Categorization of Frequent Errors in Solution Codes Created by Novice Programmers

Md. Mostafizer Rahman[1,*], *Shunsuke* Kawabayashi[1,**], and *Yutaka* Watanobe[1,***]

[1]Graduate Department of Computer and Information Systems, The University of Aizu, Aizu-Wakamatsu City, Fukushima, Japan.

Abstract. In recent times, e-learning has become indispensable for both technical and general education. Among all the subjects, programming education has drawn attention because of its importance for continuous development in the ICT sector. Finding errors in a solution code is a laborious task for novice programmers, teachers and instructors. Novice programmers are spending a lot of valuable time to search errors in the solution codes. In this paper, a method for the categorization of frequent errors in solution codes is presented. In the proposed method, the differences between wrong solutions and accepted solutions are used to define feature vectors for a clustering algorithm. A longest common subsequence (LCS) algorithm is leveraged to find the differences between wrong and accepted codes, then all the inequalities are converted into feature vectors. The k-mean clustering algorithm is applied to cluster the elements of the feature vector to find the most common errors in solution codes. In our experiment, the method was applied to a set of program solution codes accumulated in an e-learning system. Experimental results show that the proposed method is efficient and capable to detect the most common errors occurred in solution codes that can be helpful for novice programmers to resolve errors quickly as well as useful for teachers to prepare lesson plan.

1 Introduction

Programming is one of the most vital skills in information and communication technology (ICT) as well as computer education. No significant development in ICT is possible without good programming skill. Many countries and educational institutions have given priority to programming for computer science and engineering students. Programming learning is not an easy task for students, because it requires repeated practice to do well in programming. Dealing with errors in program code is also a challenging task for students, especially novice programmers. Error occurs in code for many reasons, including algorithmic, misconceptions, and misinterpretations [1]. Not all errors are easy to find and resolve by students and novice programmers. So, helping students and novice programmers to identify and classify code errors can be important research topics.

Empirical research on detecting and classifying errors in program codes has drawn attention in recent times. Program code is a complex text where many errors are occurred. Most of the syntax errors are identified during compilation. Some recent works have focused on evaluating common syntax errors in the code, improving error detection strategies, improving compiler messages after identifying syntax errors to help students and novice programmers [2–5]. In addition to the compiler, many deep learning techniques are employed to detect syntax errors in program codes that have achieved significant success [6–8].

When we consider errors other than syntax they are called logic errors. Logic error is sometimes called semantic error in code. The program code successfully passes the compilation process when a logic error occurs in the code and gives results, but fails to provide correct results in all possible test cases [1]. Ahmadzadeh et al. [9] explained that the purposeful meaning of programmers is not reflected with the language when such logic errors occurred in codes. Searching for logic errors in codes is frustrating for programmers, where there is no clue or available feedback about the location of the error or the nature of the error. Logic errors in the code are particularly difficult to detect, especially for first-year students (computer science or other departments who have basic programming courses) and novice programmers due to their inadequate error debugging experience, misconceptions, and poor programming knowledge [10–12].

With this motivation we are focusing on identifying and classifying the most common errors, including logic errors in code. We have collected all source codes for the experiment from Aizu Online Judge (AOJ) system [13]. Initially, we have selected paired source codes (wrong → accepted). The source code may have several submissions to be accepted, we collected the last incorrect and accepted submission as a pair of source codes. We have extracted the debugging pattern in the wrong code to be accepted from the paired source codes. First, paired source codes are converted into token sequences using Lexical Analyzer and then each token is encoded with ID. We calculated

*e-mail: mostafiz26@gmail.com
**e-mail: m5231138@u-aizu.ac.jp
***e-mail: yutaka@u-aizu.ac.jp

the difference between paired source codes (wrong → accepted) using the longest common subsequence (LCS) algorithm. All asymmetry properties of paired source code are converted to feature vectors. Finally, k-means clustering algorithm is used to classify the detected errors. We have conducted our experimental work using several hyper parameters and the obtained results of the error classification can be helpful for students, novice programmers, teachers and instructors.

