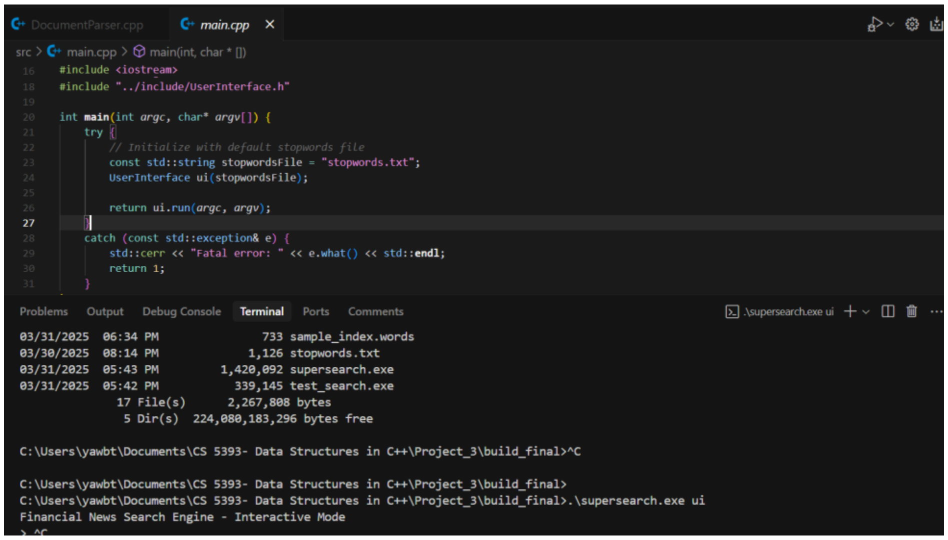
Yaw Boateng, CS 5393 Section 002

Project 3

Link to GitHub Repository: https://github.com/yawbtng/Data-Structures-Project_3-Search-Engine



```cpp
#include <iostream>
#include "../include/UserInterface.h"

int main(int argc, char* argv[]) {
    try {
        // Initialize with default stopwords file
        const std::string stopwordsFile = "stopwords.txt";
        UserInterface ui(stopwordsFile);

        return ui.run(argc, argv);
    }
    catch (const std::exception& e) {
        std::cerr << "Fatal error: " << e.what() << std::endl;
        return 1;
    }
```

```
03/31/2025  06:34 PM               733 sample_index.words
03/30/2025  08:14 PM             1,126 stopwords.txt
03/31/2025  05:43 PM         1,420,092 supersearch.exe
03/31/2025  05:42 PM           339,145 test_search.exe
              17 File(s)      2,267,808 bytes
               5 Dir(s)  224,080,183,296 bytes free

C:\Users\yawbt\Documents\CS 5393- Data Structures in C++\Project_3\build_final>^C

C:\Users\yawbt\Documents\CS 5393- Data Structures in C++\Project_3\build_final>
C:\Users\yawbt\Documents\CS 5393- Data Structures in C++\Project_3\build_final>.\supersearch.exe ui
Financial News Search Engine - Interactive Mode
> ^C
```

Final Project Summary

The C++ financial news search engine project is a comprehensive implementation that demonstrates advanced data structures and search algorithms.

Project Architecture

1. **AVL Tree-Based Index** - The core data structure for fast search operations
1. **Document Parser** - Processes JSON articles with NLP techniques (stopwords, stemming)
1. **Query Processor** - Handles search queries with boolean operations and ranking
1. **User Interface** - Command-line tools for indexing, searching, and viewing

Key Features Implemented

- Custom AVL tree implementation with balancing
- Indexed search with stopword removal and stemming
- Entity extraction for organization and person searches
- TF-IDF relevancy ranking
- Persistent indices with serialization and deserialization
- Command-line interface with interactive mode

Technical Achievements

- Successfully compiled with CMake

- Implemented Porter stemming algorithm
- Used RapidJSON for efficient document parsing
- Built custom AVL trees for O(log n) search complexity
- Created a persistent storage system for indices

Running the Application

1. Build with CMake using: `cmake .. && cmake --build .`
1. Index JSON documents with: `.\supersearch.exe index <path>`
1. Search using: `.\supersearch.exe query <terms>`
1. Or use interactive mode: `.\supersearch.exe ui`

The search engine can process large volumes of financial news articles, extract entities, and provide relevant search results ranked by importance.

Next Steps

- Further optimize query processing for larger datasets
- Enhance relevancy ranking with additional algorithms
- Add more advanced search operators
- Implement result highlighting and snippets
- Support for more complex boolean queries

The project demonstrates a solid understanding of C++ programming, data structures (particularly AVL trees), and search engine architecture principles.