Common Errors in High School Novice Programming

Radaković, Davorka; and Steingartner, William

Abstract:

Identifying and classifying the commonness of errors made by novices learning to write computer programs has long been of interest to both: researchers and educators. When teachers understand the nature of these errors and how students correct them, instruction can be more effective. Some errors occur more frequently than others. In this paper, we examine the most common programming errors made by beginning first-year high school gifted mathematics students in Mathematical High School. Notwithstanding the extensive coverage of these error types in lectures and learning materials, we found that these errors still occur when students write programs. Our results suggest that students who habitually make all common errors have lower grades, but even excellent students make logical errors in loop conditions. Therefore, we advise more practice in logical reasoning for novice programmers and an introduction to formal semantics.

Index Terms: ***Computer Science Education, C# Language, Errors, Novice Programmers, Programming***

Manuscript received April 1, 2023.

This work was supported by the project 048TUKE-4/2022 – "Collaborative virtual reality technologies in the educational process", and by the project 030TUKE-4/2023 – "Application of new principles in the education of IT specialists in the field of formal languages and compilers" both granted by the Cultural and Education Grant Agency of the Slovak Ministry of Education.

Davorka Radaković (Corresponding author) is with the Faculty of Sciences, University of Novi Sad, Serbia (e-mail: davorkar@dmi.uns.ac.rs).

William Steingartner is with the Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia (e-mail: william.steingartner@tuke.sk)

1. Introduction

In today's world, computer science education is an important part of the curriculum STEM (Science, Technology, Engineering, and Mathematics). Students exposed to STEM content at an early age typically expand their interest in STEM subjects through elementary and high school and into the faculty level [1, 2]. Computer science education offers an abundance of new learning concepts and opportunities across all domains. Computational thinking in education prepares today's students to live and work in a fully digitized world [3].

Programming is an essential part of Computer science education, but learning programming is sometimes very difficult for first-year high school students. Therefore, students produce significant errors in their code when faced with difficulties in learning programming [4, 5]. The analysis of students' common errors is necessary for computer science professors to understand students' problems in learning programming. Accordingly, a computer science professor needs to obtain expertise in the subject matter and pedagogical knowledge of teaching the subject's content.

Due to the worldwide COVID-19 pandemic, teaching in the past two years has been conducted mainly in the home environment. Face-to-face teaching in schools was substituted with teaching online, using online platforms as new virtual classrooms. This pandemic brought remarkable disruption to education, as well as to High School Education, locally and all over the world [6, 7, 8, 9, 10, 11].

Motivated by this lack of constant face-to-face student-teacher interaction, in [12] we analyzed high school students' solutions to programming exercises in an introductory C#

course and identified the most common errors made by high school students. We used data collected during a first-year introductory course in C# programming to determine what mistakes are commonly made by high school mathematics students in two generations. One generation attended a class before the pandemic COVID-19 and the second generation attended a class during the pandemic COVID-19. We considered all the different types of errors that led to an incorrect solution. In addition, we investigated correlations between students' error-related behavior and their performance in the introductory C# programming course.

We investigate the following research questions:

- RQ1 Which are the most common errors made by high school students?
- RQ2 Is there a significant difference between errors made in students' tasks done in face-to-face and online classes?
- RQ3 Is there a significant difference between errors made in students' tasks done in face-to-face classes before pandemic and post pandemic?

In this paper, we extend our previous work [12] in the following ways:

- We added one more generation to our study, the generation that attended classes in a face-to-face setting after COVID-19 and been taught online for more than two years;
- We detail the students' programming errors;
- We extended Related Work;
- We extended Results with statistical analysis.

Our work is innovative in several respects. First, the population of the course is gifted mathematicians in their first year of high school mathematics. Second, there was an inconsistency between the errors identified in papers by the researchers and those errors experienced by the high school scholars.

The paper is organized as follows: the next section reviews related work. Section 3 describes the methodology used for detecting common programming errors. Section 4 illustrates the performed analysis and the obtained results. Lastly, Section 5 outlines the conclusion and future work on the presented topic.

2. Related Works

A brief overview of the most closely related research presented in this section covers the categorizations of programming errors and the tools developed to help new programmers.

In the last few years Computer Science (CS) education has increased interest as a new school subject, and programming is a key part of computer science and computing [13]. Learning to program involves the acquisition of complex new knowledge and related strategies and pragmatic skills [14].

Ko and Myers in [15] gave the categorizations of programming errors linking the causes of errors. Identifying and helping to correct Java programming errors for Introductory Computer Science students using the education tool, Expresso, made for Java programming is presented in [16]. This interactive tool generates error messages and additionally provides instructions on how to fix the code. The main purpose is to be used all along the beginning process of learning programming and for students to become more skilled with Java and gain a better comprehension of the essential programming concepts.

In 2005, Jackson, Cobb and Carver presented an integrated semantic and syntax error pre-processing system to help new programmers decipher the otherwise cryptic compiler error messages so that they can focus more on design issues than implementation issues used in the United States Military Academy for an introductory programming course [17].