The rest of the paper is organized as follows: Section 2 presents related literature for error identification and classification. In Section 3, model architecture is presented. Section 4 presents experimental results and discussion. Finally, section 5 presents conclusion and future work of the research.

2 Related Literature

Ettles et al. [1] presented a logic error classification model. Students and novice programmers made logic errors in the program code that errors were broadly classified into three types of errors, including algorithmic, misinterpretation, and misconception. About 15,000 logic erroneous code segments (created by students and novice programmers) written in C programming language were analyzed for classification. Most of the logic errors in the codes were caused by misconceptions that reflected a low level of programming knowledge. A similar work was reported [3] where students' ability to resolve syntax errors was examined. The research paper also examined how students spend their time solving very common types of syntax errors and rare types of syntax errors. Targeting time-taking errors by teachers and instructors during class helps students quickly resolve rare types of syntax errors that improve students' productivity.

Intelligent source code evaluation models based on deep neural networks are proposed [6**?** –8]. These models are capable of detecting logic errors as well as syntax errors in codes. Experimental results show that the models are detecting logic errors at a high rate and also providing code repair suggestions. Fitzgerald et al. [10] described that debugging is a challenging task for novice programmers and students. The results of their survey show that good debuggers are more likely to be good programmers but good programmers may not be good debuggers. As they presented, most students made arithmetic errors, incorrect loop conditions, etc. that occurred due to misconceptions and lack of language knowledge.

In [14], numerous syntax erroneous source codes were categorized by manual analysis instead of compiler message analysis. However, this study did not provide a detailed description of the classification of logic errors. Hristova et al. [15] introduced a tool for Java programming called "Expresso" which was developed by faculty members and advanced students. This paper also presented the reasons why "Expresso" is better than other programming tools. The "Expresso" tool has provided a list of typical errors made by novice programmers and students. Altadmri and Brown [16] presented a comprehensive analysis based on about 37 million source codes collected from 250,000 students and novice programmers. Source codes are analyzed based on frequency, spreading of errors, time-to-fix and so on. They found in their analysis that the average time to solve logical (semantic) errors in a semester course is decreasing. They further presented that students and novice programmers faced more difficulties resolving logic errors than syntax errors.

Our research is unique and different from existing research. We focus on identifying frequent errors from paired source codes to investigate error debugging patterns for students and novice programmers. One of our main aims to classify these frequent errors in different groups that help students and teachers in problem solving and teaching. One of our main goals is to classify these frequent errors into different groups that help students and teachers.

3 Proposed Methodology

3.1 Overview

The architecture of the proposed model is presented in Figure 1. At the very beginning, paired (wrong → accepted) source codes are collected from the AOJ system and then the codes are converted into token IDs using Lexical Analyzer. The LCS algorithm is leveraged to find the difference between wrong and accepted codes then all inequalities between paired codes are listed as feature vectors. Finally, the K-mean clustering algorithm is applied to find and classify frequent errors. Similar errors will be categorized in the same category which helps teachers and instructors to know which types of errors occurred in most cases. Accordingly, they can plan programming lectures.

3.2 Word Sequencing of Program Codes

In computer science, lexical analysis, scanner or tokenization is a process used to convert high-level inputs such as web pages, programming codes into sequence of tokens. In the proposed model, we have used lexical analyzer to convert codes into word sequences and then each word is encoded to ID. The process of word sequencing of codes is shown in Figure 2.

3.3 Clustering the Frequent Errors

In the proposed model, the K-means clustering algorithm is leveraged to group the frequent errors in codes [17]. Initially, the Elbow (an optimal k value selection) method has applied to determine the optimal number of clusters for K-means algorithm. Similar errors are located in the same cluster, so the inter-cluster similarity index is lower and the intra-cluster similarity index is higher.

