration for a continuation project.

A more abstract goal we discarded was the creation of a new initial project to teach basic syntax. We felt it would be a useful addition to the course to have a primary project based on minor coding problems, and theorized creating a website on which to host this project. This turned out to not be core to our project, and was not explored further.

The final reach goal we did not accomplish was the adaptation of the autograder for additional assignments. We instead include our recommendations on how to finish this.

5 Conclusions

This section includes analysis of the code created, our thoughts on its usefulness, and an analysis of the testing.

5.1 Generic Autograder Client

We think that our generic autograder client is a satisfactory starting point for an autograder for other assignments. We expect other MQP teams or TAs to build their assignment specific autograder upon our generic autograder client without having to do many edits. Since our autograder template is not specific to the C programming language, we also expect graders from other CS courses to be able to use our autograder.

5.2 Assignment 1 Specific Client

We implemented all the necessary features of a working Assignment 1 specific client. It worked well during our testing with the students. Further features like a more sophisticated comparison function can be developed, but overall the quality of our Assignment 1 client meets our expectations. It can be handed out to students with no major issues.

5.3 Assignment 1 Client Testing

Overwhelmingly the feedback from students was positive. Students using our script more than once shows they found it useful enough to incorporate into their testing process. In addition, the majority of students found it helpful in confirming their project was completed, and more than half of students found it helpful in finding problems with their output. While only six of the surveyed students found it useful in identifying where in their own code the problem occured, that still represents nearly one third of the students who found problems with their output using our autograder.

The number of students who initially had trouble with our autograder was higher than we hoped for, and we have identified several items to add to our documentation. However, only fifteen percent of

students found the autograder consistently difficult to use, as the majority of students resolved their issues without needing help from grading staff. Considering our short timeline and the limited documentation we presented to students, we found this to be a very successful trial run of our project. Most notable, this test did not uncover any significant bugs in our code. While our code did not always correctly handle user error, it did work correctly every time it was used correctly. Given our lack of user testing prior to the course, this was another big success.

6 Further Work

This section will include our thoughts on the possible routes for continuation of our project.

6.1 Completion of Client for Remaining Assignments

Our recommended edits for the remaining assignments are shown in section [3.3 Editing Given Assignments for Compatibility with Grading Scripts]. Assignments 2, 3 and 5 would be autogradable based on output, building off our generic client. The 4th assignment can be autograded on output if randomness is removed from the assignment. We recommend instead storing many test cases and looking for appropriate mathematical trends. The 6th assignment can be autograded by tracking entities.

6.2 Integration with InstructAssist (or other LMS)

To integrate our project with an LMS, a server side client would need to be adapted from our scripts. We would recommend using InstructAssist, as the source code is available to be worked on with Professor Craig A. Shue. He considers it a useful future addition to the site already and would likely be a helpful resource. We envisioned the server receiving student submissions from InstructAssist, unzipping files, running the autograder scripts, and returning a .csv or other file with feedback to InstructAssist to be viewed by the student. The most recent or most successful submission could be saved as a grade, or as part of a grade.

6.3 Increased Feedback for Students

The autograder outputs the results of ten test cases, spanning a variety of testing inputs. However, it does not clearly label why inputs were chosen, nor does it summarize which type of inputs were successful and which were not. For further assignments, more in depth feedback than simple output comparison should be considered, such as runtime feedback or testing error handling.

6.4 Post Testing Edits

Using the feedback given from students, we added an explanation of the comparison output to our README included with the autograder. However, we recommend in-person user testing to further understand the issues students have with reading output, and to confirm if the changes we made to the README are sufficient explanation. Several students also placed files incorrectly, which we did not address. In the README, it is explained where to place files, but the README needs to have a more clear explanation. Other user difficulties may be found through more extensive user testing.

