frontiers | Frontiers in Education

TYPE Curriculum, Instruction, and Pedagogy
PUBLISHED 19 June 2025
DOI 10.3389/feduc.2025.1498100

OPEN ACCESS

EDITED BY
Attila Gilanyi,
University of Debrecen, Hungary

REVIEWED BY
Varun Dutt,
Indian Institute of Technology Mandi, India
Akash K. Rao,
Manipal Academy of Higher Education, India
José Figueiredo,
Instituto Politécnico da Guarda, Portugal

*CORRESPONDENCE
Feifan Zhang
ffz@ahmu.edu.cn

RECEIVED 18 September 2024
ACCEPTED 05 June 2025
PUBLISHED 19 June 2025

CITATION
Zhang F, Peng Z, Wang C and Yang F (2025) Design and application of teaching cases based on heuristic teaching in C programming language curriculums—taking the loop structure for an example. *Front. Educ.* 10:1498100. doi: 10.3389/feduc.2025.1498100

Design and application of teaching cases based on heuristic teaching in C programming language curriculums—taking the loop structure for an example

Feifan Zhang*, Zhenwan Peng, Changqing Wang and Fei Yang

School of Biomedical Engineering, Anhui Medical University, Hefei, China

C programming is a general-purpose, processor-oriented, and powerful language, widely used in various daily life applications. As a prerequisite for many core courses in computer science and information technology such as data structures, it serves as a fundamental course for multiple majors in universities globally. However, it remains one of the most challenging subjects to learn and master, with consistently high failure rates being a global problem. To improve learning efficiency and develop problem-solving skills through programming, we implemented a heuristic teaching method incorporating specially designed case studies. The participants were freshmen majoring in Medical Information Engineering at Anhui Medical University. Taking the loop structure as an example, we explain the design of case studies to engage students. The average scores improved from 72.9 to 77.2 between semesters without and with the heuristic teaching approach. Independent *T*-test results confirmed the statistical significance of this improvement. Student evaluations of teaching performance increased from 90.73 to 94.53. These results demonstrate the effectiveness of this heuristic teaching method with specially designed cases. We think that the proposed method may be also suitable for other programming courses.

KEYWORDS

C programming, heuristic teaching method, cases, loop-structure, programming courses

1 Introduction

The C programming language is one of the most widely used programming languages, with applications ranging from embedded systems, operating systems, to performance-critical applications (Hart et al., 2023). As a prerequisite for core courses in computer science, information technology, and engineering, such as Data Structures, Single-Chip Microcontroller Systems, and Principles of Microcomputers and Interface Techniques, it serves as a fundamental course for numerous majors in Chinese universities (Liu et al., 2013). The primary objectives of C programming courses include developing students' programming skills, cultivating computational thinking, and improving the ability to solve complex real-world problems through programming (Yu et al., 2023; Keppens and Hay, 2008; Wang et al., 2017). These competencies are critical for succeeding in follow-up courses. Students who fail to master C programming fundamentals will struggle in subsequent subjects. For instance, students unfamiliar with pointers and structures may require significantly more time to comprehend data structures such as linked lists. Similarly, those lacking proficiency in C syntax will face challenges in Single-Chip Microcontroller Systems, as C serves as the primary

programming language in this domain. This demands strong self-motivation and self-discipline. Otherwise, students may experience self-doubt and lose interest in learning. Worse still, subsequent programming-related courses could become insurmountable obstacles, potentially hindering academic progress, career opportunities, and pursuit of advanced degrees.

In fact, C programming is considered one of the most challenging courses for undergraduates (Dilshodov and Adxamjonov, 2023; Xu et al., 2020). The challenges stem from multiple factors. Unlike natural languages, its complex syntax rules and low-level abstractions create inherent barriers (Cheah, 2020). Novice learners often remain confused without sustained practice. Furthermore, since most freshmen lack prior programming experience, and many are accustomed to passive learning, they rarely engage in self-directed C programming practice, preferring step-by-step instructor guidance (Liu et al., 2013). Consequently, they struggle with fundamental constructs like loops and arrays, which hampers writing functional functions and procedures. This erodes learning motivation, ultimately affecting career prospects. Additionally, the sheer volume of course content coupled with dry instructional methods exacerbates the issue. Lectures dominated by slide-reading diminish learning engagement, fostering perceptions of C programming as uninteresting. Consequently, students neglect essential practice despite its critical role in mastering programming. This creates a vicious cycle: inadequate practice perpetuates skill gaps, further discouraging learning efforts.

