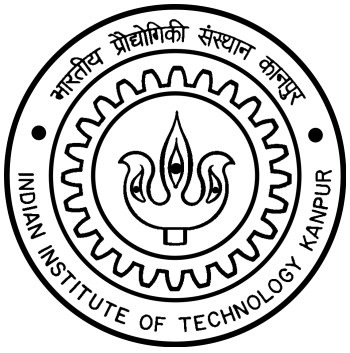
Automated Repair of Programs in Introductory Programming Courses

A thesis submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Praveen Kumar Singh
Roll Number: **14111023**



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

May, 2016

i

CERTIFICATE

It is certified that the work contained in the thesis titled **Automated Repair of Programs in Introductory Programming Courses**, by **Praveen Kumar Singh** (*Roll Number:* **14111023**), has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Amey Karkare

Dr. Amey Karkare
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Arnab Bhattacharya

Dr. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

May, 2016

ABSTRACT

Advancement in computer technology has made basic programming skills a necessity for everyone. This has led to almost all educational institutes to have introductory programming courses for all engineering students. Teaching hundreds of students and providing relevant feedback to them has become a very difficult task. This has motivated researchers to develop automated grading and feedback tools to help instructors to conduct programming courses effectively. Automated grading tools work well when the programs submitted for evaluation are compiling successfully, but fail badly with submissions containing compilation errors.

In this thesis, we will attempt to tackle the task of correcting non-compiling programs to syntactically correct programs. We will present an automated repair system which can be used to correct errors made by students in their submissions. The system will provide a web interface for integration with existing tools. The system can be used with automated grading tools to improve their performance and provide students with better and relevant feedback in a timely manner. The system also identifies students' learning patterns and topics with which students are constantly struggling. Such patterns can be used by instructors and students to identify areas which need more effort. This system thus serves as an automated repair tool which can be used with existing automated grading and feedback tools.

Dedicated to my family

Acknowledgements

I would like to thank my thesis supervisor **Dr. Amey Karkare** for his continuous support and guidance during my thesis. I would also like to extend my gratitude to my co-supervisor **Dr. Arnab Bhattacharya** for his guidance over the past semester. Without their help and guidance, it would not be possible to complete this thesis.

I would also like to thank Rajdeep Das who helped us throughout our thesis with any problems related to PRUTOR. I am also grateful to Sagar Parihar for his work in the domain of Automated grading for PRUTOR.

I would like to thank a few people for their constant support throughout my stay here at IIT Kanpur. I like to start with Ziyaan Dadachanji without whom this thesis would not be complete. I also like to thank Milan, Sawan, Vivek, Rishabh and many others for their valuable inputs.

Contents

List of Tables **vii**

List of Figures **viii**

1 Introduction **1**

1.1 Idea 3

1.2 Thesis Outline 3

2 Background and Related Work **5**

2.1 Background 5

2.1.1 Prutor : Online Programming Environment 5

2.1.2 Grading 6

2.2 Related Work 6

2.2.1 Automated Grading Tool for Introductory Programming . . . 7

2.2.2 Automated Feedback Generation for Introductory Programming Assignments 8

2.2.3 Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks 8

2.2.4 Feedback Generation for Performance Problems in Introductory Programming Assignments 9

2.2.5 Automated Error Localization and Correction for Imperative Programs 9

2.2.6 Program Repair as a Game 10

3 Error Pattern Discovery **11**

3.1 Motivation 11

3.2 Dataset 12

3.3 Results 14

4 Tackling Patterns **16**

4.1 Input and Output Statements Errors 17

4.2 Variables and Arrays Errors 19

4.3 Function Declaration and Definition Errors 23

4.4 Loop Errors 25

4.5 Pointer Errors 27

4.6 Common Errors 28

5 Repair System Architecture **32**

5.1 Input Receiver 33

5.2 Compiler Engine 35

5.3 Repair Engine 36

5.4 Evaluation Engine 37

6 Experiments and Results **40**

6.1 Performance of Repair System 40

6.2 Analysis of Repair System 43

6.2.1 Successful and Unsuccessful Submissions 44

6.2.2 Improvement in Marks 46

6.2.3 Repaired Errors and Frequency 47

7 Conclusions and Future Scope **51**

7.1 Conclusions 51

7.2 Future Scope 53

References **55**

List of Tables

3.1 List of Most Common Errors 13

4.1 Input and Output Statements Errors 19
4.2 Variables and Arrays Errors 23
4.3 Function Declaration and Definition Errors 25
4.4 Common Errors 31

6.1 Performance of Automated Repair System 41
6.2 Automated Repair System Statistics 46
6.3 Students performance for each lab event 49

List of Figures

5.1 Repair System Architecture 32
5.2 Input Receiver 34
5.3 Compiler Engine 36
5.4 Repair Engine 37
5.5 Evaluation Engine 39

6.1 Performance of Auto grader with Automated Repair System 42
6.2 Average SLOC per Submission for each Lab 42
6.3 Frequency of submissions with improvement in marks 47
6.4 Error Statistics 48
6.5 Error Frequency for some frequent errors for all lab events 49

Chapter 1

Introduction

Since its advent, computers have come a long way. Computer technology has become an integral part of our day-to-day life. Computers are being used by everyone and playing an important role in leading us to live a better life. Automated processes and systems controlled by computers have made industries and organizations efficient. Use of computers has led to better and efficient systems, faster results and revolutionized communication systems.

This development and progress in computer science technology has made computer education a necessity for everyone. Students of any field are more and more using computers to help them with their fields to solve harder problems efficiently and faster. This has led to almost every educational institute to have introductory programming courses as course work requirements. These courses are taught in computer labs to provide practical approaches to computer education.

Introductory programming courses are taught by assigning programming problems to students. Students are required to code the solutions to these problems and submit the solution for grading. The submissions are graded by institute appointed teaching assistants. As many students are learning to program for the first time, it is expected that many students will make errors and will face difficulties in solving assigned problems. TAs are expected to help students to understand programming

concepts and topics in which they are facing problems. As in any institute, hundreds of students are enrolled in the introductory programming course, providing relevant feedback in a timely manner in the form of grades has become a very difficult task. As grading policies and test cases based evaluation can be defined objectively. This has led to the design and development of an automated grading system [Par15]. Automated grading system can be used by students for real-time estimate of grades. This system can be used by instructor and TAs to reduce their task of grading students' submissions significantly.

Automated grading system is a great idea but it suffers from some fundamental problems. It works perfectly for those submissions which compile successfully and passes almost all test cases. Test cases are set of inputs and outputs on which a submission can be evaluated for correctness. The test cases are provided by instructor or problem setter. We have to remember that many students are learning programming for the first time and so it is expected that they will be making a lot of mistakes and thus, their submissions will be awarded low marks by automated grading system. Automated grading systems make some assumption while grading students submissions. Some assumptions are:

- Student's submission is compiling successfully. Automated grading system award zero or minimal marks for unsuccessful compiled submissions.
- All test cases results are matched without considering that minor changes in output format may lead to test cases passing.

In this thesis, we are trying to solve the problems mentioned above. We will explain automated repair system based grading framework which can be used to grade students' submission even when the programs are not compiling.

1.1 Idea

Our idea is to design and develop an automated grading system which will also grade submissions which do not compile and those submissions whose outputs deviate from expected outputs. This design requirement motivated us to design and develop an automated repair system which can correct compilation errors and make students submissions compilable.

We have worked with **PRUTOR** (**PR**ogramming T**utor**) [Das15] system which is a web-based programming environment designed, developed and hosted at Indian Institute of Technology, Kanpur by the department of Computer Science & Technology, to conduct **ESc101: Introduction to Computing** course for all branches of engineering. The purpose of conducting introductory programming course for all branches of engineering is to introduce students to basics of computer programming and help them to develop critical and logical approaches to solve problems with the help of computers. Because of these reasons, we have focused on correcting only syntactic errors. We have avoided correcting logical errors as correcting logical errors in students submissions is indirectly helping them with solving problems thus defeating the purpose of this course. Also, as automated repair system generates repairs based on limited information like compilation error messages and the line containing the error, repaired program may not be the one intended by the student.

1.2 Thesis Outline

Thesis outline is as follows :

- Chapter 2, **Background and Related Work**

  This chapter discusses previous work and development done in automated grading and automated repairs of programs.

- Chapter 3, **Error Pattern Discovery**

  This chapter explains the process and motivation behind choosing which errors to repair.

- Chapter 4, **Tackling Errors**

  This chapter explains the framework on which Automated Repair System works.

- Chapter 5, **Repair System Architecture**

  This chapter explains how different components of Automated Repair System interact with each other.

- Chapter 6, **Experiments and Results**

  This chapter explains all the experiments and testing done to evaluate the performance of Automated Repair System.

- Chapter 7, **Conclusions and Future Scope**

  This chapter concludes our thesis and presents improvements to the performance of automated grading system. We have briefly discussed future scope to improve Automated Repair System for better automated grading.

Chapter 2

Background and Related Work

This chapter discusses related work done in the field of automated grading and automated repairing of programs. We will briefly explain the programming environment used by students and previous related research and studies done to improve learning experience of students.

2.1 Background

2.1.1 Prutor : Online Programming Environment

Prutor [Das15] is a web based programming tutoring platform focused for both students and instructors. It wass designed, developed and hosted at IIT Kanpur under the supervision of Dr. Amey Karkare by the Department of Computer Science & Engineering. Students are provided with an online editor with advanced code editing features including tools for compilation, execution, and evaluation. Students can view their progress in the course like the number of assignments solved by them, grade awarded etc. As the tutoring system is a login based application, it tracks all the activities of students and how a student is solving a problem.

Instructors and teaching assistants are provided with tools to monitor students progress. They can see any student progress and his approach to solve any problem.

Also, instructors have access to tools to schedule different course events like lab assignments, quizzes and lab exams. They can schedule grading events for labs and exams, can decide grading policies and can monitor students grades. Teaching assistants can view students submissions, how they are approaching lab problems and provide relevant feedback.

