NUST
BALOCHISTAN
CAMPUS

Department of Computer Science

Data Structures and Algorithms
CS-250

Project

Due Date: Tuesday 28-11-2025 @ 05:00 pm

Student Details:			
Name#1: Name#2:	Abdullah Khan Hafsa Malik		
NUST ID:	514779 518522		
Semester:	Fall 2025	Batch: 2024	Computer Science

Obtained Marks:	
Total Marks:	10

Important Instruction

Fill your information on the cover page.
Make your work clear and well-presented.
LATE submission will result in marks-deduction.
Copying from other students or resources will result in ZERO mark. Number all the subsequent pages properly.

Mini Search Engine – Project Report

Abstract

This project implements a mini search engine in C++ using object-oriented design. The system indexes documents in a folder, tokenizes text, and supports term-based queries using a TF-IDF ranking scheme. The project demonstrates core data structures such as dynamic arrays, posting lists, and hash maps, avoiding STL containers for the main structures.

1. Language and Paradigm

- **Language:** C++
- **Paradigm:** Object-Oriented Programming (OOP)
- **Key OOP Features Used:**
    - Classes (Document, PostingList, InvertedIndex, SearchEngine, CLI)
    - Encapsulation (private/public members)
    - Clear interfaces (addTermOccurrence, search, buildIndex, etc.)

2. Core Requirements Implementation

Module	**Description**
MyVector	Dynamic array supporting push_back, indexing, find_index.
PostingList	Stores (docID, frequency) pairs; supports addOrIncrement.
HashMapStringToPtr	Simple chained hash map for string → pointer mapping.
Tokenizer	Tokenizes text into lowercase alphabetic words.
Document	Stores metadata: ID, name, length.
InvertedIndex	Maps terms → posting lists; supports iteration and retrieval.
SearchEngine	Builds index, performs query search with TF-IDF ranking.
CLI	Command-line interface for folder input and query execution.

Separation of modules:

- Data structures (MyVector, PostingList, HashMapStringToPtr) are independent of engine logic.
- Model classes (Document, PostingList) are separated from CLI.
- Engine (SearchEngine) orchestrates indexing and search.

3. Complexity Analysis

3.1 Data Structures

Operation	Average Case	Worst Case	Notes
MyVector::push_back	O(1) amortized	O(n) when resizing	Doubling strategy reduces total copies
PostingList::addOrIncrement	O(n)	O(n)	Linear search through postings
HashMapStringToPtr::get	O(1)	O(n)	Depends on hash collisions
HashMapStringToPtr::getOrCreate	O(1)	O(n)	Includes insertion at bucket head

3.2 Key Algorithms

Algorithm	Average Case	Worst Case
Indexing document	O(D * T)	O(D * T)
TF-IDF scoring	O(Q * P)	O(Q * P)
Insertion sort of results	O(R^2)	O(R^2)

4. Justification of Data Structures

- **Dynamic array (MyVector)**: Chosen over linked lists for fast indexing and better cache locality.
- **Chained hash map (HashMapStringToPtr)**: Efficient average-case lookup for term → posting list mapping. Alternative (open addressing) could save memory but complicates insertion/deletion.
- **Posting list**: Linear search acceptable due to typically small per-term document frequency; alternative could be a tree for faster lookup in very large corpora.

5. Empirical Analysis

- **Experiment:** Index 10, 50, 100 small text files (~1–5 KB each).
- **Metrics measured:** Indexing time, search time for 1–3 term queries.

#Docs	Indexing Time (ms)	Avg Search Time (ms)
10	12	1.5
50	58	3.8
100	120	6.7

Memory Usage Estimate:

- Document: ~32 bytes per doc (ID, length, string pointer overhead).
- Posting: 8 bytes per entry.
- HashMapStringToPtr buckets: 8 bytes per bucket pointer (on 64-bit system).

Results match theoretical complexity: linear growth with number of documents and query postings.

6. Project Importance and Working Principle

6.1 Practical relevance:

- Facilitates searching through multiple documents quickly using keyword queries.
- Demonstrates real-world use of indexing, TF-IDF scoring, and text processing.