Many methods have been proposed to improve teaching quality. Common approaches for teaching programming include lecture-based instruction, lab sessions, software visualization tools, and problem-based learning frameworks (Santos et al., 2020). For example, researchers have introduced adaptive learning activities that incorporate the Revised Bloom Taxonomy (RBT) to align with students' cognitive skills in programming education (Troussas et al., 2020). Empirical evidence suggests that adaptive teaching methods consistently outperform non-adaptive approaches (Troussas et al., 2020). Leveraging the computational power of mobile devices, educators are increasingly adopting educational games. Gamification—the integration of game-design elements (e.g., points, badges) into instructional contexts—has proven effective in enhancing student engagement and interest (Elshiekh and Butgerit, 2017). This strategy has been successfully implemented in C programming courses (Ibanez et al., 2014), with studies demonstrating its positive impact on learning outcomes (Ibanez et al., 2014). Further research evaluated the gamified Android application C-rocks as a pedagogical tool for C programming (Talingdan and Llanda, 2019), concluding that the app significantly improved student performance (Talingdan and Llanda, 2019). Similarly, the App Inventor platform has been utilized to motivate engineering students through game-based C syntax learning (Dolgopolovas et al., 2018). Additionally, precision teaching frameworks grounded in behavioral psychology principles have been developed for programming education (Yu et al., 2023). These frameworks operationalize three structured teaching phases and five core instructional skills, resulting in enhanced teacher-student interactions and higher learning efficiency compared to traditional classrooms (Yu et al., 2023). Despite these innovations, a substantial proportion of students continue to struggle with acquiring programming competencies and failing course assessments. Emerging approaches such as heuristic teaching—recently validated in domains like Chinese composition (Xue, 2022)—may offer promising solutions for programming education. The heuristic teaching method has been demonstrated as a highly effective approach for improving pupils' teamwork skills through structured organization of physical education lessons (D'Isanto et al., 2022). Heuristic learning, a learner-centered pedagogy, enables students to actively construct knowledge, redefine curricular objectives, and autonomously develop learning frameworks via self-directed processes that integrate continuous diagnosis, metacognitive reflection, and systematic knowledge organization (Pisarenko and Zatona, 2024). This method offers distinct advantages, including fostering student initiative, stimulating intrinsic motivation, and facilitating self-actualization within the learning process (Pisarenko and Zatona, 2024). Broadly defined, heuristics represent experience-based problem-solving strategies that guide cognitive exploration (Wakhata et al., 2023). The concept of heuristics traces its origins to an ancient Greek philosophical paradox: "How can we search for what we do not know, and if we know what we are looking for, then why should we look for it?" (summarized as the science of discovery) (Nokhatbayeva, 2020). Its pedagogical roots lie in Socratic dialogs, where knowledge emerged through resolving contradictions in discourse (Nokhatbayeva, 2020). In modern education, the Socratic method informs the design of heuristic activities, exemplified by mathematician George Pólya's systematic heuristic framework for problem-solving through sequenced questioning aimed at cultivating critical thinking (Nokhatbayeva, 2020). As articulated by Russian psychologist Kapterev, "Heuristic form of teaching is one in which scientific laws, formulas, rules and truths are discovered and developed by the students themselves under the guidance of the teacher" (Nokhatbayeva, 2020). While this approach has profoundly influenced mathematics education (Nokhatbayeva, 2020), its application to C programming courses remains unexplored to date. Given heuristic teaching's capacity to enhance learning motivation, promote active engagement through self-discovery, and improve academic outcomes, this study designs a heuristic-based instructional framework. We exemplify this methodology through a case study on loop structure instruction, detailing its pedagogical design and practical implementation.

The remaining of the paper is organized as follows. In Section 2, we will explain the design and application of heuristic teaching in the programming courses. The teaching of loop structure is taken as an example. The results and discussions are shown in Section 3. And the conclusions are shown in Section 4.

