UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

School of Computer Science)

MAJOR PROJECT REPORT

- C PROGRAMMING (CSE102)

LAPTOP SEARCH ENGINE

SUBMITTED BY:

Name: Vaani Kamboj
SAP ID: 590025226
Batch: 61
Program: B.Tech CSE (1st Year)

SUBMITTED TO:

Dr. Srinivasan Ramachandran
Instructor – C Programming
School of Computer Science
University of Petroleum and Energy Studies

ACADEMIC YEAR: 2025-2026

~ABSTRACT

This project focuses on designing and implementing a Laptop Search Engine using the C programming language, aimed at helping users select laptops based on personalised requirements. The program stores a database of fifty laptops categorised into five segments—Gaming, Portable, Convertible, Programming and Mainstream—inside a text file. It extracts and processes this data using file handling techniques and stores it in an array of structures.

A multi-criteria scoring system evaluates each laptop by comparing its specifications (processor, RAM, storage, graphics, display size, price and rating) with the user's input preferences. Bubble Sort is applied to rank all laptops and identify the top three recommendations.

This project highlights the importance of structured programming and demonstrates how C can be used to build practical decision-support tools.

~PROBLEM DEFINITION

Choosing a suitable laptop has become increasingly complex due to the vast variety of brands, configurations, and performance categories available in the market today. Customers often face confusion when trying to match their personal requirements—such as budget, usage type, and performance expectations—with the correct device.

A few months ago, while searching for my own laptop, I experienced this difficulty firsthand. Despite researching extensively, comparing specifications manually was slow and inefficient. This inspired the idea to build a tool that simplifies laptop selection.

This project addresses that problem by providing a **C-based Laptop Recommendation System** that loads a laptop database, matches it with user preferences, scores each device, and recommends the top 3 best-suited options.

~SYSTEM DESIGN

⭐ Major Components in the project

1. laptops.txt (Database)
   *stores 50 laptops
   *5 different categories of laptop
   *each laptop stores 10 characteristics
   *loaded in struct Laptop array

2. laptop.h

*contains the structure used to store laptop information like id, category, brand, model, processor, RAM, storage, graphics, display, price, rating, score.

3. database.c

*Responsible for opening laptops.txt, reading each line, storing values in array of struct using sscanf(), returns total number of laptops.

4. input.c

*Takes user preferences for category, brand, processor, RAM, Storage, Graphics, Display size, Budget, Rating and uses pointers to store values directly in main variables.

5. scoring.c

*applies algorithm to score each laptop based on exact match, similar match, range check, rating difference, RAM ,Processor ,Graphics priority and every laptop gets a **score**.

6. recommendation.c

* Uses **bubble sort** to sort laptops by Score (descending), Rating (tie-breaker), Price (cheapest first if tie) and prints **Top 3 laptops.**

- **ALGORITHM**

Step 1: Start the program.

Step 2: Load Laptop Database

1. Open the file **laptops.txt** in read mode.
2. Read the file **line by line**.
3. For each valid line (ignore lines starting with #):
    - Extract laptop details using sscanf.
    - Store the details in an array of struct Laptop.
4. Close the file and store the total count of laptops.

Step 3: Get User Preferences

1. Ask user to enter the following preferences:
    - Category
    - Brand
    - Processor
    - Minimum RAM
    - Storage type
    - Graphics requirement
    - Display size
    - Budget
    - Minimum rating
2. Store all values in variables passed by reference.

Step 4: Compute Scores for All Laptops

For each laptop in the array:

1. Initialize score = 0.
2. Compare laptop characteristics with user preferences:
    - Category match → +5
    - Brand match → +2
    - Processor similarity → +5 (or partial match)

- RAM ≥ required → +5
- Storage match → +3
- Graphics match → +5 (or partial match)
- Display size close → +1 or +2
- Price close to budget → +2 to +5
- Laptop rating ≥ user rating → +1 to +3

3. Add all points to compute final score for each laptop.

Step 5: Sort Laptops by Score

1. Use **Bubble Sort** on the laptop array.
2. Sort in **descending order** of score.
3. If two laptops have same score:
   - Higher rating comes first.
   - If rating also same, lower price comes first.

Step 6: Recommend Top 3 Laptops

1. Print the first **three laptops** from the sorted list.
2. Display their:
   - ID
   - Category
   - Brand
   - Model
   - Processor
   - RAM
   - Storage
   - Graphics
   - Display
   - Rating
   - Total score

Step 7: End the program

- **WORKFLOW OF THE PROGRAM**
  - User runs the program.
  - Program loads laptops from file.
  - User enters preferences.
  - Scoring algorithm compares each laptop.
  - Laptops are sorted by score.
  - Top 3 best-matched laptops are recommended.