3.4 Verdicts of Online Judge and Paired Code Selection

We collected paired source codes from the AOJ system for our experimental purposes. We have focused on the actual reasons and context of the errors that users made in

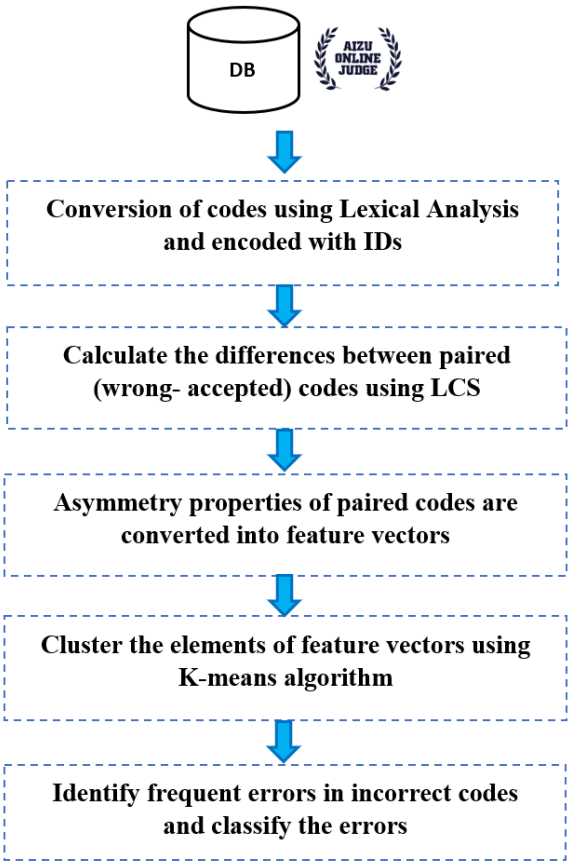


Figure 1. Workflow of the proposed methodology

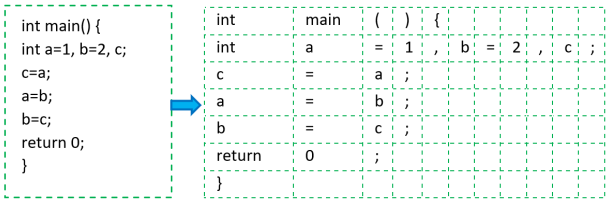


Figure 2. Word sequencing using lexical analyzer

the codes and how users responded to correct the errors in the codes. In an online judge system, different types of verdicts (judgments) are given based on code errors as shown in Table 1.

In an online judge system, students / users can submit their solutions as many as they like or until their solution is accepted, so we have a huge number of trial-error history in our database. We selected the accepted code and the last incorrect code as a pair of source code for experimental purposes as shown in Figure 3.

3.5 Source code conversion into feature vectors

In the previous section, we described that codes are converted into word sequences and then encoded with ID. We leveraged LCS algorithm to calculate the difference between incorrect and accepted codes. All the asymmetry attributes contained in incorrect and accepted codes are

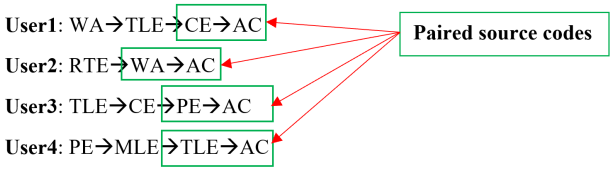
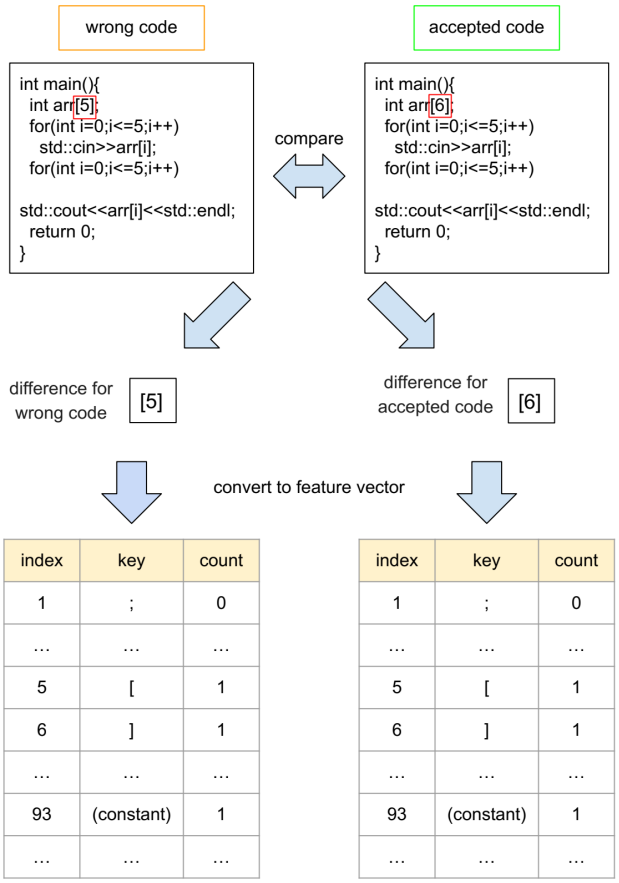


