ECE1786 Final Report
CNavigator

Zixiao Qiu 1006103556
Xinyu Song 1010180956
Ming Yu 1011466423
Ruoxi Yu 1010110201

Word Count: 1815
Penalty: 0%

Permission

Members	Video	Final Report	Code
Zixiao Qiu	Yes	Yes	Yes
Xinyu Song	Yes	Yes	Yes
Ming Yu	Yes	Yes	Yes
Ruoxi Yu	Yes	Yes	Yes

Introduction

Traditional C programming courses often follow a one-size-fits-all approach, ignoring diverse student needs and leading to inefficiencies in learning. This can lead to frustration, slow progress, and inefficiencies in mastering C programming. Additionally, students frequently need help locating resources matching their current proficiency levels. **CNavigator** addresses these challenges by leveraging AI's real-time adaptation capability to create a personalized learning platform. It dynamically tailors lectures, quizzes, and feedback to individual needs, enabling students to advance at their own pace. This approach improves learning efficiency and aligns with the growing demand for intelligent, adaptive tools in modern education, offering a more engaging and effective solution for mastering C programming.

Illustration

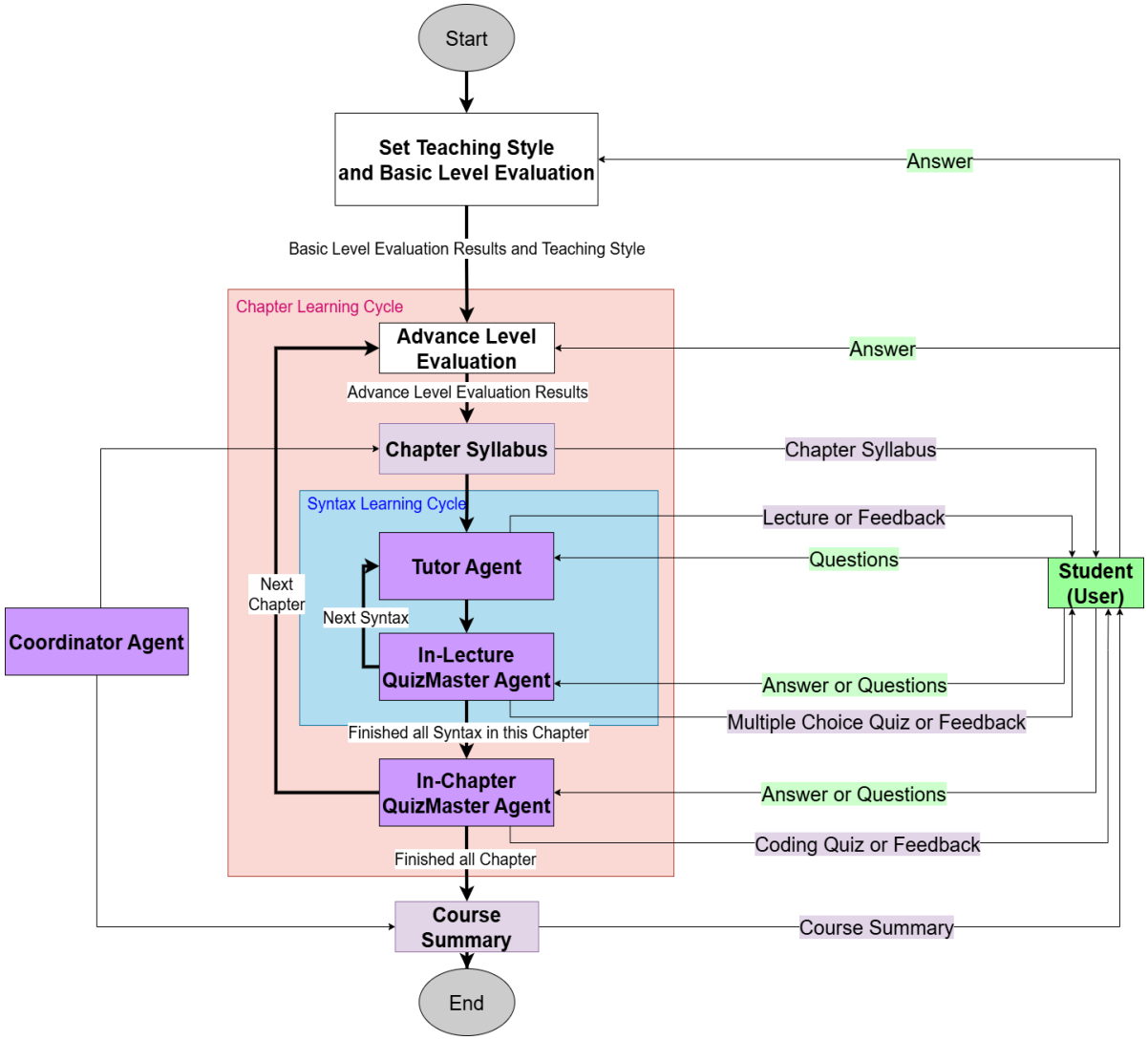


Figure 1. System Design

Background and Related Work

Several approaches have been proposed to address challenges in C programming education. Fu et al. (2017) introduced a real-time learning analytics system that detects errors and provides immediate feedback, enabling students to correct mistakes during exercises. This foundational work on adaptive feedback mechanisms informs the design of **CNavigator**, enhancing student engagement and learning outcomes [1].

Similarly, Daungcharone et al. (2019) developed a mobile game-based learning system for C programming, showing that interactive, engaging tools significantly improve student motivation and performance. **CNavigator** incorporates quizzes and personalized feedback to create a more engaging learning environment [2]. These works establish **CNavigator's** foundation, combining analytics and interactive features to address common limitations in traditional C programming education.

Data and Data Processing

Our data is derived from three sources, as shown in Figure 2: *C Primer Plus*, an authoritative textbook; *DEV_DOCS*, concise C syntax definitions; and *MISRA*, a coding standards document for professional practices.