2 Materials and methods

In designing and implementing heuristic teaching methods, we integrate principles from the Contextual Teaching and Learning (CTL) model and Problem-Based Learning (PBL) framework. The philosophical foundation of CTL lies in constructivism, which emphasizes knowledge construction through contextual connections—bridging abstract concepts with lived experiences or real-world applications—rather than rote memorization (Kia, 2023). PBL, grounded in cognitive psychology, adopts a "problem-driven learning" paradigm that inverts traditional instruction by prioritizing problem exploration before knowledge transmission. By synthesizing

heuristic methods with CTL and PBL, we formulated the following pedagogical tenets for C programming:

1. Problem Contextualization: Embed syntax learning within authentic programming scenarios to stimulate cognitive engagement (shifting focus from "how" to "why").
2. Progressive Task Decomposition: Implement stepwise complexity escalation—from code imitation to creative implementation—by modularizing systems and gradually elevating abstraction levels.
3. Error-Driven Discovery: Leverage coding errors as diagnostic tools for self-directed learning, enabling students to autonomously uncover syntactic and logical principles.
4. Multimodal Knowledge Representation: Demystify abstract concepts through schematic diagrams, algorithm animations, and tangible analogies (e.g., physical models of loop execution).

As the loop structure constitutes one of the three fundamental constructs in structured programming and serves as the conceptual foundation for subsequent chapters (e.g., arrays and linked lists), we select it as a paradigmatic case to demonstrate the design and implementation of heuristic pedagogy in C programming. Prior to this instructional unit, students have achieved mastery of sequential and selection structures. The proposed lesson plan spans 120 min. At the beginning of the loop structure, we show a clip from the movie Edge of Tomorrow to show the cycle of death and resurrection. Then, we show students a clock with second hands keeping running (lasts about 5 min). Thus, students would have an intuitive sense of the loop. Next, we tell students about the story of Gauss and ask them to list the solutions they can think of for the calculation of the sum of 1 to 100. After students share three or four solutions, we ask them to think about how we can solve this problem by C programming. Assuming that we do not know the sum formulas of arrays, students will try to write a sequential structure program as they have learned the sequential structure. One possible sequential program example is shown in Table 1.

Note that the programming tasks can be completed in class since students can implement them on their mobile phones with C compilers installed. After they finish the tasks, we ask students to analyze the disadvantages of the program in Table 1. One disadvantage is its excessive length due to repetitive expressions. Another drawback is that calculating the sum from 1 to 1,000 would require writing numerous similar statements (this phase lasts about 10 min). We then ask students to identify the commonality among these statements. These can be generalized as $sum = sum + i$, where i takes values $1, 2, 3, \ldots, 100$. Thus, the result can be computed by iteratively executing these generalized statements with incrementing i values. This process forms a loop, as it repeatedly executes two operations: $sum = sum + i$ and $i = i + 1$ — analogous to the cycle of death and rebirth or the movement of a clock's second hand. Through this analogy, the loop structure is introduced, enabling students to better grasp its execution mechanism. Finally, we provide students with the optimized program solving this problem, as shown in Table 2.

TABLE 1 A possible C program for calculating the accumulation of 1 to 100 written by students.

Program: calculating the sum of 1 to 100

#include<stdio.h>
int main(){
int sum=0;
sum=sum+1;
sum=sum+2;
sum=sum+3;
…..
sum=sum+100;
return 0;
}

Substantially, based on the program shown in Table 2, we describe the execution process of the for-loop structure in detail using debugging within the integrated development environment (IDE) Dev C++, which directly displays the changes in variables such as i. This approach helps students better understand the for-loop structure. After that, we guide students to summarize the syntax and characteristics of the for-loop structure. Based on this summary, we visually demonstrate the execution process using an animated flowchart in PowerPoint, showing the workflow with a red point traversing the diagram. Next, we ask students to modify the program to calculate the sum of numbers from 1 to 1,000 using the C compiler installed on their mobile phones. We then propose extension challenges sequentially: calculating the sum of odd numbers and even numbers within $\overline{1,1,000}$. Once these problems are solved, students gain familiarity with the basic usage of the for-loop structure. Furthermore, we allocate 5 min for them to practice calculating the sum of numbers divisible by five within $\overline{1,1,000}$. This exercise not only reinforces their mastery of the for-loop structure but also revisits the selection structure through checking divisibility by five. The entire process lasts about 25 min.

