Code Critiquer for C

FINAL REPORT

Team 34
Client: Dr. Ureel, Michigan Tech University
Advisor: Dr. Rover

Team Members:

Nicholas Carber	Regex Support
Conner Cook	AST Support
Brandon Ford	Database Admin
Emily Huisinga	Frontend
Sage Matt	Frontend
Cade Robison	Test Suite Support

Team Website:
sdmay24-34.sd.ece.iastate.edu

Team Email:
sdmay24-34@iastate.edu

Revised:
April 27th, 2024

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

Novice C programmers, specifically CPR E 288 students, need an easy-to-use tool to provide feedback on their C code and help them debug errors in their code. The C compiler alone does not always provide helpful feedback to programmers, especially novice programmers. This critiquer will use compiler feedback and conduct its own analysis of the code to help generate more detailed feedback. The code critiquer will also catch style errors that a normal compiler doesn't check for.

1.2 Intended Users and Uses

This project will benefit any novice programmers working in C, providing them with a user-friendly interface that will debug their code and check for antipatterns[1]. More specifically, this will be implemented in Iowa State's CPR E 288 class to help those students write better C code. Michigan Tech will also eventually implement this regex[2] alongside the other code critiquers they have written or are working on.

1.3 Related Products and Literature

We were extremely fortunate to have contact with Dr. Ureel and his team at Michigan Tech University, who have already created code critiquers in Python, MATLAB, and Java. They graciously allowed us to analyze their code critiquers to aid in our own development. Each of their code critiquers assisted us in different ways.

1. **Michigan Tech's Code Critiquer for Python:** Our Code Critiquer for C is written in Python. As such, MTU's Code Critiquer for Python, also written in Python, provided a good reference for us when initially creating our project.

2. **Michigan Tech's Code Critiquer for MATLAB:** MTU demoed their Code Critiquer for MATLAB to us, which gave us an idea of how to frame our UI, as well as how to get started with regexes and XPaths[3].

1.3.1 Related Papers

[1] Ureel, L. C., Brown, L., Sticklen, J., Jarvie-Eggart, M. E., and Benjamin, M. , "Work in Progress: The RICA Project: Rich, Immediate Critique of Antipatterns in Student Code," in *Proceedings of the 6th Educational Data Mining in Computer Science Education (CSEDM)*, 2022, 75-81. http://doi.org/10.5281/zenodo.6983498

[2] Ureel, L. C., and Wallace, C. , "Automated Critique of Early Programming Antipatterns," in *SIGCSE '19: Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2022, 738-744. http://doi.org/10.1145/3287324.3287463

[1] Antipattern: Poor solutions to common programming problems
[2] Regex: A sequence of characters used to find patterns in text
[3] XPath: Expression language for searching XML

2 Revised Design

2.1 Requirements

Functional Requirements

- Files in C upload successfully
- The program compiles the uploaded code
- Provide proper feedback for at least five regex antipatterns
- Provide proper feedback for at least three XPath antipatterns
- Runs code against uploaded test suites to check for runtime errors and check output against expected results
- The program communicates with a database that stores the C antipatterns
- The program should run through a GUI
- The code analysis process takes less than thirty seconds or returns an error
- The file upload is almost instantaneous and has no loss of data

Non-Functional Requirements

- GUI is simplistic and easy to use for novice programmers
- All feedback for errors is presented in a way novice programmers can understand
- The database is easy to update with new antipatterns
- The project is of the same design and standard that Michigan Tech has developed for their other code critiquers, which are for different languages
- The project is well-documented and easy for another team to pick up when we have completed our part

(Refer to section 3 for implementation details)

2.2 Engineering Standards

1. IEEE 12207-1996: Standards for software lifecycle processes.

2. IEEE 2675-2021: Standards for DevOps and deploying applications.

3. IEEE Standard for Standard SystemC® Language Reference Manual: This was used for the creation of the test files, as well as the creation of the antipatterns.

4. IETF RFC 9110: Standards for HTTP/HTTPS communication.

5. PEP-8 Style Guide for Python Code: This was used for the Python code that makes up a majority of the Flask application.

2.3 Security Concerns and Countermeasures

1. Instructors or students could upload malicious code through test files or code

- Uploaded test files are run in a separate sandbox[4] environment to prevent any malicious code from affecting the actual codebase.
- File uploads are limited only to .c and .h files.

2. Instructors could use custom antipatterns to upload a malicious regex

- Custom antipatterns are specific to each instructor, so any malicious regex would only affect that specific instructor and their classes.

3. An attacker could use SQL injections to insert malicious code

- Variables are sanitized in order to avoid SQL injection.

2.4 Design evolution from 491

Throughout this semester, numerous changes have been made to the design to meet client needs and to deal with challenges that have arisen. Listed below are a few significant design changes:

1. We decided not to integrate with Canvas. Initially, we had intended to integrate our application with Canvas. However, after discussing with Michigan Tech University, which had implemented something similar, it became apparent that it was unnecessary to integrate with Canvas. This decision was made because students disliked the interface integration with Canvas.

2. We decided not to utilize a command line interface. Instead, the entire application is accessed through a GUI. This is primarily for ease of usability for users. A GUI is much smoother and easier to use than a command line interface, especially for novice programmers.

3. We decided to add a custom antipatterns table to the database. This is because there are key differences between antipatterns and custom antipatterns. For one, custom antipatterns do not currently require an "XPath" field. More importantly, custom antipatterns require an instructor id field to attach the antipattern to an instructor.

3 Implementation Details

3.1 Design

Student Code: The student will submit the code files. These can be either C files or header files. Makefiles are not required for code analysis. This fulfills the functional

[4] Sandbox: an isolated environment to run untrusted code safely

requirements of allowing a student to upload their code and the performance requirements of uploading the code in an acceptable amount of time. This is a vital aspect of the functional requirement that students should be able to upload code to the application.

Code Critiquer Application: This interface will allow students to interact with the system (upload code, see results, etc). This will help fulfill many of the functional requirements, such as students being able to upload code and receive feedback on errors. The students will also be able to see any of the tests their instructor has uploaded which have failed.

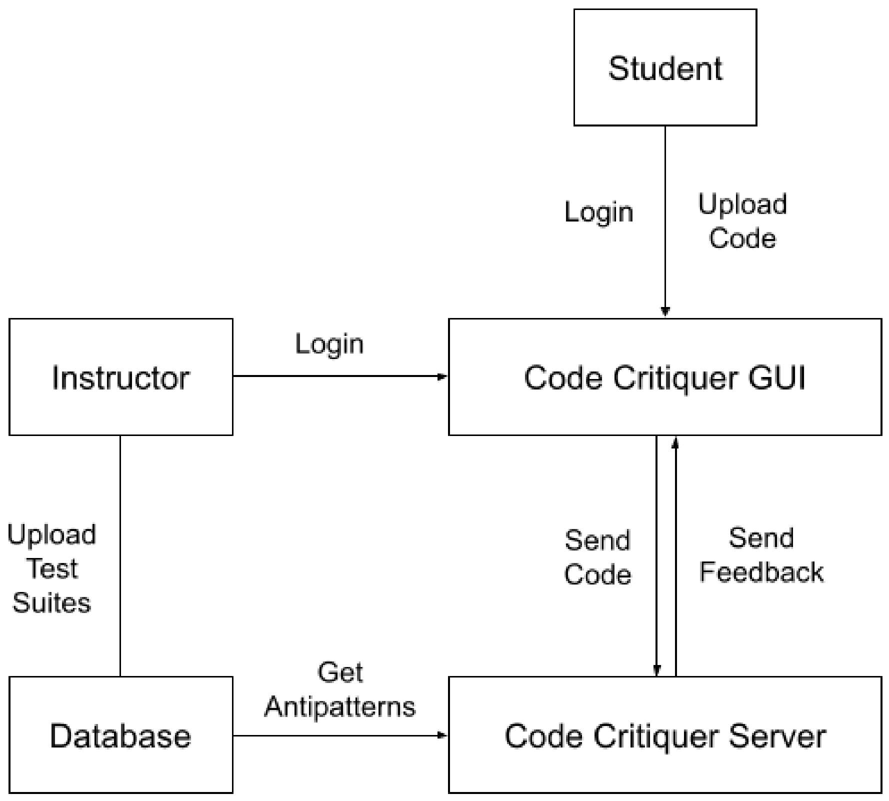


Figure 1 - System Sketch

Critiquer Server: This will consist of the code critiquer application, which houses the abstract syntax tree[5] builder, the regular expression matcher, and the test case runner.