Figure 3. Paired (wrong→accepted) code selection

converted to feature vectors. We have defined 96 feature elements for this experiment. In Figure 4, we present an example of paired code conversion into two separate feature vectors. Similarly, we can convert paired code into a feature vector as well.



index	key	count
1	;	0
…	…	…
5	[	1
6	]	1
…	…	…
93	(constant)	1
…	…	…

index	key	count
1	;	0
…	…	…
5	[	1
6	]	1
…	…	…
93	(constant)	1
…	…	…

Figure 4. Paired source codes are converted into two separate feature vectors

4 Experimental Results

We have performed several experiments with paired source codes collected from AOJ system to evaluate the performance of our proposed methodology. We selected solution codes (both incorrect and accepted) of two problems. The experimental cases are (*i*) clustering in incorrect source codes (*ii*) clustering in accepted source codes (*iii*) clustering in both incorrect and accepted codes. Experimental results and discussions are presented below.

Table 1. Verdicts of the online judge system

Sl.	Name of the Verdict	Descriptions
1	Compile Error (CE)	Compilers are unable to compile the submitted codes.
2	Runtime Error (RTE)	A program is failed to execute, possible causes are pointer value has been exceeded, some values have been divided by zero, stack overflow, etc.
3	Time Limit Exceeded (TLE)	The time allotted for a program has exceeded.
4	Memory Limit Exceeded (MLE)	The memory allotted for a program has exceeded.
5	Output Limit Exceeded (OLE)	A program produces many outputs.
6	Wrong Answer (WA)	A program generates incorrect output during the test cases.
7	Presentation Error (PE)	Output does not match (blank lines, extra spaces, etc.) with the test cases.
8	Accepted	A program has passed all the test cases.

Problem 1:
Reverse Sequence: Write a program which reverses a given string *str*.
Input: (the size of $str \leq 20$) is given in a line.
Output: Print the reversed *str* in a line.
Solution Codes: A total of 922 paired solution codes have been used.

Initially, we performed a clustering operation independently on the incorrect solution codes of Problem 1. We selected the number of optimal clusters using the Elbow method and suggested that the number of optimal clusters for the incorrect solution codes would be 25 as shown in Figure 5 (a). The mean of the error distances of each cluster is shown in Figure 5 (b) and clusters 4, 8, 20, 21 and 23 have higher mean values than the other clusters. Different color properties represent the different elements of the feature vector.

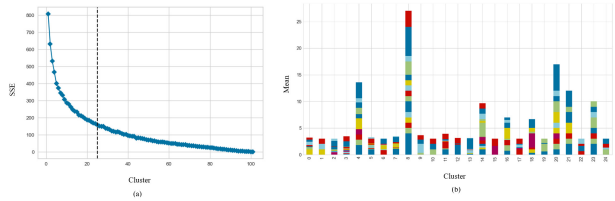


Figure 5. Incorrect codes (a) Optimal number of clusters using Elbow method (b) Mean value of each cluster

We have 25 clusters for incorrect codes, of which we have presented some cluster data in Table 2 where each cluster has the same type of errors. Cluster 1 contains errors related to incorrect array sizes, and array references. Cluster 4 contains errors related to the program "printf" function or output. Similarly, clusters 8 and 12 have errors related to assignment process, operation, etc.

