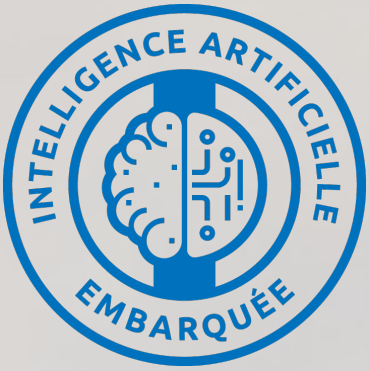
House of Code - Phase 2

By Jihade Gharby & Mohamed Hossame Baiz

March 24, 2025





# House of Code: Phase 2 "Tome of Endless Knowledge"

# Contents

# 1 Introduction: A Quest for Hidden Wisdom

In the great kingdom of House of Code, knowledge is recorded in ancient tomes, preserved through generations. These tomes do not reveal their secrets easily. A wise scholar must learn to extract knowledge line by line, never reading more than what is needed, lest the wisdom be lost in the void.

Thus, the High Maesters of the realm have decreed a new challenge: to craft a function capable of unveiling one line at a time from these sacred texts, ensuring that the flow of wisdom is never interrupted. This is the **Tome of Endless Knowledge**, where the words are infinite, but only the most disciplined minds may retrieve them properly.

# 2 Sacred Scrolls of Knowledge (Resources)

To prepare for this trial, scholars are encouraged to study the following:

- File Handling in C - GeeksforGeeks
- Static Variables in C - Wikipedia
- Buffer Management and Memory Allocation - GeeksforGeeks
- Understanding File Descriptors - Linux Programming
- How Read() Works in C - GeeksforGeeks

# 3 The Laws of the Realm (General Rules)

## 3.1 Tools of the Scholar

To embark on this noble quest, a scholar must wield:

- A Unix-based system (Linux or macOS).
- A compiler such as GCC.
- A deep understanding of file descriptors and buffer management in C.

## 3.2 Forging the Scroll Decoder (Compilation)

To prepare your tools for uncovering hidden knowledge, compile your sacred function:

```
# Compile the Tome Reader
gcc -Wall -Wextra -Werror -D BUFFER_SIZE=13 tome_reader.c tome_utils.c -o
    tome_reader
```

The `BUFFER_SIZE` is a mystical force that controls how much knowledge is retrieved at once. It may be altered to test different methods of study.

### 3.3 The Ritual of Knowledge Extraction

To begin reading from a sacred tome, invoke the command:

```
./tome_reader  ancient_texts.txt
```

The Tome Reader shall reveal the text line by line, ensuring that each secret is uncovered in due time, without overreaching into forbidden knowledge.

### 3.4 The Scholar's Code

- Each invocation of the Tome Reader shall return a single complete verse (line) from the tome.
- The function must respect the flow of knowledge, ensuring that no extra lines are read prematurely.
- The function must handle both standard input and text files with equal mastery.
- Global variables are forbidden, for discipline in knowledge must be maintained.
- The function must work under varying `BUFFER_SIZE` values, whether small or large.
- The `lseek` function is forbidden, for once knowledge has been revealed, it cannot be revisited.

## 4 Common Instructions

- Your functions should not quit unexpectedly (segmentation fault, bus error, double free, etc.) apart from undefined behaviors. If this happens, your project will be considered non-functional.
- All heap-allocated memory space must be properly freed when necessary. No memory leaks will be tolerated.
- Ensure proper error handling in all cases, especially when reading from files or standard input.

## 5 Function Specifications

### 5.1 Core Function

```
char *tome_reader(int fd);
```

### 5.2 Parameters

- `fd`: File descriptor to read from.

### 5.3 Return Value

- `char *`: Read line (correct behavior).
- `NULL`: Nothing else to read or an error occurred.

### 5.4 External Functions Allowed

- `read`
- `malloc`
- `free`

## 6 Path to Higher Enlightenment (Bonus Challenges - Advanced Optional)

For those seeking true mastery, additional feats of wisdom may be pursued:

- **The Grand Archivist's Mastery**: The scholar must learn to manage multiple tomes (file descriptors) at once, ensuring that the wisdom of different texts does not become entangled.
- **The Purest Form of Knowledge**: The function must be crafted with only one static variable, achieving ultimate efficiency and elegance.

## 7 Instructions for the Tome Utils (Helper Functions)

To complete this quest, you are encouraged to create a `tome_utils.c` file, which will contain useful helper functions, such as:

- **Memory Management Functions**: Ensure dynamic memory is allocated and freed properly.
- **String Handling Functions**: Custom implementations for handling and processing text, avoiding standard library functions where necessary.
- **Buffer Manipulation**: Managing how data is read into buffers efficiently, without unnecessary overhead.

## 8 Submission Instructions

Scholars must present their work using one of the following methods:

- **GitHub Submission**: Upload your repository and provide the link.
- **Google Drive**: Upload your source files and share the link.
- **Other File-Sharing Platforms**: If unable to use the above, any accessible method is allowed.

## 8.1 Final Verification Before Submission

- Ensure your repository or file has public access for evaluation.
- Verify that your code compiles and runs correctly with different `BUFFER_SIZE` values.
- Include comments to explain your logic and approach.

**The House of Code awaits your knowledge. Will you rise as a true scholar, or will the secrets of the Tome of Endless Knowledge remain beyond your grasp?**