After finishing the practice, we teach students the knowledge of the while-loop structure and do-while loop structure through multimedia technology such as animated flowcharts with a red point indicating execution progress. To better master the while-loop and do-while loop, students are asked to implement the functionality of the above programs using different loop structures. Furthermore, students are asked to compare the differences between these three loop structures (lasting about 20 min). Then, we ask students to write a program that outputs a 10-star pattern using a loop structure, as the visual patterns are intuitive (Figure 1). They can use any of the three loop structures. We demonstrate the while-loop structure as an example (Table 3).

After demonstrating the execution process with Dev C++, we ask students to design a program that outputs an 8-row by 10-column star pattern (Figure 2). We invite three students to demonstrate how they would write this program. After their presentations, we provide a solution where one loop controls the rows and a nested loop within it controls the number of stars per row, similar to the previous program. Through this, we introduce the concept of nested loops—embedding additional loops inside a primary loop statement—to achieve multi-level data traversal and processing. We then explain the core principles of nested loops. Subsequently, we provide students with an example program (Table 4) intended to print Figure 2, but containing intentional errors. When students execute the given program, they observe that only a single line of 10 stars (Figure 3) is printed instead of the expected pattern. We ask them to diagnose why the output differs from the target by debugging the program and monitoring changes in loop control variable values.

TABLE 2 A possible C program with loop structure written for the accumulation of 1 to 100.

Program: calculating the sum of 1 to 100

#include<stdio.h>
int main(){
int sum=0;
for(i=1; i<=100; i++)
sum+=i;
return 0;
}

FIGURE 1
The pattern to be output.

TABLE 3 A possible C program with while-loop structure written for outputting 10 stars.

Program: calculating the sum of 1 to 100

#include<stdio.h>
int main(){
int sum=0;
for(i=1; i<=100; i++)
sum+=i;
return 0;
}

FIGURE 2
The pattern to be output.

Subsequently, a student is randomly selected to explain their reasoning. We then analyze the program and clarify the reason: a loop terminates when its continuation condition becomes false. Specifically, after the inner loop completes execution, the value of its control variable no longer meets the loop's continuation condition. When the outer loop iterates again, the inner loop's control variable must be reinitialized. Students are therefore instructed to modify the program in Table 4. Next, we ask them to implement the same functionality using a for-loop structure and determine whether this issue persists with for loops. This ensures students fully comprehend

TABLE 4 A C program outputting Figure 3.

Program: output 10 stars with while-loop structure

#include <stdio.h>
int main ( ){
int i = 1, j = 1;
while(i<=8) {
while (j<=10){
putchar('*');
j++;
}
putchar('\n');
i++;
}
return 0;
}

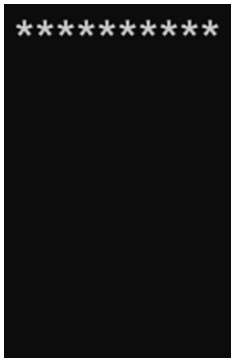

FIGURE 3
The pattern output.

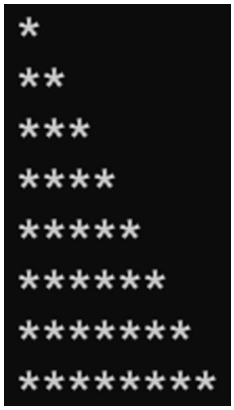

FIGURE 4
The pattern to be output.

nested loop execution and develop awareness of loop control variable management. Following this, based on Figure 2, students are tasked with writing a program to output Figure 4. They then progress incrementally to develop programs for Figures 5–7. Note that Figure 5 differs from Figure 4 in that the number of lines is dynamically determined by user input.

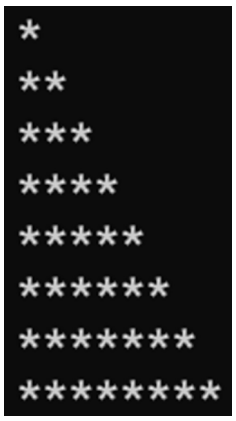

FIGURE 5
The pattern to be output.

FIGURE 6
The pattern to be output.

FIGURE 7
The pattern to be output.