2.1.2 Grading

In any course, grading provides the relevant feedback to students to help them track their progress. Grading can be defined as the task of assigning a score to student submission based on the correctness. For any course, this score is relative and based on the grading policy decided by the course instructor. From a student's perspective, grading is a very important aspect of any course. Grades help a student to track his progress and performance in the course. Also, it tells him about his relative performance with his peer students. Thus, grading plays a significant role in any course. Thus, maintaining unbiased grading and providing relevant feedback are of utmost importance from both student and instructor perspective.

ESc101: Introduction to Computing course employs a number of teaching assistants to help instructor with the grading task. TAs grade each submission based on grading policies and give a score to each submission. As this task is manually done, it is time-consuming and suffers from unintentional bias. Thus the task of maintaining the quality of relevant feedbacks becomes very difficult.

2.2 Related Work

In today's world of massive open online courses [MOO] and online learning platforms, automated feedback and grading has become of even more importance. Providing thousands of students with relevant feedback in the form of grades and suggestion has become very time-consuming. So the problem of providing automated feedback has caught the attention of education researchers. Many have designed and developed

different approaches to solve the scalability problem of automating feedback and grading system.

2.2.1 Automated Grading Tool for Introductory Programming

An automated grading system was designed and developed by Sagar Parihar [Par15] at IIT Kanpur. The motivation behind automated grading system was that as grading policy is objectively defined and thus a rule based automated grading system can be designed. Automated grading tool [Par15] uses linear regression approach to grade students submissions. Automated grading tool uses following features to generate a grading model:

- Number of test cases passed by student submission
- Time taken by student to solve the problem
- Fraction of successful compilations of all compilations made by student

Using linear regression model on existing TAs score, different values for above parameters were learned. These parameters define an automated grading policy which returns a grade score between zero and one for each submission . Final score can be deduced by multiplying grade score with maximum marks for that problem.

Linear regression-based automated grading system works very well for those submissions which are nearly perfect or failing small fraction of test cases. But if the program fails to compile due to some minor compilation error or some small logical error, automated grading system fails terribly. As marks awarded depends on number of test cases passed, a submission with minor compilation error is awarded zero or very low marks. This is a major problem as most of the students are learning to program for the first time and awarding zero or very low marks for minor mistakes do not make much sense.

2.2.2 Automated Feedback Generation for Introductory Programming Assignments

[SGSL13] paper presents an automated feedback technique based on determining a minimum number of fixes required to be made in student submission that will make it behave like the reference solution provided by the instructor. This feedback system provides accurate feedback to students about what they did wrong and how to correct their mistakes. This paper is different from our approach of automated repair system in following two respects:

- **Complete Specification is unknown with Automated Repair System:** Automated feedback technique is based on providing feedback by comparing student submission with instructor provided the solution. This only works if complete problem specification is known or if students are using the same approach as instructor's approach to solving the problem.

- **Predictability of students errors:** Automated feedback technique is based on the assumption that students are solving problems after attending same lectures and learning from same notes. Our system does not make any assumption about students learning approaches or environment .

2.2.3 Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks

[BS16] presented a Recurrent Neural Network based technique for providing feedback on syntax errors. Their approach is inspired from the developments done in field of learning language models from Big Code. The approach is to learn an RNN to model all valid token sequences using all the correct students submissions. Then for a incorrect submissions, learned RNN is used with prefix token sequences to predict token sequences that can fix the error by inserting or replacing the predicted token sequence at the error location.

2.2.4 Feedback Generation for Performance Problems in Introductory Programming Assignments

[GRZ14] presented a dynamic analysis based feedback approach to test whether a student's submission matches the specification provided by the instructor. Their approach is based on two important observations

- The same problem can be solved using different algorithm approach having different levels of efficiency and performance characteristics.
- The same algorithmic solution can be implemented in different ways.

[GRZ14] approach is to identify what algorithmic strategy is implemented by the student. As algorithmic strategy can be identified by values computed during program execution, the strategy can be converted into a program specification. They present a dynamic analysis approach that decides whether a student's implementation matches any of the algorithmic specification provided by the instructor. The instructor maintains a set of efficient and inefficient specifications. Student's submission is checked against all specifications. If student submission matches any specification, associated feedback is generated. If no match is found and student submission is correct against all test cases, a new specification based on student submission is added to the set of specification for future matchings.

2.2.5 Automated Error Localization and Correction for Imperative Programs

[KB11] presented a debugging technique based on automatic error correction and localization. Their error localization method uses a model-based diagnosis and SMT solving. To correct errors, they use a template-based approach to ensure that computer repairs are readable. Their method expects an input in the form of incorrect program and a corresponding specification which can be a reference implementation or assertions statements. This input is preprocessed to express faulty components

and then symbolic execution is applied to transform the problem into the domain of logic. This technique applies repairs in iterative refinements and supports only simple corrections.

2.2.6 Program Repair as a Game

[JGB05] presents a method to automatically fix errors in programs by considering the repairing system as a game. The program is represented in linear time logic (LTL) specification. The program repair problem is a LTL game which captures the possible repairs in the program. A repair is represented as a move in which system can provide a correct value for unknown expressions. If repair makes the program satisfy the problem specification, that move represents a winning move. A winning strategy can lead to change in program logic so converted the problem of repairing into a memoryless strategy. This strategy is shown to be NP-Complete and thus they presented a heuristic approach to find an efficient repair for a given memoryless strategy.

Chapter 3

Error Pattern Discovery

3.1 Motivation

In any introductory programming course, it is expected that students are going to make a lot of errors and will be facing difficulties in correcting those errors. To help students with the corrections of their code, ESc101 course employs a number of teaching assistants (TAs). Students can ask assigned TAs to help them with their lab assignments. These problems are set by the instructor. Students are supposed to provide solutions in C programming language for these lab assignments. TAs are specifically instructed not to help students with the program logic but only with syntactic errors. However helping around two hundred students with the help of $15 - 20$ TAs is not feasible and many students feel there is a lack of a consistent feedback system. Moreover, every student requires a different level of feedback which is very difficult to provide with a limited number of TAs. TAs feedback may be very generic or maybe he is not able to address the underlying problem without actually solving the problem for the student. Also, instructor has no way to know in which areas students are struggling. If instructor knew these areas, he can focus more on these areas specifically or give problems and assignments for practice.

To design an automated program repairing system, it is necessary to identify which errors are made most by students. We analyzed the data collected by PRUTOR

[Das15] and identified those areas in which students are struggling. We also analyzed incomplete solutions to lab assignments and identified recurring errors with which students are struggling continuously. This analysis combined with manual feedback from students, TAs and instructors helped us to identify most recurring problem areas which can be repaired in an automated manner.

3.2 Dataset

The following dataset is collected from **ESc101: Fundamentals of Computing**, an introductory programming course taught in the even semester of the academic year $2015-2016$ at IIT Kanpur. The course was taught in C programming language. ESc101 course conducted weekly labs of three hours each with two or three problems in each lab. Students were to submit the solutions for lab assignments and those assignments were graded by TAs. All student activities including typing patterns, all assignments solutions, program compilation results, intermediate code, etc. were recorded. These recorded activities were analyzed to identify those areas in which most of the students are struggling or need better feedback.

Sr.No.	**Error Type**	**Count**
1	use of undeclared identifier ‘var’	122375
2	control may reach end of non-void function	49016
3	Missing ‘&’ before variable name in scanf statements	48327
4	Use of wrong format specifier in scanf and printf statements	
5	Extra ‘&’ before variables in printf statements	
6	initialize the variable ‘var’ to silence this warning	43767
7	expected ‘;’ after expression	20578
8	expected ‘)’ ,‘]’,‘}’	18558

Sr.No.	Error Type	Count
9	use '==' to turn this assignment into an equality comparison	9823
10	type specifier missing, defaults to 'int'	9427
11	subscripted value is not an array, pointer, or vector	6633
12	previous definition is here	5642
13	extraneous closing brace ('}')	5589
14	use of undeclared identifier 'varr'; did you mean 'var'?	5264
15	please include the header $< header.h >$ or explicitly provide a declaration for 'func'	4454
16	multi-character character constant	3804
17	for , while loop or if or switch statement has empty body	3598
18	expected ';' in 'for' statement specifier	2928
19	member reference type 'ptr' is a pointer; maybe you meant to use '→'?	1917
20	invalid conversion specifier '.'	1519
21	member reference type 'var' is not a pointer; maybe you meant to use '.'?	1304
22	definition of variable with array type needs an explicit size or an initializer	1094
23	missing terminating '"' character	1041
24	void function 'func' should not return a value	656

Table 3.1: List of Most Common Errors

3.3 Results

Analyzing above data, we can divide the above errors into six categories. These categories are:

- **Input/Output Statements**
    1. Missing '&' before variable name in scanf statements
    2. Extra '&' before variables in printf statements
    3. Use of wrong format specifier in scanf and printf statements
    4. Invalid format specifier in scanf
- **Variables and Array Related Errors**
    1. Redefinition of variable
    2. Incorrect spelling of variable
    3. Undeclared variable
    4. Unused variable
    5. Char type variable with multi-character value
    6. Uninitialized variable OR use of scanf statement after using the variable
    7. Size missing in declaring array
    8. Variable not array/pointer
- **Function declaration and definition Related Errors**
    1. Return statement missing
    2. Return data type missing from function definition
    3. Void function should not return a value
    4. Non-void function should return a value

- **Loops related Errors**

    1. Empty Body of for or while loops
    2. Expected semicolon ';' in a for statement

- **Pointers**

    1. Use of '.' operator in place of '→' operator in reference type pointer variables
    2. Use of '→' operator in place of'.' operator in non-pointer variables

- **Common Errors**

    1. Semicolon missing
    2. Missing Header file
    3. Extra braces or parenthesis, missing braces or parenthesis
    4. Missing terminating '"' character
    5. Interchange of comparison and assignment operator

We group the errors into different categories such that errors in each category are similar in concepts. Grouping of errors is necessary because most of the concepts are interconnected and linked. The grouping helps us to abstract common programming concepts which are used to repair errors.