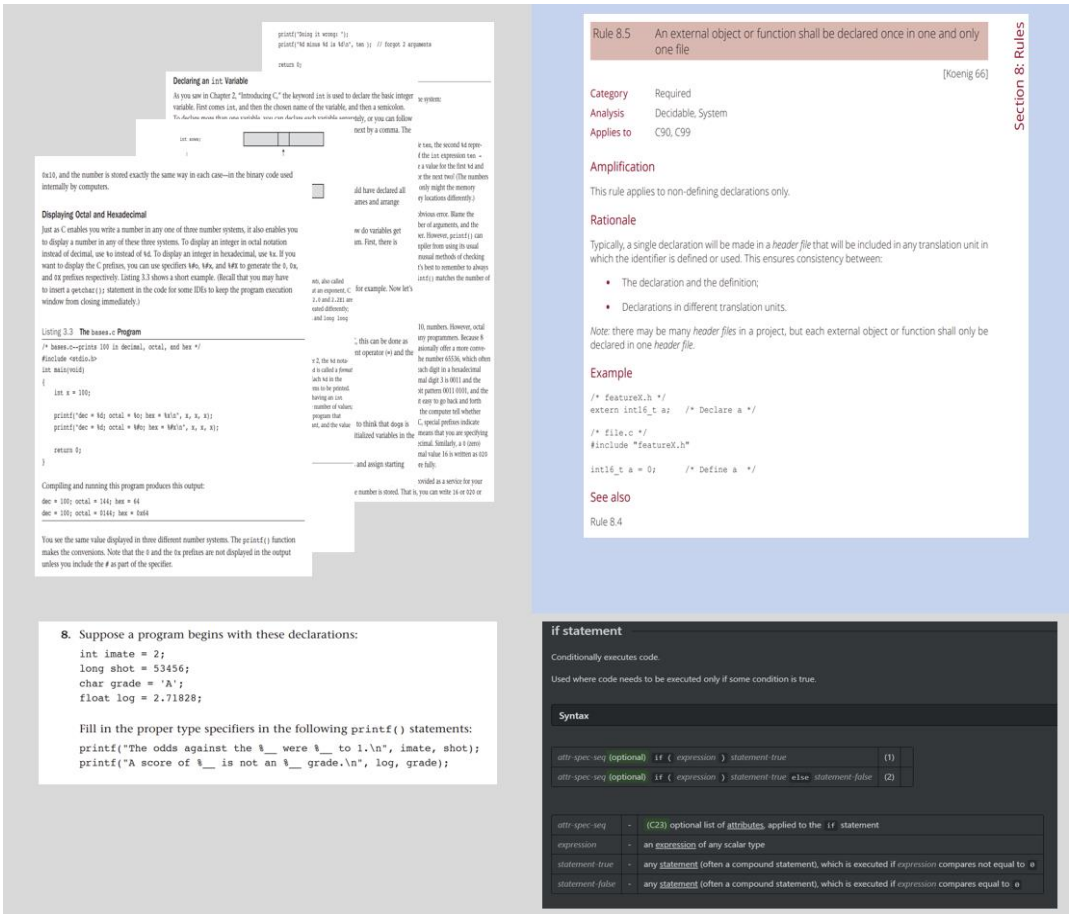


Figure 2. Data Sources

The raw data required manual cleaning to remove extraneous content, such as lengthy examples. We extracted 45 knowledge points across nine chapters from *C Primer Plus* and *DEV_DOCS*, labelling them with two difficulty levels to create a structured dataset (Figure 3) that aligns with educational goals and is the basis for generating system prompts. This structured dataset is the database for implementing the Retrieval-Augmented Generation (RAG) technique, enabling the system to dynamically access relevant and well-organized knowledge points to generate accurate and contextually appropriate prompts.

Chapter	Knowledge Point	Basic Content	Advanced Content
Data Types	int	The int type is used to store integers	Explore how signed and unsigned integers work, including overflow behavior and performance implications in arithmetic operations.

Figure 3. Knowledge Points Dataset

To facilitate personalized learning, we developed basic and advanced fixed initial tests (Figures 4 and 5) to assess students' proficiency. These tests were created using review questions from *C Primer Plus* and focused on fundamental and advanced concepts.

Chapter	Question	Option A	Option B	Option C	Option D	Answer

Data Types	Which of the following options correctly declares variables with data types int, float, and char in C?	int age = 25;
		float height = 5.9;
		char grade = 'A';
			integer age = 25;
			decimal height = 5.9;
			character grade = 'A';
				int age = "25";
				float height = "5.9";
				char grade = A;
					int age = '25';
					float height = 5.9;
					char grade = "A";
						A

Figure 4. Basic Fixed Initial Tests

Chapter	Knowledge Point	Question	Option A	Option B	Option C	Option D	Answer
Data Types	int	In a system performing high-frequency calculations which approach best handles potential integer overflow?	Use long long for all calculations	Convert all calculations to float	Implement overflow checks with appropriate unsigned types	Use only unsigned integers	C

Figure 5. Advanced Fixed Initial Tests

We also compiled a coding rules dataset from *MISRA* (Figure 6), targeting professional practices.

Rule	Requirement
Rule 6.1	Bit-fields shall only be declared with an appropriate type
Rule 6.2	Single-bit named bit fields shall not be of a signed type
Rule 7.1	Octal constants shall not be used
Rule 7.2	A 'u' or 'U' suffix shall be applied to all integer constants that are represented in an unsigned type
Rule 7.3	The lowercase character 'l' shall not be used in a literal suffix
Rule 7.4	A string literal shall not be assigned to an object unless the object's type is 'pointer to const-qualified char'

Figure 6. Coding Rules Dataset

All datasets were formatted into CSV files for integration into the system. Additionally, we labelled the generated content (Figure 7) into categories such as syllabus, quizzes, and feedback for evaluation on two aspects outlined in our rubric: **Accuracy** and **Relevance**.

Generated content
Syllabus
Lecture
In-lecture quiz
In-lecture quiz feedback
Chapter quiz
Chapter quiz feedback
Summary

Figure 7. Generated Content Labelling

This approach ensures the data is clean, well-structured, and tailored for adaptive learning.

Architecture

Figure 1 in the Illustration section outlines the system's architecture, which comprises four agents: **Coordinator**, **Tutor**, **In-lecture QuizMaster**, and **In-chapter QuizMaster**, using GPT-4o. These agents collaboratively deliver adaptive C programming education, each performing distinct roles to create an efficient and personalized environment. f

The system begins with fixed initial evaluations (Figure 8), split into basic questions covering all nine chapters and advanced questions targeting each chapter's syntax. This structure prevents students from being overwhelmed and allows the **Coordinator** and **Tutor** to deliver tailored course content effectively.