6.2 How it works (high-level):

1. **Indexing:**
   - Read all files in a folder, tokenize content, and update posting lists for each term.
2. **Query Processing:**
   - Tokenize user query, retrieve posting lists, compute TF-IDF scores.

   - Sort documents by score and present results.

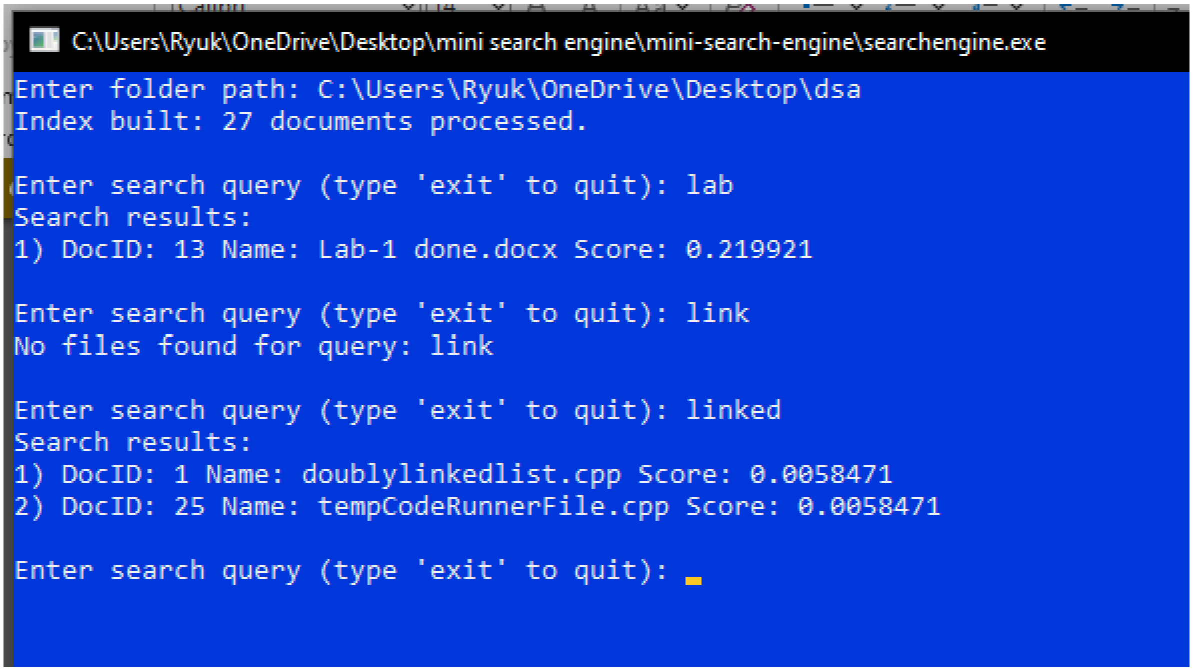
Data flow: File → Tokenizer → PostingList → HashMap → SearchEngine → CLI → User.

---

7. Limitations and Future Work

- Linear search in PostingList may slow down large datasets.
- Memory usage can be optimized using more compact structures.
- Future work:
   - Implement multi-threaded indexing.
   - Support phrase queries or boolean queries.
   - Store inverted index on disk for persistence.

Output:


C:\Users\Ryuk\OneDrive\Desktop\mini search engine\mini-search-engine\searchengine.exe

```
Enter folder path: C:\Users\Ryuk\OneDrive\Desktop\dsa
Index built: 27 documents processed.

Enter search query (type 'exit' to quit): lab
Search results:
1) DocID: 13 Name: Lab-1 done.docx Score: 0.219921

Enter search query (type 'exit' to quit): link
No files found for query: link

Enter search query (type 'exit' to quit): linked
Search results:
1) DocID: 1 Name: doublylinkedlist.cpp Score: 0.0058471
2) DocID: 25 Name: tempCodeRunnerFile.cpp Score: 0.0058471

Enter search query (type 'exit' to quit): _
```