Chapter 4

Tackling Patterns

In this chapter, we will present a framework which is used to design a model to repair students' submissions and provide consistent and systematic feedback to the students. As shown in Chapter 3, students struggled with different areas which can be divided into six major categories. Each of these categories has a core concept which the students are not able to grasp completely. As most errors and concepts in each category are interrelated, we have repaired these errors in a modular manner.

We have selected most common errors done by students. Most of these errors are easy to understand and correct. Also, a detailed feedback can be given to the student explaining the cause of these errors. To correct these errors, we design a framework which is based on the information returned to us by the compiler. The repairs are rule based and worked in a deterministic manner. Repairing wrong code statements requires that we know the exact position of the error and determine deterministically what the category of error. Different categories of errors and repairing approaches are explained below. All the repairs are based on limited information like compilation errors and code containing the error. Thus, resultant repaired code may not be the one intended by the student.

4.1 Input and Output Statements Errors

When solving programming assignments and problems using C programming language, most students prefer to use scanf and printf statements for data input and output respectively. It is expected that a student will have a good grasp on scanf and printf statements syntax and will understand their working. However, our analysis shows that scanf and printf statements are one of the most common source of error with which most of the students struggle. Even in advanced lab assignment problems, it has been observed that many students were not completely comfortable with data input and output. Some of the common errors made by students are :

- Missing '&' before variable name in scanf statements
- Extra '&' in front of variables in printf statements
- Use of wrong format specifier in scanf and printf statements
- Invalid format specifier in scanf

As most of these errors are very similar in structure and principle, so repairing them can be grouped.

<table>
<tr>
<td>Error</td>
<td>

Missing ‘&’ before variable name in scanf statements
Extra ‘&’ in front of variables in printf statements

</td>
</tr>
<tr>
<td>Repairing Approach</td>
<td>Both are similar in sense that either address operator ‘&’ is missing from scanf statement which expects a pointer to the variable in which the value is to be read or extra address operator ‘&’ is placed in front of variable in printf statement which expects a variable name instead of pointer to that variable. Repairing these two errors can be grouped based on in which type of statements the error has occurred and address operator can be added or removed from the statement correcting the error.</td>
</tr>
<tr>
<td>Example</td>
<td>Incorrect Code snippet :

<pre>int num;
scanf("%d",num);
printf("%d",&num);
...</pre>
Correct Code snippet after repair:

<pre>int num;
scanf("%d",&num);
printf("%d",num);
...</pre>
</td>
</tr>
</table>

<table>
<tr><td>Error</td><td>

- Use of wrong format specifier in scanf and printf statements
- Invalid format specifier in scanf

</td></tr>
<tr><td>Repairing Approach</td><td>Both these errors refer to errors which are caused due to using wrong format specifiers. To correct these types of errors, we have to determine the correct data type of the variable which is being used in the statement. This information can be extracted from compiler error messages referring to these statements. From learned data type, incorrect format specifiers can be replaced with correct format specifiers.</td></tr>
<tr><td>Example</td><td>Incorrect Code snippet :
<pre>int num,var;
scanf("%f",&num);
scanf("%3.f",&var);
printf("%f",num);
...</pre>Correct Code snippet after repair:
<pre>int num,var;
scanf("%d",&num);
scanf("%d",&var);
printf("%d",num);
...</pre></td></tr>
</table>

Table 4.1: Input and Output Statements Errors

4.2 Variables and Arrays Errors

Almost all statements and expressions in a program use some variables which can be simple variables, array type variable or pointer variables. Variable declaration and usage are basic concepts but after analyzing students' submissions for four semesters, it has been observed that many students struggle with variable declaration and usage.

<table>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Redefinition of variable
This error occurs when a student has already defined a variable with some data type, let say ‘datatype1’ and now he has again defined that same variable with the same data type or another data type. This error can be corrected by removing second declaration statement or expression.

Incorrect Code snippet :
<pre>int num;
...
float num;
...</pre>Correct Code snippet after repair:
<pre>int num;
...
...</pre></td>
</tr>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Char type variable with multi-character value
Many students make the error to initialize a char variable with multi character constant. This error can be corrected by removing all extra characters from the constant.

Incorrect Code snippet :
<pre>char var = ‘ab’;
...</pre>Correct Code snippet after repair:
<pre>char var = ‘a’;
...</pre></td>
</tr>
</table>

<table>
<tr>
<td>Error</td>
<td>

Incorrect spelling of variable
Undeclared variable

</td>
</tr>
<tr>
<td>Repairing Approach</td>
<td>These errors are similar in the sense that if a variable spelling is incorrect, or it is undefined, the compiler throws similar errors. To check if a variable is spelled wrong, we use an edit distance algorithm which is an in-built feature of LLVM based clang compiler [Cla]. If a variable is found which is suspected to be spelled wrong, the spelling can be corrected. If no such declared variable is found, we deduce the type of the variable from the expression it is being used in. Using the deduced type, we can declare and initialize the variable.</td>
</tr>
<tr>
<td>Example</td>
<td>Incorrect Code snippet :

<pre>int num;
numm = 10;
var =5;
...</pre>
Correct Code snippet after repair:

<pre>int num;
int var=0;
num= 10;
var =5;
...</pre>
</td>
</tr>
</table>

<table>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Uninitialized variable / Use of scanf statement after using the variable
This error is an interesting error. Students are using variables in expressions without initializing them. It has been observed that they use those variable to take input but after using them in expressions. This type of errors can be corrected by swapping the two statements.

Incorrect Code snippet :

<pre>int len,bre,area=0;
area = len*bre;
scanf("%d%d",&len,&bre);
...</pre>
Correct Code snippet after repair:

<pre>int len,bre,area=0;
scanf("%d%d",&len,&bre);
area = len*bre;
...</pre></td>
</tr>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Size missing in declaring array
If an array type variable is declared and array size is not given, a compile time error is thrown. This error can be handled by setting a default size to the array type variables. Based on the problems and assignments, the array size has been set to 100.

Incorrect Code snippet :

<pre>int num[];
...</pre>
Correct Code snippet after repair:

<pre>int num[100];
...</pre></td>
</tr>
</table>

Error	Variable not array/pointer
Repairing Approach	If a variable which is not of array type but used with a subscript, the subscripted values can be removed to correct this error. The assumption behind this repair is that student intended to declare a simple variable not array type variable. As we can not unambiguously decide which array subscript to use, we safely remove subscripted values.

Example	Incorrect Code snippet :
	`int num,var=20;`
	`num[1] = 5;`
	`...`
	`num = var[1] -10;`
	`...`
	Correct Code snippet after repair:
	`int num,var=20;`
	`num = 5;`
	`...`
	`num = var -10;`
	`...`

Table 4.2: Variables and Arrays Errors

4.3 Function Declaration and Definition Errors

Writing custom functions and using them are an important part of any basic programming course. But they have proved to be quite puzzling for students. Some common mistakes made by students are:

1. Return statement missing
2. Return data type missing from function definition
3. Void function should not return a value

<table>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Return statement missing
If a function definition has a return data type and does not include a return statement, a default return statement can be added in the function. This addition may result into breaking the functionality of the function

Incorrect Code snippet :
<pre>int calArea(int len,int bre){
...
// No return statement
}</pre>
Correct Code snippet after repair:
<pre>int calArea(int len,int bre){
...
// return statement added
return 0;
}</pre></td>
</tr>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Void function should not return a value

If a function is defined as a void function, it should not return any value. This can be corrected by removing the return statement.

Incorrect Code snippet :
<pre>void printFactor(int num){
  return 0;
}
...</pre>
Correct Code snippet after repair:
<pre>void printFactor(int num){
  // no return statement
}
...</pre></td>
</tr>
</table>

<table>
<tr><td>Error

Repairing Approach

Example</td><td>Return data type missing from function definition
If a student forgets to provide a return type in the function definition, default data type such as ‘void’ or ‘int’ can be added to the definition after analyzing the function usage in rest of program.

Incorrect Code snippet :

<pre>printFactor(int num){
 // no return statement
}

calArea(int len,int bre){
return area;
}
...</pre>
Correct Code snippet after repair:

<pre>void printFactor(int num){
  // no return statement
}

int calArea(int len,int bre){
   return area;
}
...</pre></td></tr>
</table>

Table 4.3: Function Declaration and Definition Errors

4.4 Loop Errors

Loops are an integral part of any programming language and C programming language support three types of loops: for loop, while loop and do-while loop. Loops are easy to understand and code but many students make errors related to their syntax. They tend to forget the following:

- **Empty Body of for or while loops** :This error is not a compilation error but a logical error. Many students put a semicolon at the end of 'for' or 'while' statements causing the body of loop effectless. This introduces many logical bugs which have been proved quite difficult for students to debug. As loops can have single or multi statement body, we have two different cases

  - **Multi Statement Body:** We solved this bug by checking if a loop statement ending with semicolon has a block starting on next line. If this is the case,we can safely remove the semicolon from the loop statement.

    Original Code:

```
#include<stdio.h>
int main()
{
          int i=0;
          for(i=0;i<10;i++);
          {
                        ...
          }
          return 0;
}
```

    Repaired Code:

```
#include<stdio.h>
int main()
{
         int i=0;
         for(i=0;i<10;i++)
         {
                       ...
         }
         return 0;
}
```

  - **Single Statement Body:** If there is no block after the loop body. This case can be handled by checking if the loop statement is performing any operations on any variable except loop variable. If not, next statement is part of loop body else we cannot unambiguously detect loop body.

Original Code:

```
#include<stdio.h>
int main()
{
        int i=0,num=1;
        for(i=0;i<10;i++);
                num = num + i ;
        return 0;
}
```

Repaired Code:

```
#include<stdio.h>
int main()
{
        int i=0;
        for(i=0;i<10;i++)
                 num = num + i ;
        return 0;
}
```

- **Expected semicolon ';' in a for statement:** A for statement expects three expressions: initialization of loop variable, loop condition and variable modifier expression. All three expressions can be empty but need to be separated by ';'. Many students forget to add a semicolon after initialization or loop condition expression or add a comma instead of semicolon. This error can be corrected by detecting the initialization and condition expression and placing a semicolon after them. If comma is found, it can be replaced with a semicolon.