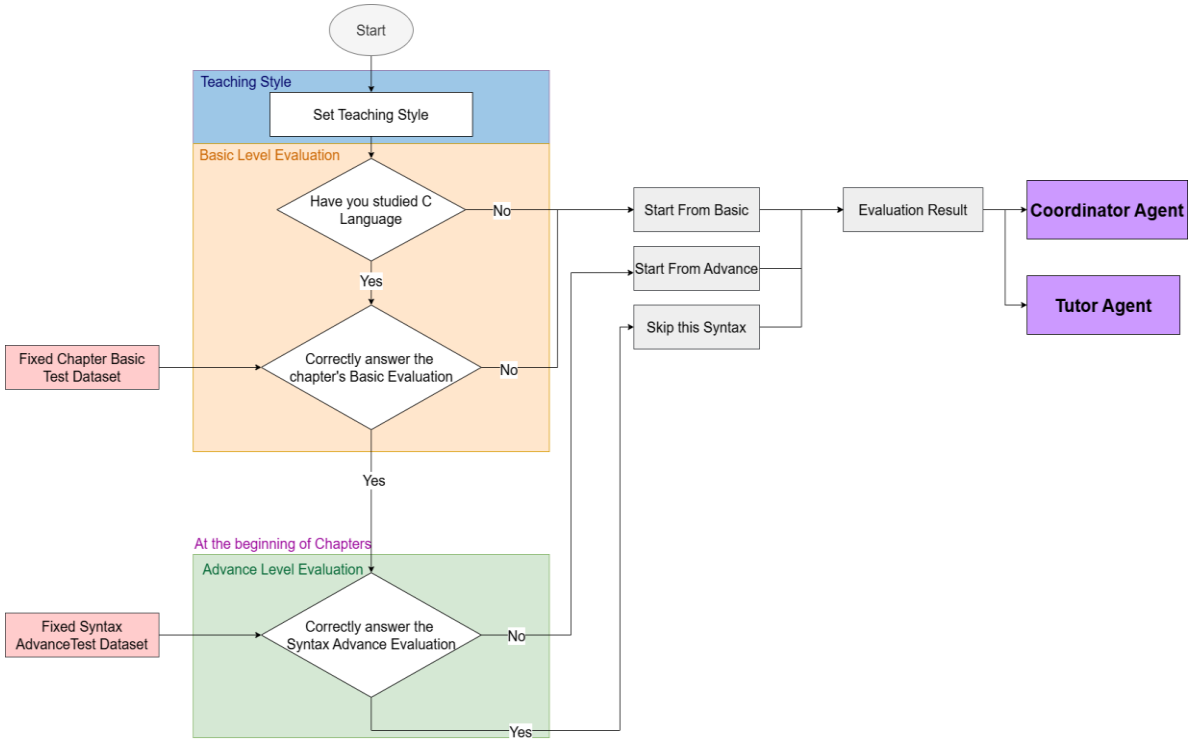


Figure 8. Level Evaluation Logic

The **Coordinator** introduces a chapter syllabus, providing a clear framework. This helps students understand the chapter's learning goal. The **Tutor** delivers interactive lectures, adapting content based on evaluations and conversation history to match students' understanding. Students can interact with the **Tutor** by asking questions, and the **Tutor** can give alternative explanations to ensure that students grasp each concept fully.

After each lecture, the **In-lecture QuizMaster** reinforces concepts with customized quizzes and feedback. Upon completing a chapter, the **In-chapter QuizMaster** evaluates understanding with coding quizzes, offering hints, corrections, and praise.

After all chapters, the **Coordinator** generates a personalized course summary based on the accumulated interactions.

This adaptive evaluation and teaching process tailors the learning journey to individual proficiency levels. The interaction among these agents enables **CNavigator** to dynamically adapt content, ensuring that each student's experience is personalized and effective. Additionally, the modular design ensures each agent performs a specific role and interacts seamlessly, making the system scalable and reproducible. Its scalability and replicability make it a model for creating similar adaptive systems.

Comparison

Traditional methods like BERTScore focus on textual similarity and cannot evaluate dynamic, customized systems. No standard quantitative evaluation method is explicitly tailored for personalized learning systems like ours. Existing adaptive systems primarily

assess user interaction, content relevance, and learning outcomes [3]. So, we developed a manual evaluation process to address the gap and chose the Coursera course *Programming Fundamentals* as a baseline. This course, offering static, non-adaptive lecture content and quizzes, provides a meaningful contrast to assess **CNavigator's** adaptive capabilities. Unlike document-based resources like W3Schools, Coursera delivers structured video lectures and quizzes, simulating a teacher-led learning experience. This structured and guided approach makes Coursera a more relevant and effective benchmark for evaluating the personalized and adaptive features of **CNavigator**.

Evaluation Method

We evaluated **CNavigator** and the baseline system under identical conditions, focusing on the following:

1. **Rubric-Based Evaluation**: Using a predefined rubric (Table 1), three experts rated **Accuracy** and **Relevance** (factual correctness and topic alignment), while three students assessed **Difficulty**, **Tone/Attitude**, and **Interaction** (content suitability, system friendliness, and feedback quality).

   Evaluators scored each aspect on a scale of 0, 0.5, and 1, with total scores aggregated for overall performance. This method ensures consistency and reproducibility, aligning with adaptive learning research practices [3].

Table 1. Evaluation Rubric

Aspects	**Details**	**Score**
Accuracy	Are the generated contents free from factual errors?	0-1
Relevance	Does the course content match our current topic?	0-1
Difficulty	Does the lecture content match the user's level?	0-1
Tone/Attitude	Friendly and patient during the courses?	0-1
Interaction	Reasonable interaction & helpful feedback?	0-1
Overall	Total score across all aspects.	0-5

2. **User Feedback**: Evaluators provided qualitative feedback on content clarity, engagement, learning efficiency, and satisfaction. These insights complement the quantitative results by highlighting system strengths and identifying areas for improvement.

This approach combines quantitative scoring with qualitative insights for a comprehensive evaluation.

Quantitative Results

Our evaluation framework employs a comprehensive rubric (Table 1) that measures five key dimensions: **Accuracy**, **Relevance**, **Difficulty**, **Tone/Attitude**, and **Interaction**. Multiple evaluators independently scored both systems for each lecture, ensuring objectivity and reliability. The scores represent averages across all lectures and evaluators, providing a holistic view of system performance.

Accuracy and **Relevance** were calculated as ratios of accurate to inaccurate and relevant to irrelevant content, respectively. **Difficulty, Tone/Attitude,** and **Interaction** scores were averaged across chapters from testers. Appendix A contains detailed evaluation data.

Based on Table 2, both systems perform well in **Accuracy** and **Relevance**, with the baseline scoring perfectly in both. **CNavigator** scored slightly lower in **Relevance** (0.99) due to occasional mismatches but outperformed the baseline significantly in **Difficulty** (0.86 vs 0.57), demonstrating its adaptability. It also achieved perfect scores in **Tone/Attitude** and **Interaction**, highlighting its effectiveness in creating an engaging learning environment.

CNavigator's cumulative score of 4.82 versus the baseline's 3.85 validates its effectiveness in delivering personalized, high-quality learning. These results strongly support our adaptive programming education approach.

Table 2. Performance Comparison Scores

Aspects	**CNavigator**	**Baseline**
Accuracy	1.00	1.00
Relevance	0.99	1.00
Difficulty	0.86	0.57
Tone/Attitude	1.00	0.71
Interaction	1.00	0.57
Overall	4.82	3.85

Qualitative Results

For the qualitative evaluation, we relied on feedback from six evaluators. The results confirm **CNavigator's** personalized and adaptive learning experience, aligning with quantitative results across **Accuracy**, **Relevance**, **Difficulty**, **Tone/Attitude**, and **Interaction**.

The system effectively tailors lecture difficulty based on initial test results (Figures 9-10) and offers teaching style options.