a. **AST builder and XPath matcher:** The abstract syntax tree builder reads in the students' code and generates an abstract syntax tree from which an XML

[5]Abstract syntax tree: a tree representation of the syntactic structure of the code

format is constructed for the critiquer to analyze. The XPath matcher then takes that XML representation and applies all of the XPaths in the database to see if the code has any of the matching antipatterns

b. **Regex matcher:** The regex matcher will look at the database and match all regular expressions that are found in the student's uploaded code. These antipatterns, and their respective feedback, will be combined with the found XPath antipatterns and then shown to the student.
c. **Test case runner:** The test case runner will take the uploaded test files from the instructor and, using the files uploaded by the student, will compile and run these test files. The test files from the professor should be Unity test files specified in the Unity documentation (see Appendix D for documentation). These tests are then run in a sandboxed environment that does not have access to the source code of the project and has limited time and memory constraints as well as limited permissions. The failed tests and what the expected values are are then shown to the student.
d. **Feedback generator:** The feedback generator is the pipeline that combines both the matching antipatterns for regex and XPath and returns the corresponding feedback for each antipattern found.

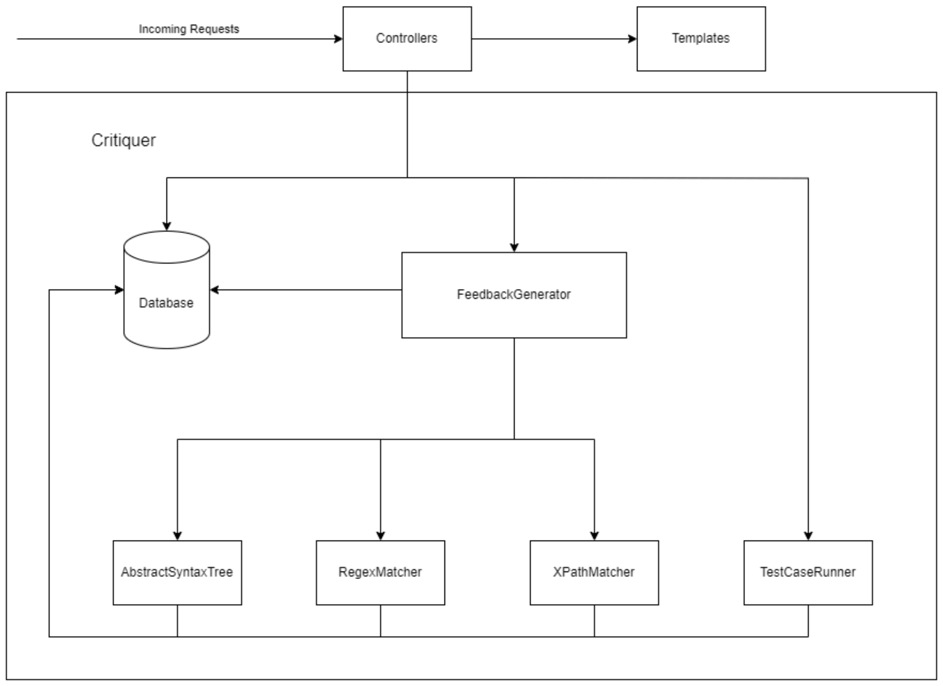


Figure 2 - Component Diagram

Database: The database, implemented using SQLite, holds all the information about the system, including all antipatterns and feedback, assignments, instructor account information, and assignment test files. This data is used to not only critique the students' code, but run and test the uploaded code. Instructors can also create custom antipatterns for their students. These are also stored in their own table on the database. Furthermore, the instructors' accounts are stored securely with the password hashed using bcrypt encryption.