After completing the previously mentioned patterns, we increase the complexity of programming practice by integrating character manipulation with standard I/O operations. Students are tasked with generating patterns composed of characters (Figure 8). Next, they are required to produce patterns based on a user-input uppercase letter—for example, inputting "G" generates the pattern in Figure 9. To account for potential errors, the program prompts users to re-enter if the input is non-uppercase, continuing this validation loop until valid data is received (Figure 10). Through this exercise, students progressively learn to design interactive menus using loop structures and selection statements (taught prior to loops). The entire activity is designed to be completed within approximately 60 min.

Throughout these heuristic teaching activities, students actively engage in thinking, learning, and practicing during class. With frequent questioning and interactive practices progressing incrementally from simple to complex tasks, students remain motivated and cognitively engaged. Finally, we assign homework requiring them to generate a specific pattern based on a user-input uppercase character (Figure 11).

3 Results and discussions

We began teaching C programming courses to Medical Information Engineering majors at Anhui Medical University in the fall 2021 semester. Based on preliminary surveys conducted during the first class, these freshmen (predominantly from Anhui Province) had minimal programming exposure, with C programming being their first formal coding course. Most lacked foundational computer science knowledge. By the semester's end, we observed persistent difficulties in their ability to acquire programming skills and develop computational problem-solving competencies. Consequently, we implemented pedagogical reforms for the 2022 fall semester cohort of the same program (comprising students mainly from Anhui Province, with one exception from Shanxi Province). Adopting heuristic teaching methodologies with customized case studies, we maintained instructional continuity as outlined in Section II. Notably, due to logistical constraints—including shared theoretical coursework with master's degree students—we could not conduct parallel controlled experiments within the same semester (i.e., traditional vs. reformed methods). Furthermore, implementing differential teaching approaches risked student objections, as course grades directly impacted scholarship eligibility. Therefore, uniform instructional strategies were applied within each cohort. However, the students of the 2021 and 2022 fall semesters can be seen as two contrast groups to a certain extent because they are taught by the same teacher with the same teaching hours. Only the teaching methods are different. In fact, these students are with the same curriculum. The C programming lessons are both 45 teaching hours. Thus, the examination scores and student evaluation scores of teaching performance are used as indicators to show the efficiency of the proposed teaching methods. Note that each topic in the 2022 fall semesters was taught by the same teacher consistent with that in the 2021 fall semesters. Moreover, the examination papers have the same level of difficulty with the same emphasis. The examination scores are shown in Table 5.

As shown in Table 5, the average examination scores for C programming courses in 2021 and 2022 were 72.9 and 77.2,

FIGURE 8
The pattern to be output.

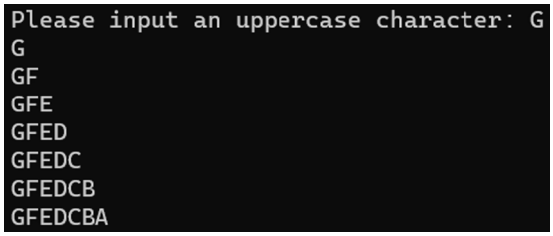

FIGURE 9
The pattern to be output.

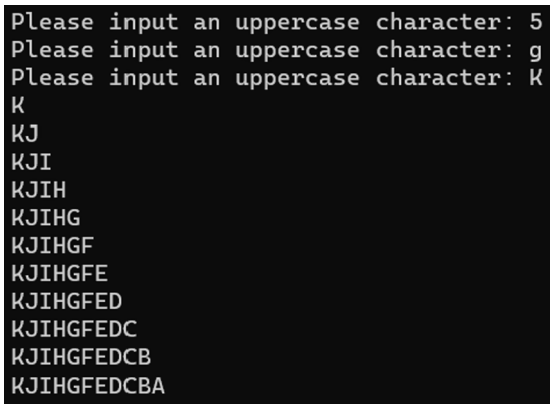

FIGURE 10
The pattern to be output according to the character input.