Our autograder is most helpful when students have completed most of their assignment and need a way to verify their outputs. When we first designed our autograder, we tested it against perfect student outputs and near-perfect student outputs. Our autograder worked well in those cases because there was no error in compiling and running the students' code. However, the student outputs are not always near-perfect and they can make formatting and usage mistakes contrary to assignment instructions.

One common mistake would be running the Python 3 script with Python 2. Since Python 2 is the default Python interpreter in Xubuntu, when the students run 'python Autograder.py', Python 2 is selected instead of Python 3. It is written clearly in the instructions that students should run our script with Python 3, but some students may still omit the version number. What can be done to improve this would be changing the Python script to check the Python version before running the actual grading code.

Another common mistake would be the students placing the autograder in the wrong directory. As a result, the autograder would not find the necessary file it needs to run. One solution to this problem is to warn the students that certain files are missing. Although it is stated clearly in the instructions about where to place the autograder, it could help students to tell them again the correct directory to put our autograder in if the autograder fails to find the necessary files.

Our autograder is encapsulated in a .zip file, which contains the main program Autograder.py and other supporting files. One student reported that he received an error of missing files, and it turned out that he did not copy all the files. It could help prevent the students from making such a mistake if the autograder checks that all the necessary files are present.

Our autograder looks for a string “CALENDAR\n” in the student output and ignores all the characters before it. However, in the early versions of our design, our autograder raises an error if no such string is found in the student output. Printing “CALENDAR\n” is critical for our autograder to run, so it has been emphasized both in the assignment instructions and our autograder readme. Nevertheless, some students may still fail to print it probably because they have failed their own testing. In the final version of out script, the autograder warns the student that “CALENDAR\n” is missing instead of treating the student outputs as completely.

7. Bibliography

[1] “Lab Assignments,” *CS:APP2e*. [Online]. Available: http://csapp.cs.cmu.edu/2e/labs.html. [Accessed: 14-Mar-2019].

[2] “GeeksforGeeks | A computer science portal for geeks,” *IDE*. [Online]. Available: https://ide.geeksforgeeks.org/. [Accessed: 14-Mar-2019].

[3] “Ideone.com,” *Ideone.com*. [Online]. Available: https://ideone.com/. [Accessed: 14-Mar-2019].

[4] “Scalable Autograder and LMS Integration,” *Procedia Technology*, 29-Jan-2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212017313003617. [Accessed: 14-Mar-2019].

[5] “Providing Meaningful Feedback for Autograding of Programming Assignments,” *SIGCSE Technical Symposia*, Feb-2018. [Online]. Available: https://www.cs.rutgers.edu/~tdnguyen/pubs/haldeman.sigcse.2018.pdf. [Accessed: 14-Mar-2019].

A1: Assignment 1 Instruction Edits

The following changes are suggested to the instructions for assignment 1:

Edits:

> ~~Graders~~ The downloadable client will compile your assignment by executing the following command on an Ubuntu Linux system compatible with your course virtual machine:–
>
> gcc –Wall –o PA1 PA1.c
>
> Your program must compile without errors in order to receive any credit for this assignment. You must not use any extra switches — for example, -ansi or -std=C99. These will cause incompatibilities that will cause it to fail to compile for the graders. If you do your work on some other system, you may find that your system adheres to a slightly different standard and that some details of the C language may be different from those on the course virtual machine. Before submitting your assignment, be sure that it compiles on the ~~course virtual and correct it if it does not.~~downloadable client.

Addition:

> Please note that your project will be graded by a script directly comparing your output to the correct output. Therefore it is essential that you follow the above formating precisely; this includes capitalization, spelling, and whitespace. You are encouraged to copy-paste out of this document to ensure you have the correct headers. The grading script will output for you where your formatting mistakes are, so be sure to test your code against the grading script yourself: waiting for a TA to run the scripts may result in an unexpectedly low score due to minor formatting mistakes. TAs will only inspect your code directly for grading of comments and code readability.

A2: Informal Interview: Craig Shue; Lead Designer of InstructAssist - September 21st, 2018

Question: What was your reason for creating InstructAssist?