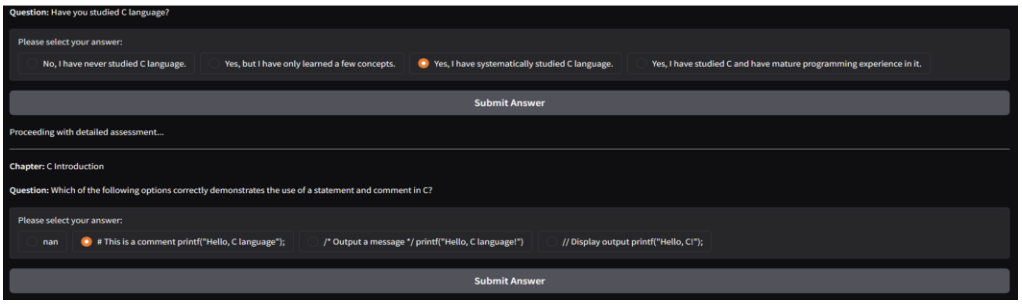


Figure 9. Initial Test

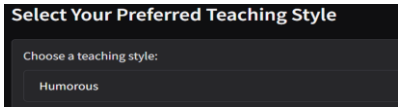


Figure 10. Teaching Style Selection

Each chapter presents a structured syllabus (Figure 11) with clear learning objectives. The lecture content integrates with learning standards and provides customized feedback (Figure 12).

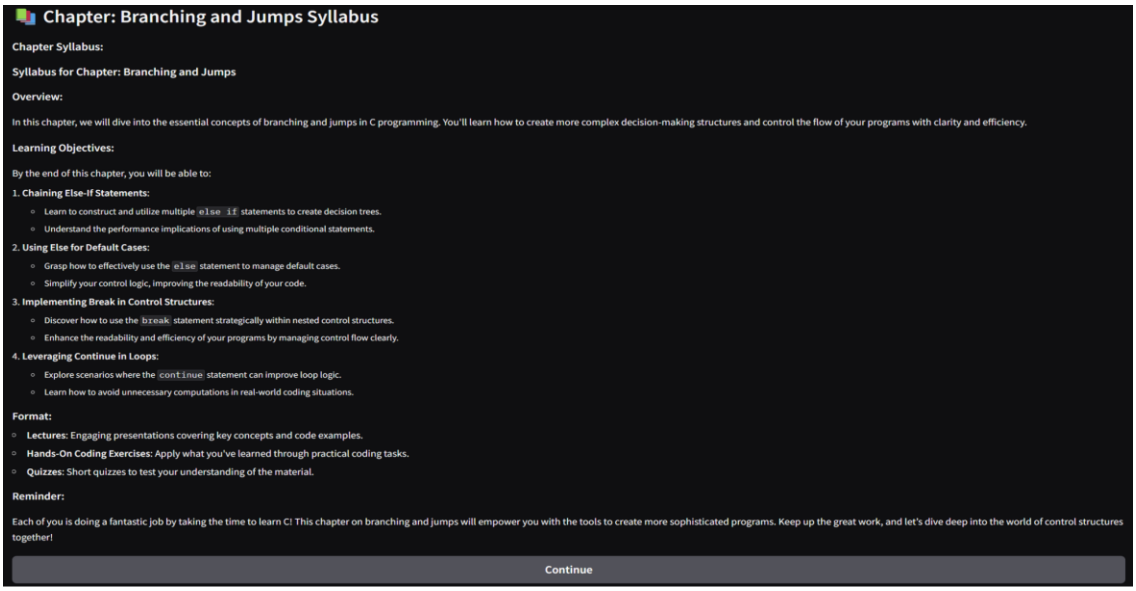
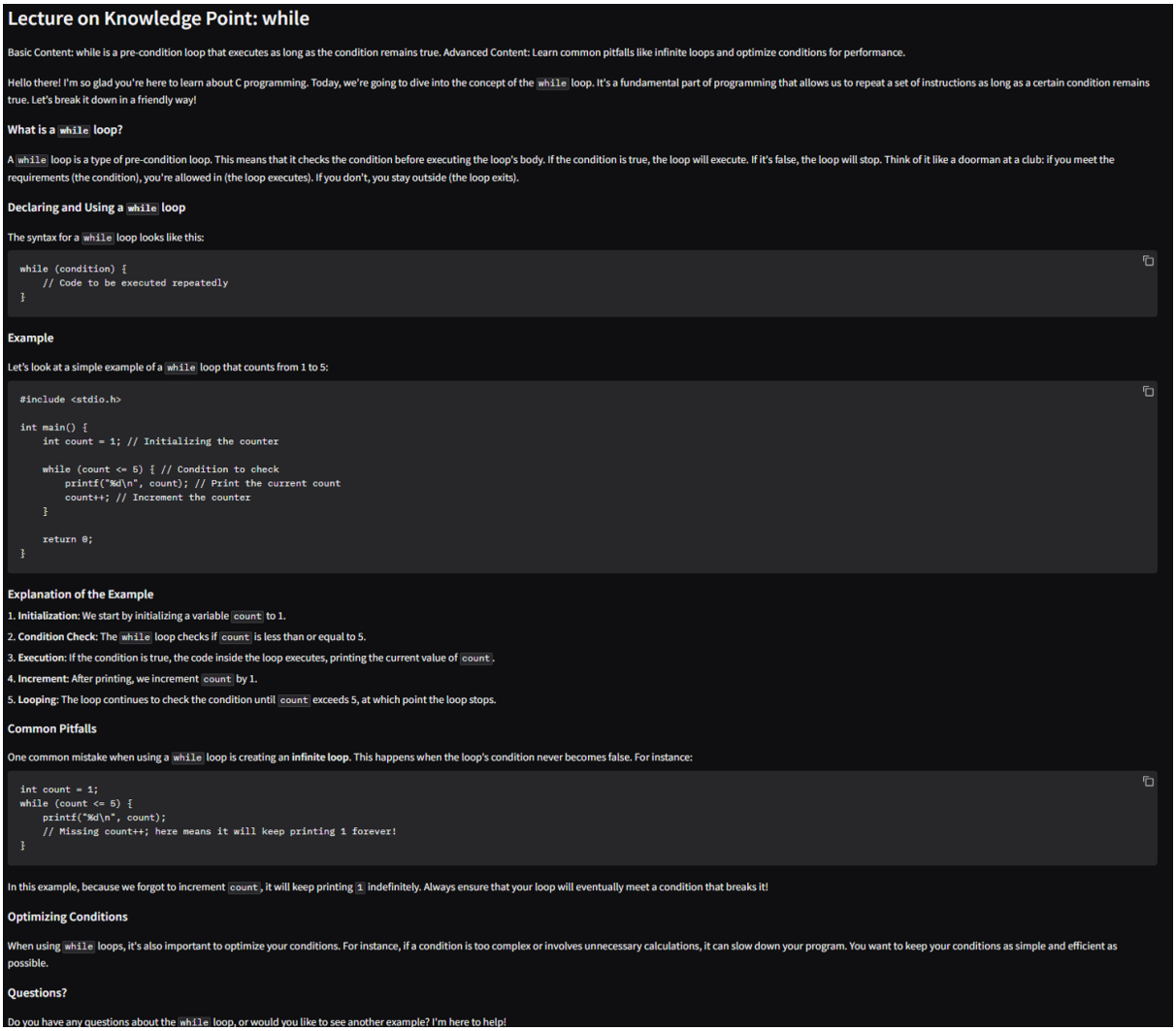


Figure 11. Chapter Syllabus



Lecture on Knowledge Point: while

Basic Content: while is a pre-condition loop that executes as long as the condition remains true. Advanced Content: Learn common pitfalls like infinite loops and optimize conditions for performance.

