

Course Construction and Practice of "Fundamentals of Programming -C Language" Oriented towards Applied Talent Cultivation

Yaqin Wu *, Zhenhan Liu

School of Artificial Intelligence, China University of Mining and Technology (Beijing), Beijing, 100083, China

* Corresponding author: Yaqin Wu (Email: wuyq@cumtb.edu.cn)

Abstract: "Fundamentals of Programming – C Language" is a crucial foundational course for computer science and related engineering majors, playing a significant role in cultivating students' computational thinking, programming abilities, and engineering practice literacy. Addressing issues in traditional teaching such as weak practical components and monotonous teaching methods, this paper conducts research on course construction and practice centered on the goal of cultivating applied talents. Adhering to the concept of "student-centered, ability-oriented, task-based, and supported by digital resources," the course reconstructs the content system modularly, integrating syntax knowledge into tasks. It promotes blended learning and task-driven teaching to enhance classroom interaction and student engagement. Furthermore, it constructs micro-lecture videos, exercise banks, common error case libraries, and online judging platforms to support blended teaching. Practice indicates that the course reform helps improve students' programming and comprehensive application abilities, enhances the learning experience, and promotes the continuous optimization of teaching methods and course resources, offering reference significance for the construction of similar programming foundation courses.

Keywords: Fundamentals of Programming; C Language; Blended Learning; Task-Driven; Diversified Evaluation.

1. Introduction

"Fundamentals of Programming – C Language" is an important foundational course for computer science, electronic information, and related engineering majors, tasked with cultivating students' computational thinking, programming abilities, and problem-solving skills. With the advancement of "New Engineering" construction and the reform of applied talent cultivation models, the programming foundation course should serve as vital support for helping students form programming capabilities, engineering awareness, and innovative literacy. In traditional teaching, C language courses often suffer from an overemphasis on syntax explanation, relatively weak practical links, and singular teaching methods, making it difficult to fully support the transition from "learning syntax" to "solving practical problems." Therefore, systematic construction regarding course content, teaching methods, practical systems, resource building, and evaluation mechanisms is of great significance for improving teaching quality and forming replicable reform experiences.

2. Background and Current Issues of Course Construction

"Fundamentals of Programming – C Language" is a foundational programming course offered to freshmen at our university. Students exhibit significant differences in their programming backgrounds. While some have been exposed to simple programming knowledge, the majority lack programming experience and tend to feel intimidated by complex concepts such as loops, functions, and pointers. The course objectives require not only ensuring students master basic C language syntax and programming methods but also guiding them to form standardized programming habits,

rigorous logical thinking, and preliminary engineering practice capabilities.

Regarding teaching content, traditional C language courses typically follow the syntax system as the main thread, sequentially explaining data types, operators, control structures, arrays, functions, pointers, structures, and file operations. Although this organization follows a progression from easy to difficult, it easily traps the classroom in a fixed pattern of "explanation–demonstration–practice." [1] There is a lack of support from real-world problem contexts between knowledge points, making it difficult for students to comprehensively apply multiple concepts to solve practical problems.

In terms of teaching methods, classroom lectures still occupy a high proportion. Students have limited opportunities to actively participate in programming practice, lacking complete training in problem analysis, algorithm design, program implementation, debugging, and optimization.[2][3] The use of online teaching resources often remains at the level of watching videos and submitting assignments, leaving room for improvement in classroom interaction, process feedback, and personalized guidance. In teaching practice, the proportion of verification experiments is high, while comprehensive, design-oriented, and open-ended experiments are insufficient, resulting in weak abilities for students to independently analyze problems and debug programs. Regarding assessment and evaluation, final exam scores often account for a large proportion, making it difficult to fully incorporate daily assignments, experimental performance, classroom participation, and task completion into the evaluation. A singular evaluation method tends to cause students to neglect the programming process, debugging abilities, and innovative practice. Therefore, course construction requires overall optimization in terms of course objectives, content systems, teaching methods,

practical links, resource platforms, and evaluation mechanisms.

3. Course Construction Ideas

This course construction adheres to the basic concept of being "student-centered, ability-oriented, task-driven, and supported by digital resources." Course objectives are reconstructed around three dimensions: knowledge, ability, and quality.[4][5] At the knowledge level, students should master basic C language knowledge and core content such as functions, pointers, and structures. At the ability level, they should be able to analyze problems, design algorithms, and solve problems using programming methods. At the quality level, they should form standardized programming habits, rigorous logical thinking, collaborative awareness, and continuous learning capabilities.