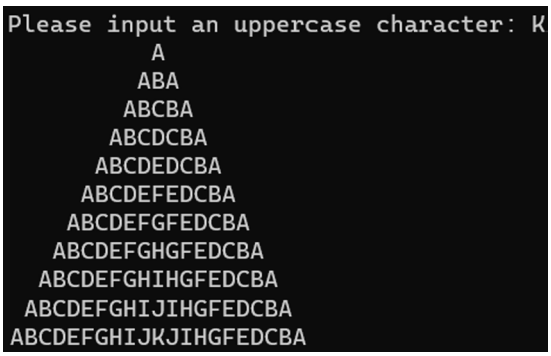

FIGURE 11
The pattern to be output according to the character input.

respectively. While differences in scores could theoretically arise from factors like student backgrounds, instructor variations, or teaching methodologies, several observations suggest these variables were controlled: All students except one originated from Anhui Province; both cohorts consisted of Medical Information Engineering freshmen with comparable age profiles (average ages 17.91 in 2021 vs. 17.90 in 2022) and minimal programming experience. The 2021 cohort included 59 students (30 male, 29 female), while the 2022 cohort had 88 students (52 male, 36 female). Crucially, the same instructor delivered all course content. Given these stabilized conditions, the 4.3-point score increase in 2022 implies the heuristic teaching reforms contributed to improved outcomes. Statistical validation was conducted through a one-tailed *t*-test (justified by similar standard deviations: 9.1 for 2021 vs. 9.9 for 2022), yielding a *p*-value of 0.00447—significantly below the 0.05 threshold. The 95% confidence interval $\bar{}\left[-\infty, -1.61\right]$ indicates the 2021 cohort's mean score was at least 1.61 points lower than 2022's. A Cohen's d value of 1.38 confirms a large effect size. Supporting evidence emerges from score distributions: the 2021 modal range was 70–79 (Figures 12, 13), shifting to 80–89 in 2022. Notably, both the highest and lowest scores in 2022 marginally surpassed 2021's extremes. These findings collectively demonstrate the pedagogical effectiveness of the redesigned heuristic teaching framework.

We also find that although students of the same semester attend the same classrooms together, however, the scores differ with classes (Figure 14). For freshmen of the 2021 fall semester, they were divided into two small classes. The average scores of these two classes are 71.93 and 73.86, respectively. For freshmen of the 2022 fall semester, they were divided into three classes. The average scores of these three classes are 79.17, 77.79, and 74.31. As students of the same semester attended theory classes of C programming together, they were instructed with the same teaching contents and methods. However, the scores are still different. We guess that this may be caused by class discipline and academic atmosphere as they doing C programming practice in the units of classes. According to our experience in experimental programming classes, for classes with higher scores, more students would ask questions or discuss problems during programming. In contrast, some students were addicted to mobile video games. Some of them even tried to play mobile video games when others were programming. Of course, they would be stopped from playing games in classes. Thus, we think the differences in examination scores between classes of the same semester may be caused by class discipline and academic atmosphere. This will be our future research direction to further improve the teaching efficiency while analyzing factors affecting teaching efficiency and minimizing the difference between classes. Also, we need to develop more suitable evaluation approaches such as formative assessment and project assessment into our teaching process in the future research. This is one of the limitations of the work.

After the C programming course concluded, students were asked to evaluate teaching performance through scoring. Following the implementation of the proposed method, teaching evaluation scores increased from 90.73 to 94.53. Many students noted that complex

TABLE 5 Examination scores of C programming courses of two different semesters.

Semester	Number of students	Average scores	The highest score	The lowest score	Standard deviation
2021 fall	59	72.9	94	53	9.0
2022 fall	88	77.2	96	55	9.9

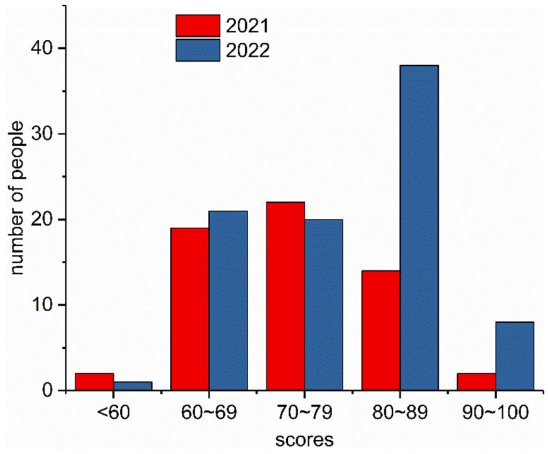

FIGURE 12
The score distribution.