>CAS authentication doesn't work with other LMS (learning management systems)

Planned Question (Not asked): How did it change as you created and edited it?

Planned Question (Not asked): What were the biggest challenges of creating it?

Question: What further functionality do you wish it had?

>It has an ~30 item long bug file (mostly mundane) to be dealt with

>Calendar feed doesn't work well, would be useful to professors and students

>Autograding would be great, has been suggested but not seriously looked into

>Plagiarism detection would help, currently professors handle it on their own

How possible would it be to integrate our project with InstructAssist?

>Create an event that through Php does a file transfer to the linux server (PHP form submission or other server side API)

Linux server

Server-side API

Respond to Instruct Assist through an API (unique identifier)

Synopsis:

InstructAssist is lightweight, and therefore easy to add to. Autograding would be useful, and efforts to do so would be supported by Professor Shue.

A3: Informal Interview: Ciaraldi; Professor teaching CS2303 - September 20th, 2018

Discussed object of our MQP

Question: What difficulties have you had with the grading of assignments for this course?

>Suggestion - TAs and SAs do all the grading. (Talk with them?)

>Getting things to run, someone doesn't use the right VM. The more strict the formatting the less problems occur. Sometimes multiple formats are considered correct, make sure you know which are accepted.

>Files left out, make sure everything is correctly turned in

>Flexibility of changing the assignment so old code is no longer a viable solution.

>Interactive projects: command line arguments

>Style and commenting

Planned Question (Not answered): What needs to be retained for new assignments?

Discussed direction so far

Asked for questions / reactions / input.

>Look into what testing software is available, open source / commercial

>Expect, Jenkins,

Question: What would your ideal final product look like?

>Grading rubrics

>Consider output for graders (excel?)

N.B. Beta testing in C-term possible

A4: MQP Project Plan 9/30/18

Note that we refer to a generic client and assignment specific clients. We intend to edit a local client for each student assignment. They may be combined into a single downloadable client later, but for the purposes of working on the project step by step it will be individual clients per assignment.

By the end of A-term: Autograder Template complete. Runs in a directory with student files. Check for appropriate files, runs make, runs program with functionality for arguments as necessary, pipes output to file or testing program.

Halfway through B term: Generic local client complete. Generates random inputs or test cases, makes and runs program (autograder template), compares output to expected output, outputs a student grade file.

End of B term: Possibly worked on instruct assist integration, definitely edited assignments as necessary to make them possible to autograde correctly. Most important B term deadline will be to have edited the generic local client to create one specifically for the first autograded assignment.

Halfway through C term: If other parts of the project are on time, we will create clients specific to the next few assignments at the start of C-term. This is a reach goal for the project, so we won't know how far into this we will get until other parts are completed. This may be replaced entirely be testing and bug fixing the rest of our project.

The rest of C term will be spent on finishing the paper.