In [18], the authors investigated the types of errors students most often make when writing short fragments of Java code using the CodeWrite exercise tool. They also investigated how much time students spend solving the most common syntax errors. It was found that certain types of errors are not solved faster by stu-

dents with higher skills. It was also found that these errors waste a large portion of students' time, suggesting that targeted instructional interventions can significantly increase student productivity.

The frequency of diagnostic reports in [5] shows a relatively high occurrence of the diagnoses "cannot find symbol", "';' expected", "')' expected", and "Illegal start of expression". The results are similar to the study [17]. In a recent study [4], a data-driven approach was implemented to identify the most common errors made by Chinese students in a Java-based introductory programming course using data collected by an automated assessment tool "Mulberry". The students' error-related behaviors were further analyzed and their relationship to success in introductory programming was examined. The study suggests that students' competency in improving their code is important to their success in introductory programming. Also, in [19] authors explored the effectiveness of Enhanced Programming Error Messages (EPEMs) in a Python-based introductory programming course in the middle school using an automated assessment tool Mulberry.

Ettles, Luxton-Reilly, and Denny analyzed 15.000 code fragments [20] created by novice programmers that contain logic errors. They classified the errors as algorithmic errors, misinterpretations of the problem, and basic misunderstandings. They also found that misunderstandings are the most common source of logic errors and lead to the most complicated errors for students to fix.

In [21], a method for categorizing common errors in solution codes was proposed. The authors used paired source codes (false-accepted) for these experiments. The LCS (Longest Common Sub-sequence) algorithm is influenced to find the differences between false and accepted codes.

With the development of the internet, there is an increase in online learning resources. Massive Online Open Courses (MOOC) obtain suitable learning resources for students adjusting the learning resources recommended to each student according to the student's learning style or knowledge mastery [22]. Also, faculties develop software tools which help students to master the subject's material as the visualization in modern teaching is of extensive importance [23, 24, 25, 26].

Due to the COVID-19 pandemic, many studies confirm the learning loss caused by school closures and online learning [6, 7, 10, 22, 27, 28]. In addition to this confirmation, the studies also propose solutions to make up for the missed knowledge: consolidating the curriculum, increasing instructional time, or improving learning skills by helping teachers apply structured pedagogy and targeted instruction.

3. Methodology

In this section, we describe the design of our study, which aims to find out which errors are commonly made by students in the first year of a mathematics high school compared to the three generations before, during, and after COVID-19 the pandemic.

3.1 Context

We analyze the data collected during a first-year programming course for gifted mathematicians at the Mathematical Grammar School "Jovan Jovanović Zmaj" (abbreviated: JJZ), Novi Sad, Serbia. The school has a specialized curriculum which ensures that teaching in mathematics, physics and computer science is delivered on a highly advanced level.The students are carefully selected through a specific admission process, which includes a specialized entry exam and the assessment of previous achievements.

The introductory course in C# programming was held five school hours per week. The main topics covered in this course were program structure, input/output, variables and operators, conditionals and loops. In the course during the first semester (i.e. first four months), students created simple C# programs, working on a variety of programming tasks.

Students were exposed to active-learning pedagogy and tasked with interacting with their teacher in a class. All programming tasks were demonstrated from scratch on the computer. Unfortunately, the classrooms have only one computer on the teacher's desk. Therefore, at

a beginning of the school year students had at least 5 hours block in a computer classroom when they learned how to write a C# program, to compile it and run it in different environments: Visual Studio, JDoodle, command line. In this way students could test their solutions in computer at home.

Three separate surveys were conducted, one before the pandemic COVID-19 in the 2019/2020 school year; one during the pandemic in 2020/2021 and one after the pandemic in 2022/2023.

3.2 Participants

Research results presented in the paper are obtained from data sample of a total of 53 students: the first group (2019/2020 school year) consisted of 18 students, the second (2020/2021 school year) of 19 students, and the third (2022/2023 school year) of 16 students. Some students had completed seventh and eighth grade at the Mathematical Grammar School: 10 from the first group, 15 from the second group (in 2018/2019 school year there were formed 3 classes for the 7th grade) and 8 from the third group. Thus, they had prior knowledge of programming. Table (Table 1) shows descriptive statistics concerning the number of students participating in our survey.

Students who completed 7th and 8th grade in JJZ learned basics of C#. Further, some students that participated in competitions had experience with C++, and some students learned Python in elementary school. There was no pre-tests for measuring students' existing programming knowledge.

Table 1: Participants by generation

Generation	Number of students	7th and 8th grade finished in JJZ
2019/2020	18	10
2020/2021	19	15
2022/2023	16	8

The gender composition of the classes of each generation are presented in the Table 2. Also, this table has for each generation a number of males and females who finished 7th and 8th grade in "Jovan Jovanović Zmaj".

Table 2: Gender composition of the classes

Gender	Generation 2019/2020	Generation 2020/2021	Generation 2022/2023
M	11	8	9
F	7	11	7
7th and 8th grade finished in JJZ			
M	7	6	3
F	3	9	5

The average age of the students was 15 years as students had to have by the beginning of the school year at least 14 and a half years. The most students came from various regions of Vojvodina, the north part of Serbia, and smaller number came from the central Serbia. There is also similar school in Belgrade: Mathematical Grammar School specialized for students talented in mathematics, physics and computer science.