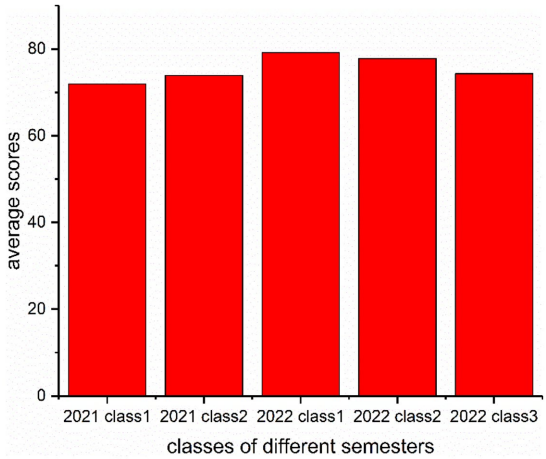

FIGURE 14
The relationship between scores and classes.

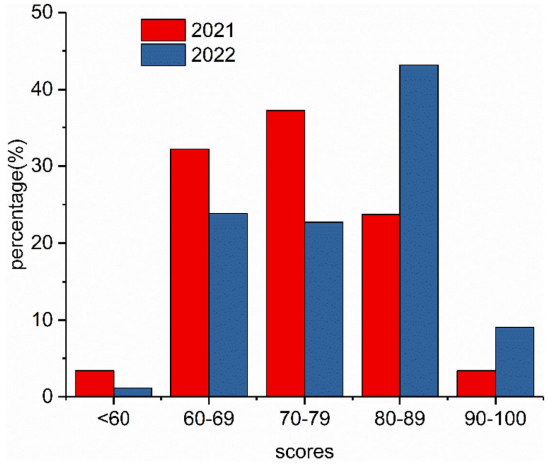

FIGURE 13
The percentage of score ranges.

concepts became easier to understand as the designed cases and targeted analyses by teachers effectively connected abstract theories with practical understanding. Some students highlighted that the praxis-oriented approach, integrating theoretical frameworks with contextualized applications, strengthened their learning processes. Student feedback further confirms the efficacy of these teaching methods. The proposed method may be applied to other computer science courses. For example, object-oriented programming classes such as Python and Java also involve loop structures. The teaching process can be adapted to suit these courses. The framework can further extend to data structure instruction. For instance, animations may directly demonstrate linked list operations or binary tree traversal, while physical analogies (e.g., stacking plates) could explain stack push/pop mechanisms. This approach thus helps students better understand abstract concepts.

4 Conclusion

The C programming language has various applications and is also a prerequisite for courses such as Data Structures, Object-oriented Programming, and Embedded Systems. However, C programming remains one of the most difficult courses in many majors, with a high failure rate. Failure to learn C programming may not only decrease students' motivation in learning programming languages but also increase the difficulty of mastering knowledge related to subsequent courses. Furthermore, this may affect their future career prospects.

To improve the teaching efficiency of C programming classes offered to freshmen majoring in Medical Information Engineering at Anhui Medical University, we designed teaching cases based on heuristic teaching methods and implemented them starting in the 2022 fall semester. Since the loop structure is foundational for subsequent topics such as arrays and pointers, we used it as an example to demonstrate case design strategies and stimulate students' learning interest. We utilized examination scores as an indicator to evaluate the effectiveness of the designed cases and heuristic teaching methods. As the students were freshmen with no prior programming experience, this metric is valid. Results indicate that with the heuristic-based cases, the average score rose from 72.9 (2021) to 77.2 (2022), while teaching evaluation scores increased from 90.73 to 94.53. This demonstrates that the heuristic teaching cases significantly enhance instructional efficiency in C programming courses. We believe these methods could inform pedagogy in other computer-related subjects.

Data availability statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

Author contributions

FZ: Methodology, Writing – original draft. ZP: Methodology, Writing – original draft. CW: Methodology, Writing – original draft. FY: Methodology, Writing – original draft.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. This research was funded by Department of Education Anhui Province, grant numbers 2023zyxwjxalk039, 2023jyxm0235, and 2022jyxm710. This research was also funded by Anhui Medical University, grant number 2024xjxm117.

Acknowledgments

We appreciate the editors and reviewers for their earnest work and insightful comments. We appreciate Deepseek for helping us correcting grammar errors.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Cheah, C. S. (2020). Factors contributing to the difficulties in teaching and learning of computer programming: a literature review. *Contemp. Educ. Technol.* 12:ep272. doi: 10.30935/cedtech/8247

D'Isanto, T., Aliberti, S., Altavilla, G., Esposito, G., and D'Elia, F. (2022). Heuristic learning as a method for improving students' teamwork skills in physical education. *Int. J. Environ. Res. Public Health* 19:12596. doi: 10.3390/ijerph191912596

Dilshodov, A., and Adxamjonov, M. (2023). "Application of the opengl library in the course of programming and computer simulation" in Engineering Problems and Innovations, 112–114.