A5: Gantt Chart Projected Timeline

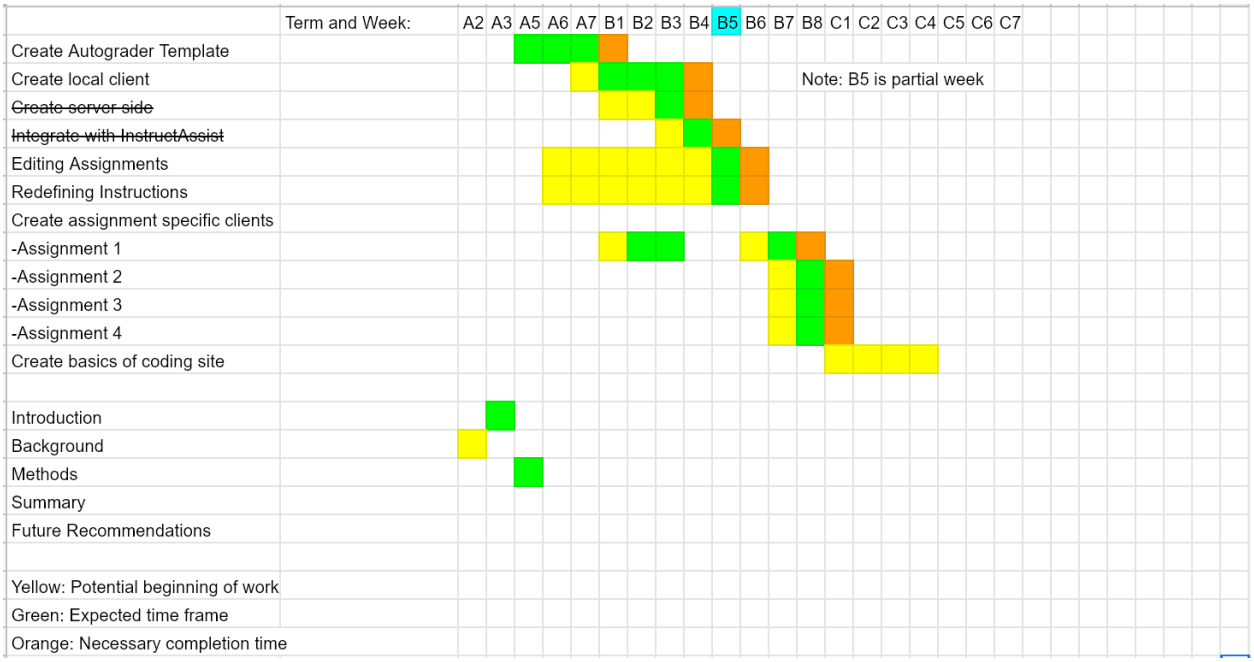


Figure 1: Projected Timeline

A6: README

CS2303 Calendar Autograder

This autograder is a tool designed for Student and TA use to aid in evaluating submissions for the Calendar assignment. When run, it creates a set of randomized test cases and compares the output of your program to the output of our solution. You may use this to identify where formatting errors occur in your output. Because it does exact character comparison, your formatting must exactly match that shown in your assignment pdf. Whitespace is included, and will be graded by our autograder and the grading staff.

Using the Autograder:

To run the autograder, your assignment must be named HW1.c, such that the compiled executable is named HW1. After unzipping the downloadable autograder file, move the Autograder folder into the same folder as the folder HW1, where your project is. Open terminal by right clicking on the Autograder folder and selecting 'open terminal here', then run '$ python3 Autograder.py'. Our autograder looks at all output following the string CALENDAR, which should be directly prior to your production code output, as shown in the example given in your assignment, and assumes it ends with 'Production seems to have worked.'

Understanding the output:

Our autograder runs your code against 10 random test cases. If your output doesn't match our standard output, our autograder will display the differences between your output and the expected output. A line preceded by a '-' will show the output of your code. A line preceded by a '+' will show the correct output. Characters that need to be added or deleted to correct your output will be marked with a '+' or '-' sign on a line preceded by a '?'. For example:

NB: For formatting purposes we switch to a monospace font to properly align characters. Viewing this in a .txt will not maintain proper spacing. Copy-pasting to a word editor and selecting ‘Consolas’ or another monospace font will regain proper formatting:

```
- 23    24   25    2   27   28   2        <- Your output
?      -          -                      <- Incorrect characters
+ 23   24   25   26   27   28   29       <- Correct Output
?                +              +        <- Missing characters
```

Grading:

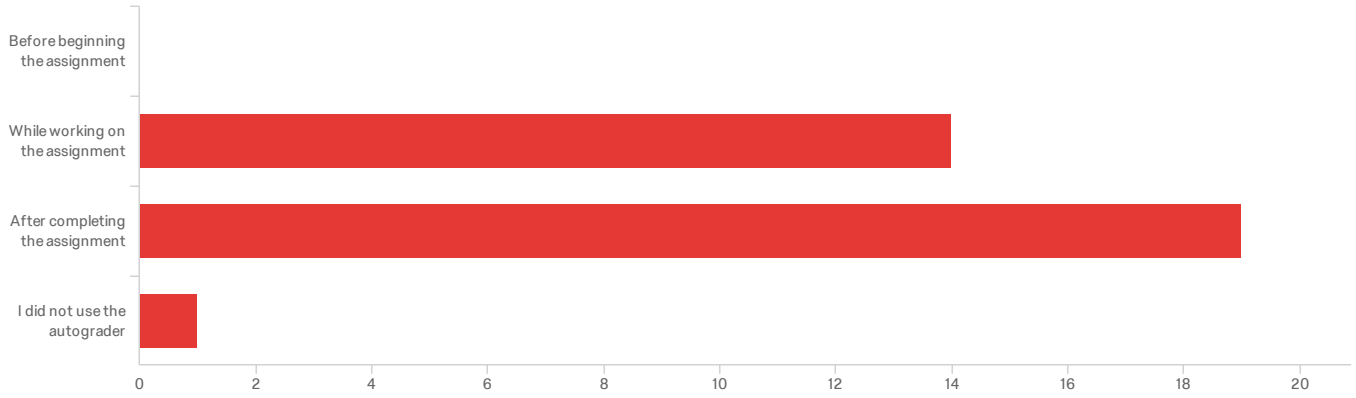
Keep in mind that the grading staff will be using the autograder as a tool to test your code, but will still be manually grading your assignments. That is, they will not directly use the score out of 10 that our autograder supplies, as our autograder does not provide partial credit.

Default Report

CS2303 Autograder MQP

March 15, 2019 10:02 AM MDT

Q1 - At what point did you first use our autograder?

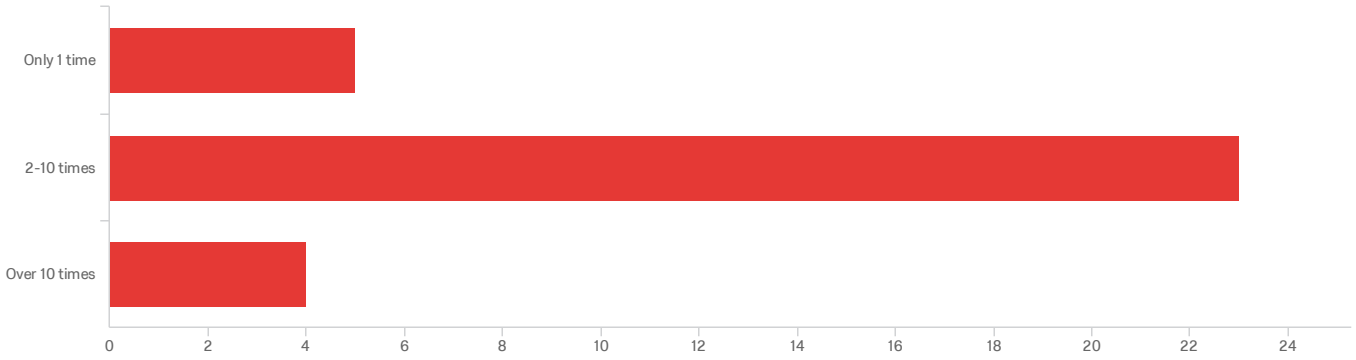


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	At what point did you first use our autograder?	2.00	4.00	2.62	0.54	0.29	34

#	Field		Choice Count
1	Before beginning the assignment	0.00%	0
2	While working on the assignment	41.18%	14
3	After completing the assignment	55.88%	19
4	I did not use the autograder	2.94%	1
			34

Showing rows 1 - 5 of 5

Q2 - How many times did you utilize our autograder? (Running it repeatedly without changing your code would count as utilizing the autograder once)

