# Implementation and Analysis of a Knowledge-Based Chatbot in C++ Practical

Report September 11, 2025 Prepared by: Grok (xAI Assistant)

# 1 Title

Implementation and Analysis of a Knowledge-Based Chatbot in C++

# 2 Objective

The objective of this practical is to develop a C++ program that implements a simple chatbot, named SmartBot, capable of:

- Responding to user queries based on a stored knowledge base.
- Learning new question-answer pairs interactively from the user.
- Persisting the knowledge base to a file for future use.
- Demonstrating string manipulation, file I/O, and data structures like unordered maps.

This practical aims to showcase basic natural language interaction, persistent storage, and the use of C++ standard libraries for building an extensible conversational system.

# 3 Introduction

Chatbots are software agents designed to simulate human conversation, often used in customer service, education, or entertainment. This implementation creates a console-based chatbot that uses a knowledge base stored in a text file to answer user queries. The bot supports case-insensitive matching, learns new responses, and saves them for persistence. The program leverages C++ standard libraries such as <iostream>, <string>, <unordered_map>, <fstream>, and <algorithm> for input/output, string processing, hash table storage, file handling, and text transformation, respectively.

The N-Queens problem, discussed in a previous practical, used backtracking to solve a combinatorial puzzle. In contrast, this chatbot focuses on interactive data management and string processing, highlighting different aspects of algorithmic design and practical application in C++.

# 4 Methodology (What Was Done)

The program was developed with the following steps:

1. Initialization: Load an existing knowledge base from a text file (knowledge.txt) into an unordered map, where keys are questions (lowercase) and values are answers.
2. User Interaction: Prompt the user for input in a loop until they type ”exit”.
3. Query Processing: Convert user input to lowercase and check the knowledge base for a matching question.
4. Response Handling: If a match is found, display the stored answer. Otherwise, offer to learn a new answer, which is then saved to the knowledge base and file.

5. Persistence: Save new question-answer pairs to the file to ensure persistence across sessions.

The program was tested with sample inputs, including known and unknown questions, to verify functionality and file operations.

# 5 Implementation Details (How It Was Done)

The code is modular, with functions for string processing, file I/O, and the main interaction loop.

## 5.1 toLowerCase(string str)

- Purpose: Converts a string to lowercase for case-insensitive matching.
- How It Works: Uses std::transform with ::tolower to convert each character.
- Efficiency: O(n), where n is the string length.

## 5.2 loadKnowledgeBase(const string& filename)

- Purpose: Loads question-answer pairs from a file into an unordered map.
- How It Works: Reads the file line by line, treating alternating lines as questions and answers, converting questions to lowercase.
- Key Aspects: Returns an empty map if the file doesn't exist. Assumes correct file format (question and answer pairs).

## 5.3 saveKnowledgeBase(const unordered_map<string, string>& kb, const string& filename)

- Purpose: Saves the knowledge base to a file.
- How It Works: Writes each question-answer pair to the file, separated by newlines.
- Key Aspects: Overwrites the file, ensuring the latest knowledge base is stored.

## 5.4 main()

- Purpose: Manages user interaction and program flow.
- How It Works:
  - Loads the knowledge base.
  - Enters a loop prompting for user input.
  - Exits if input is "exit".
  - Checks the knowledge base for the lowercase input.
  - If no match, asks to teach a new answer, updating the knowledge base and file if provided.

### 5.5 Compilation and Execution

- Compiled with a C++ compiler (e.g., g++ with C++11 or later).
- Example Run: User inputs "hello", gets a response if known, or teaches a new response like "Hi there!" which is saved.

## 6 Results

The program was tested with a sample knowledge.txt file containing:

what is the capital of france
Paris
who is elon musk
CEO of Tesla and SpaceX

Sample interaction:

SmartBot: Hello! I am your assistant. Type 'exit' to quit.

You: what is the capital of france
SmartBot: Paris

You: what is python
SmartBot: I dont know the answer. Can you teach me? (yes/no)
You: yes
Please provide the correct answer:
A programming language
SmartBot: Thank you! I will remember this.

You: exit
SmartBot: Goodbye!

After teaching, knowledge.txt updated to include:

what is python
A programming language

The program correctly loaded, responded, learned, and saved new knowledge, with no crashes or errors.

## 7 Analysis in Detail

### 7.1 Algorithm Correctness

The chatbot correctly handles:

- Case-insensitive queries using toLowerCase.
- File I/O for persistent storage.
- Interactive learning, updating the knowledge base in memory and on disk.

Correctness was verified by matching known responses and checking file updates.

## 7.2 Time Complexity

- toLowerCase: O(n), where n is string length.
- loadKnowledgeBase: O(m), where m is the number of lines in the file.
- saveKnowledgeBase: O(k), where k is the number of entries in the knowledge base.
- Lookup in unordered_map: O(1) average case.
- Main loop: Depends on user interactions, but each query is O(n) for string conversion plus O(1) for lookup.

## 7.3 Space Complexity

- O(k) for the unordered_map, where k is the number of question-answer pairs.
- O(n) for temporary strings during input processing.

## 7.4 Performance Observations

- Responsiveness: Instantaneous for small knowledge bases due to O(1) hash table lookups.
- File I/O: Bottleneck for large files, but negligible for typical use (e.g., <1000 entries).
- Limitations: No error handling for file issues (e.g., missing or corrupted knowledge.txt). Assumes well-formed input. No partial matching or fuzzy search for queries.
- Scalability: Suitable for small to medium knowledge bases. Large bases may slow down file I/O but not lookups.

## 7.5 Strengths

- Simple and extensible design.
- Persistent storage ensures knowledge retention.
- Case-insensitive matching improves usability.

## 7.6 Weaknesses

- No error handling for invalid file formats or I/O failures.
- Exact-match queries limit flexibility (e.g., no synonyms or partial matches).
- Console-based; no GUI for better user experience.

# 8 Conclusion

This practical successfully implemented a knowledge-based chatbot in C++, demonstrating file I/O, hash table usage, and interactive learning. The program is efficient for small-scale use but could be enhanced with error handling, partial matching, or a graphical interface. Future improvements might include natural language processing for smarter query matching or database integration for larger knowledge bases.

# 9 References


- C++ Standard Library documentation for unordered_map and fstream.
- "The C++ Programming Language" by Bjarne Stroustrup.