Dolgopolovas, V., Jevsikova, T., and Dagiene, V. (2018). From android games to coding in C—an approach to motivate novice engineering students to learn programming: a case study. *Comput. Appl. Eng. Educ.* 26, 75–90. doi: 10.1002/cae.21862

Elshiekh, R., and Butgerit, L. (2017). Using gamification to teach students programming concepts. *Open Access Library J.* 4, 1–7. doi: 10.4236/oalib.1103803

Hart, R., Hays, B., McMillin, C., Rezig, E. K., Rodriguez-Rivera, G., and Turkstra, J. A. (2023). Eastwood-tidy: C linting for automated code style assessment in programming courses. Proceedings of the 54th ACM Technical Symposium on Computer Science Education, 1, 799–805.

Ibanez, M. B., Di-Serio, A., and Delgado-Kloos, C. (2014). Gamification for engaging computer science students in learning activities: a case study. *IEEE Trans. Learn. Technol.* 7, 291–301. doi: 10.1109/TLT.2014.2329293

Keppens, J., and Hay, D. (2008). Concept map assessment for teaching computer program ming. *Comput. Sci. Educ.* 18, 31–42. doi: 10.1080/08993400701864880

Kia, A. D. (2023). The use of heuristic reasoning in Christian education. *AI-Ishlah: Jurnal Pendidikan* 15, 1571–1580. doi: 10.35445/alishlah.v15i2.3888

Liu, S., Chen, Z., and Tang, J. (2013). The improved methods of teaching practice based on C language programming. 2013 Conference on Education Technology and Management Science (ICETMS 2013), 807–810.

Nokhatbayeva, K. (2020). The effects of heuristic teaching methods in mathematics. *Proc. Int. Young Scholars Workshop* 9, 142–156. doi: 10.47344/iysw.v9i0.158

Pisarenko, V., and Zatona, D. (2024). Using heuristic methods in primary school. In EDULEARN24 Proceedings: 16th International Conference on Education and New Learning Technologies, 9486–9495.

Santos, S. C., Tedesco, P. A., Borba, M., and Brito, M. (2020). Innovative approaches in teaching programming: a systematic literature review. In Proceedings of the 12th International Conference on Computer Supported Education, 1, 205–214.

Talingdan, J. A., and Llanda, C. R. (2019). Assessment of the effectiveness of learning theories using gamified android app in teaching C programming. *IOP Conf. Ser. Mater. Sci. Eng.* 482:012030. doi: 10.1088/1757-899X/482/1/012030

Troussas, C., Krouska, A., and Sgouropoulou, C. (2020). A novel teaching strategy through adaptive learning activities for computer programming. *IEEE Trans. Educ.* 64, 103–109. doi: 10.1109/TE.2020.3012744

Wakhata, R., Mutarutinya, V., and Balimuttajjo, S. (2023). Relationship between active learning heuristic problem-solving approach and students' attitude towards mathematics. *Eurasia J. Math. Sci. Technol. Educ.* 19:em2231. doi: 10.29333/ejmste/12963

Wang, X. M., Hwang, G. J., Liang, Z. Y., and Wang, H. Y. (2017). Enhancing students' computer programming performances, critical thinking awareness and attitudes towards programming: an online peer-assessment attempt. *J. Educ. Technol. Soc.* 20, 58–68.

Xu, B., Yan, S., Jiang, X., and Feng, S. (2020). SCFH: a student analysis model to identify students' programming levels in online judge systems. *Symmetry* 12:601. doi: 10.3390/sym12040601

Xue, B. (2022) The use of picture books and heuristic teaching method for Chinese writing skill of grade 3 Chinese students in Sichuan Province, China, (doctoral dissertation, Rangsit University).

Yu, F., Liu, Y., and Xiao, F. (2023). Research on construction and practice of precision teaching classroom for university programming courses. *IEEE Access* 11, 9560–9576. doi: 10.1109/ACCESS.2023.3240105