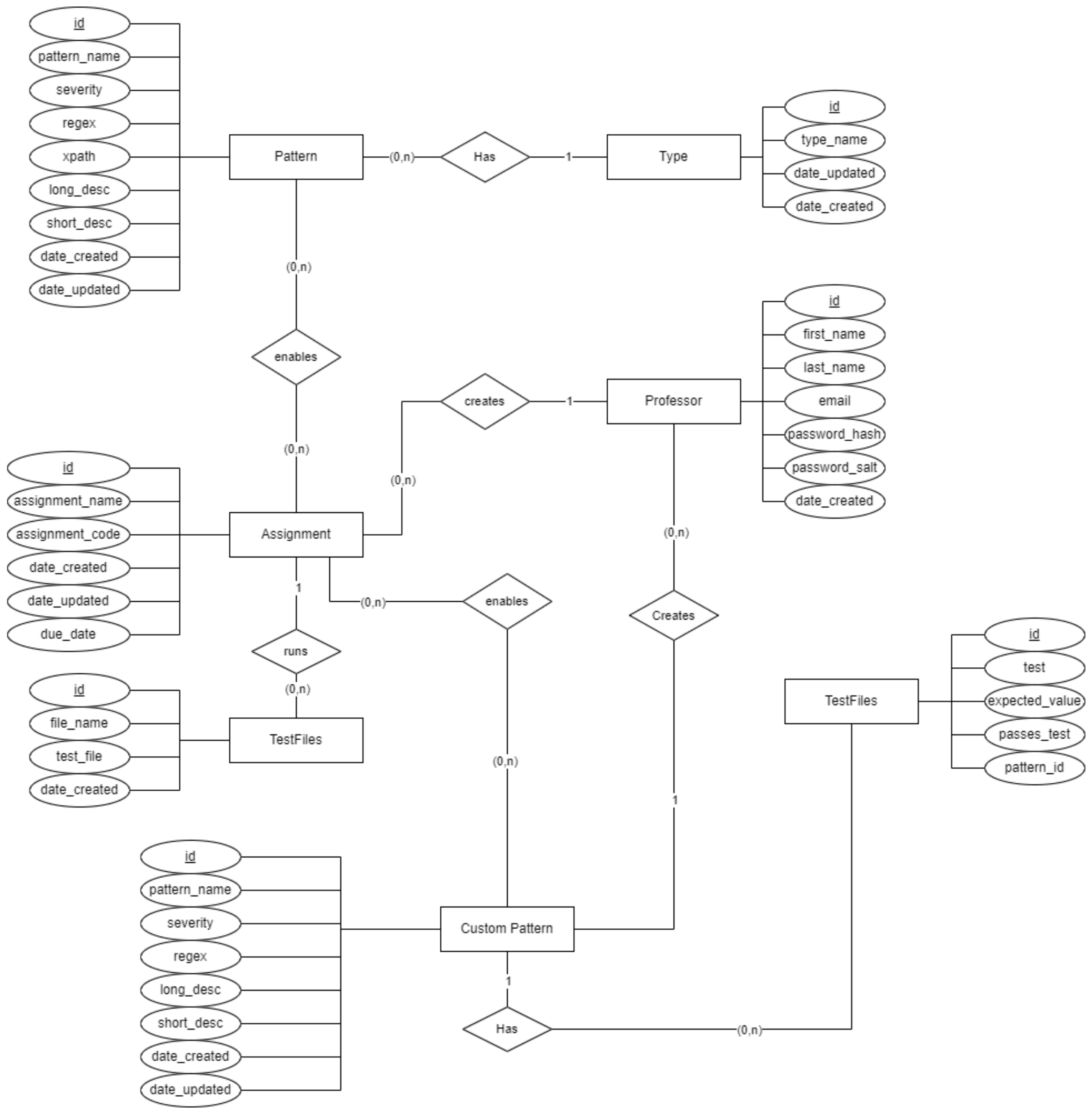


Figure 3 - ER Diagram

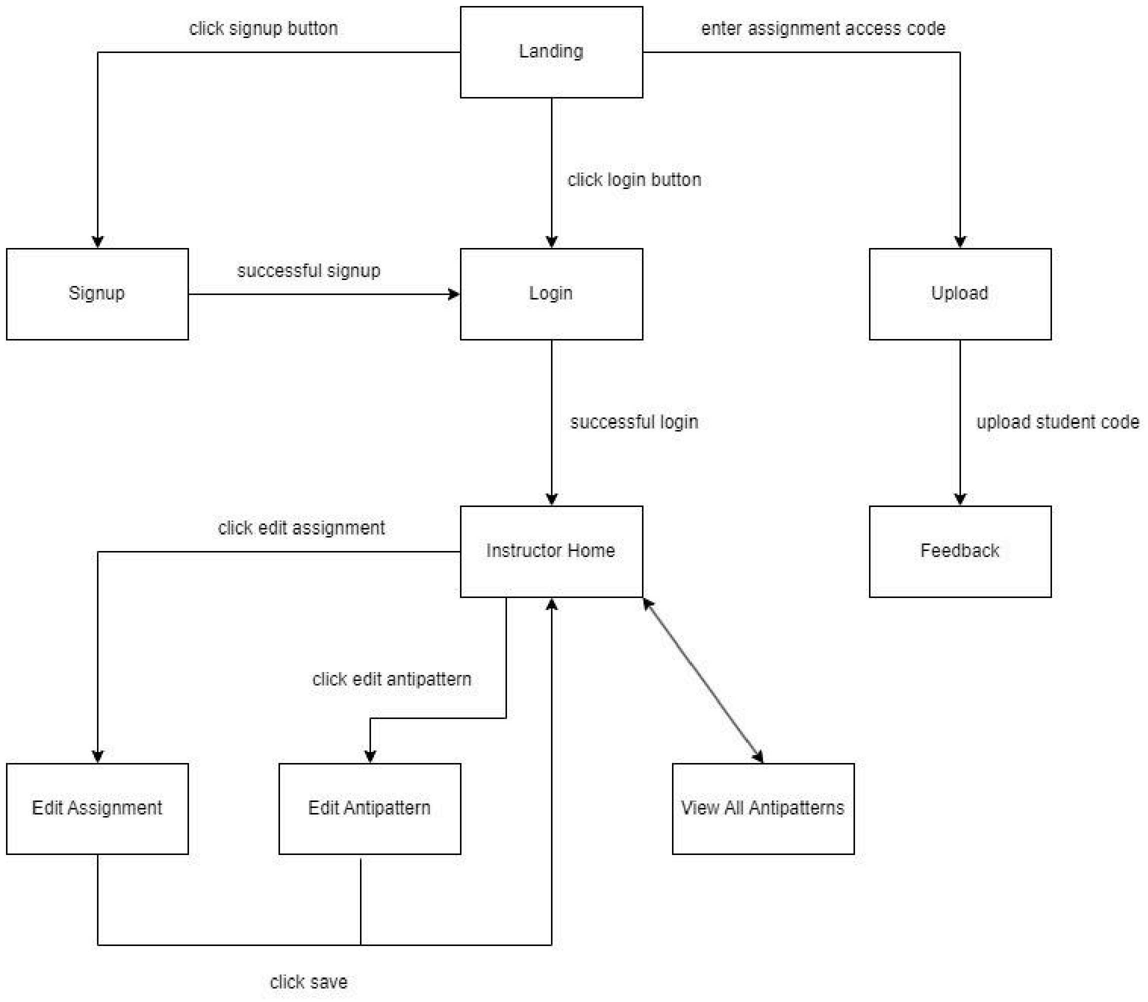


Figure 4 - UI Diagram

3.2 Functionality

The Code Critiquer for C is, first and foremost, an application where students can upload their work and receive feedback on it. To do this, students obtain an access code from their instructor, which they use to access the assignment from the landing page. This brings the students to a code upload page, where they are allowed to upload c files, header files, and Makefiles.

From here, students are brought to the code feedback page. In the backend, potential antipatterns are identified using a mix of regex patterns and XPaths. The backend also compiles and runs the student's code as part of the professor's uploaded test files, if there are any. On the feedback page, students are greeted with assignment details, a summary of found issues, a summary of instructor tests, and a detailed chart with found antipatterns. The table with found antipatterns is sorted by severity, includes file and line number, and includes the specific novice-geared critique.

On the landing page, instructors are greeted with one of two options: either create an account or log in to an existing account. Once they do so, instructors are brought to an instructor home page. On the instructor home page, instructors see two columns: one of the antipatterns they have created and one of the assignments they have created. In the antipatterns columns, instructors can edit existing custom antipatterns, create a new antipattern, or view all antipatterns (including non-custom antipatterns). The view-all-antipattern functionality lists details of existing antipatterns, including regexes and XPaths, which can be beneficial for instructors to look at when they are creating their own. In the assignments column, instructors can either edit an existing antipattern or create a new one. Additionally on this page, instructors can modify their account.

In the create/edit antipattern page, instructors can create their own antipatterns of varying severity with their own regex and description. Additionally, instructors can create test cases to test their newly created regexes.

In the create/edit assignment page, instructors can create their own assignments. On this page, a unique four-digit access code is created for each assignment for the instructor to provide to their students. Instructors can also upload their own test code to run with the students' code. These tests will be run when the students submit their code. Additionally, instructors can configure which antipatterns they want to include or not include.

4 Testing

4.1 Unit Testing

The framework we will be using to test our software will be the unittest library for Python. This should help us easily test all the functions needed for each class. We can ensure each component is working by breaking down our system into these units. Aiming for 100% coverage of the backend functions will ensure that future changes do not negatively affect previous code. At a minimum, a failed test will give a warning that old code and tests will need to be updated. One test was implemented for each developed antipattern.

4.2 Acceptance Testing

For acceptance testing, we acted as an end user to ensure all front-end navigation worked as intended. In the future, this testing should also be done with novice programmers to obtain valuable feedback so that the application can be improved. We should clearly be able to upload the test code that is being used by the unit tests and have the proper responses returned. Dr. Rover and Michigan Tech both provided assistance with front-end style suggestions to make the application easier to use and to look at.

4.3 Integration and Consistency

Writing tests for every component, even when it seems straightforward and unlikely to break, can help catch changes that break unexpected parts of the code. Keeping all tests can also help catch unexpected effects on old code. When writing code locally, ensuring that our local codebase is up to date (pulling from GitLab frequently) can help avoid merge conflicts. Running all tests before pushing code to GitLab can ensure breakages are found before being merged into the codebase. Running tests with a local copy of the database can also ensure changes don't break anything in the global database.

5 Broader Context

Code Critiquer for C adds another programming language to the critiquer toolbelt already being developed by MTU, which includes Java, Python, and MATLAB.