On this basis, the course construction forms an overall framework of "content reconstruction – method innovation – practice reinforcement – resource support – evaluation improvement." Content reconstruction emphasizes integrating syntax knowledge into project tasks.[6] Method innovation emphasizes combining blended learning with task-driven teaching to improve classroom participation and learning initiative. Practice reinforcement emphasizes building a step-by-step experimental system and utilizing open experimental platforms to better serve classroom teaching. Resource support emphasizes constructing micro-lecture videos, courseware, exercise banks, and online judging platforms to continuously optimize teaching through learning data feedback.

4. Reconstruction of Teaching Content and Reform of Teaching Methods

To change the disconnection between syntax knowledge and practical problems in traditional teaching, course content is reorganized in a modular manner. In addition to basic modules such as basic syntax, statements, and statement structures, emphasis is placed on helping students establish basic concepts of program execution and training their ability to use basic control structures to solve typical problems. Advanced modules such as pointers, structures, linked lists, and files focus on enhancing students' understanding of memory, data organization, and persistent storage. For students with surplus capacity, extension modules such as comprehensive practical tasks, competition problem training, and engineering application cases are set up to enhance the course's applicability and challenge.

The course employs a tiered instructional design to accommodate students with diverse backgrounds. It offers a spectrum of difficulty levels—ranging from fundamental exercises and standard programming tasks to open-ended projects and competitive challenges. [7] This structure ensures that the majority of students can master core learning objectives, while simultaneously providing ample room for advanced learners to excel.

Case-based and task-based content organization is an important method of course reform. Cases and tasks such as student grade statistics, simple calculators, address book management, and library borrowing record processing are designed to embed syntax knowledge points into cases.[8] Each case reflects the complete process of "problem abstraction – algorithm design – program implementation – testing and debugging," allowing multiple knowledge points

to be comprehensively applied within a continuous project.

In terms of teaching methods, the course constructs a blended learning model. Before class, teachers release micro-lecture videos, preview tasks, and self-test questions via online platforms to help students understand basic concepts in advance. In class, teachers conduct case demonstrations, live coding, and error analysis focusing on key and difficult points. After class, students complete layered assignments, case tasks, and extension exercises through the platform. Blended learning allows classroom time to be used more for problem-solving and ability training while transmitting knowledge.

Case and task-based teaching methods are heavily applied in the classroom. Teachers guide students to analyze and design around a specific problem, determining how to process the problem using C language programs. Task-driven teaching utilizes small tasks, comprehensive tasks, and open-ended tasks. Students complete requirements analysis, module division, code implementation, testing verification, and result presentation in groups. This not only trains students' programming abilities but also cultivates their teamwork and communication skills.

The course implements layered teaching and personalized guidance to address the significant differences in student foundations. Students with weak foundations undergo basic training, error analysis, and individual tutoring; students with stronger abilities can participate in competition problem training and extension tasks. Stage-based tests are conducted using online programming platforms, allowing students to view scores and errors immediately and receive feedback. Teachers can also understand student learning and test completion status at any time based on platform data, making teaching more targeted.

5. Practical Teaching System and Course Resource Construction

Practical teaching is the key link for ability improvement in programming courses. In recent years, we have continuously reformed and optimized the experimental teaching links of the course.

We optimized experiment types by appropriately reducing the proportion of single verification experiments and increasing design-oriented, comprehensive, and open-ended experiments. Basic experiments mainly train basic syntax, such as variable usage, branch structures, loop structures, function calls, and array operations. Comprehensive experiments require students to comprehensively apply multiple knowledge points to complete tasks of a certain scale, such as student grade management systems, address book programs, simple number-guessing games, and text file processing programs. Innovative experiments encourage students to choose their own topics based on their professional backgrounds, such as data statistical analysis, simple simulation programs, and file management tools. Through ladder-style experimental design, students can improve their programming abilities from easy to difficult and from local to overall.

The course introduces an Online Judging (OJ) platform to provide immediate feedback for students. After students submit their programs, the platform automatically completes compilation, execution, and test case comparison, helping students promptly discover logical errors, improper handling of boundary conditions, and input/output format errors.