3.3 Procedure

An important part of learning are evaluations, mostly taken as written tests or exams. Accordingly, all data collection was conducted using the students' written exams. Thus, the students solved their tasks by writing on paper, without the opportunity to compile the program. Therefore, the students did not have the help of the compiler to warn them of errors, as is common in other studies [5, 15, 16, 18, 29]. In addition, all solutions were checked manually. We analyzed the number of occurrences of the eight common type of errors.

We analyzed the solutions from the first three written exams, i.e. data analysis was based on 159 student solutions. The structure of the written exams is as follows:

- The first written exam: two theoretical tasks and three programming tasks with the following parts:
  - define variables;
  - load variables;

  - calculate some standard value e.g. area, volume;
  - control flow statements;
  - print answer.
- The second written exam had five programming tasks:
  - loading variables under a condition;
  - loops, nested loops;
  - counting with conditions.
- The third exam had three theoretical tasks and three programming tasks:
  - loading variables under a condition;
  - loops, nested loops;
  - counting with conditions;
  - efficient summarization; and
  - finding the n^{th} member of the sequence iteratively.

5. Write a program that computes the function

$$f(x) = \begin{cases} x^5 + \sin(x), & x \geq 6 \\ \frac{6+x}{x-5}, & \text{otherwise} \end{cases}$$

Figure 1: Task in the 1^{st} exam.

5. Write a program that loads a number $n \in (2, 10)$ and prints:

$$\text{for } n = 3, \quad \begin{matrix} 1 & 0 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 9 \end{matrix}$$

Figure 2: Task in the 2^{nd} exam.

5. Write a program that computes the n^{th} element in series of numbers, given by the formula:

$$a_1 = 4, a_2 = -2, a_3 = 3, a_4 = 6,$$
$$a_n = -3a_{n-2} + a_{n-4}$$

Figure 3: Task in the 3^{rd} exam.

The loading variables under a condition means the loading variables inside a loop under a condition while the condition is not met. Figures 1-3 show the translation of the original tasks in exams (written in Serbian) into English. For each exam, there is given an example of a task. The first exam had only one task with logical conditions, while in other exams each task had at least one use of logical conditions. Also, the first exam does not cover loops and type casting.

3.4 Categories of Programming Errors

Our review of the literature revealed that researchers use a variety of methods to identify common errors. Commonly accepted categorizations of errors include:

- lexical errors (when a sequence of characters that does not match the pattern of any token),
- syntactic errors (when the rule of the language writing techniques or syntax has been broken),
- semantic errors (when a sentence is syntactically correct but has no meaning, e.g. wrongly typed), and
- logical errors (even if the syntax and other factors are correct, we may not get the desired results due to logical issues).

Furthermore, these errors can be either static (compile-time) or dynamic (run-time) in nature. However, the nature of the errors is highly dependent on the task at hand [16].

Logical errors are the most difficult of all error types to detect since they cannot be identified by the compiler. Typically, where there are situations where the programmer's code compiles and executes successfully, but does not produce the proposed output for all possible inputs [20], those situations are considered as logical errors. Therefore we consider as suitable to provide some foundations from formal semantics of programming languages to high school students because it offers them valuable tools for critical thinking, language comprehension, logical reasoning, and error prevention, mostly logical errors.

4. Results

In this section, we focus on the results and observations obtained.

4.1 List of Common High-school Student Errors

The following errors were identified in our research made by both groups of students:

- The comparison operator (==) and the assignment operator (=);
- Unbalanced:
  - parentheses '(', ')';
  - square brackets '[', ']';
  - curly brackets '{', '}'; and
  - quotation marks.
- Inserting a semi-colon after the parentheses defining conditions in the if, for, foreach, or while constructs.
- Separating the for-loops with commas (',') instead of semi-colons (';').
- The equal sign in front of:
  - the greater-than sign (or) ('=>') for a greater than or equal; and
  - the (or less than or equal) ('=<'), instead of following them.
- Improper type casting, i.e. missing of the cast.
- Logical errors in loop conditions.

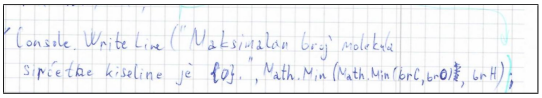

Figure 4: Example of unbalanced or missing brackets '[', ']', '(', ')', '{', '}'.

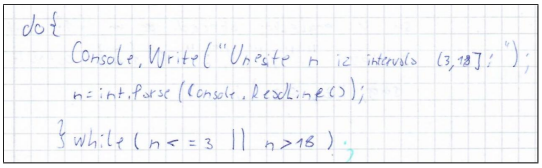

Figure 5: Example of missing semicolon ';' at the end of statement.