Code Critiquer for C is designed for programming learners to provide instant feedback on their code and improve their C programming skills. It also benefits instructors, as it can be used in tandem with coursework and save them valuable time that would be spent correcting simple errors.

Area	**Effects of Code Critiquer for C**
Public health, safety, and welfare	This project will greatly improve the mental health of students struggling to learn programming in C but feel that they have nowhere to go for help to learn or interpret cryptic compiler messages.
Global, cultural, and social	There are several ways that Code Critiquer for C can both positively and negatively impact our profession. It can be used to create a generation of programmers who have incredibly strong programming skills. On the other hand, if the Code Critiquer for C were to become good enough, future programmers may become lazy and rely on the critiquer to make corrections. It would be similar to how our generation has become increasingly poor spellers as we rely on autocorrect.
Environmental	This project will have a minimal environmental impact, except for the energy it takes to run the device on which the application will be run.
Economic	This project has the potential to inflate the market with programmers, as students will feel more supported during the learning process and, therefore, will be less likely to drop out. Additionally, if the Code Critiquer for C technology could become advanced enough, it could decrease the demand for programming tutors.

Table 1 - Effects of Code Critiquer for C

6 Conclusion

6.1 Progress Review

We have met all of our functional requirements set at the beginning of the year and we met our non-functional requirements to the best of our ability (design choices are subjective). The core application has been designed, developed, and well-documented for a future team to pick up and improve upon. We believe that this project can be used to help students in their journey to learn a difficult topic and hopefully make it a better experience for them. These are the primary features that we have implemented:

- Regex and XPath matching
- Test suite support
- Custom regex antipattern creation
- Assignment creation and student access
- Instructor account creation

6.2 Design Value

The current design provides the necessary functionality to address the problem concerning antipatterns within novice programmers' code. The design provides the basis for analyzing, testing, and feedback on students' code. A small set of default expressions exists to identify antipatterns with the ability for instructors to add their own. With this design, it is possible to extend the existing antipattern expressions to create better coverage of possible antipatterns within the code. The significant value of the design lies in the ability for instructors to add expressions to identify antipatterns, allowing them to customize the possible antipatterns found and have it be specific for an assignment. This customization allows fine control over the help the critiquer can have over a single assignment.

6.3 Potential Future Steps

Given limited design and development, different aspects deserve further attention and implementation. These aspects include additional regular expressions and XPaths to identify antipatterns, complete automated end-to-end application testing, preprocessing the code before antipattern identification, returning feedback when an error occurs with running testing files, the ability for instructors to share expressions to identify antipatterns, and providing custom feedback for default feedback.

Antipattern identification is the backbone of the problem being addressed. Creating more regular expressions and XPaths allows more coverage of the number of antipatterns discovered within a student code. Currently, there is a limited number of default expressions, but this should be expanded to reduce the demand for instructors to develop their antipatterns.

Complete automated end-to-end testing provides a sense of sanity that the application functions in a desirable manner. By developing these styles of tests, future development can be ensured that future implementation does not jeopardize the integrity of the system.

Preprocessing the code before antipattern identification allows better identification of antipatterns. By doing such, comments are removed, macros are applied, and header information is inserted. If a comment contains antipatterns, the regular expression will identify them by removing them. This concern is removed. Macros provide a way for developers to short-hand coding that may remove antipatterns previously there. For example, a macro could replace true with a value of 1. Since true is not default in C, it is identified as an antipattern, but by applying the preprocessor the antipattern gets removed. Lastly, header information provides a way for function declaration to be at the top of the program, making it difficult to call a function before it is declared. Currently, the system cannot identify if a function is called before declared if that function is declared in the header file.

Feedback is an essential part of the code critiquing processes. Due to a lack of time, when errors occur when running test files, the feedback could be more helpful than currently reported. The ability to better create feedback when errors occur during running test files allows novice programmers to better understand antipatterns in their code.

Since different instructors can teach the same class and some antipatterns are uniform regardless of the context, sharing antipattern identification expressions saves time so that other instructors do not have to implement the same expression themselves.

Lastly, the current implementation does not allow instructors an easy way to override the feedback provided by a default implemented antipattern. By allowing instructors to override the feedback, they can provide feedback more directed toward their students concerning the class.

Appendices

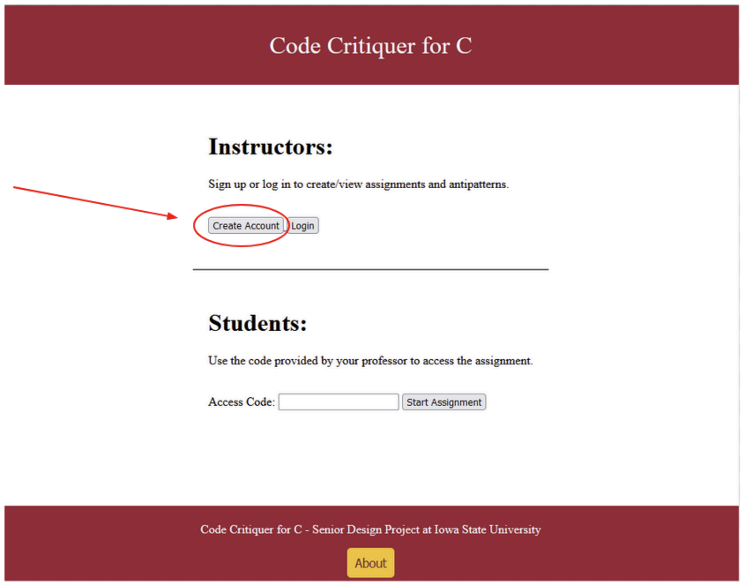
A. Operation Manual

Setup

1. Download project code
2. Create .env file
    a. Create .env file in root directory of project
    b. Run import secrets and secrets.token_hex() in a python shell
    c. Enter the following lines into the file (don't include the square brackets in secret key line):
        i. ENVIRONMENT="qa"
        FLASK_SECRET_KEY=[secrets.token_hex() result here]
        FLASK_DATABASE=sqlite
3. Install project requirements: python -m pip install -r requirements.txt
4. Initialize database: python -m flask --app . init-db
5. Start application: python -m flask --app . run --port 8000
6. Open localhost:8000 in browser

Account Creation / Login Demo (as instructor)

1. Click the "Create Account" button if you do not already have an account



Code Critiquer for C

Instructors:

Sign up or log in to create/view assignments and antipatterns.

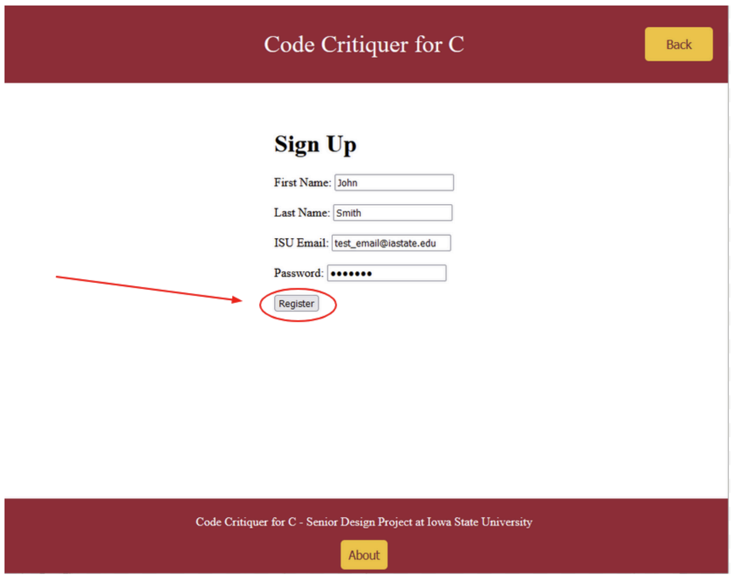
Students:

Use the code provided by your professor to access the assignment.