We performed the clustering operation on accepted codes for the same problem described in the problem description (Problem 1). We have 26 clusters and the mean value of each cluster is also calculated as shown in Figure 6. Clusters 0, 4, 7, 8, 20, 22, 23 have the highest mean values among others. In Table 3, we presented some clusters data that users made their accepted solution codes. Clusters 0 and 5 contain corrections related to the *printf* or *output* function and *array*, respectively. Clusters 14 and 17 have corrections related to adding new line feeds as output.

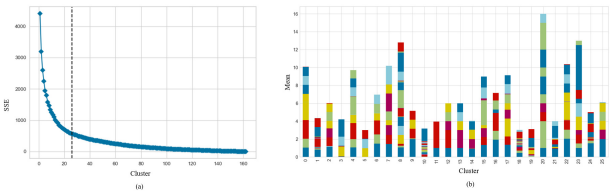


Figure 6. Accepted codes (a) Optimal number of clusters using Elbow method (b) Mean value of each cluster

Considering all the solution codes of Problem 1, we calculated the difference between incorrect and accepted codes and then converted the asymmetry properties to a single feature vector whose dimension is 192 (96 * 2). The clustering operation is performed on the feature vector, the optimal number of clusters 32 generated by the Elbow method. The mean values of each cluster are shown in Figure 7.

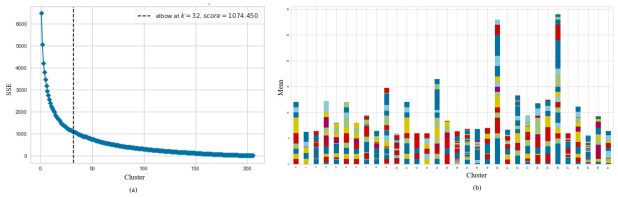


Figure 7. Incorrect and accepted codes of Problem 1 (a) Optimal number of clusters using Elbow method (b) Mean value of each cluster

Table 4 presents cluster data of different clusters where the clustering operation is performed using both incorrect and accepted solution codes. Cluster 3 shows that users made a mistake adding *endl* after the output. In Cluster 13, the users made a mistake in entering the correct array size. Similarly, in Cluster 27, users added incorrect numbers and then corrected them. On the other hand, the

Table 2. Clustering in incorrect solution codes of Problem 1

Cluster	Detected Errors	Descriptions
1	[20], [21], [10], [i], +2]	Wrong array size, array reference
4	printf (" % d \n " , 1) ; printf (" % s \n " , str) ; printf (" % d \n " , num) ; % d " , i) ; printf (" % c	Error around printf statement
8	for (i = 0 ; i < 20 ; i ++) in [i] = ' \0 ' ; scanf for (i = 0 ; i < 22 ; ++ i) str [i] = ' \0 ' ; fgets	Error related with assignment process
12	= 21 ; = 18 ; = 20 ; j = 0 ; = - 1	Assignment operation

Table 3. Clustering in accepted solution codes of Problem 1

Cluster	Changes in accepted codes	Descriptions
0	} printf (" \n ") ; return, } printf (" \n ") ; }, ; printf (" \n ") ; }, } printf (" \n ") ; return 0 ; }, } fprintf (fout , " \n ") ; return, } printf (" % s " , " \n ") ; return	Add or changes in printf function or output
5	[100], [21], [22], [j], [k], + 1]	Changes around array sizes and value
14	s « endl ; t « endl ; str « endl ;)) « endl ;] « endl ;) « endl ; « str « endl; c « endl ; a « endl ;; return	Added new line for output
17	} cout « " \n " ; return ; cout « " \n " ; } ; cout « " \n " ; return } cout « " \n " ; } ; std :: cout « " \n " ; return	Added new line for output

cluster 16 shows miscellaneous miscalculations in incorrect codes.