Figure 4 shows an example of unbalanced or missing some of the brackets '[', ']', '(', ')', '{', '}'. Figure 5 shows missing semicolon ';' at the end of statement. Logical errors are often the most general errors [16] and Figure 6 shows one of the typical ones.

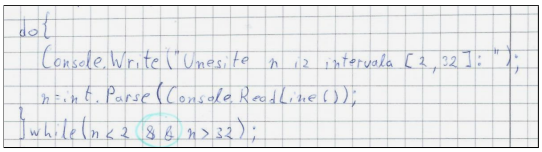

Figure 6: Example of logical errors.

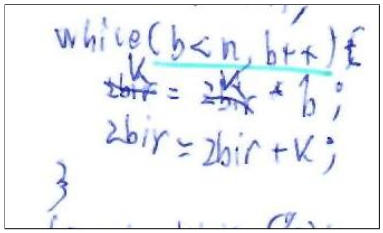

Figure 7: Example of incorrect syntax.

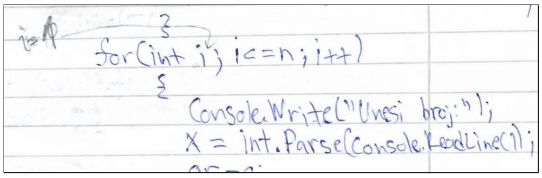

Figure 8: Example of an uninitialized variable.

Example of incorrect syntax is given in Figure 7, and Figure 8 presents uninitialized variable in the for-loop.

4.2 Results and Discussion

To analyze all the obtained results, the Statistica version 14.1 is used [30]. The Factorial Analysis of Variance (ANOVA), LS Means, and Posthoc Tukey HSD tests were used to analyze the number of errors in all three generations. The statistical significance level is set to $p < 0.05$.

To answer RQ1, we show the overview of the occurrence of errors in the all three written exams for all tree generations in tables: Table 3, Table 4 and Table 5. From the data given in the tables we can see that the first and the second generation have less errors than the third generation.

Table 3 presents details about the number of errors of the generation 2019/2020. If we

compare the number of errors and number of students which made these errors we can see that there is much higher number of mistakes than students which made them, e.g. 20 mistakes to 5 students, or some mistakes are made only once per student. In the same manner, the Table 4 from the generation 2020/2021, gives the similar findings.

Table 5 has much higher occurrence of errors and they are made by larger number of students: 1st task 34 errors (17 students), 2nd task 42 errors (23), and 3rd task 33 errors (20).

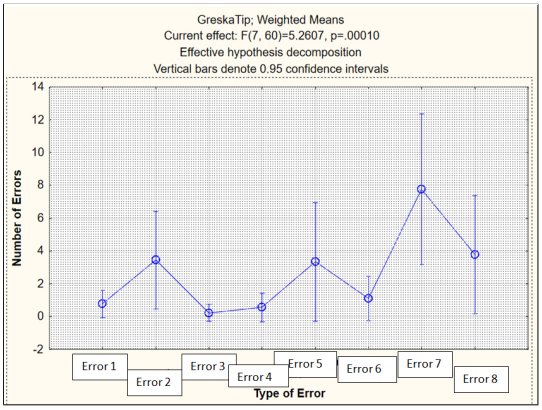

Figure 9: LS Means of number of errors by types of errors.

Figure 9 shows that Incorrect syntax has the biggest occurrence, and second are Logical errors in loop conditions. Errors are numbered in the same order as they are listed in Tables 3, 4, and 5.

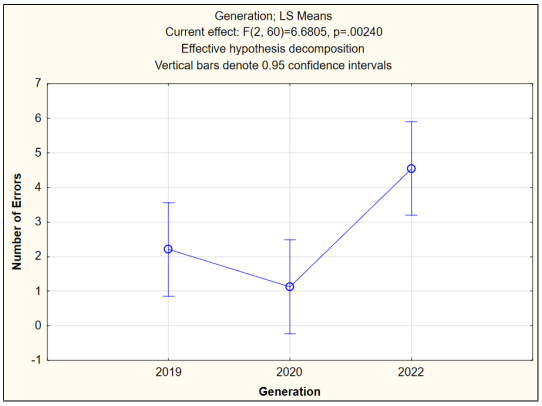

Figure 10: LS Means of number of errors by generations.

Figure 10 shows LS Means of number of errors in all tree generations, with $p = 0.00240$. It confirms that generations 2019 and 2020 have similar occurrence of errors, while generation 2022 has larger number of errors. Therefore, to answer RQ2 results show that we have similar results for face-to-face generation 2019 and online generation 2020, but face-to-face generation 2022 has a significant difference with both 2019 and 2020 generations. Hence, RQ3 is confirmed.

Univariate Tests of Significance for Number of Errors (Characteristics)
Sigma-restricted parameterization
Effective hypothesis decomposition

Effect	SS	Degr. of Freedom	MS	F	p
Intercept	496.1250	1	496.1250	45.29864	0.000000
Generation	146.3333	2	73.1667	6.68048	0.002402
ErrorType	403.3194	7	57.6171	5.26072	0.000101
Test	4.0833	2	2.0417	0.18641	0.830409
Error	657.1389	60	10.9523		

Figure 11: Univariate Tests of Significance for Number of Errors.