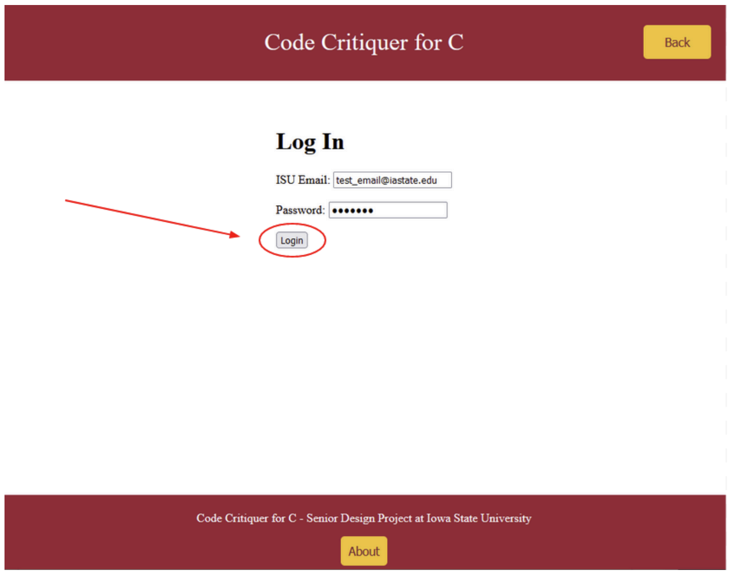
Code Critiquer for C - Senior Design Project at Iowa State University

    a. If you already have an account, click "Login" and skip to step three

2. Enter your information and click "Register"

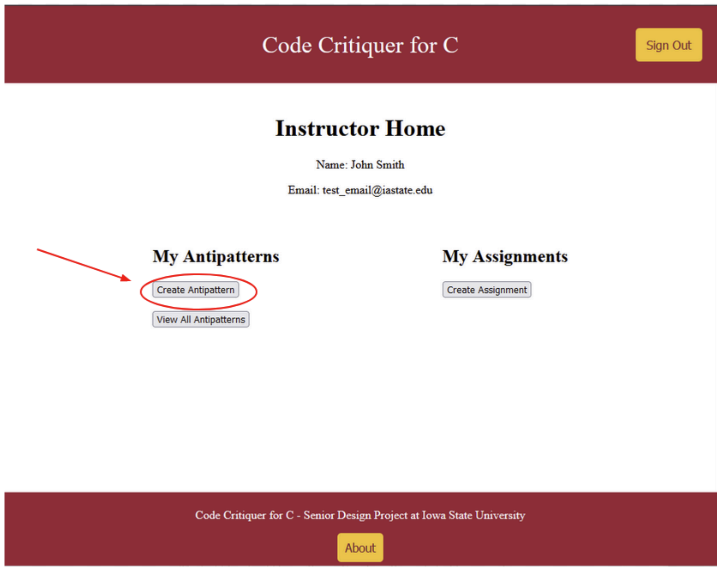

Code Critiquer for C
Back
Sign Up
First Name: John
Last Name: Smith
ISU Email: test_email@iastate.edu
Password: •••••••
Register
Code Critiquer for C - Senior Design Project at Iowa State University
About

3. Enter your email and password and click "Login" to go to the instructor home page


Code Critiquer for C
Back
Log In
ISU Email: test_email@iastate.edu
Password: •••••••
Login
Code Critiquer for C - Senior Design Project at Iowa State University
About

Custom Antipattern Creation Demo (as instructor)

1. From the instructor home page, click "Create Antipattern" under the "My Antipatterns" section



Code Critiquer for C

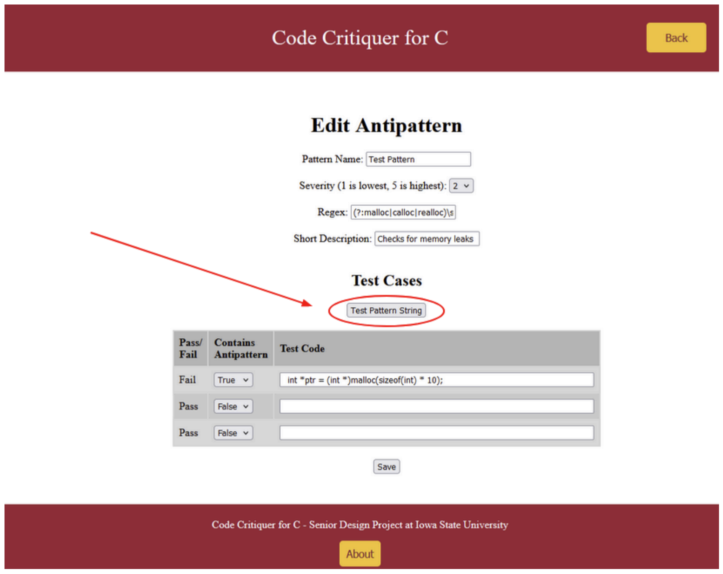
Instructor Home

Name: John Smith

Email: test_email@iastate.edu

Code Critiquer for C - Senior Design Project at Iowa State University

2. Enter antipattern information
3. You can additionally add test code to see if the regex matches under the "Test Cases" section



Code Critiquer for C

Edit Antipattern

Pattern Name: Test Pattern

Severity (1 is lowest, 5 is highest): 2

Regex: (?:malloc|calloc|realloc)\s

Short Description: Checks for memory leaks

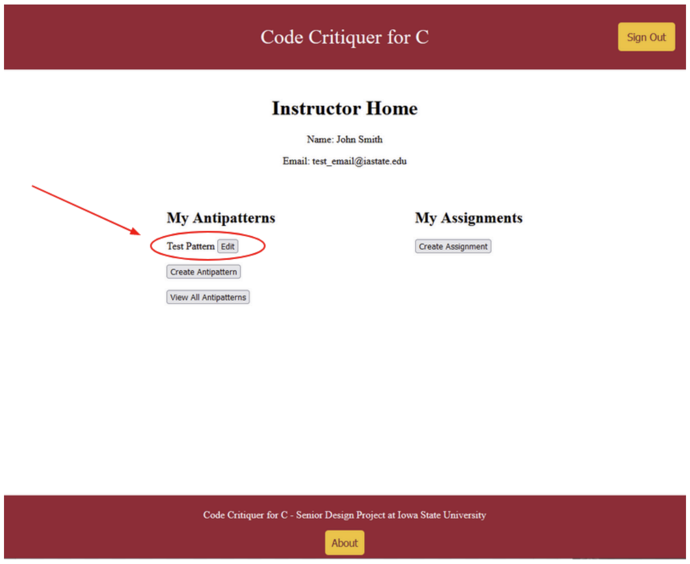
Test Cases

Pass/ Fail	Contains Antipattern	Test Code
Fail	True	int *ptr = (int *)malloc(sizeof(int) * 10);
Pass	False	
Pass	False	