Teachers can also view student submission counts, pass rates, error types, and learning progress through the platform, thereby more accurately grasping the overall learning status of the class. Platform data provides an objective basis for process evaluation and offers convenience for students' autonomous training.

Extracurricular programming training and competition integration are important extensions of practical teaching. The course can organize programming training camps, programming check-in activities, and stage-based challenge tasks to encourage students to maintain continuous programming habits. For students with surplus capacity, they can be guided to participate in programming competitions, innovation and entrepreneurship training projects, or course extension research. Some competition problems, after appropriate simplification, can also be transformed into classroom teaching resources, allowing students to improve their algorithmic thinking and program implementation capabilities while solving challenging problems.

Course resource construction revolves around the goal of being "learnable, usable, testable, and updatable." The teaching resource library includes micro-lecture videos, teaching courseware, typical cases, experiment guides, project task sheets, typical program examples, and a common error case library. Micro-lecture videos should not be too long and should focus on a single knowledge point or a typical problem, facilitating student preview before class and review after class. Experiment guides should clarify experiment objectives, task requirements, implementation ideas, and evaluation criteria to avoid students mechanically following steps. The common error case library collects high-frequency problems students encounter, such as uninitialized variables, array out-of-bounds, pointer misuse, loop condition errors, and file opening failures, to help students improve their debugging abilities.

Exercise banks and project libraries are constructed by classifying knowledge points, difficulty levels, and ability objectives. The exercise bank contains basic, improved, comprehensive, and open-ended questions, suitable for both daily practice and stage-based tests. The project library preserves typical project tasks formed in previous teachings and is continuously updated based on student feedback and technological developments. The online course platform undertakes functions such as resource release, assignment submission, online testing, discussion exchange, and data statistics, allowing the pre-class, in-class, and post-class learning processes to be fully recorded. Learning behavior data recorded by the platform, such as video viewing duration, quiz completion, assignment submission quality, and experiment pass rates, can provide a basis for teaching improvement.

Textbook and auxiliary material construction is equally important. The course can compile supporting handouts based on existing textbooks to highlight the characteristics of our university's students and teaching priorities. At the same time, it recommends programming specification documents, high-quality online resources, and reference textbooks to help students establish standardization awareness. Learning roadmaps can guide students to clarify learning objectives at different stages, reducing blind learning. Resource construction is not a one-time task but requires continuous supplementation, revision, and optimization in teaching practice to form a continuously updated course resource ecosystem.

6. Reform of Assessment and Evaluation and Analysis of Implementation Effects

Traditional C language course evaluation often relies mainly on final exam scores, making it difficult to fully reflect the process of students' programming ability formation. Therefore, the course establishes a diversified evaluation system, combining formative evaluation with summative evaluation, knowledge assessment with ability and quality assessment, and individual evaluation with group evaluation. Process evaluation mainly includes classroom participation, online learning progress, daily assignments, experiment completion, experiment report quality, project process performance, and stage-based tests. Summative evaluation examines students' program reading, program design, and debugging abilities through final exams.

Process evaluation indicators need to be specific and operable. Classroom participation looks not only at attendance but also at whether students participate in code discussions, error analysis, and result presentations. Online learning progress is recorded through video viewing, preview quizzes, and platform login status. Daily assignments should distinguish between completion, correctness, and standardization, avoiding scoring based solely on submission. Experiment evaluation looks not only at whether the program runs successfully but also at whether students complete it independently, can explain key code, and possess basic debugging abilities. Project evaluation combines performance in requirements analysis, module design, code quality, test results, document writing, and group presentation, reducing the "free-rider" phenomenon through peer evaluation, teacher evaluation, and process records.

Summative evaluation also needs corresponding optimization. Final exams should reduce pure memory-based questions and increase program reading, fill-in-the-blank, error correction, and program design questions. Test design should reflect comprehensive application, such as requiring students to select appropriate data structures, design function interfaces, handle boundary conditions, and provide test results based on problem descriptions. In this way, the exam no longer just tests whether students remember syntax but whether they truly possess the ability to solve problems using C language.

Evaluation results should form a continuous improvement loop. Teachers can use platform data to analyze student learning situations, such as which knowledge points have high error rates, which experiment tasks are difficult to complete, and which project links students are generally weak in. Based on this data, teachers can adjust classroom explanation focuses, supplement targeted exercises, optimize experiment task difficulty, and improve project guidance methods. In the next round of teaching, the effects of adjustments are observed and compared, forming a continuous improvement mechanism of "evaluation – feedback – improvement – re-evaluation."