- **IMPLEMENTATION DETAILS**

The Laptop Search Engine is implemented in the C programming language using the concepts of structures, arrays, file handling, string functions, sorting algorithms. The complete project is divided into multiple .c, .h files to maintain clarity and separation of logic.

1. **Laptop Database Handling (database.c)**

- The laptop details are stored in an external file laptops.txt.
- The program uses fopen(), fgets(), and sscanf() to open the database, read each line, and extract fields separated by |.
- Each laptop record is stored inside an array of struct Laptop.
- Comment lines beginning with # are ignored.
- A counter keeps track of how many laptops are successfully loaded into memory.

2. Structure Definitions (laptop.h)

- A structure named Laptop stores all characteristics such as ID, category, brand, RAM, storage, display size, price, rating, and score.
- Header file also allows safe inclusion of structure in multiple .c files.

3. Input Module (input.c)

- User preferences such as category, brand, processor, RAM, price, and rating are taken as input.
- Pointers are used for numerical inputs (RAM, display, price, rating) so the values update directly in the main() variables.
- scanf() and " %[^\n]" are used to accept multi-word values like “Ryzen 5”.

4. Scoring Logic (scoring.c)

- Every laptop in the array is compared with the user's preferences.
- For each matched characteristic, the laptop gains points.
- Functions used:
  - strcasecmp() for case-insensitive string comparison
  - strstr() for checking partial matches

    - fabs() to compare floating-point display values
- The scoring formula includes:
    - Category match
    - Brand match
    - Processor similarity
    - RAM ≥ preferred RAM
    - Storage match
    - Graphics requirements
    - Display similarity
    - Smart price scoring (+/– ₹5000, ₹10000 etc.)
    - Minimum rating

4. <u>Recommendation Module (recommendation.c)</u>

- All laptops are sorted in descending order using the Bubble Sort.
- Sorting preference:
    1. Higher score
    2. If scores equal → higher rating
    3. If rating equal → lower price
- The top 3 best laptops are displayed with all details.

<u>6. Main Controller (main.c)</u>

- Calls all the other modules:
    1. Loads the database
    2. Takes user input
    3. Calls scoring function
    4. Calls recommendation function
    5. Displays final results
- Implements clean function calling.

<u>7. File Structure & Modularity</u>

- Program separated into:
    - database.c
    - input.c
    - scoring.c
    - recommendation.c
    - main.c
- This ensures easier debugging, readability, and professional project structure.

- ## Testing and Results

  To confirm the correctness of the Laptop Search Engine, the following testing methods were used:

a) Functional Testing

Checking whether each module works correctly:

- File reading (database.c)
- User input (input.c)
- Scoring logic (scoring.c)
- Sorting + recommendation (recommendation.c)
- Overall integration (main.c)

b) Boundary Testing

Testing edge cases such as:

- Lowest and highest prices
- Minimum/maximum RAM
- Very strict rating requirement
- Wrong category input

c) Invalid Input Testing

Checking how the program behaves when:

- User enters unexpected values
- Multi-word processor names
- Lowercase / uppercase mismatch

d) Integration Testing

Checking if all modules correctly work together in sequence.

- ## CONCLUSION AND FUTURE WORK

Building this project not only enhanced my technical skills but also solved a problem I personally experienced while purchasing a laptop. The Laptop Search Engine simplifies the decision-making process by analyzing multiple specifications and providing the top 3 best-matched options. This system proves how programming can convert a real-life difficulty into a meaningful solution using logic, structure, and algorithms.

This project demonstrates how core C programming concepts—such as file handling, string manipulation, dynamic scoring, and sorting algorithms—can be combined to build a practical and user-friendly application.

The results validate the efficiency of the scoring logic and show that the system can be further expanded into a full-fledged recommendation engine.

Future Work

The current version of the Laptop Search Engine successfully recommends laptops using a structured scoring algorithm. However, several enhancements can significantly extend its usefulness:

1. Allow users to add new laptop entries into the database, making the system dynamic.
2. Provide guided assistance for beginners who do not understand technical specifications by asking simplified questions and auto-generating preferences.
3. Add more filters like battery life, build quality, and portability.
4. Implement an advanced scoring algorithm or ML-based ranking.
5. Develop a GUI or web-based interface for better user experience.
6. Add real-time price tracking and alerts.
7. Store user profiles for personalized recommendations.
8. Introduce a laptop comparison mode.
9. Add sort-by-price, rating, or performance features.
10. Support voice-based input and output.

- **REFERENCES**

1. *Let Us C* — Yashavant Kanetkar
2. GitHub Documentation – README & Repository Setup Guidance
3. UPES Classroom Notes & Lab Materials
4. GeeksforGeeks.com – Bubble Sort, Strings, Pointers tutorials