4.5 Pointer Errors

Pointers are one of the basic concepts to learn programming with C language. When students are introduced to pointers in the introductory programming course, many of them are confused regarding the apt usage of following operators:

- Indirection operator '*'
- Address-of operator '&'

- structure indirection operator '→'
- structure direction operator '.'

We found that many students were using wrong operators. Some common cases were

1. Use of '.' operator in place of '→' operator in reference type pointer variables
2. Use of '→' operator in place of '.' operator in non-pointer variables

To correct these errors, we detect wrong placement of operator and replace those with the correct operator by deducing variable type.

4.6 Common Errors

There are many errors which do not fit in any specific category but are very common to make. These errors are simple enough to miss and if a student is not paying attention, very difficult to debug.

<table>
<tr>
<td>Error</td>
<td>Extra braces or parenthesis, missing braces or parenthesis</td>
</tr>
<tr>
<td>Repairing Approach</td>
<td>When a student is writing a nested block of code, it is very easy to lose track of all the braces and parenthesis. This may lead to extra or missing braces and parenthesis error. These error can be repaired by added or removing those extra braces and parenthesis.</td>
</tr>
<tr>
<td>Example</td>
<td>Incorrect Code snippet :

```
int main(){
    int num=5;
    if(num > 10){
        printf("Number greater than 10");
    }
    return 0;
 }
 }
```

Correct Code snippet after repair:

```
int main(){
   int num=5;
   if(num > 10){
       printf("Number greater than 10");
   }
   return 0;
}
```

</td>
</tr>
</table>

<table>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Semicolon missing
This error is one of the most common mistake made by students who has no experience in programming in C. Every statement in C expects a semicolon to denote its completeness. But many students struggle with this requirement. They forget to add the semicolon at the end of statements. This can be corrected by inserting a semicolon at the end of the statements which were expecting semicolon.

Incorrect Code snippet :

<pre>int num;
scanf("%d",&num)
...</pre>
Correct Code snippet after repair:

<pre>int num;
scanf("%d",&num);
...</pre></td>
</tr>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Missing terminating double quote ” character
String and characters array are very similar but have proved to be quite challenging for students who have started working with them. Many students forget to enclose the string in double quotes or may use single quotes instead of double quotes. These errors can be repaired by enclosing the strings in double quotes.

Incorrect Code snippet :

<pre>...
char s[10] = "hello ;
printf("%s",s);
...</pre>
Correct Code snippet after repair:

<pre>...
char s[10] = "hello" ;
printf("%s",s);
...</pre></td>
</tr>
</table>

<table>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Missing Header file
Students may have used some library functions but forget to include the respective header file. This causes the compiler to throw undefined or undeclared function use. This error can be removed by including the header file in which the used function is defined.

Incorrect Code snippet :

<pre>
#include<stdio.h>
int main(){
        char s[10] = "hello";
        int len = strlen(s);
        ...
}
</pre>
Correct Code snippet after repair:

<pre>
#include<stdio.h>
#include<string.h>
int main(){
        char s[10] = "hello";
        int len = strlen(s);
        ...
}
</pre>
</td>
</tr>
<tr>
<td>Error

Repairing Approach

Example</td>
<td>Interchange of comparison ‘==’ and assignment operator ‘=’
When a student is introduced to programming, they seem to struggle with comparison and assignment operator. They place comparison operator in place of assignment operator in binary expressions or assignment statements. This error can be repaired by identifying those expressions in which students might have used wrong operator and replace them with correct operator.

Incorrect Code snippet :

<pre>
int num = 5;
if (num =5){
    ...
}
</pre>
Correct Code snippet after repair:

<pre>
int num = 5;
if (num == 5){
                ...
}
</pre>
</td>
</tr>
</table>

Table 4.4: Common Errors

Chapter 5

Repair System Architecture

The repairing system comprised of four main components. These components are **Input Receiver, Compiler Engine, Evaluation Engine** and **Repair Engine**. These components work with each other with help of message passing in JSON [JSO] format.

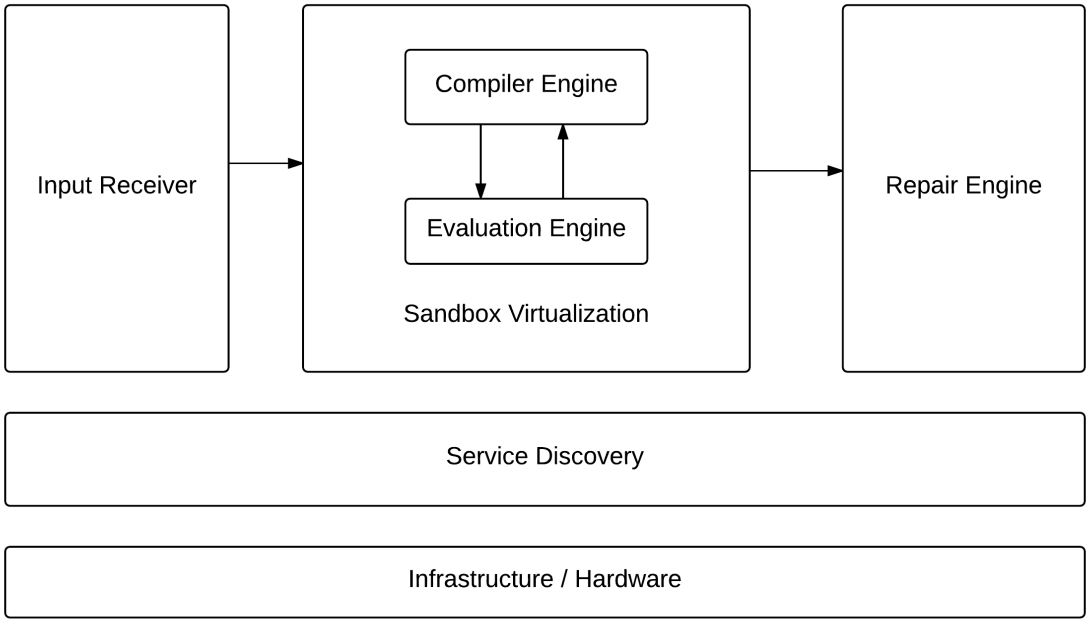


Figure 5.1: Repair System Architecture

5.1 Input Receiver

Input receiver is the first component which receives all the request from external systems or from repairing system users. This component acts as a proxy which handles all incoming requests. Current system is designed to work with only HTTP POST [POS] requests which must contains all the data in JSON [JSO] form encoded in base64 format [bas]. This encoding at user end ensures data security and avoids data corruption. This component expects following data all encoded in base64 format [bas]:

1. Source code (required)
2. Test Cases input list on which repaired code can be tested (not required)
3. Expected output list to compare with actual outputs generated with the repaired program (not required)
4. Maximum marks for the problem - this data is not necessary for repairing engine but required by Auto grader System which is the complementary system based on Repairing System. Marks are not encoded and expected as plain text as a integer

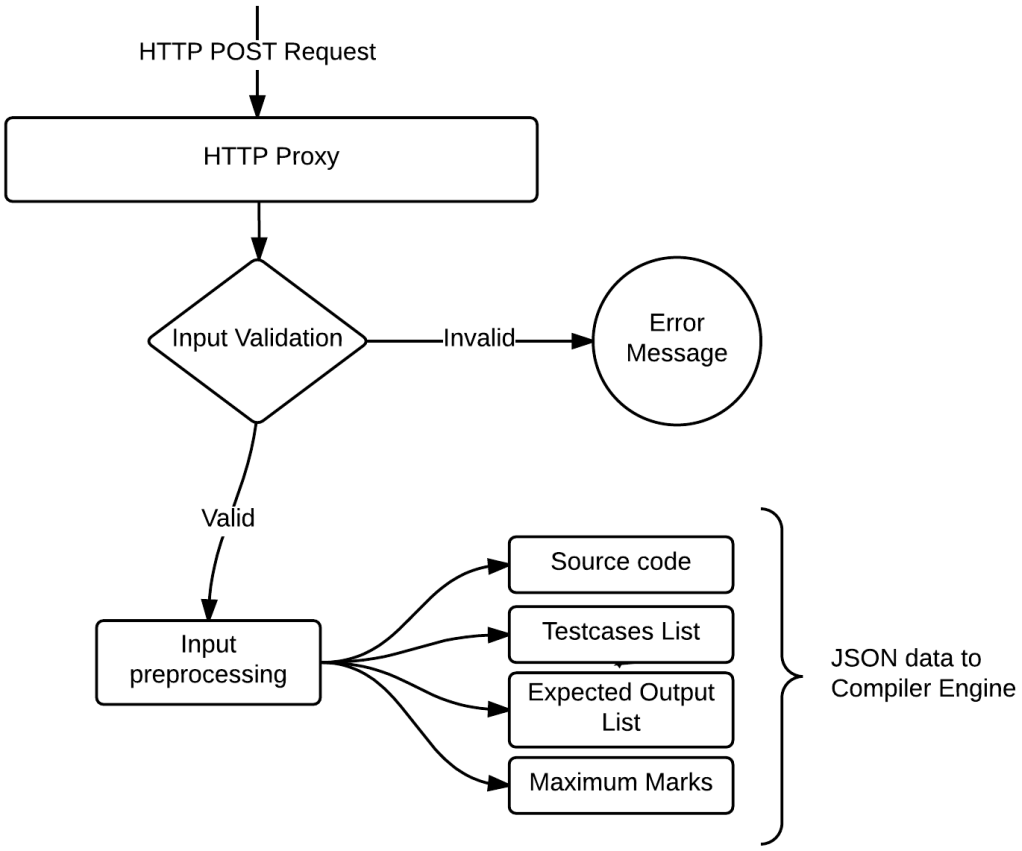


Figure 5.2: Input Receiver

All received data is validated for data corruption. After data validation, initial code is passed to Compiler Engine which tries to compile the code. This may result into following cases

- **Successful Compilation:** If code successfully compiles without any errors or warning, then generated executable file is passed to Evaluation Engine.
- **Unsuccessful Compilation:** If code compilation produce any errors or warning, it is considered as unsuccessful compilation. In this case, the code is passed to Repair Engine which tries to repair compilation errors and warnings.