Code Critiquer for C - Senior Design Project at Iowa State University

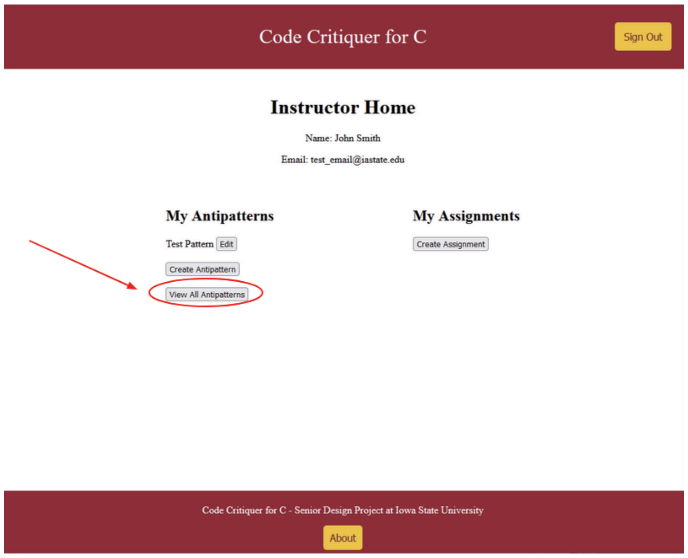
   a. Click "Test Pattern String" to run your regex expression on the test code and see the results
4. Click "Save" to save the custom antipattern

5. On the instructor home page, you should now see your new antipattern listed under the "My Antipattern" section



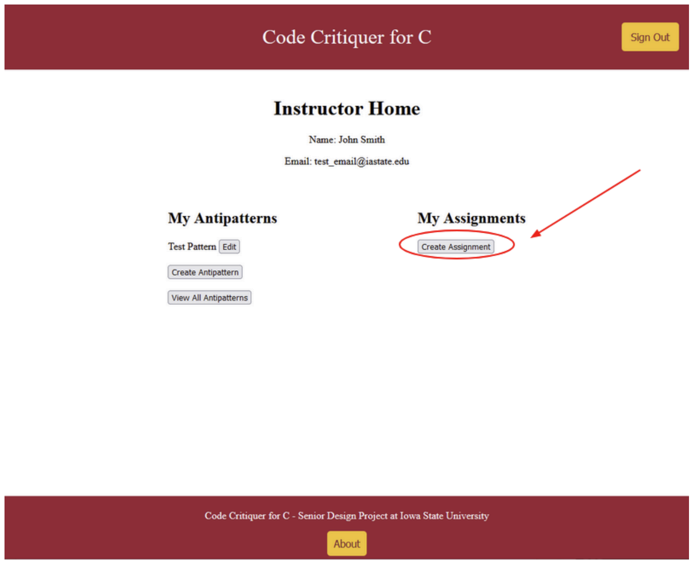
   a. You can edit this antipattern by clicking the "Edit" button next to it

6. To view all antipatterns in the database including your custom antipatterns, click the "View All Antipatterns" button

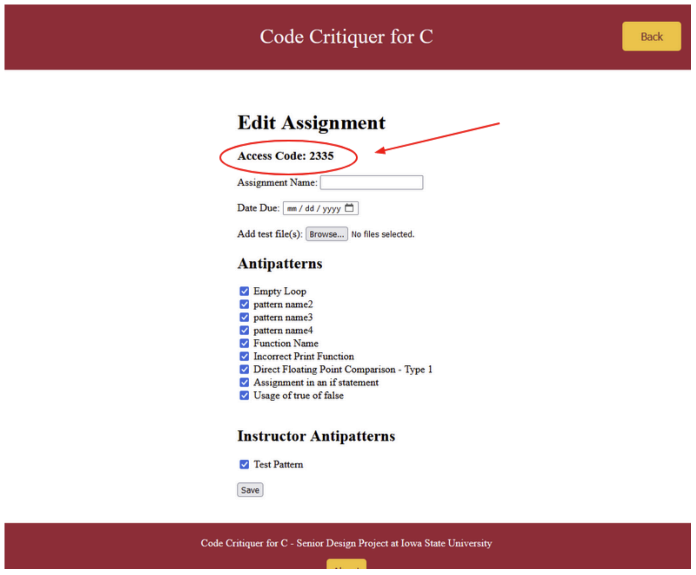


Assignment Creation Demo (as instructor)

1. From the instructor home page, click "Create Assignment" under the "My Assignments" section

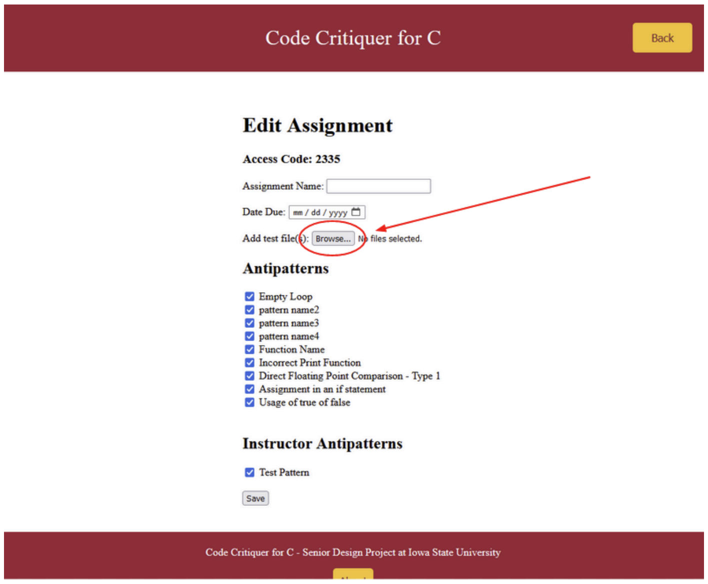


2. You can view the auto-generated access code at the top of the page



3. Enter assignment information

4. Add any test files you want to run with the students' code



Edit Assignment

Access Code: 2335

Assignment Name:

Date Due: mm / dd / yyyy

Add test file(s): Browse... No files selected.

Antipatterns

- [x] Empty Loop
- [x] pattern name2
- [x] pattern name3
- [x] pattern name4
- [x] Function Name
- [x] Incorrect Print Function
- [x] Direct Floating Point Comparison - Type 1
- [x] Assignment in an if statement
- [x] Usage of true of false

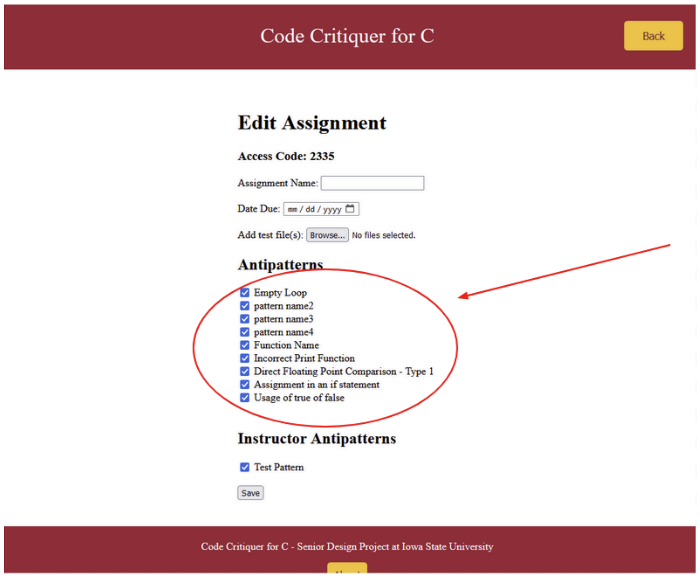
Instructor Antipatterns

- [x] Test Pattern

Save
Code Critiquer for C - Senior Design Project at Iowa State University

   a. The test files should be written in C and use the unity test framework

5. You can select which of the antipatterns (including any custom ones) you would like to be included in the feedback for this assignment



Edit Assignment

Access Code: 2335

Assignment Name:

Date Due: mm / dd / yyyy

Add test file(s): Browse... No files selected.