Hello there! I'm so glad you're here to learn about C programming. Today, we're going to dive into the concept of the `while` loop. It's a fundamental part of programming that allows us to repeat a set of instructions as long as a certain condition remains true. Let's break it down in a friendly way!

What is a `while` loop?

A `while` loop is a type of pre-condition loop. This means that it checks the condition before executing the loop's body. If the condition is true, the loop will execute. If it's false, the loop will stop. Think of it like a doorman at a club: if you meet the requirements (the condition), you're allowed in (the loop executes). If you don't, you stay outside (the loop exits).

Declaring and Using a `while` loop

The syntax for a `while` loop looks like this:

```
while (condition) {
    // Code to be executed repeatedly
}
```

Example

Let's look at a simple example of a `while` loop that counts from 1 to 5:

```
#include <stdio.h>

int main() {
    int count = 1; // Initializing the counter

    while (count <= 5) { // Condition to check
        printf("%d\n", count); // Print the current count
        count++; // Increment the counter
    }

    return 0;
}
```

Explanation of the Example

1. **Initialization**: We start by initializing a variable `count` to 1.
2. **Condition Check**: The `while` loop checks if `count` is less than or equal to 5.
3. **Execution**: If the condition is true, the code inside the loop executes, printing the current value of `count`.
4. **Increment**: After printing, we increment `count` by 1.
5. **Looping**: The loop continues to check the condition until `count` exceeds 5, at which point the loop stops.

Common Pitfalls

One common mistake when using a `while` loop is creating an **infinite loop**. This happens when the loop's condition never becomes false. For instance:

```
int count = 1;
while (count <= 5) {
    printf("%d\n", count);
    // Missing count++; here means it will keep printing 1 forever!
}
```

In this example, because we forgot to increment `count`, it will keep printing `1` indefinitely. Always ensure that your loop will eventually meet a condition that breaks it!

Optimizing Conditions

When using `while` loops, it's also important to optimize your conditions. For instance, if a condition is too complex or involves unnecessary calculations, it can slow down your program. You want to keep your conditions as simple and efficient as possible.

Questions?

Do you have any questions about the `while` loop, or would you like to see another example? I'm here to help!

Figure 12. Lecture Notes

Real-time interaction (Figure 13 and 14) reduces pressure and boosts confidence, allowing users to ask questions freely.

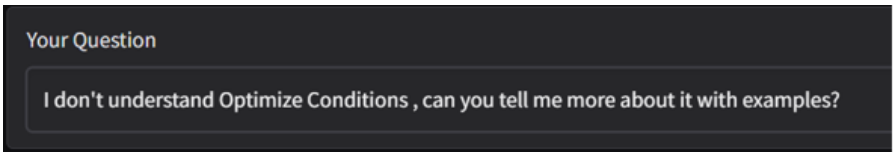


Figure 13. Question Box

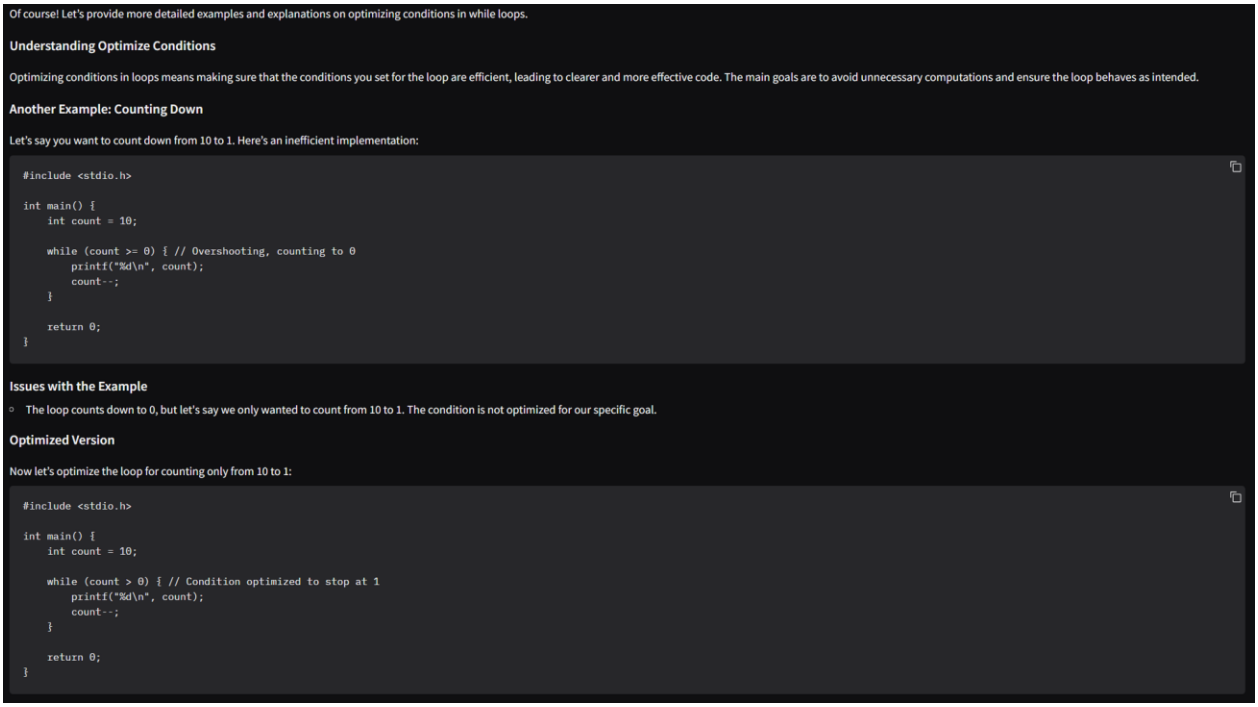


Figure 14. Real-Time Interaction

The system incorporates customized in-lecture quizzes based on conversation history, including personalized feedback. (Figure 15)