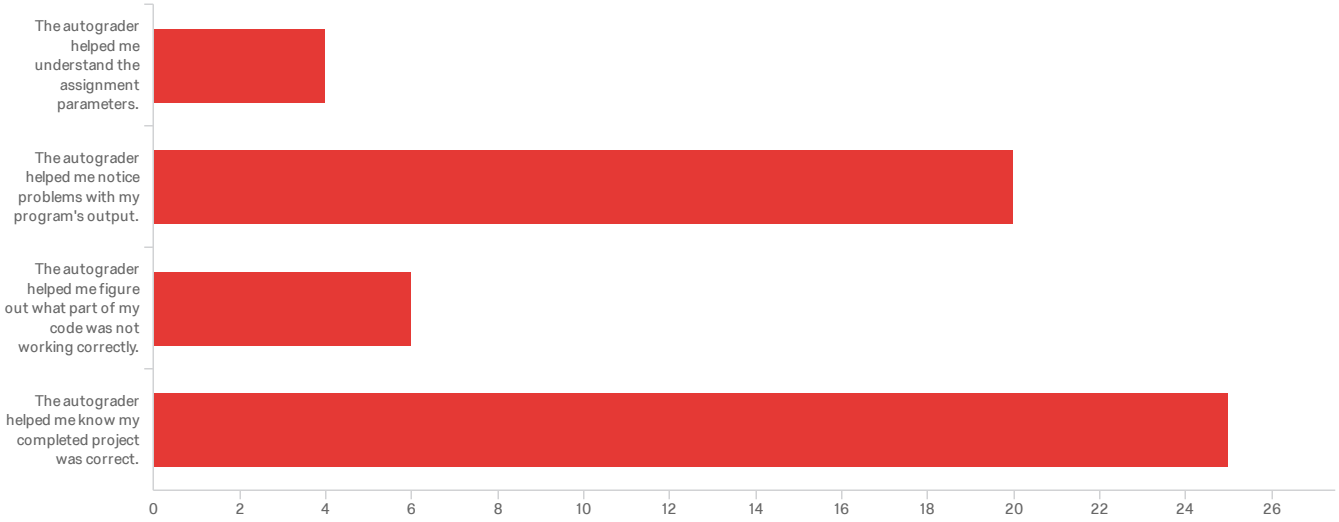


#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	How many times did you utilize our autograder? (Running it repeatedly without changing your code would count as utilizing the autograder once)	1.00	3.00	1.97	0.53	0.28	32

#	Field		Choice Count
1	Only 1 time	15.63%	5
2	2-10 times	71.88%	23
3	Over 10 times	12.50%	4
			32

Showing rows 1 - 4 of 4

Q3 - Which of the following do you feel apply?



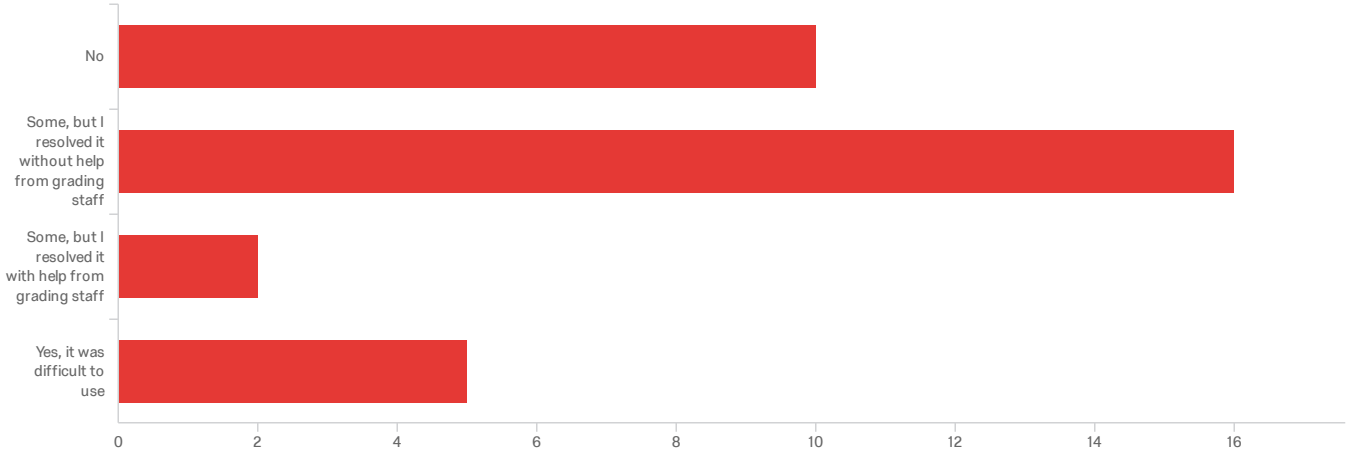
#	Field		Choice Count
1	The autograder helped me understand the assignment parameters.	7.27%	4
2	The autograder helped me notice problems with my program's output.	36.36%	20
3	The autograder helped me figure out what part of my code was not working correctly.	10.91%	6
4	The autograder helped me know my completed project was correct.	45.45%	25
			55