Antipatterns

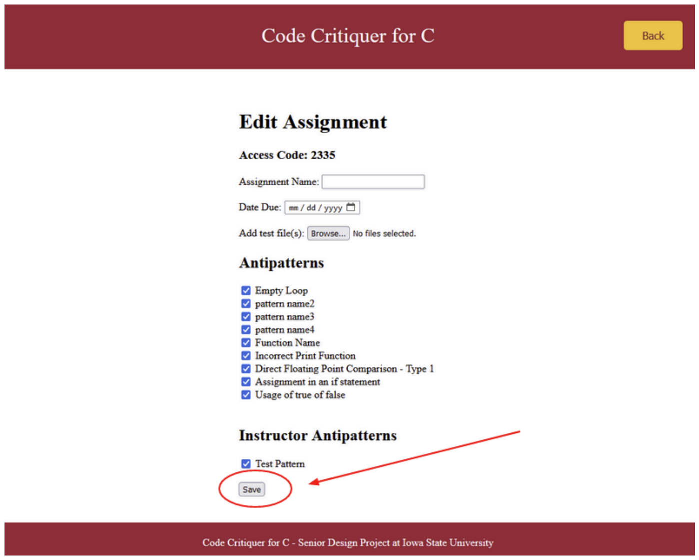
- [x] Empty Loop
- [x] pattern name2
- [x] pattern name3
- [x] pattern name4
- [x] Function Name
- [x] Incorrect Print Function
- [x] Direct Floating Point Comparison - Type 1
- [x] Assignment in an if statement
- [x] Usage of true of false

Instructor Antipatterns

- [x] Test Pattern

Save
Code Critiquer for C - Senior Design Project at Iowa State University

6. Click "Save" to save the assignment


Code Critiquer for C
Back

Edit Assignment

Access Code: 2335

Assignment Name:

Date Due: mm / dd / yyyy

Add test file(s): Browse... No files selected.

Antipatterns

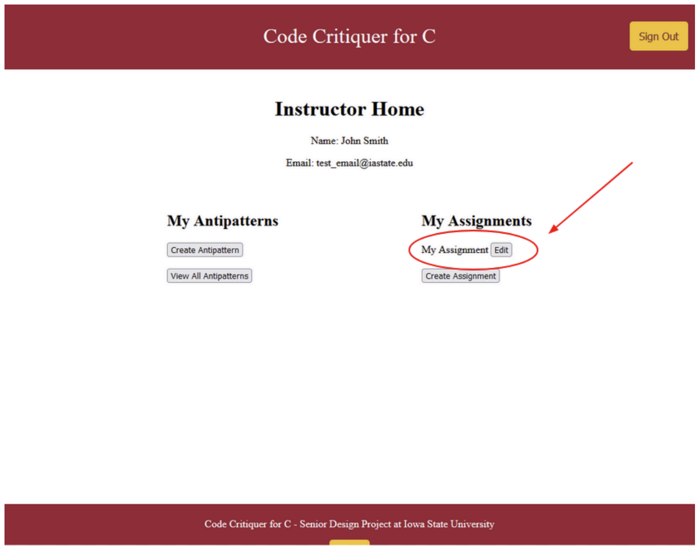
- ✓ Empty Loop
- ✓ pattern name2
- ✓ pattern name3
- ✓ pattern name4
- ✓ Function Name
- ✓ Incorrect Print Function
- ✓ Direct Floating Point Comparison - Type 1
- ✓ Assignment in an if statement
- ✓ Usage of true of false

Instructor Antipatterns

- ✓ Test Pattern

Save
Code Critiquer for C - Senior Design Project at Iowa State University

7. On the instructor home page you should now see your new assignment listed under the "My Assignment" section



Instructor Home

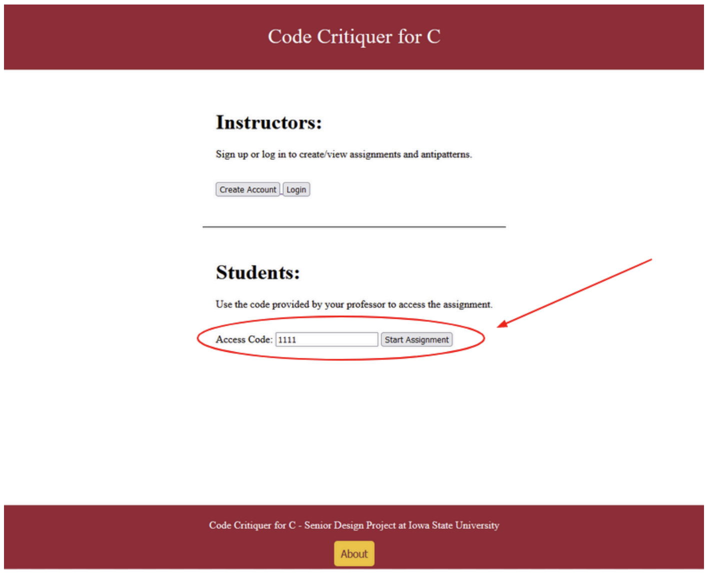
Name: John Smith

Email: test_email@iastate.edu

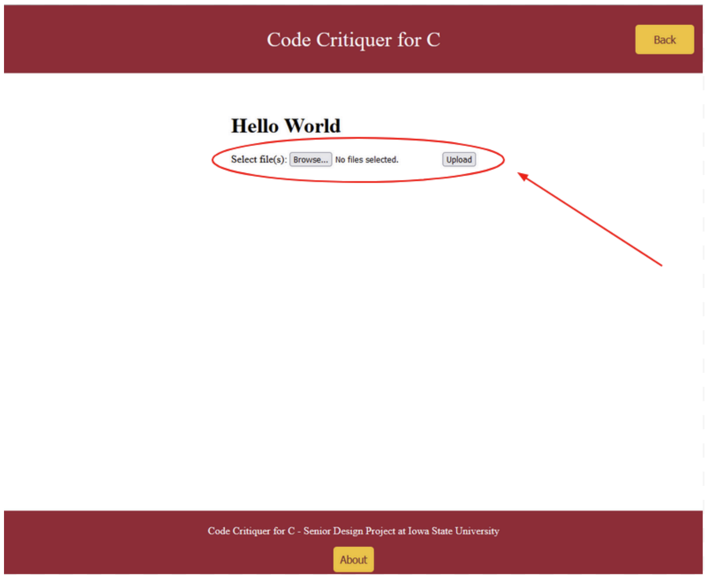
   a. You can edit this assignment by clicking the "Edit" button next to it

Retrieve Code Feedback Demo (as student)

1. Open application in browser by going to localhost:8000 or whatever address the application is running on
2. Type in your access code for assignment under the "Student" section and click "Start Assignment"


Code Critiquer for C
Instructors:
Sign up or log in to create/view assignments and antipatterns.
Create Account
Login
Students:
Use the code provided by your professor to access the assignment.
Access Code: 1111
Start Assignment
Code Critiquer for C - Senior Design Project at Iowa State University
About

3. Click "Browse…" and select C code files (either C source files or header files) you want to upload


Code Critiquer for C
Back
Hello World
Select file(s): Browse... No files selected.
Upload
Code Critiquer for C - Senior Design Project at Iowa State University
About