Table 4. Clustering in both incorrect and accepted solution codes

Clust.	Errors in incorrect codes and Changes in accepted codes	Descriptions
3	WA: } return AC: } std :: cout « std :: endl ; return WA: ; } AC: ; std :: cout « std :: endl ; } WA: } return AC: } std :: cout « "" « std :: endl ; return WA: } } AC: } std :: cout « std :: endl ; } WA: ; return AC: ; std :: cout « std :: endl ; return	Users made mistake to add new line feed as output.
13	WA: [20], AC: [100] WA: [20], AC: [21] WA: [21], AC: [22] WA: [i], AC: [j] WA: [i], AC: [k] WA: + 2], AC: + 1]	Wrong array size, array reference, etc.
16	WA: i > 10, AC: i - 10 WA: i > 0, AC: i ≥ 0 WA: f > 1, AC: f ≥ 1 WA: i > 10, AC: i ∧ 10 WA: length + 1, AC: length - 1 WA: 2 > I, AC: 2 < i WA: i > 10, AC: i ≠ 10	Miscalculation of constants and variables
27	WA: = 21 ; AC: = 20 ; WA: = 18 ; AC: = 19 ; WA: = 20 ; AC: = 19 ; WA: - 2 ; AC: - 1 ; WA: - 3 ; AC: - 2 ; WA: - 1 ; AC: - 2 ; WA: + 19 ; AC: + 41 ; WA: + 40 ; AC: + 41 ;	Incorrect number value

Problem 2:
Print a Rectangle: Draw a rectangle which has a height of H cm and a width of W cm. Draw a 1-cm square by single '#'.
Input: The input consists of multiple datasets. Each dataset consists of two integers H and W separated by a single space. The input ends with two 0 (when both H and W are zero).
Output: For each dataset, print the rectangle made of $H \times W$ '#'. Print a blank line after each dataset.
Solution Codes: A total of 1173 paired solution codes have been used.

We used both incorrect and accepted solution codes of Problem 2 for clustering, the Elbow method shows the optimal number of clusters is 2 as shown in Figure 8. The mean value of Cluster 3 is much higher than other clusters.

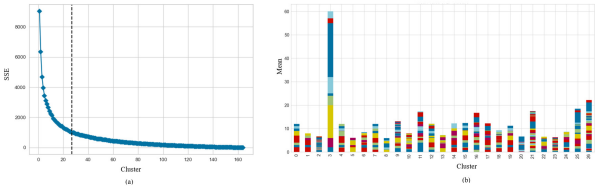


Figure 8. Incorrect and Accepted codes of Problem 2 (a) Optimal number of clusters using Elbow method (b) Mean value of each cluster

Clustering operation has been performed using both incorrect and accepted solution codes of Problem 2. Some cluster data is presented in Table 5. After analyzing the cluster data, we found that many users made common mistakes in the solution codes.

5 Conclusion

In this paper, we have proposed a method for the categorization of frequent errors in solution codes. We used

Table 5. Clustering in both incorrect and accepted solution codes of Problem 2

Cluster	Errors in incorrect codes and Changes in accepted codes	Descriptions
0	WA: } } AC: } printf (" \n ") ; } WA: } } AC: } printf (" % c " , ' \n ') ; }	Missing to print new lines in incorrect codes that are modified into accepted codes
4	WA: } } AC: } putchar (' \n ') ; } WA: } scanf AC: } putchar (' \n ') ; scanf WA: } return AC: } putchar (' \n ') ; return WA: } } AC: } cout « ' n ' ; }	Missing to print new lines in incorrect codes that are changed into accepted codes
16	WA: { for AC: { if (a == 0 && b == 0) break ; for WA: { for AC: { if (H == 0 && W == 0) break ; for WA: { for AC: { if (H == 0 && W == 0) break ; for WA: { for AC: { if (x == 0 || y == 0) break ; for WA: { for AC: { if (H == 0 & W == 0) break ; for WA: { REP AC: { if (H == 0 && W == 0) break ; REP WA: { drawRect AC: { if (H == 0 && W == 0) break ; drawRect	"Break" statement is missing in incorrect codes that are corrected into accepted codes.
20	WA: h ≤ H AC: h < H WA: l ≤ k AC: l < k WA: count ≤ z AC: count < z WA: d , b AC: d < b WA: x > w AC: x < w WA: i = H AC: i < H	The comparison operation was incorrect in the wrong codes that changed to accepted codes.
10	WA: } cout « endl ; return AC: } return WA: ; cout « endl ; } AC: ; } WA: } cout « endl ; for AC: } for WA: } cout « endl ; if AC: } if WA: { cout « endl ; return AC: { return WA: } cout « endl ; } AC: } }	Wrong output line ("endl") was added to incorrect codes that deleted in accepted codes.