5.2 Compiler Engine

Compiler Engine is responsible for compilation of given code. This component can be called by Input receiver or Repair Engine. Compiler Engine provides a secure sandbox environment based on Linux containers [Lin] and Linux AppArmor [App] security model which provides security by confining programs to a limited set of resources. This secure sandbox is shared by Compiler Engine and Evaluation Engine.

This modules receives the code in plain text format which is compiled with LLVM based clang compiler [Cla]. The compiler output can result into a successful compilation or unsuccessful compilation. If compilation is successful, executable file is passed to Evaluation Engine for grading purposes. In case of unsuccessful compilation, code is send to Repair engine which sends back the repaired code to this module for recompilation. System administrator can set a global variable which control the maximum number of recompilations of code. Along with code, this engine sends a list of all the errors which includes following information

- **Type:** which can be either 'error' or 'warning'
- **Line no:** line no in the source code which has some syntax error
- **Column no:** position of the error in the statement
- **Error message:** error message generated by the compiler

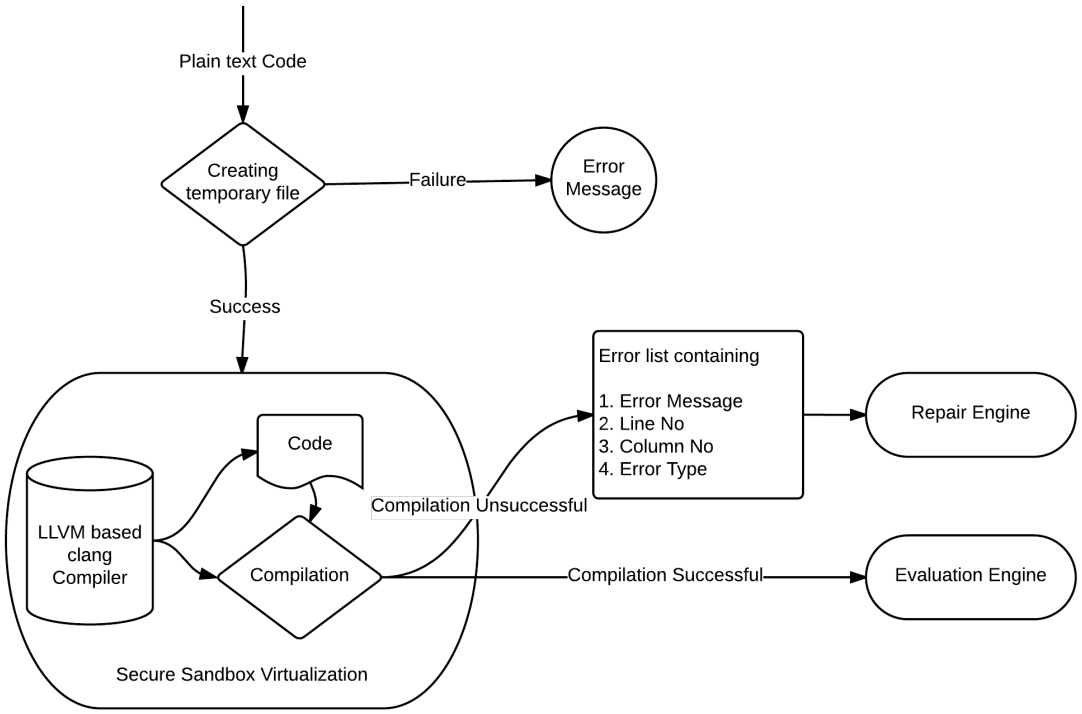


Figure 5.3: Compiler Engine

5.3 Repair Engine

This component is the heart of our repairing system. It is responsible for sorting and analyzing different types of errors and how to repair them. Repair engine receives code and a list which includes all the information about the errors and warnings. This list is sorted based on position of errors in code.

Repairing code is a rule based deterministic task. This ensures that for same type of errors on similar code always produce similar repairs. Every error invokes a repairing function which is designed to repair that specific error. The framework supporting repairing modules is designed to support extensibility and portability. New repairing modules can be developed and added to existing system with minimal changes. This also allows the instructor to decide which type of repairs to support. The instructor can choose which repairs to be allowed for students submissions and he can choose different penalties for each error which is repaired.

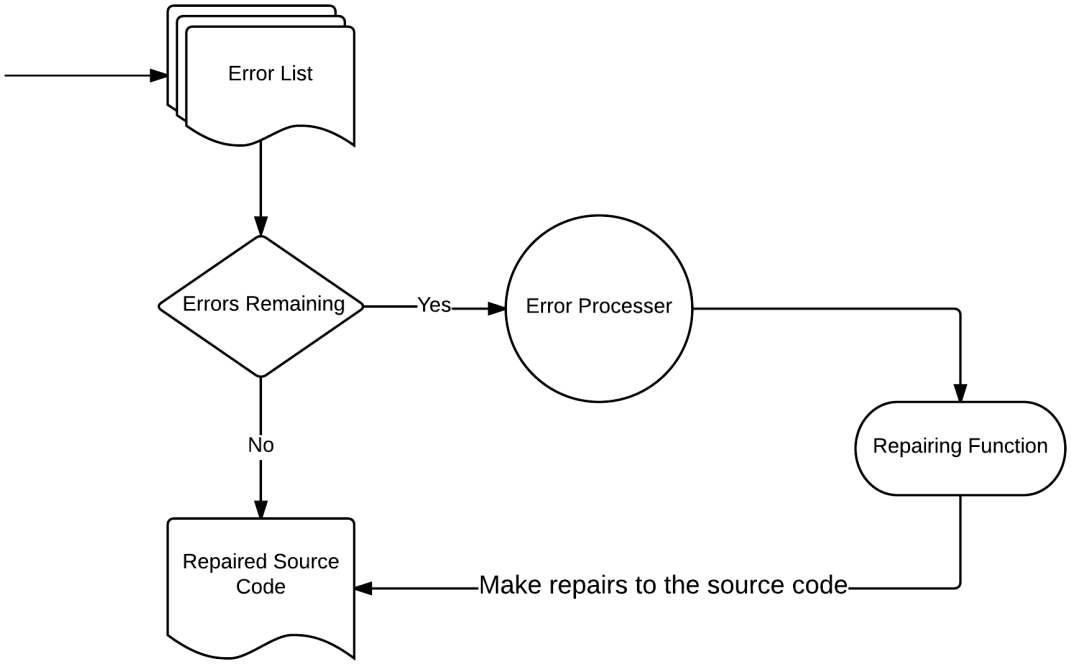


Figure 5.4: Repair Engine

5.4 Evaluation Engine

Evaluation engine is the final step in repairing system. It is a validation module which checks whether the repairs done in repair engine has caused any improvement in student's code. It is called only if code is successfully compiled. Both evaluation engine and compiler engine work in secure sandbox environment to run and compile user code. This protects underlying operating system from running malicious code.

We will be using following terms in relation to the functioning of evaluation engine

1. **Visible test case:** A visible test case is the input which is visible to the students. Students can see the input and corresponding expected output.

2. **Hidden test case:** A hidden test case is hidden from students. The input and corresponding output is not shown to student. A correct submission has to pass all the hidden and visible test cases. The students are informed about the status whether the submission has passed the hidden test case or not.

3. **Actual Output:** Output generated by executing student's submission on a test case.

4. **Expected Output:** Output provided by the problem setter or the instructor for test case.

5. **Output matching:** If actual output and expected output are same according to matching criteria set by problem setter then it is considered a successful matching else it is an unsuccessful matching. Successful matching means a test case has passed.

Evaluation engine's task is to execute the students code on given test cases and keep track of the respective outputs. It matches actual outputs with corresponding expected outputs. Also, it keeps track of how many hidden and visible test cases have passed. This data is used by auto grader module for grading lab assignments and problems.

