ISSN: 1813-4890

A Three-Step Pedagogical Approach to Teaching C Programming

Ren Yuan

School of Electronics and Information, Shanghai Dianji University, Shanghai 201203, China.

Abstract

Teaching a programming language is a challenging job for college teachers. This paper proposes a three-step pedagogical approach to teaching c programming. Step 1: Students are required to complete their code and make their programs executable. Step 2: Students are asked to debug their programs so the results will be completely correct. Step 3: Students are encouraged to enhance their programs from the perspective of users. This approach has been adopted on international students' courses in Shanghai Dianji University (SDJU) for several years, and has turned out to be effective.

Keywords

Pedagogical approach, C programming, three-step, international student, SDJU.

1. Introduction

Teaching programming has always been a challenging job for educators in all levels of schools as well as learning programming for students [1-4]. In this paper, we propose a three-step pedagogical approach to teaching programming in the C language [5].

In step 1, students must make their programs runnable by eliminating all grammatic errors. Furthermore, in step 2, students should fix all logical errors so the output will be correct. Finally in step 3, students are encouraged to improve the results in various aspects. This approach has been experimented on international students in Shanghai Dianji SDJU for several years, and the results turn out to be as expected. Fig. 1 illustrates the core this approach.

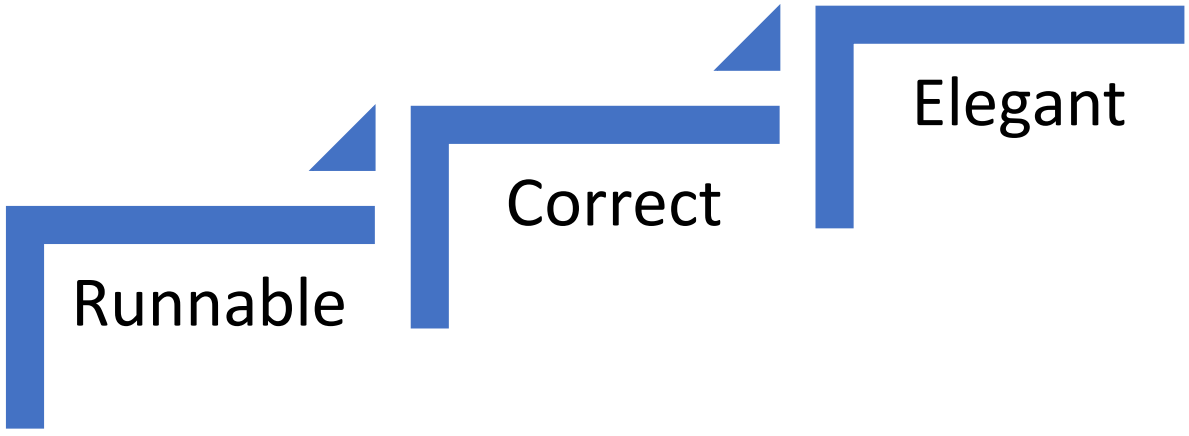

Fig. 1 Three steps of teaching programming

2. Step 1: Make the code runnable

Like studying a natural language such as English, Chinese or French, grammar is the very foundation for using that language preliminarily by forming key words into statements. Beginners always make a lot of grammatic errors when learning a new programming language. It is therefore the first step to eliminate all grammatic errors so the code will compile into an

ISSN: 1813-4890

executable program. Usually, it takes much longer time for students to locate and correct errors than to write code. If some students cannot succeed in fixing errors and finally making the code runnable like most others will do in class, they are likely to give up learning [6]. In this extremely crucial phase, teachers should do whatever they could to help students run their programs without red-wavy underlined code or a long list of errors or warnings. Students' confidence and interests will greatly rise if they are able to obtain results from their programs after having tried hard to make every single mistake disappear.

For most, if not all courses, students' learning outcomes in the first several weeks (usually less than a month) will largely determine their behaviors in the whole semester. In this period, students are encouraged to focus on problems that prevent their programs from going through compilations. Even if the programs produce incorrect results or no result at all, they can still feel a sense of accomplishment. The process of error correction also enhances students' comprehension of abstract grammar.

3. Step 2: Make the result correct

Even if the code is compiled into a runnable program, the program may not produce logically correct results at all times or may unexpectedly crash with an exception thrown at runtime. It usually takes a great amount of time to debug a program to eliminate logical errors and to make the results completely correct [7]. A lot of work and efforts are required in this step. Textbooks and handbooks can explain grammar very clearly, under the guidance of which students can locate and fix compilation errors. But few books can explicitly illustrate show how to look for and solve problems from a program that the compiler cannot even find errors.