Showing rows 1 - 5 of 5

Q4 - Please describe other ways (if any) that you found the autograder helpful:

Please describe other ways (if any) that you found the autograder helpful:
Made sure it was formatted correctly
Easy to use
It was not very helpful but it was very difficult to use and understand. Running the autograder felt like playing a game of Mastermind instead of understanding what was "wrong" with my program output.
I ended up changing the script to point to a different file and it worked great
It made the exact formatting requirement of the assignment specific instead of still vague.

Q6 - Did you have any difficulty using the autograder?



#	Field	Minimum	Maximum	Mean	Std Deviation	Variance	Count
1	Did you have any difficulty using the autograder?	1.00	4.00	2.06	0.98	0.97	33

#	Field		Choice Count
1	No	30.30%	10
2	Some, but I resolved it without help from grading staff	48.48%	16
3	Some, but I resolved it with help from grading staff	6.06%	2
4	Yes, it was difficult to use	15.15%	5
			33

Showing rows 1 - 5 of 5

Q5 - What difficulty did you have with the autograder?

What difficulty did you have with the autograder?
I didn't understand what the _ _ ? symbols meant at first, which made it difficult to see what I needed to change about my code.
i wasn't sure if the problem was with my program or the autograder, because the program ran perfectly in eclipse but wouldn't build in the autograder
Originally my eclipse workspace had a space between the words eclipse and workspace, so the autograder could not find my file.
It was unclear how what the corrections were supposed to mean as it shipped with zero documentation.
The autograder was hard to understand and it felt like you had to play of a game of Mastermind every time you ran it to sort-of understand what the "problem" with your output was.
I was unsure of what some of the symbols meant when the code was running (+,-, etc)
No matter how many times I tried to match my output to the autograder one, it wouldn't work because of weird things with white spaces and such.
At first it was not finding my executable but I then realized it was because it was setup for eclipse directories so I changed it
The output was very strange and made it difficult to view the error
I was a bit confused on which folder to place the autograder to get it to work. I managed to solve it by opening up the python file and seeing what path it used to access the files.
Getting it to work properly
It was tough to place in the files, make the instructions better.(add pictures)
The notations were confusingly placed, which caused me to be unsure about what was what.
I just think that it would be more helpful if the autograder was more explicit in why the program was failing or at least a clearer depiction of what my program produced and what was expected.
I had trouble trying to start it, but after re-reading the instructions I figured it out.
no clear directions on how to use it

Q7 - Thank you for your input. Please add any other feedback below:

Thank you for your input. Please add any other feedback below:
Allow for input from command line to the HW.c file as an argument
Clean up the source code.
You should have it so the user confirms the file path before running the script
The autograder should really come with instructions or a read me. Once you understand the syntax it is pretty simple to use, but all the + and - signs are hard to understand at first.
O.o
The output had to match exactly which was too strict so I failed all of the tests but it must've only been off one " "

End of Report