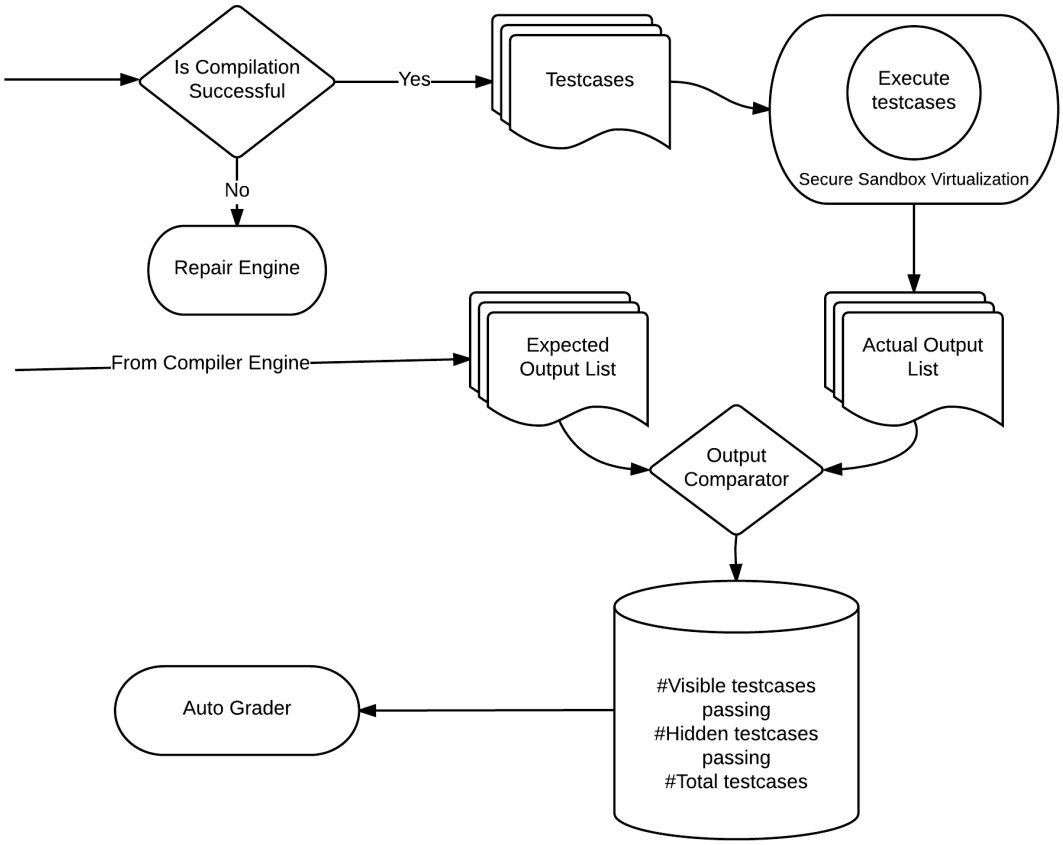


Figure 5.5: Evaluation Engine

Chapter 6

Experiments and Results

For our experiments, we have used the dataset collected from PRUTOR [Das15] as part of ESc101: Fundamentals of Computing course conducted in the even semester of the academic year of 2015-2016 at IIT Kanpur. All lab assignments and exams are solved using C programming language.

We have two datasets, lab dataset and exam dataset. There are twelve labs and two exams: mid-sem exam and end-sem. Lab dataset refers to all the data related to students submissions and grades collected during labs. Exam dataset refers to the data collected during exams. PRUTOR [Das15] logs every student's activity in lab and exam including problems, submissions, number of compilations, compilation errors, etc. and grades awarded by TAs for each assignment problem.

6.1 Performance of Repair System

Performance and efficiency of any computer system can be measured by multiple parameters [Per] like :

- **Response time for any computing task:** the total amount of time it takes to respond to a request for service.
- **The throughput of the system:** the rate at which some computing task can be processed. It is calculated as inverse of response time.

- **Utilization of computing resource:** Amount of computing resources used by the system. It includes CPU cycles, RAM and disk usage, network throughput, I/O operations, etc. [Res]

To measure the performance and efficiency of automated grading tool with automated repair system, we measure the above parameters for different tasks like:

- Particular lab or exam submission
- All submissions for each lab type.
- All submissions of mid-sem and end-sem exam.

Event	**Response Time (ms)**	**Throughput(per sec)**	**Avg. SLOC**
Lab 1	118.90	8.41	7
Lab 2	123.00	8.13	9
Lab 3	153.76	6.50	34
Lab 4	215.69	4.64	23
Lab 5	368.68	2.71	31
Exam 1	1,029.79	0.97	29
Lab 6	278.90	3.59	43
Lab 7	515.11	1.94	49
Lab 8	575.97	1.74	36
Lab 9	292.57	3.42	46
Lab 10	394.75	2.53	46
Lab 11	529.37	1.89	78
Lab 12	441.18	2.27	122
Exam 2	2,032.89	0.49	53

SLOC: Source lines of code.

Throughput: Submissions repaired per second.

Table 6.1: Performance of Automated Repair System

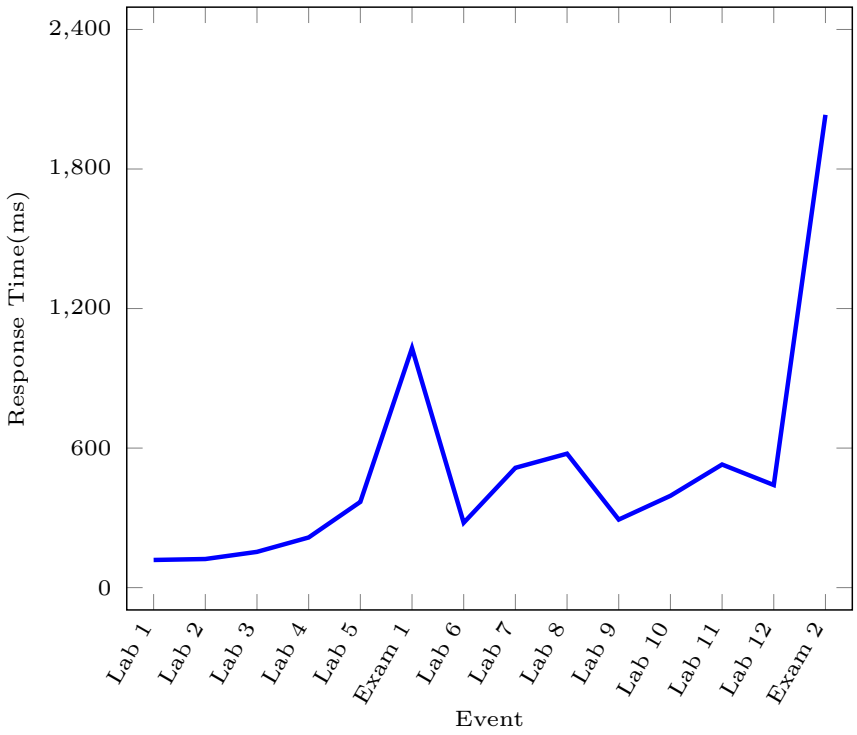
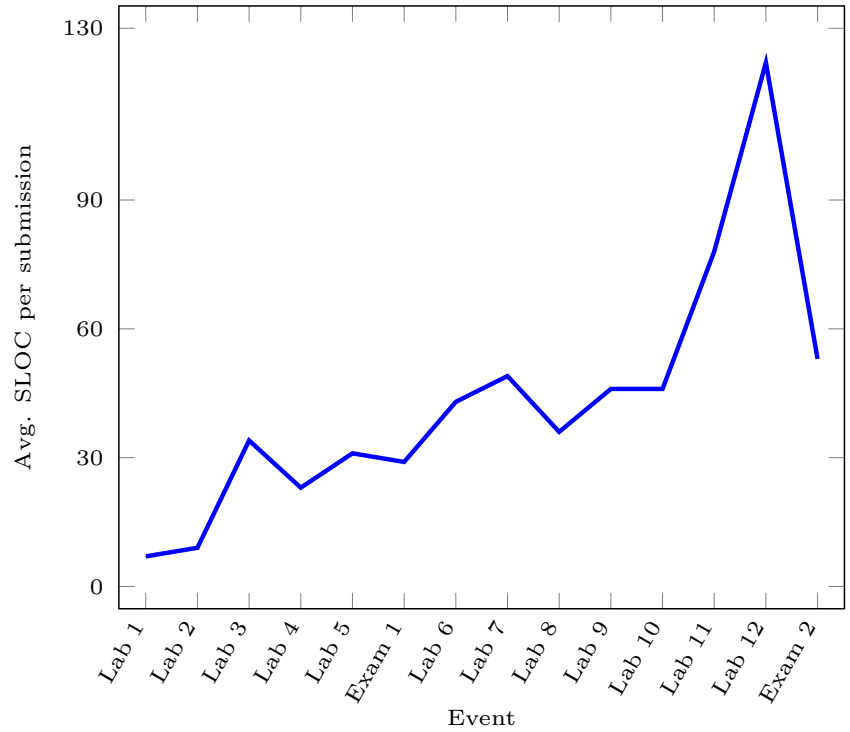


Figure 6.1: Performance of Auto grader with Automated Repair System



SLOC: Source lines of code.

Figure 6.2: Average SLOC per Submission for each Lab

Program runtime depends on multiple parameters like CPU usage, memory usage, lines of code, execution time, etc. As we are performing experiments on thousands of submissions, calculating CPU and memory usage for each submission is a very difficult task. As we are considering the submissions submitted for introductory programming course, we can safely ignore the CPU and memory usage. Each submission is only accepted if it executes within the time limit decided by the instructor. So, we have used lines of code in source code as the metric to calculate the runtime of student's submissions.

We can observe a direct correlation between response time of the system and the number of lines in the given source code. As the size of source code increase, the response time of the automated repair system also increases. A deviation from this trend can be observed for the lab exam submissions. The reason behind this deviation is the increase in the difficulty level of exam problems. Also, in lab 11 and 12 topics like pointers, linked list, structures etc. are taught. Our repair system does repair some basic errors related to pointers and structures but ignores rest of the errors. But in exams, problems are asked from all topics and so, frequency of those errors which are corrected by automated repair system increases. Thus a significant increase in response time can be observed for exams. In labs students take help from teaching assistants and their friends, thus make less errors. In exams lack of any help from teaching assistants causes increase in number of errors thus requires more time for repairing the submissions.

6.2 Analysis of Repair System

The motivation behind the development of automated repair system was to enable auto-grader to grade those programs which were compiled unsuccessfully. Auto grader without automated repair system awards zero or very low marks to unsuccessfully compiled submissions. We have analyzed the performance of auto-grader with automated repair system.

6.2.1 Successful and Unsuccessful Submissions

Unsuccessful compiled submissions refer to those submissions which when compiled generate warning and errors. Successful compilation means the program gets compiled without any compile time error or warning. Submissions which contains only warning are also considered as unsuccessfully compiled as in many cases it is observed that repairing the warning may lead to the correct program which may make the submissions passing more test cases than earlier.

We analyzed the working of automated repair system on unsuccessfully compiled submissions. The successfully compiled submissions were not considered for analysis of working of rep air system. This is due to the fact that analysis of repair system on successfully compiled submissions does not produce any significant results related to repair system.

Example 6.1 The student has used a scanf statement with wrong format identifier and forgot to use '&' before the variable name. Compiler will show these as warning but no compile time error. But this mistake changes the runtime behavior of the submissions.