For example, a program is supposed to add two input integers but ends up with the product of the numbers. Most beginners will feel helpless after reading the code many times without having any idea about the source of the error. In this phase, teachers should focus on step-by-step demonstrations of debugging the program by simply printing out variable values and intermediate results or more preferably by setting breakpoints.

Testing a program is also an important job for programmers. A program may be correct sometimes but incorrect under other circumstances. For example, a program that finds the absolute of input may prints undesirable results with negative input is some arithmetic operators are missing in the code. But when students test the program, they tend to type in simple input such as positive integers and ignore the other cases.

4. Step 3: Make the program elegant

After obtaining correct results from the program, students are encouraged to further improve the interface through which the program interacts with users including the way how the program takes input and presents output [8]. They are supposed to switch their roles and think from the perspective of users instead of programmers. For instance, programmers know how their programs are designed to work, so they will enter all input as expected even without any prompt messages. Users, nevertheless, will have no idea what they are supposed to do if input is required with no hints, or they will even think the program is frozen or unresponsive because no output will be shown no matter what key they hit. Programs like these will be considered user-unfriendly. There are many aspects in which students can enhance their programs and make them elegant:

(1) To show unambiguous prompt messages before requiring input, such as the format of expected input. Sometimes, it is even necessary to provide a sample if textual descriptions are not clear enough for users to enter input in the exact format. For example, when reading date, it will be more favorable to show a sample like "24/12/07" than to say "Enter year/month/day".

ISSN: 1813-4890

(2) Allow users to make mistakes and give them more chances to try. Not only users but programmers themselves will make mistakes when entering input to programs. The programs will be considered unrobust or unstable if they crash whenever unexpected data are received, leaving users completely confused about what's happening. High-level programming languages like C provide mechanism for handling exceptional cases at runtime. Programs should warn users that errors occur due to their incorrect input, explain the reasons if possible, and then give them more chances to try until correct results can be generated.

5. Conclusion

Using the abovementioned three-step approach, students can achieve desirable learning outcome in the C programming course. This approach can also be applied to teaching of other programming languages such as Java, C++ and Python. In the future, this approach will be constantly improved by encouraging students to enhance their programming skills in more aspects.

References

[1] Yu, Fang; Liu, Yan; Xiao, Fengyan: Research on Construction and Practice of Precision Teaching Classroom for University Programming Courses, IEEE Access, Vol. 11(2023), p. 9560-9576.

[2] Roldán-Álvarez, David; Mesa, Francisco J.: Intelligent Deep-Learning Tutoring System to Assist Instructors in Programming Courses, IEEE Transactions on Education, Vol. 67(2024), No. 1, p. 153-161.

[3] Lv, Na; Zhao, Xiuyang; Tian, Jinglan; Zhang, Qianqian; Xu, Meihui; Fan, Xue: The application of mixed teaching mode in programming courses, 14th International Conference on Computer Science and Education, 2019, p. 627-630.

[4] Khomokhoana, Pakiso: Understanding Elements, Strengths and Challenges of Explicit Instruction for The Teaching of Computer Programming, The Independent Journal of Teaching and Learning, 2023, p. 59-80

[5] Rong, Wenge; Xu, Tianfan; Sun, Zhiwei; Sun, Zian; Ouyang, Yuanxin; Xiong, Zhang: An Object Tuple Model for Understanding Pointer and Array in C Language, IEEE Transactions on Education, Vol. 66(2023), Issue 4, p. 318-329.

[6] Oka, Hiroki; Ohnishi, Ayumi; Terada, Tsutomu; Tsukamoto, Masahiko: System for Detecting Learner Stuck in Programming Learning, Sensors, Vol. 23(2023), Issue 12.

[7] Alaboudi, Abdulaziz; LaToza, Thomas D.: What Constitutes Debugging? An Exploratory Study of Debugging Episodes, Empirical Software Engineering, Vol. 28(2023), Issue 5.

[8] Yuen, Kevin K. F; Liu, Dennis Y. W; Leong, Hong Va: Competitive Programming in Computational Thinking and Problem Solving Education, Computer Applications in Engineering Education, Vol(2023). 31, Issue 4, p. 850-866