Univariate Tests of Significance for Number of Errors in Figure 11 show that factors Generation and Type of Error are statistically significantly different from each other with $p < 0.05$ (Generation $p = 0.002402$; Type of Error $p = 0.000101$).

Tukey HSD test; variable Number of Errors (Characteristics)
Approximate Probabilities for Post Hoc Tests
Error: Between MS = 10.952, df = 60.000

Cell No.	Generation	2019 2.2083	2020 1.1250	**2022 4.5417**
1	2019		0.497250	0.045660
2	2020	0.497250		0.002082
3	2022	0.045660	0.002082	

Figure 12: Tukey HSD test for Generations.

Tukey HSD test; variable BrojGresaka (Characteristics)
Approximate Probabilities for Post Hoc Tests
Error: Between MS = 10.952, df = 60.000

Cell No	Type of Error	Error 1 .77778	Error 2 3.4444	Error 3 .22222	Error 4 .55556	Error 5 3.3333	Error 6 1.1111	Error 7 7.7778	**Error 8 3.7778**
1	Error 1		0.681311	0.999964	1.000000	0.725647	0.999999	0.000934	0.541171
2	Error 2	0.681311		0.448561	0.588384	1.000000	0.806745	0.120320	0.999999
3	Error 3	0.999964	0.448561		0.999999	0.494357	0.999145	0.000348	0.322059
4	Error 4	1.000000	0.588384	0.999999		0.635332	0.999964	0.000616	0.448561
5	Error 5	0.725647	1.000000	0.494357	0.635332		0.842388	0.102297	0.999992
6	Error 6	0.999999	0.806745	0.999145	0.999964	0.842388		0.001796	0.681311
7	Error 7	0.000934	0.120320	0.000348	0.000616	0.102297	0.001796		0.189826
8	Error 8	0.541171	0.999999	0.322059	0.448561	0.999992	0.681311	0.189826	

Figure 13: Tukey HSD test for Types of Errors.

The Posthoc Tukey HSD test for Generations obtained that generations 2019 and 2022 have statistically significantly different results ($p = 0.0456660$), as well as generations 2020 and 2022 ($p = 0.002082$) (Figure 12). Also, the Posthoc Tukey HSD test for Types of errors (Figure 13) obtained that error incorrect syntax

is statistically significantly different from errors: (==) vs (=); unbalanced or missing quotation marks; separating the for loops with commas instead of semi-colon and Improper type casting.

Further, the errors of unbalanced or missing brackets, quotation marks, and missing semicolons at the end of statements can be caused by sloppiness as discussed in [31]. If the students have done these tasks on computer they would corrected them at the most cases.

Furthermore, we found that students who write their code neatly usually do not have unbalanced errors. As can be seen from tables, another common error (up to 10%) was that conditions in the if-else statement were incorrect because the comparison operator was replaced with the assignment operator. Also, inserting a semicolon before an if, for, foreach, or while block was common among 10% of the students.

Surprisingly, the most common errors were logical errors in loop conditions, made by 30% of the students in the initial evaluations, even though this type of error was consistently covered in the lectures. If we compare 1^{st} and 2^{nd} exams, there is a much higher number of logical errors in the 2^{nd} exam, but that is due to a bigger number of tasks which have logical conditions. In the 3^{rd} exam, in which students had more experience, the number of mistakes had fallen off.

No significant difference was found between students‘ errors and grades achieved between face-to-face and online classes, in contrast to reports in [20, 21], which noted that there were alarming signs in core subjects such as math and reading that students may be falling even further behind pre-pandemic expectations in some grades. On the other hand, students in generation 2022/2023 had a worse grade point average, although they had face-to-face classes, in the previous two years they didn't have it constantly, and a lack of attention was noticed. This conclusion can be also confirmed by the average marks given in the Figure 15. Our second hypothesis is not accepted if we observe the first two generations.

The 3rd generation is a bit specific: they had all classes in 1st year in face-to-face, but two and a half years before they were online, and that could be the main reason why the results are such different as the 1st generation.

In addition, authors in [32] presented the average learning loss scaled by the length of time schools were closed during the pandemic. The main reasons that student performance was the same before and during the pandemic could be that classes were held online without interruption and at the same times as face-to-face classes and that groups were rather small, with up to 19 students.

Finally, we have to notice that some student had made many errors although they had a prior programming experience.

4.3 Student Errors and Marks in the Course

Introduction to programming is usually difficult for high school students. However, our students are gifted and have academic achievement in math and science, which is one of the most important predictors of their success in introductory programming courses [4]. Although the observed groups of students passed an additional exam to be included in the gifted mathematics course, some students abandoned their excellent mastery of the subjects in high school. Consequently, the students with many errors in their exams had pure grades (grades 1, 2, and 3). The range of grades is from 1 to 5, where 5 is highest mark. Figure 14 show that students from generation 2022/2023 have lower marks in all 3 written tasks, according to the high number of mistakes made in the tasks.