Original Code:

```
#include<stdio.h>
int main()
{
    float num;
    scanf("%d",num);
    ...
    return 0;
}
```

Repaired Code:

```
1  #include<stdio.h>
2  int main()
3  {
4      float num;
5      scanf("%f",&num);
6      ...
7      return 0;
8  }
```

In above example, in line no 5, the student has used a wrong format specifier **'%d'** in place of **'%f'** and forget to place **'&'** before the variable. This will cause a compile time warning and will change the run time behavior of the student's submission. This repair will make the program work as expected and if rest of the program is correct, it may lead to passing more test cases than before.

We have analyzed the dataset from labs and exams separately. We did this due to the fact that in a lab environment, a student can take help from his notes, friends or teaching assistants so he is expected to make fewer errors. This is not available in an exam environment which leads to students making more errors.

For labs, 12,872 programs were submitted as the requirement for course completion. Around 40 teaching assistants graded these submissions and provided feedback to students. Out of these 12,872 submissions, 2,359 submissions had compilation warnings and 881 submissions were unsuccessfully compiled with compilation errors. For these 3,240 (2,359 and 881) , teaching assistants tried to correct these errors and warnings and check the correctness of the logic of the submissions. For exams, 2,741 submissions were submitted. Out of these submissions, 724 submissions had compilation warnings and 536 submissions failed to compile successfully.

After running our tool on these 3,240 lab submissions and 1,260 exam submissions, 323 lab submissions and 185 exam submissions got corrected and compiled successfully (no warnings and errors) . In some cases after every correction, new errors and warnings can be detected by the compiler. As our repairing tool only tries a limited

times to repair any submissions and repairing tool does not correct every type of warning and errors, some error and warning remains. 199 lab submissions and 144 exam submissions were repaired with compilation warnings remaining. These submissions can be evaluated for validation of logic.

		Lab Submissions	**Exam Submissions**
Total Submissions		12872	2741
Before Repairs			
Unsuccessful Compilations	warnings only	2359	724
	errors	881	536
After Repairs			
Successful Compilations	warnings & errors → no warnings & no errors	323	185
	warnings → no warnings	241	136
	errors → no errors & no warnings	82	49
	errors → successful compilation (may contain warnings)	199	144
Total submissions containing repairs		1265	713

Table 6.2: Automated Repair System Statistics

6.2.2 Improvement in Marks

From Table 6.2 we can see that 323 lab submissions and 185 exam submissions were corrected with no compilation time errors or warnings. 199 lab submissions and 144 exam submissions were corrected with some compilation time warnings remaining. Out of these submissions, 91 (58 lab and 33 exam) submissions have improved grades compared to auto-grader used without repairing system. The improvement

in grades ranges up to 24 marks. 32 submissions were awarded more than 50% of total marks. Out of 91 submissions which have improved grades, 38 submissions had compilation errors which were awarded zero marks by the auto-grader system before using repairing system.

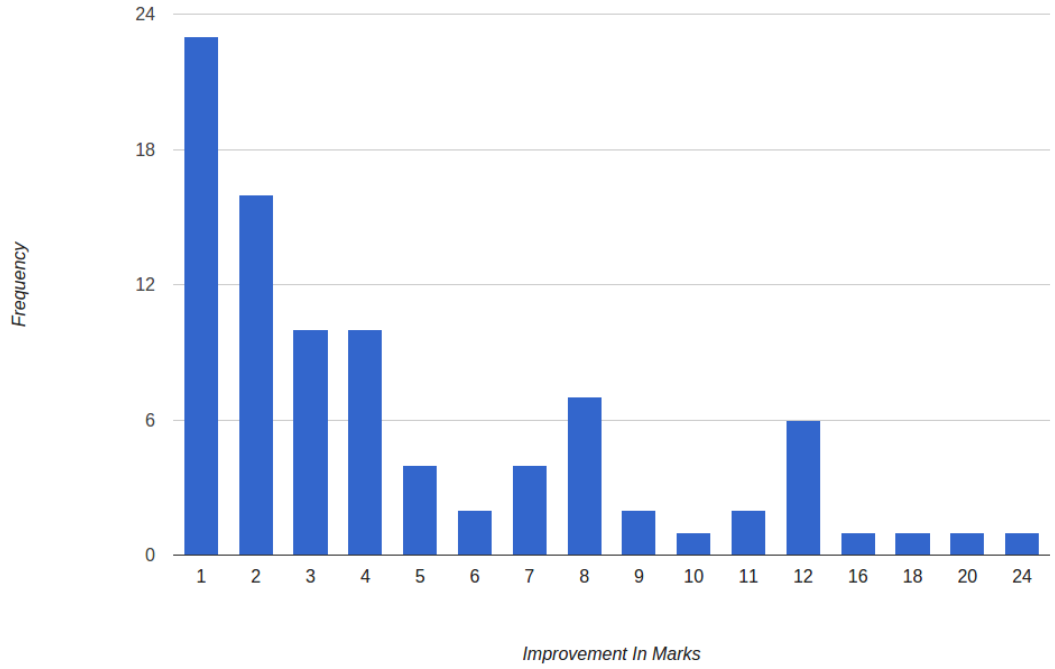


Figure 6.3: Frequency of submissions with improvement in marks

In Figure **??**, we can see the improvement in marks before and after using automated repair system with auto grader. Blue curve shows the marks before and red curve shows the marks after auto repair of submissions.We can observe that many submissions have improvement of 1 to 5 after using auto repair system. This is because our repair system only tries to fix compile time errors, it does not repair logical errors. A compiled program does not necessarily result into a valid submission for a given problem. Table **??** shows the frequency of submissions having improved marks. We observe an average improvement of 4.81 ($\simeq 5$) marks per submission.

6.2.3 Repaired Errors and Frequency

An interesting pattern in error frequency and their types emerge after analyzing the repairs done on students submissions. Many errors which were very common in

beginning labs were rarely occurred in later labs. Also, as new concepts were taught during the course, new errors were made.

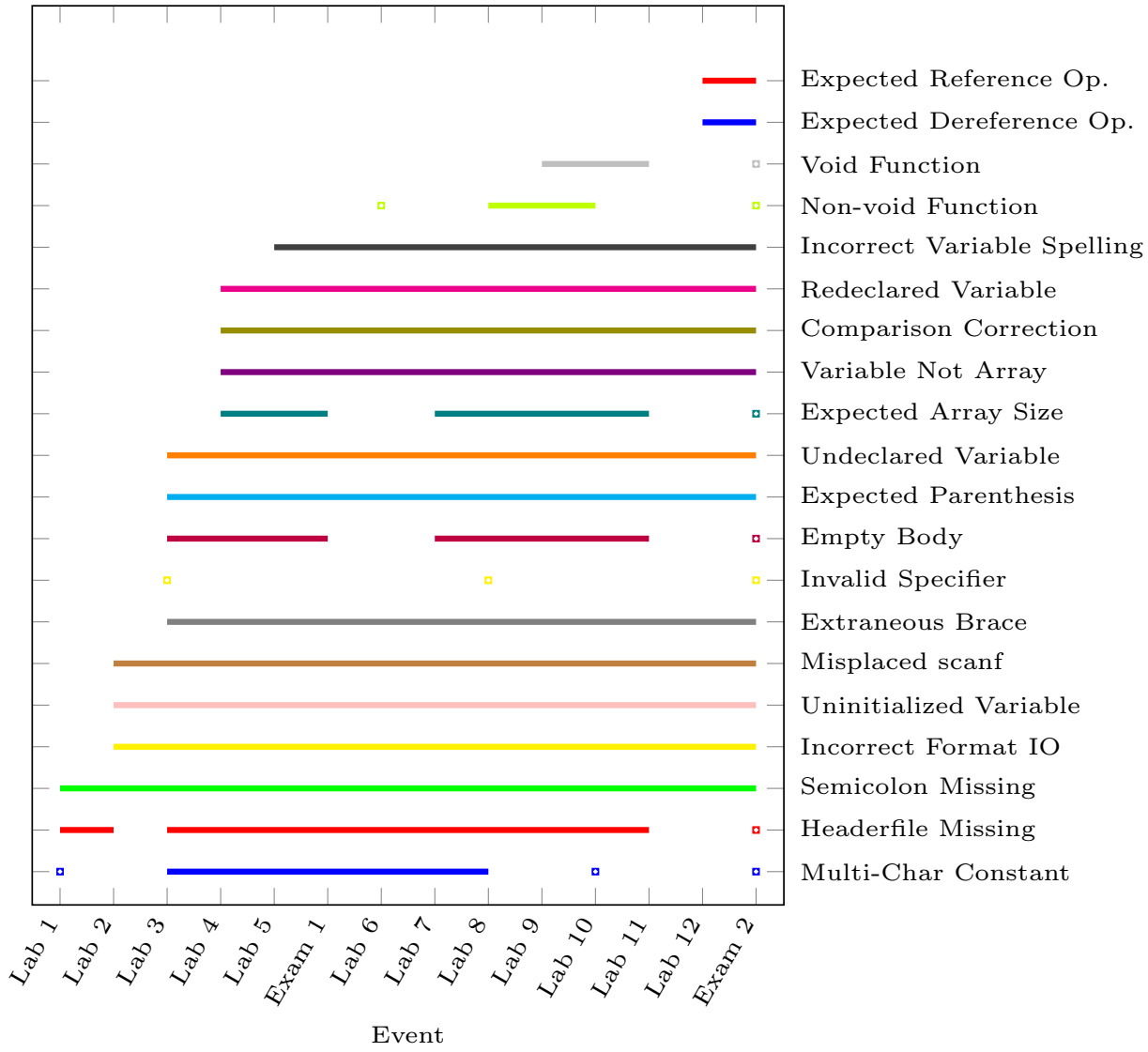


Figure 6.4: Error Statistics

Lab Type	Concepts Taught	Avg. Time(s)	Avg. Marks
Lab 1	Practice	18.21	20
Lab 2	Practice	30.6	12.41
Lab 3	Conditional Statements	80.09	9.11
Lab 4	Loops	73.14	14.64
Lab 5	Array	92.2	11.71