paired source codes (incorrect → accepted) for our experiments. We have analyzed the reasons behind the errors, we find the most common errors in the solution codes made by the users. We also analyzed the correction process in the accepted solution codes. Experimental results show that the model can detect common errors made by users and cluster errors on the basis of similarity. The outcome of this research paper can be useful for students as well as teachers and instructors. Teachers can prepare accurate lecture plans to help students, so that they can solve common mistakes in solution codes. In future work, automatic correction of incorrect codes can be interesting.

Acknowledgement

This research work was supported by the JSPS Kakenhi grant number 19K12252.

References

[1] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. 2018. Common logic errors made by novice programmers. In Proceedings of the *20th Australasian Computing Education Conference (ACE '18)* . Association for Computing Machinery, New York, NY, USA, 83–89.

[2] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In Proceedings of the *47th ACM Technical Symposium on Computer Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 126–131.

[3] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. 2012. All Syntax Errors Are Not Equal. In Proceedings of the *17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '12)*. ACM, New York, NY, USA, 75–80.

[4] Raymond S. Pettit, John Homer, and Roger Gee. 2017. Do Enhanced Compiler Error Messages Help Students? Results Inconclusive. In Proceedings of the *2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 465–470.

[5] James Prather, Raymond Pettit, Kayla Holcomb McMurry, Alani Peters, John Homer, Nevan Simone, and Maxine Cohen. 2017. On Novices' Interaction with Compiler Error Messages: A Human Factors Approach. In Proceedings of the *2017 ACM Conference on International Computing Education Research (ICER '17)*. ACM,New York, NY, USA, 74–82.

[6] Md. Mostafizer Rahman, Yutaka Watanobe, and Keita Nakamura. 2020. A Neural Network Based Intelligent Support Model for Program Code Completion. *Scientific Programming*, vol. 2020, Article ID 7426461, 18 pages.

[7] Md. Mostafizer Rahman, Yutaka Watanobe, and Keita Nakamura. 2020. Source Code Assessment and Classification Based on Estimated Error Probability Using Attentive LSTM Language Model and Its Application in Programming Education. *Applied Sciences*, vol. 10, no. 8: 2973.

[8] Md. Mostafizer Rahman, Yutaka Watanobe, and Keita Nakamura. 2021. A Bidirectional LSTM Language Model for Code Evaluation and Repair. *Symmetry*, 13(2):247.

[9] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. 2005. An Analysis of Patterns of Debugging Among Novice Computer Science Students. SIGCSE Bull. 37, 3 (June 2005), 84–88.

[10] Sue C Fitzgerald, Gary Lewandowski, Renée Mccauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education*, June 2008, 93–16.

[11] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A Multi-national, Multi-institutional Study of Assessment of Programming Skills of First-year CS Students. SIGCSE Bull. 33, 4 (Dec. 2001), 125–180.

[12] James C. Spohrer and Elliot Soloway. 1986. Novice Mistakes: Are the Folk Wisdoms Correct? Commun. ACM 29, 7 (July 1986), 624–632.

[13] Yutaka Watanobe. Aizu online judge. https://onlinejudge.u-aizu.ac.jp. (2018)

[14] Davin McCall and Michael Kölling. 2014. Meaningful categorisation of novice programmer errors. In the Proceedings of the *2014 IEEE Frontiers in Education Conference (FIE)*, Madrid, 2014, pp. 1-8.

[15] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. SIGCSE Bull. 35, 1 (Jan. 2003), 153–156.

[16] Amjad Altadmri and Neil C.C. Brown. 2015. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In Proceedings of the *46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 522–527.

[17] Md. Mostafizer Rahman and Yutaka Watanobe. 2019. An efficient approach for selecting initial centroid and outlier detection of data clustering. In Advancing Technology Industrialization Through Intelligent Software Methodologies, Tools and Techniques; IOS Press: Amsterdam, The Netherlands, 2019; vol. 318, 616–628.