On the other hand, it can be seen that up to 20% of students with very good and excellent grades make logical mistakes under loop conditions. Therefore, logical thinking should be practiced more in class.

4.4 The Role of Formal Semantics in Teaching

Formal semantics is a way to precisely define the meaning of programming languages using mathematical principles. It helps to understand how computer programs work and how they interact with the computer [33]. It helps to catch mistakes early on by analyzing the produced code before execution, making error detection and correction easier. When the user's code

Table 3: Overview of the occurrence of errors in the written exams for generation 2019/2020

Generation 2019/2020	1st exam		2nd exam		3rd exam	
Type of error	Number of errors	Number of students made errors	Number of errors	Number of students made errors	Number of errors	Number of students made errors
(==) vs (=)	/	/	/	/	1	1
Unbalanced or missing '[', ']','(', ')', '{', '}'	5	1	/	/	/	/
Unbalanced or missing quotation marks	/	/	/	/	/	/
Separating the for loops with commas instead of semi-colons	/	/	/	/	/	/
Missing ';' at the end of statement	/	/	/	/	3	2
Improper type casting	/	/	1	1	/	/
Incorrect syntax	20	5	3	1	14	4
Logical errors in loop conditions	/	/	4	4	2	2

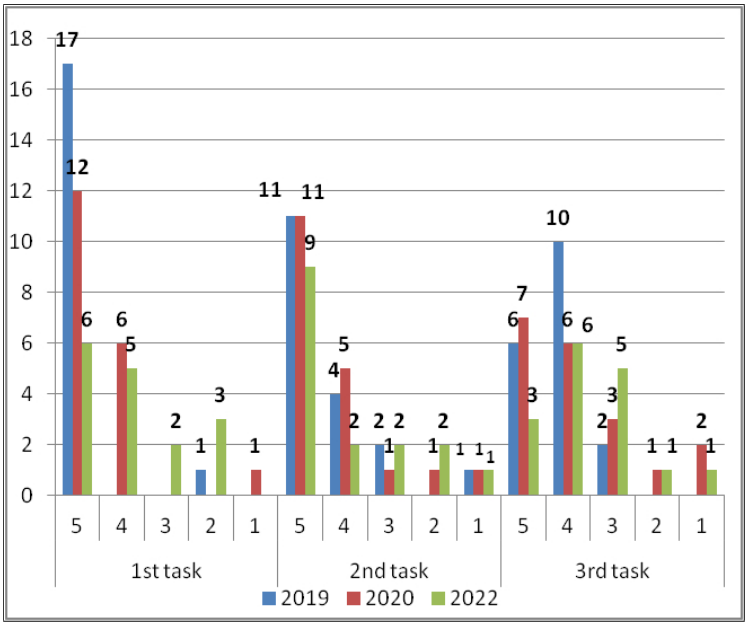

Figure 14: Marks distribution of each task by generations

does not behave as expected, formal semantics provides a structured approach to understanding and troubleshooting its behavior, enabling to find and fix problems.

In our opinion, learning about formal semantics is important for high school students because it helps us write clearer, more efficient, and reliable code. It allows to catch mistakes early, understand and fix problems, optimize the produced code for speed, and even create new things. By understanding formal semantics, students can become better programmers (and IT experts) and gain a deeper understanding of how programming languages work.

Table 4: Overview of the occurrence of errors in the written exams for generation 2020/2021

Generation 2020/2021	1st exam		2nd exam		3rd exam	
Type of error	Number of errors	Number of students made errors	Number of errors	Number of students made errors	Number of errors	Number of students made errors
(==) vs (=)	/	/	/	/	/	/
Unbalanced or missing '[', ']','(', ')', '{', '}'	2	1	/	/	3	1
Unbalanced or missing quotation marks	/	/	/	/	/	/
Separating the for loops with commas instead of semi-colons	/	/	/	/	/	/
Missing ';' at the end of statement	/	/	/	/	/	/
Improper type casting	/	/	3	3	/	/
Incorrect syntax	10	1	2	2	4	2
Logical errors in loop conditions	/	/	3	3	/	/

Table 5: Overview of the occurrence of errors in the written exams for generation 2022/2023

Generation 2022/2023	1st exam		2nd exam		3rd exam	
Type of error	Number of errors	Number of students made errors	Number of errors	Number of students made errors	Number of errors	Number of students made errors
(==) vs (=)	3	3	1	1	2	1
Unbalanced or missing '[', ']','(', ')', '{', '}'	11	3	8	4	2	1
Unbalanced or missing quotation marks	/	/	2	1	0	0
Separating the for loops with commas instead of semi-colons	/	/	2	1	3	1
Missing ';' at the end of statement	10	3	5	4	12	4
Improper type casting	/	/	1	1	5	5
Incorrect syntax	5	3	8	4	4	3
Logical errors in loop conditions	5	5	15	9	5	5

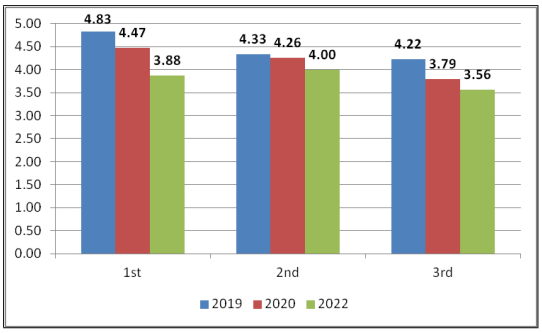

Figure 15: Average marks of each task by generation.