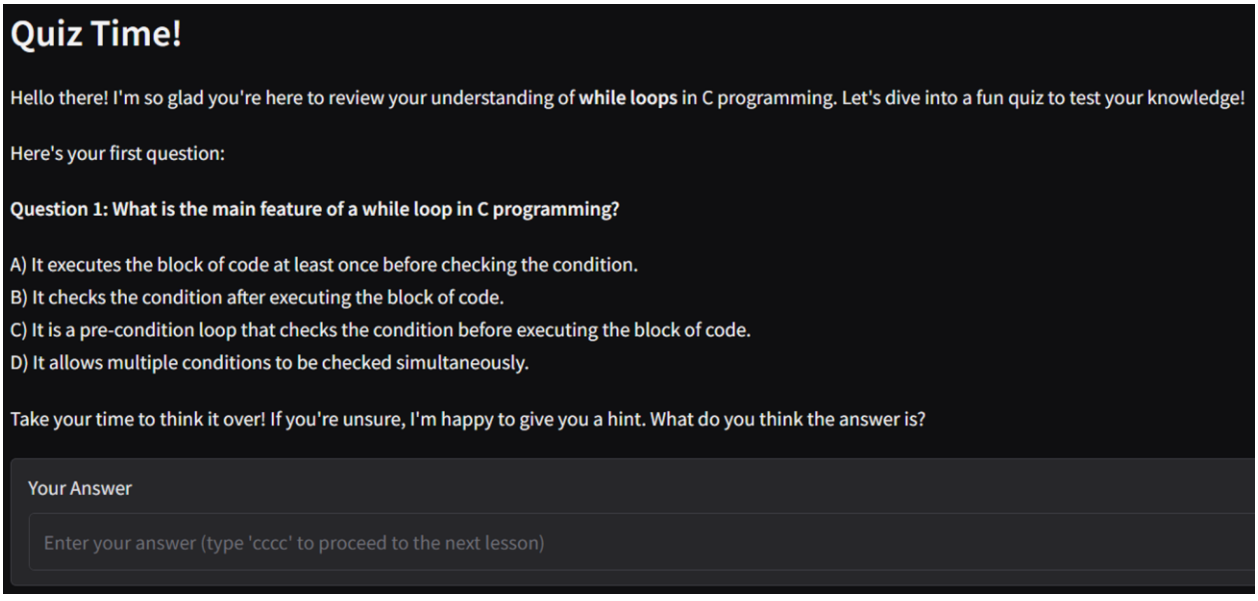


Figure 15. Lecture Quiz

While the system performs well, we identified some minor content sequencing issues, like including “else” in a quiz about “else if,” which hadn’t been taught yet. (Figure 16)

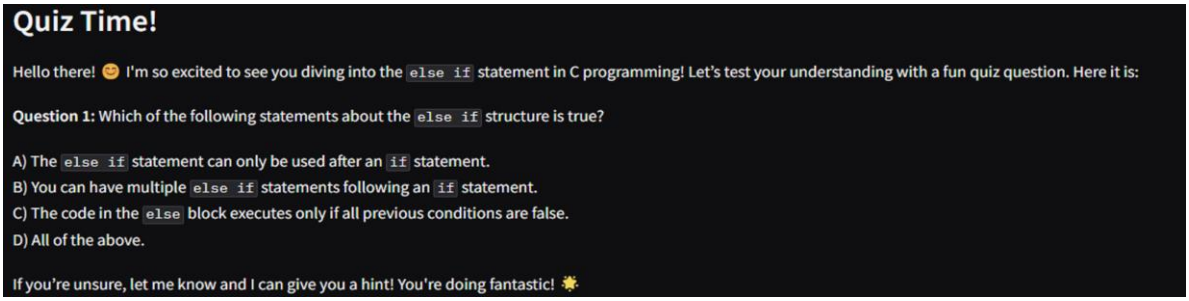


Figure 16. Irrelevant Example

The system also generates tailored coding questions, guides coding standards, and highlights non-standard practices. (Figures 17 and 18)

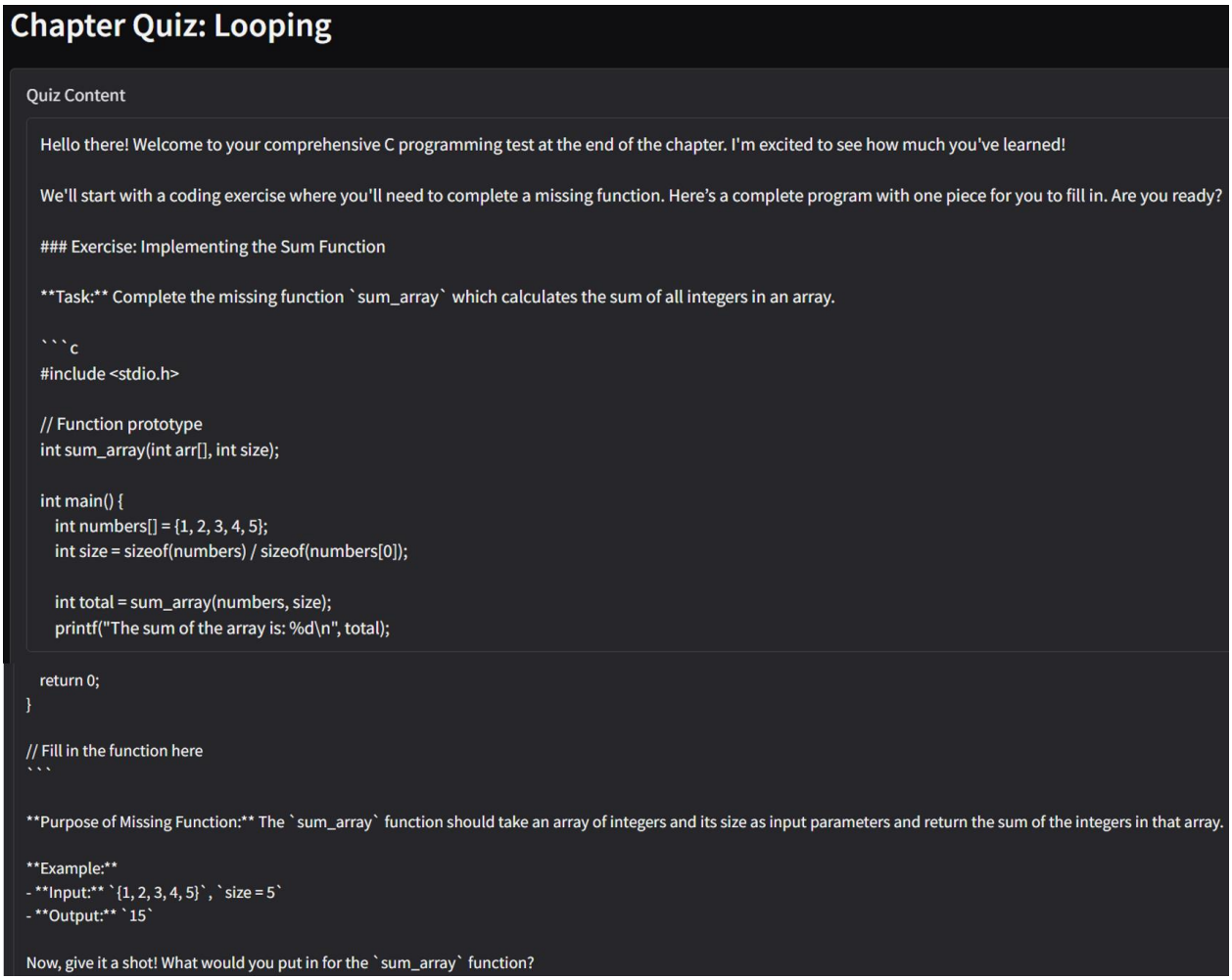


Figure 17. Chapter Quiz

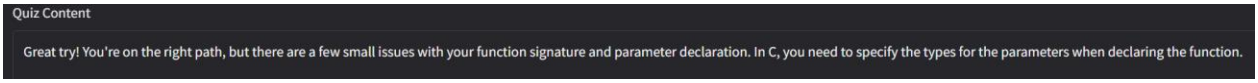


Figure 18. Feedback

CNavigator concludes the course with a comprehensive and personalized summary. (Figure 19)