Exam 1	All Concepts till date	87.41	11.88*
Lab 6	Array & Strings	82.47	11.36
Lab 7	Advanced Array & Strings	105.13	7.03
Lab 8	Recursion	101.74	11.58
Lab 9	Pointers	87.89	13.08
Lab 10	Data Structure and Algorithm	97.8	10.81
Lab 11	Linked List	90.51	5.97
Lab 12	Structures & Advanced Linked List	87.98	6.55
Exam 2	All Concepts	118.41	8.67*

*Normalized Marks

Table 6.3: Students performance for each lab event

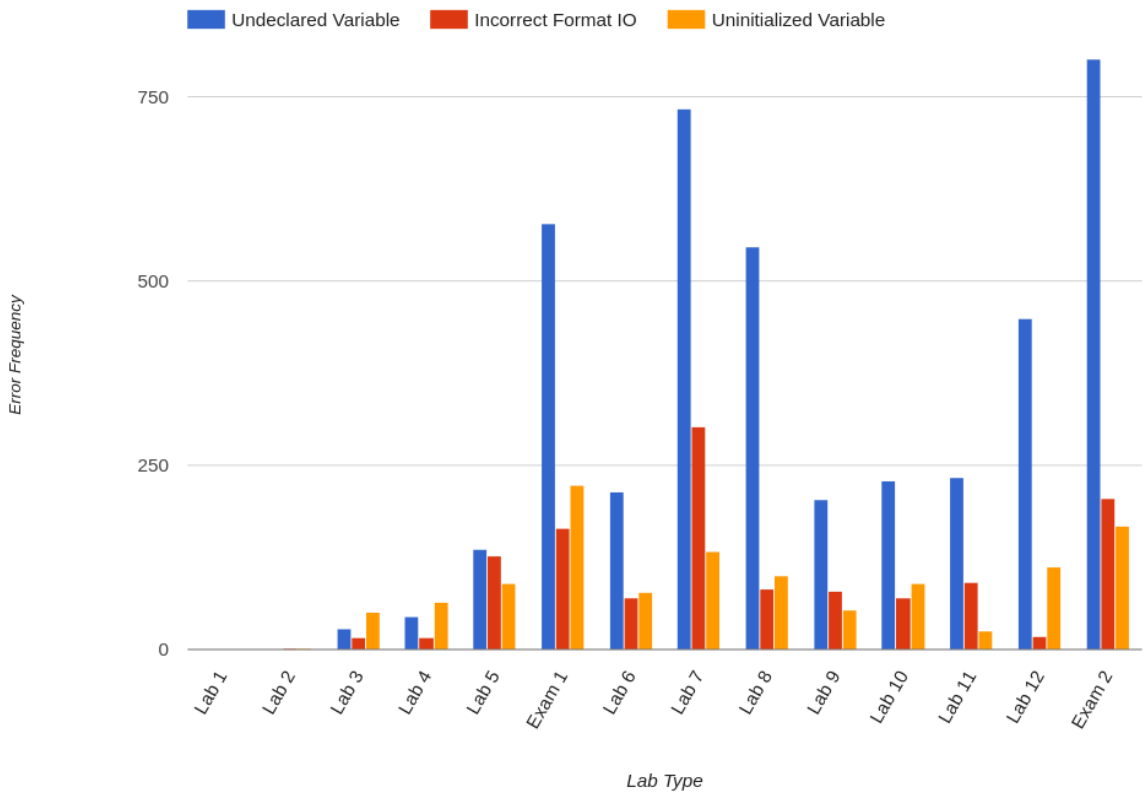


Figure 6.5: Error Frequency for some frequent errors for all lab events

We can notice that many errors were introduced in earlier labs and students continued to be struggling with them. The graph 6.4 represents a time line showing the prevalence of an error in all the labs and exams. A continuous line represents that students are making that error repeatedly. The graph 6.5 shows the frequency of selected errors made by the students in the labs and exams. The increase in frequency of errors in exam 1, exam 2, lab 7 and 8 is due to increase in difficulty level of problems. This also results into the increase in average time taken by students to submit solutions for lab assignments and low average marks for the respective lab events.

This data can be used by instructors and teaching assistants to help students learn better. Instructors can focus on above errors related topics in lectures and lab assignments. Lab assignment problems can be designed to focus more on these areas. Furthermore, the students who are struggling in labs and are continuously making these errors can be identified. Special lectures and more detailed notes can be arranged for those students.

Chapter 7

Conclusions and Future Scope

In this chapter, we will conclude this thesis and discuss about the implications and future scope in Automated repairing of programs.

7.1 Conclusions

Automated Repair System has opened many new avenues to new teaching methods and provided insight into students learning curve at new level. Current system is designed to work in different modes and can be modified easily according to new requirements. With the integration of automated repair system and newly designed architecture, performance and efficiency of automated grading system has improved many times.

Automated repair system can also be modified very easily to provide relevant and real time feedbacks to students. Any submission can be corrected and from those repairs, very custom feedbacks about the errors can be generated. These repair can also be used to correct student code on the fly and thus minimizing the need to teaching assistants.

Our goal at the beginning of this thesis was to design and develop an automated repair system which will help automated grading system to work with unsuccessfully compiled submissions also. Our goal was to design a system which will detect and

repair errors and warnings in student's submissions, so that student's submission can be graded in an automated manner. After conducting various experiments and analyzing the performance we can safely say that automated repair system has made automated grading system performance at par with the grading of teaching assistants. Also we have noticed that automated repairing of programs has made grading bias free and more consistent.

We have designed the automated repairing system in a pluggable model so that existing or new teaching system can be added easily. Automated Repair System can be easily integrated with PRUTOR [Das15] with help of exposed web services. The system can also be used independently without auto grading system to repair any program.

The current system successfully runs with auto grader system and automated feedback system [Dad16] which is an extension to the automated repair system. Performance wise, current grading system can grade around two hundred submissions per minute with automated repairing. The system is designed in pipeline manner which means all operations are independent and there is no single point of failure or bottleneck in all sub components.

There are many advantages of automated repair system, some of which are:

- Integration with automated grading system to enable grading system to grade even unsuccessfully compiled program. This improves the performance of automated grading system many folds.
- Automated feedback system can be designed on top of automated repair system. A student can be provided with feedback explaining what errors he has made with correct code and how to correct them.
- Automated repair system works with correcting those errors which students are making continuously. Instructor can use this data to focus on those topics

with which students are struggling.

7.2 Future Scope

The system offers a wide range of corrections and repairs to programs and is designed to work very efficiently. Yet there are some areas which can be improved to make automated system even better. Some of these areas are:

- **Adding more types of errors to repair system:**

  Current system is designed to work with only syntactical errors. Almost all of these errors are detected at compile time. Current system does not detect runtime errors or semantic errors. After analyzing students submissions, we have noticed that many students are making these type of errors. Designing repair modules for semantic errors will improve the functioning of automated repair system and these improvements could lead to better automated grading and automated feedback system.

- **Repair system for logical errors based on problem statement analysis:**

  The goal of current system was to help student with syntactic errors. We have purposely avoided repairing logical errors because it was felt that helping students with logic of program will hamper their learning process to solve and tackle problems. But discussing with many students and teaching assistants, it is being observed that when student's submission has logical errors due to which his submission is not accepted, he starts making guesses with the program logic and syntax and introduce more errors in program. If the repair system can correct logical errors and give relevant feedback to student, the student can avoid making more errors.

- **Providing solution blueprint with problem statement:**

  When the instructor designs a problem, he also provides a sample solution to

that problem. We could generate a blueprint in form of flowchart of the solution and provide students with feedback as flowchart of the solution. This could help students to understand how to think and design solutions to problems and think in form of code. This feedback can be controlled based on progress made by student and different grading policies can be designed to monitor a student's progress in the lab.

References

[App] Linux apparmor.
URL: https://en.wikipedia.org/wiki/AppArmor.

[bas] Base64.
URL: https://en.wikipedia.org/wiki/Base64.

[BS16] Sahil Bhatia and Rishabh Singh. Automated correction for syntax errors in programming assignments using recurrent neural networks. *CoRR*, abs/1603.06129, 2016.

[Cla] Clang. A c language family frontend for llvm.
URL: http://clang.llvm.org/.

[Dad16] Ziyaan Dadachanji. Automated feedback and grading for programs in introductory programming courses. M.tech. thesis, Indian Institute of Technology Kanpur, India, 2016.

[Das15] Rajdeep Das. A platform for data analysis and tutoring for introductory programming. M.tech. thesis, Indian Institute of Technology Kanpur, India, 2015.

[GRZ14] Sumit Gulwani, Ivan Radicek, and Florian Zuleger. Feedback generation for performance problems in introductory programming assignments. *CoRR*, abs/1403.4064, 2014.

[JGB05] Barbara Jobstmann, Andreas Griesmayer, and Roderick Bloem. Program repair as a game. In *Proceedings of the 17th International Conference on*

Computer Aided Verification, CAV'05, pages 226–238, Berlin, Heidelberg, 2005. Springer-Verlag.

[JSO] Introducing json.
URL: `http://www.json.org/`.

[KB11] Robert Könighofer and Roderick Bloem. Automated error localization and correction for imperative programs. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, FMCAD '11, pages 91–100, Austin, TX, 2011. FMCAD Inc.

[Lin] Linux containers.
URL: `https://linuxcontainers.org/`.

[MOO] Massive open online course.
URL: `https://en.wikipedia.org/wiki/Massive_open_online_course`.

[Par15] Sagar Parihar. Automated grading tool for introductory programming. M.tech. thesis, Indian Institute of Technology Kanpur, India, 2015.

[Per] Computer performance.
URL: `https://en.wikipedia.org/wiki/Computer_performance`.

[POS] Post (http).
URL: `https://en.wikipedia.org/wiki/POST_(HTTP)`.

[Res] System resource.
URL: `https://en.wikipedia.org/wiki/System_resource`.

[SGSL13] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. Automated feedback generation for introductory programming assignments. *SIGPLAN Not.*, 48(6):15–26, June 2013.