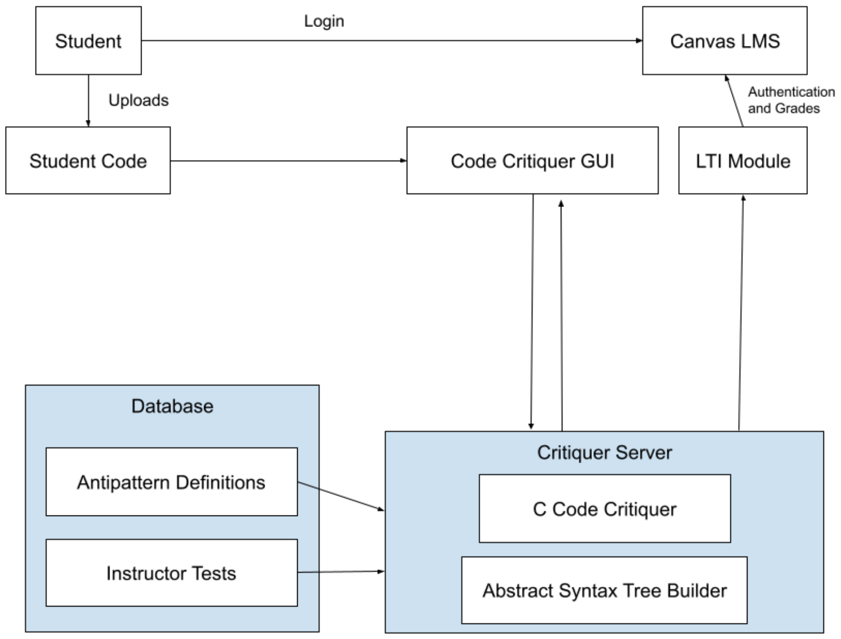
4. Click "Upload" to generate feedback
5. View feedback

Test

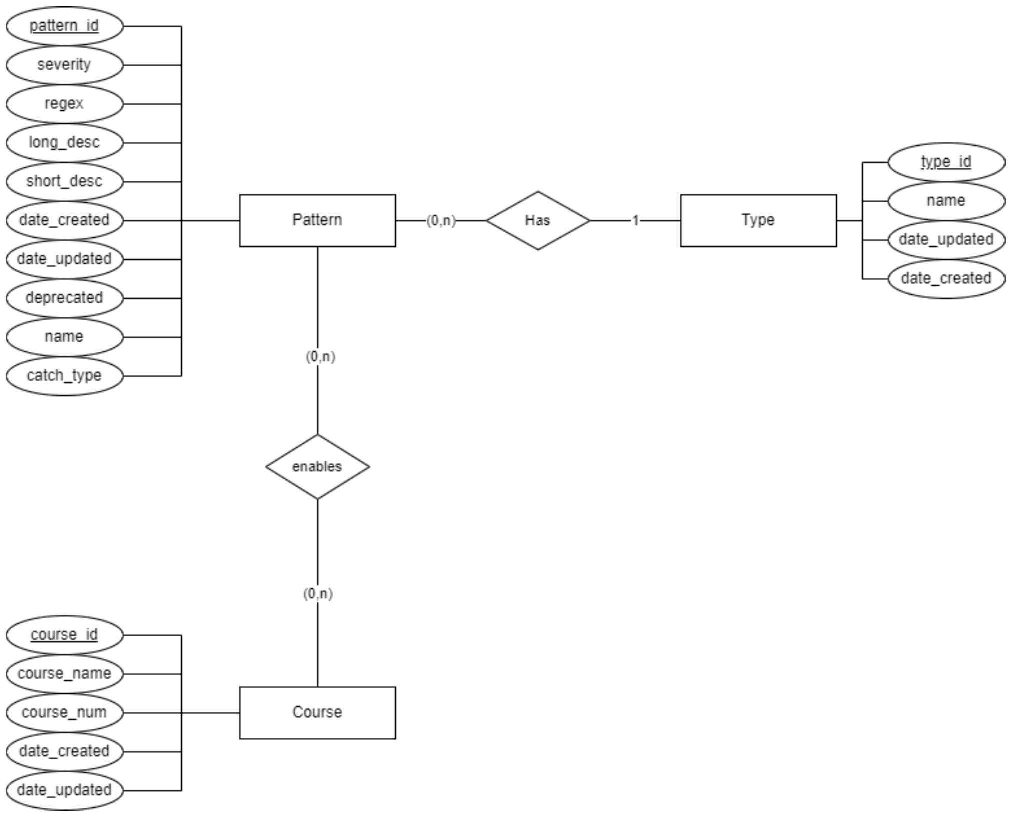
1. Run python -m unittest in project root directory to run all test suites

B. Initial Design

The Code Critiquer for C system consists of 3 main components: the website, the database, and the critiquer server. After the student logs in, they can upload their code to the website or canvas. From there, the code critiquer logic will be called from the server. This component will check the uploaded code against the antipatterns stored in the database. After finding all the errors in the student's code, the critiquer will return the feedback to the student through where the code was uploaded.



System Sketch V1



ER Diagram v1

Student Code: The student will submit the code files. These can be in the form of a single C file or a makefile for compiling multiple files. This should fulfill the functional requirements of allowing a student to upload their code and the performance requirements of uploading the code in an acceptable amount of time. This is a vital aspect of the functional requirement that students should be able to upload code to the application.

Code Critiquer for C Application: This interface will allow students to interact with the system (upload code, see results, etc). This will help fulfill many of the functional requirements, such as students being able to upload code and receive feedback on errors.

LTI Module: Connection point between Canvas and the Code Critiquer for C System. The LTI Module follows standard practice for connecting LMS and LTI systems. This will help fulfill the UI Requirement that the application will have to be run through a GUI (in this case, Canvas).

Canvas LMS: This is the Canvas everyone knows and loves here at Iowa State. Eventually, using the LTI Module, we will connect the application to Canvas for students to upload their code and receive feedback. This will fulfill the UI Requirement that the application will be run through a GUI (in this case, Canvas).

Critiquer Server: This will consist of the code critiquer application and the abstract syntax tree builder. The code critiquer will analyze the student's code using the anti-pattern data in the database and generate appropriate feedback based on any errors found. The abstract syntax tree builder reads the students' code and generates an abstract syntax tree for the critiquer to analyze. This will help fulfill the functional requirement that students will receive feedback from antipatterns stored on a database (which will be hosted on the server).

Database: The database will hold all the information for the antipatterns to check against the uploaded code. This is critical to the software's functionality as the system needs a place to store all the antipatterns. The database will also hold the tests the instructors upload for the particular assignment. This is also crucial to the requirement that the system needs to run the uploaded code. Without these tests, there is no way to run the code and have an expected output. This will help fulfill the functional requirement that students receive feedback from antipatterns stored in a database.

Functionality

The Code Critiquer for C will be, first and foremost, an application where students can upload their work. They can either upload a single file to get it critiqued or upload a project, which would require a makefile. Once the file or project is uploaded, they would click a very prominent "Critique" button and receive feedback. On the backend, the application would be hooked up to a database with antipatterns, and the program would scan the file or project for the presence of these antipatterns. The feedback would be similar to compiler feedback, except more specific and worded understandably by even programming novices. Eventually, the application would be hooked up to Canvas as an External App using LTI tools.

The current design satisfies the functional and non-functional requirements very well.

Reasons for Changes

- We removed Canvas integration after a discussion with MTU revealed that it was unnecessary, both due to complexity and previous negative student experiences with Canvas integration.
- We scrapped the command line interface, as a GUI is much more usable for users, particularly novice programmers.
- We added a custom antipattern table to the database to keep track of instructor antipatterns, as they are unique to each instructor and the antipatterns have different attributes.

C. Other Considerations

These past two semesters, our team has learned quite a bit about the following:

- How to build and deploy Flask applications, as well as how to integrate Flask with HTML and Python.
- How regexes and XPaths work and how to use them to detect different things within our code.
- How to identify bad habits within our own code while also figuring out how to detect those antipatterns.

As for something that entertained us during the last few weeks of this project as we finalized all of the details, we leave you with this:

D. Code

All code for this project can be found on our GitLab: https://git.ece.iastate.edu/sd/sdmay24-34

The unity source code and documentation can be found here: https://github.com/ThrowTheSwitch/Unity