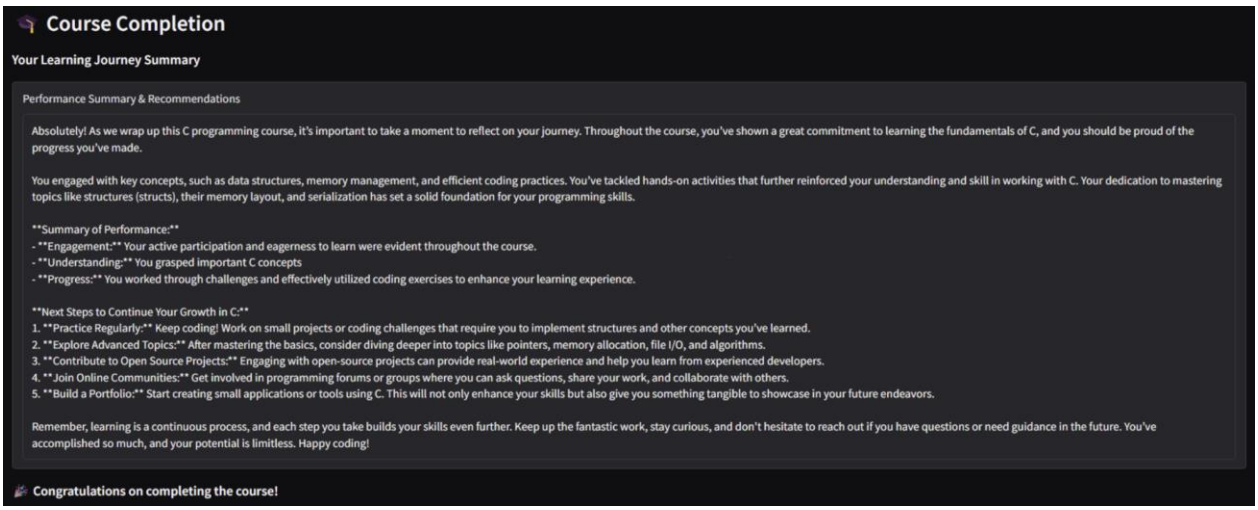


Figure 19. Course Summary

The system presents basic concepts for users who need prior programming experience (Figures 20 and 21).