From the implementation effects, students' performance in programming design questions has improved after course reform, and experiment completion rates and project participation have significantly increased. Blended learning allows students to preview and review at their own pace, while classroom coding and immediate feedback enhance learning engagement. Project-driven teaching helps students understand the connections between multiple knowledge

points and improves their document writing, teamwork, and result presentation abilities. The use of online programming platforms allows students to conduct programming training more frequently and discover their own problems through immediate feedback.

Student learning experience surveys also reflect positive changes. Most students believe that case-based teaching and project tasks make course content closer to practical applications. Layered assignments reduce pressure on students with weak foundations and meet the extension needs of students with stronger abilities. Some students reported that content such as pointers and structures remains difficult, indicating that advanced modules need further refinement in explanation paths and training gradients. Some students also hope for more project demonstrations and code specification guidance, suggesting that subsequent resource construction should strengthen typical project analysis and programming specification training.

Teacher teaching reflections indicate that course reform has achieved good results in improving students' practical abilities but also faces some challenges. Significant differences in student foundations require more refined task design and sufficient individual guidance for layered teaching. Project evaluation standards need further refinement to avoid uneven contributions in group cooperation. Online resource construction and platform maintenance require continuous investment, and it is difficult for individual teachers to maintain high-quality updates in the long term. Additionally, the programming foundation course should strengthen connections with subsequent courses such as Data Structures, Embedded Programming, and Software Engineering to form a continuous chain in the cultivation of students' programming abilities.

7. Conclusion

The construction of the "Fundamentals of Programming – C Language" premium course should aim at cultivating applied talents and carry out systematic reforms around course content, teaching methods, practical systems, resource platforms, and evaluation mechanisms. Through modular content reconstruction, case-based teaching organization, project-driven training, blended learning models, and diversified evaluation systems, the course can better support the formation of students' computational thinking and programming abilities. Practice shows that premium course construction not only improves students' learning initiative and programming practical abilities but also promotes the continuous updating of teachers' teaching methods and course

resources.

This course construction model has certain reference value for the reform of similar programming foundation courses. In the future, we can further deepen online-offline blended learning, improve micro-lecture resources, exercise banks, and project libraries, and strengthen the integrated application of online judging platforms and teaching process data. At the same time, we should explore the connection mechanism between C language courses and Python programming, Data Structures, Embedded Development, and other courses, promoting the cultivation of programming abilities throughout the entire talent cultivation process, and driving the course to develop continuously towards first-class and digital courses.

Acknowledgments

This work is financially supported by the Education and Teaching Reform Research Project of China University of Mining and Technology (Beijing) (J25ZD06).

References

- [1] Wu, Y. (2018). Building China's "Golden Courses". *China University Teaching*, (12), 4–9. (in Chinese)
- [2] Lin, J., & Ni, T. (2024). Exploration of teaching practice of C language programming course from the perspective of integration of specialization and innovation. *Computer Education*, (8), 11–13, 37. (in Chinese)
- [3] Tang, Q., Li, T., Wang, Y., et al. (2025). Practical teaching reform of C language programming course oriented to new engineering. *Computer Education*, (8), 54–57. (in Chinese)
- [4] Zhang, X., Gu, P., Guan, W., et al. (2024). Analysis of teaching reform practice of programming courses from the perspective of new engineering construction. *Modern Information Technology*, 8(14), 169–171, 180. (in Chinese)
- [5] Chang, F. (2025). Design and practice of C language programming teaching model based on OBE concept. *Modern Information Technology*, 9(11), 93–95. (in Chinese)
- [6] You, H., Nan, Y., Xu, Y., et al. (2026). Research on teaching reform of programming courses based on competitions. *Education Progress*, 16(1), 1612–1617. (in Chinese)
- [7] Liu, J., Wei, Z., & Tang, R. (2026). Data governance empowering hybrid teaching of computer fundamentals. *Cybersecurity and Data Governance*, 45(7), 47–54. (in Chinese)
- [8] Hu, S., She, Y., Lu, J., et al. (2021). Reform and practice of C programming teaching based on online-offline integration. *Journal of Yunnan Minzu University (Natural Sciences Edition)*, 30(2), 156–161. (in Chinese)