Hence, building upon the previous strategy of employing innovative teaching methods in programming, we will gradually explore potential frameworks suitable for incorporating semantic approaches into high school programming education. This entails expanding the utilization of formal semantics as a foundation for teaching programming at the high school level.

5. Conclusion

In this paper, we analyzed high school students' solutions to programming exercises in an introductory C# course. We were interested in describing students' errors. We identified the most common errors made by high school students and classified them. After above presented analysis we can draw some general conclusions, and we hope that teachers of programming courses can find it useful.

We agree with the findings in [31] that the deficits in students' strategic knowledge are one of the main reasons that programming is challenging for students.

As the analysis showed the biggest occurrence of errors has incorrect syntax, and second are logical errors in loop conditions. Incorrect syntax in C# may result from many different mistakes in code, such as failing to put strings within quotation marks, incorrectly using operators, and so forth. The reason can be found in the lack of student attention, or insufficient exercise by themselves. Students who did not do their homeworks regularly had bigger number of errors. In loops, there is a problem with interval boundaries that is not paid enough attention to, for example, whether we are talking about one interval, or the union of two intervals, when negating the required interval. If the student had tried entering a number from the given interval on the computer and tested how the program behaves when entering limit values, there would not have been so many errors in the code. Additionally, students in the second group were also observed in their second year, and only students with pure grades still make logical errors in loop conditions.

Given the current state of affairs, our recommendation for an extension of programming instruction would be to introduce a theoretical introduction to formal semantics. This would familiarize students with the theoretical features of languages, which would help them to recognize the important features at the level of syntax and how they are related to the resulting semantics, thus helping them to avoid some mistakes when writing programs [34, 35].

Our results also provide potential directions for future studies, we could additionally consider code and "algorithmic" smells to guide students to write not only correct programs but also appropriate and correct programs. The result of our research is relevant to teachers and researchers who wish to advance programming pedagogy. However to obtain more reliable conclusions, we are aware of the fact that it is required to repeat the research by enrolling new generations of students and to observe their progress in learning programming.

References

[1] H. B. Gonzalez and J. J. Kuenzi. Science, Technology, Engineering, and Mathematics (STEM) Education: A Primer. `https://fas.org/sgp/crs/misc/R42642.pdf`, 08 2012. Accessed 1 April 2022.

[2] N. DeJarnette. America's children: providing early exposure to stem (science, technology, engineering and math) Initiatives. *Education*, 133(1):77–84, 2012.

[3] TeachEngineering STEM Curriculum for K-12. `https://www.teachengineering.org/curriculum/browse`. Accessed 1 April 2022.

[4] Y. Qian and J. Lehman. An Investigation of High School Students' Errors in Introductory Programming: A Data-Driven Approach. *Journal of Educational Computing Research*, 58(5):919–945, 2020.

[5] D. McCall and M. Kölling. A new look at novice

programmer errors. *ACM Transactions on Computational Education*, 19:1–38, 2019.

[6] M. Bakator and D. Radosav. Managing Education in the COVID-19 era. In *International Conference on Information Technology and Development of Education — ITRO 2020*, pages 134–137. University of Novi Sad, 10 2020.

[7] S. Pokhrel and R. Chhetri. A Literature Review on Impact of COVID-19 Pandemic on Teaching and Learning. *Higher Education for the Future*, 8(1):133–141, 2021.

[8] V. Vilić. Cyber security and privacy protection during coronavirus pandemic. In *Sinteza 2021 – International Scientific Conference on Information Technology and Data Related Research*, pages 158–164, 2021.

[9] W. Steingartner, M. Jankura, and D. Radaković. Visualization of Formal Semantics – Possibilities of Attracting Formal Methods in Teaching. In *Sinteza 2021 – International Scientific Conference on Information Technology and Data Related Research*, pages 235–239, 2021.

[10] Sabine Meinck, Julian Fraillon, and Rolf Strietholt. *The impact of the COVID-19 pandemic on education International evidence from the Responses to Educational Disruption Survey (REDS)*. 02 2022.

[11] Kathleen Godber and Denise Atkins. COVID-19 Impacts on Teaching and Learning: A Collaborative Autoethnography by Two Higher Education Lecturers. *Frontiers in Education*, 6, 07 2021.

[12] D. Radaković and W. Steingartner. High school students' common errors in programming. In *Sinteza 2022 – International Scientific Conference on Information Technology and Data Related Research*, pages 104–108, 2022.

[13] J. Waite and S. Sentance. Teaching programming in schools: A review of approaches and strategies. `https://www.raspberrypi.org/app/uploads/2021/11/Teaching-programming-in-schools-pedagogy-review-Raspberry-Pi-Foundation.pdf`, 2021. Raspberry Pi Foundation.