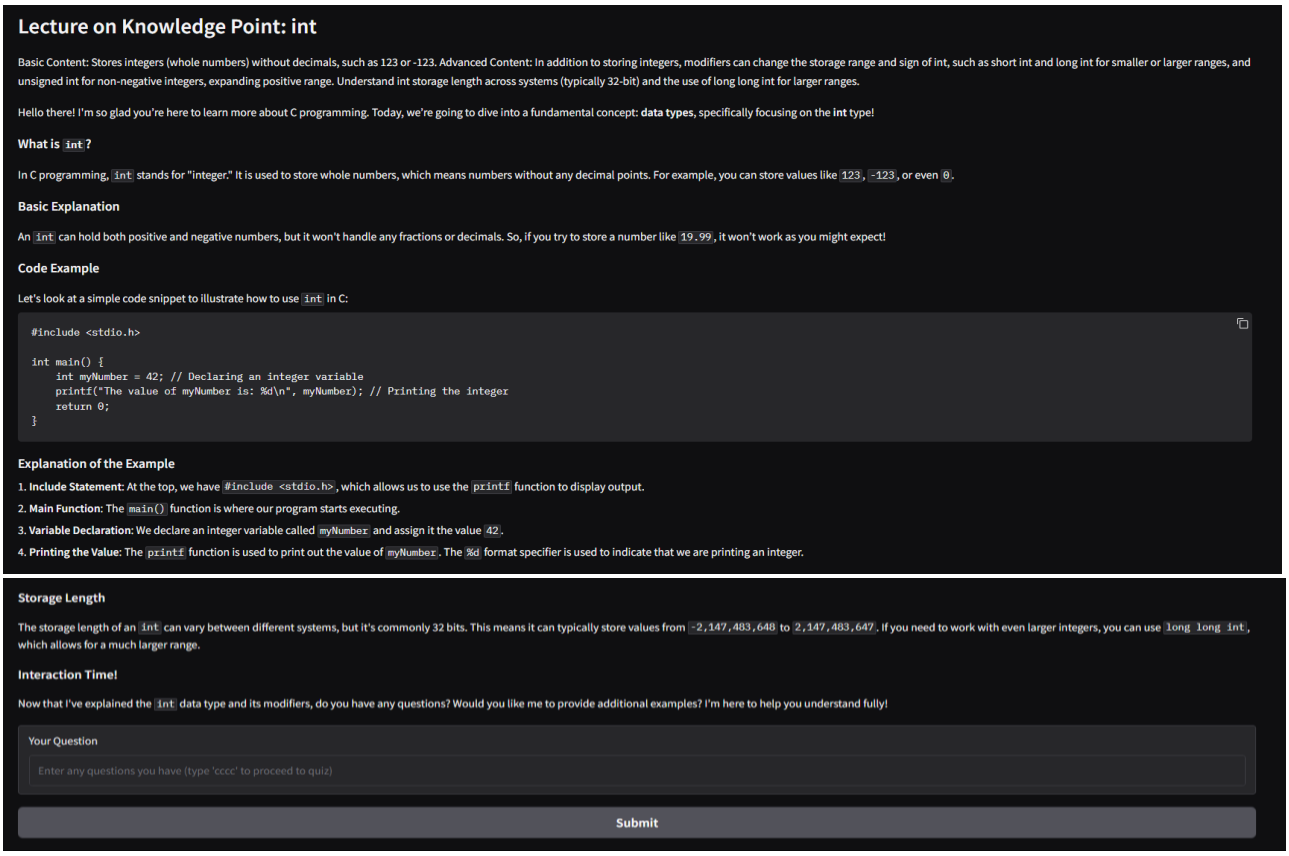


Figure 20. Basic Level Lecture

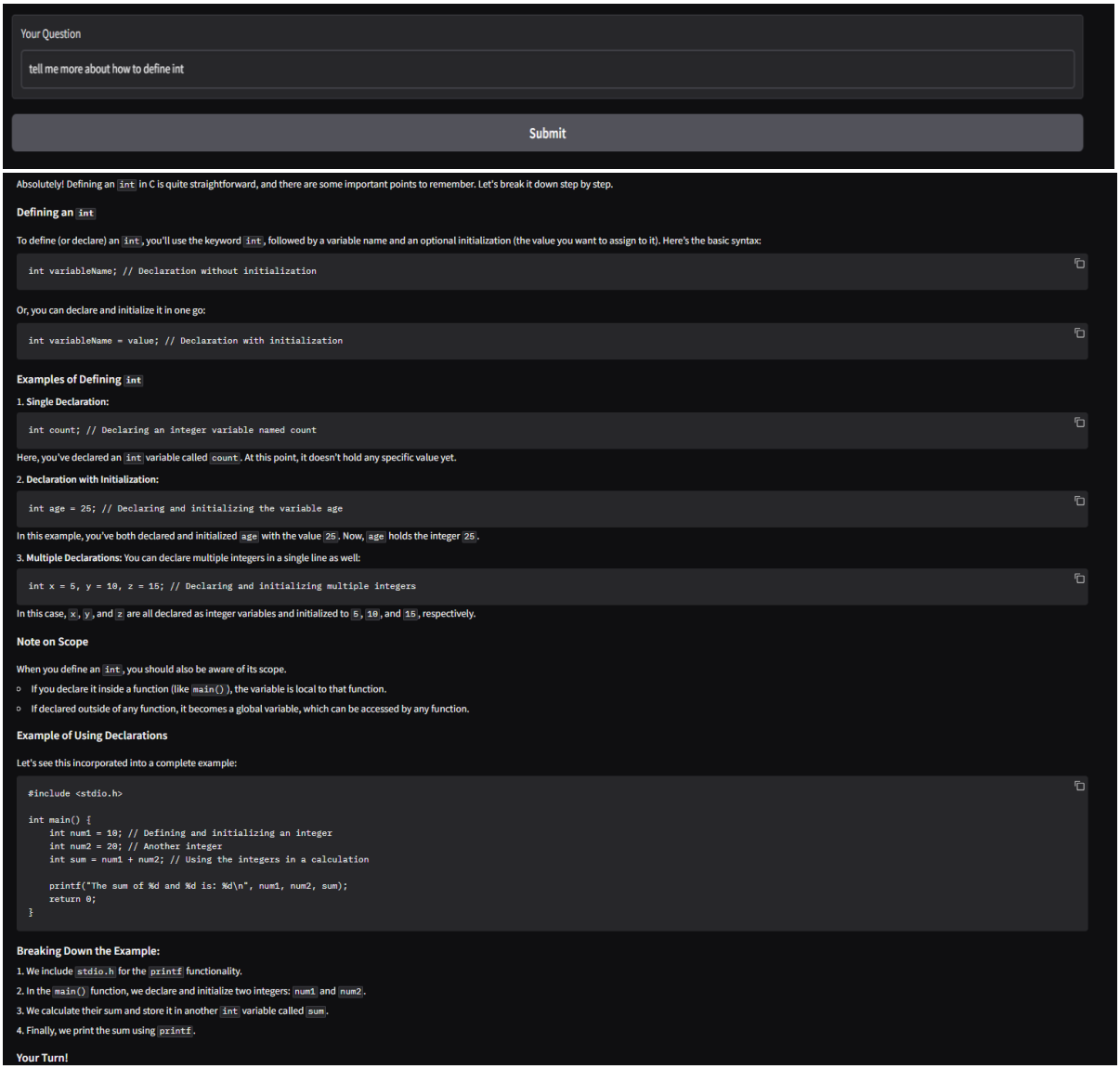


Figure 21. Basic Level Interaction

These observations highlight **CNavigator's** strengths in the **Interaction** and **Difficulty** dimensions, where it achieved perfect scores of 1 and 0.86. Evaluators emphasized its adaptability and real-time feedback as key advantages.

Discussion

CNavigator performs well in providing a user-friendly and adaptive learning experience. Evaluators praised its teaching loop for effectively transforming traditional learning methods with personalized instruction. Powered by RAG, its reliable content establishes it as a robust educational tool.

However, **CNavigator** has limitations. The basic text input for answering programming questions lacks interactivity, and the absence of multimedia formats, like voice or video,

reduces accessibility and engagement. Future updates will prioritize addressing these issues to enhance functionality.

This project highlighted the value of precise project management, including clear goals, defined roles, and regular progress reviews. We also realized GPT-4o's potential by focusing on prompt optimization rather than coding changes. These insights will guide future projects, including integrating real-time coding environments and multimedia features.

Individual Contributions

Zixiao Qiu:

- Designed prompts;
- Built user interaction module;
- Built Coordinator workflow code.

Xinyu Song:

- Processed datasets;
- Built chapter module;
- Implemented front end.

Ming Yu:

- Designed prompts;
- Built teaching module;
- Implemented RAG;
- Implemented initial tests and quiz modules.

Ruoxi Yu:

- Processed datasets;
- Designed logging system;
- Implemented front end.

All members contributed to the architecture design, evaluation, and report.

References

[1] X. Fu, A. Shimada, H. Ogata, Y. Taniguchi, and D. Suehiro, "Real-time learning analytics for C programming language courses," in *Proc. 7th Int. Conf. Learning Analytics & Knowledge*, 2017, pp. 171-178.

[2] K. Daungcharone, P. Panjaburee, and K. Thongkoo, "A mobile game-based C programming language learning: results of university students' achievement and motivations," *Int. J. Mobile Learning Organ.*, vol. 13, no. 2, pp. 171-192, 2019.

[3] P. S. Pawar, A. Pal, and A. Chavan, "A systematic literature review on adaptive learning systems using learning analytics," *Educational Technology Research and Development*, vol. 68, no. 6, pp. 3491-3515, 2020.

Appendix A

Generated content	**Accuracy**	**Inaccuracy**
Syllabus	27	0
Lecture	135	0
In-lecture quiz	134	1
In-lecture quiz feedback	135	0
Chapter quiz	27	0
Chapter quiz feedback	27	0
Summary	3	0
Overall	488	1

Generated content	**Relevance**	**Irrelevance**
Syllabus	27	0
Lecture	134	1

In-lecture quiz	131	4
In-lecture quiz feedback	135	0
Chapter quiz	26	1
Chapter quiz feedback	27	0
Summary	3	0
Overall	483	6

Chapter	**CNavigator – Difficulty**	**Baseline – Difficulty**
C Introduction	0.83	0.67
Data Types	1	0.83
Character Strings and Formatted Input and Output	1	0.5
Operators	1	0.67
Looping	0.67	0.67
Branching and Jumps	0.67	0.5
Functions	0.67	0.5
Arrays and Pointers	0.83	0.33
C Structures	1	0.5
Non-chapter content	1	0.5
Overall	0.86	0.57

Chapter	**CNavigator – Tone/Attitude**	**Baseline – Tone/Attitude**

C Introduction	1	0.83
Data Types	1	0.83
Character Strings and Formatted Input and Output	1	0.67
Operators	1	0.67
Looping	1	0.67
Branching and Jumps	1	0.83
Functions	1	0.67
Arrays and Pointers	1	0.5
C Structures	1	0.67
Non-chapter content	1	0.83
Overall	1	0.71

Chapter	**CNavigator – Interaction**	**Baseline – Interaction**
C Introduction	0.83	0.5
Data Types	1	0.83
Character Strings and Formatted Input and Output	1	0.33
Operators	1	0.5
Looping	0.67	0.83
Branching and Jumps	0.67	0.5
Functions	0.67	0.5
Arrays and Pointers	0.83	0.5
C Structures	1	0.67

Non-chapter content	1	0.5
Overall	0.86	0.57