[14] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[15] A. J. Ko and B. A. Myers. Development and evaluation of a model of programming errors. In *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments*, HCC '03, page 7–14, USA, 2003. IEEE Computer Society.

[16] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. Identifying and correcting java programming errors for introductory computer science students. volume 35, pages 153–156, 01 2003.

[17] J. Jackson, Mike Cobb, and Curtis Carver. Identifying top java errors for novice programmers. pages T4C – T4C, 11 2005.

[18] Paul Denny, Andrew Luxton-Reilly, and Ewan Tempero. All syntax errors are not equal. ITiCSE '12, page 75–80, New York, NY, USA, 2012. Association for Computing Machinery.

[19] Zihe Zhou, Shijuan Wang, and Yizhou Qian. Learning from errors: Exploring the effectiveness of enhanced error messages in learning to program. *Frontiers in Psychology*, 12, 2021.

[20] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. Common logic errors made by novice programmers. pages 83–89, 01 2018.

[21] Md Rahman, Shunsuke Kawabayashi, and Yutaka Watanobe. Categorization of frequent errors in solution codes created by novice programmers. *SHS Web of Conferences*, 102:04014, 01 2021.

[22] Y. Zhang, D. Hongle, L. Zhang, and J. Zhao. Personalization exercise recommendation framework based on knowledge concept graph. *Computer Science and Information Systems*, 20(2):857–878, 2023.

[23] V. Tsimbolynets and J. Perháč. Visualization Tool for Structural Operational Semantics of Simple Imperative Languages. *IPSI Transactions on Internet Research*, 19(1):66–74, 2023.

[24] W. Steingartner, D. Radaković, and R. Zsiga. Some Aspects about Visualization of Natural Semantics for a Selected Domain-Specific Language. *IPSI Transactions on Internet Research*, 19(1):46–54, 2023.

[25] M. Sulír, M. Bačíková, S. Chodarev, and J. Porubän. Visual Augmentation of Source Code Editors. *Journal of Visual Languages & Computing*, 49:46–59, 2018.

[26] Marek Kvet. Complexity and Scenario Robust Service System Design. In *2019 International Conference on Information and Digital Technologies (IDT)*, pages 271–274, 2019.

[27] S. G. Jaramillo. COVID-19 and primary and secondary education: the impact of the crisis and public policy implications for Latin America and the Caribbean. `https://www.latinamerica.undp.org/content/rblac/en/home/library/crisis_prevention_and_recovery/covid-19-y-educacion-primaria-y-secundaria--repercusiones-de-la-.html`, October 2020. Online, Accessed April 1st, 2022.

[28] Education in a Pandemic: The Disparate Impacts of COVID-19 on America's Students. `https://www2.ed.gov/about/offices/list/ocr/docs/20210608-impacts-of-covid19.pdf`, 2021. Online, Accessed April 1st, 2022.

[29] Ana Díaz and Carmen Lacave. The impact of covid-19 in collaborative programming. understanding the needs of undergraduate computer science students. *Electronics*, 10, 07 2021.

[30] TIBCO. Statistica® 4.1.0. `https://docs.tibco.com/products/tibco-statistica-14-1-0`, 2023. Accessed: 31.8.2023.

[31] Ella Albrecht and Jens Grabowski. Sometimes it's just sloppiness – studying students' programming errors and misconceptions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 340–345, New York, NY, USA, 2020. Association for Computing Machinery.

[32] E. Ahlgrenjoão et al. The global education crisis – even more severe than previously estimated. https://blogs.worldbank.org/education/global-education-crisis-even-more-severe-previously-estimated, 2022. Accessed, Accessed April 1st, 2022.

[33] David A. Schmidt. Programming language semantics. In Teofilo F. Gonzalez, Jorge Diaz-Herrera, and Allen Tucker, editors, *Computing Handbook, Third Edition: Computer Science and Software Engineering*. CRC Press, 2014.

[34] W. Steingartner, J. Perháč, and A. Biliński. Visualizing Tool for Graduate Course: Semantics of Programming Languages. *IPSI Transactions on Internet Research*, 15(2):52–58, 2019.

[35] Wolfgang Schreiner. Logic and Semantic Technologies for Computer Science Education. In *Informatics'2019, 2019 IEEE 15th International Scientific Conference on Informatics*, pages 7–12, Poprad, Slovakia, November 20–22, 2019. IEEE.

Davorka Radaković received her B.Sc. in Mathematics in 2001, M.Sc. (former Mr, 2 years) in Informatics in 2010, and PhD degree in Computer science in 2019 from the University of Novi Sad (Serbia), Faculty of Sciences. Her scientific research is focused on the development of a platform for dynamic geometry.

William Steingartner works as Associate Professor of Computer Science at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia. He defended his PhD thesis "The *Rôle* of Toposes in Informatics" in 2008. His main fields of research are the semantics of programming languages, category theory, compilers, data structures and recursion theory. He also works with type theory and software engineering.