**NBKR INSTITUTE OF SCIENCE AND TECHNOLOGY**

**BLOOD BANK MANAGEMENT SYSTEM**

**COURSE:** DATA STRUCTURES

**DEPARTMENT:** COMPUTER SCIENCE ENGINEERING

**SECTION:** E

**YEAR:** I

**SEMESTER:** II

**DATE:** 06 MAY 2025

**SUBMITTED TO:** ASHOK SELVAKUMAR SIR

**SUBMITTED BY:**

A.NIKITHA(24KB1A0519)

B.LAKSHMI(24KB1A0529)

B.KAVYA(24KB1A0529)

B.L.PRASANNA(24KB1A0553)

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to all those who have supported and guided me throughout the development of this project, “Blood Bank Management System.”

First and foremost, I would like to thank my faculty and mentors for their invaluable guidance, encouragement, and constructive feedback, which helped me better understand the practical aspects of data structures and file handling in C programming.

I am also thankful to my friends and classmates for their constant support, suggestions, and motivation during the course of this project.

This project helped me enhance my programming skills and deepened my understanding of linked lists, dynamic memory allocation, file operations, and modular code design. It has been a great learning experience and has significantly contributed to my growth as a Computer Science student.

Lastly, I would like to thank my family for their unwavering support and belief in my abilities.

**ABSTRACT**

The Blood Bank Management System is a C-based console application designed to efficiently manage donor information using linked lists. It allows users to add, view, and search donors by blood group and location. The system ensures data persistence through file handling, enabling storage and retrieval of donor records. This project demonstrates core concepts of data structures and file operations, providing a simple yet effective solution for managing blood donation data.

**INTRODUCTION**

Blood donation plays a vital role in saving lives during medical emergencies, surgeries, and treatments. Managing donor information manually can be inefficient and error-prone. This project, Blood Bank Management System, provides a digital solution using the C programming language to store, search, and display donor records. It uses linked lists for dynamic data handling and file operations to maintain persistent storage. The system is designed to be simple, efficient, and user-friendly.

**OBJECTIVE**

The main objective of this project is to develop a simple and efficient system to manage blood donor information. It aims to allow users to add, view, and search for donors based on blood group and location. The project also focuses on implementing dynamic memory management using linked lists and ensuring data persistence through file handling. Ultimately, the goal is to provide an easy-to-use application that can assist in organizing blood donation data effectively.

**SYSTEM REQUIREMENTS**

**Hardware Requirements:**

Processor: Intel Pentium or higher

RAM: Minimum 2 GB

Hard Disk: Minimum 100 MB free space

Display: VGA or higher resolution monitor

Keyboard and Mouse

**Software Requirements:**

Operating System: Windows / Linux

Compiler: GCC (MinGW for Windows) or any C-compatible compiler

Text Editor/IDE: Code::Blocks, Dev C++, Turbo C, or Visual Studio Code

File System: Support for reading/writing .txt files**METHODOLOGY**

**Problem Identification**

Recognized the need for a system to manage blood donor information effectively and digitally.

**Requirement Analysis**

Determined essential features such as adding donors, viewing the list, searching by blood group and location, and saving data.

**System Design**

Used modular programming concepts in C to break the system into smaller functional units (e.g., add, display, search).

Designed a singly linked list data structure to store donor information dynamically.

**Implementation**

C language was used for coding.

Functions were developed to handle insertion, traversal, and search operations on the linked list.

Used file handling for data persistence across program executions.

**Testing and Debugging**

Tested the program with various inputs to check for correctness.

Debugged errors related to file I/O, string handling, and memory management.

**Execution and Evaluation**

Ran the final version to ensure it meets user requirements.

Evaluated based on ease of use, accuracy of data, and successful file operations.

**PROJECT DESCRIPTION**

The Blood Bank Management System is a console-based application developed in the C programming language. It is designed to store, manage, and search blood donor information efficiently. The system allows users to add new donors, display all existing donors, and search for donors based on blood group and location.

The program uses a singly linked list to manage donor data dynamically in memory, ensuring efficient insertion and traversal. Additionally, it incorporates file handling to store donor records persistently in a text file, allowing data to be retained even after the program is closed. The system is menu-driven, providing a simple and user-friendly interface for interacting with the donor database. This project demonstrates core concepts of data structures, string handling, and file I/O in C, and is ideal for small-scale blood bank operations or educational purposes.

**ALGORITHM**

1. Start the program.

2. Load Existing Donor Data:

* Open the file donors.txt in read mode.
* Read each line and extract name, blood group, and location.
* Insert each record into the linked list.

3. Display Menu and Get User Choice:

* Show options:
* Add Donor
* Display All Donors
* Search Donor
* Exit
* Read user input for the selected option.

4. Based on Choice:

If choice is 1 (Add Donor)

* Prompt user for name, blood group, and location.
* Create a new node and insert it at the end of the linked list.
* Save the updated list to the file.
* If choice is 2 (Display All Donors):
* Traverse the linked list and print all donor details.

If choice is 3 (Search Donor):

* Ask for blood group and location to search.
* Traverse the list and compare each node’s blood group and location.
* Display matching donors.

If choice is 4 (Exit):

* Save all data to file.
* Exit the program.

5. Repeat steps 3–4 until the user selects Exit.

6. End the program.

**PROGRAM CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Donor {

char name[50];

char bloodGroup[5];

char location[50];

struct Donor\* next;

} Donor;

Donor\* head = NULL;

// Function to create a new donor node

Donor\* createDonor(char name[], char bloodGroup[], char location[]) {

Donor\* newDonor = (Donor\*)malloc(sizeof(Donor));

strcpy(newDonor->name, name);

strcpy(newDonor->bloodGroup, bloodGroup);

strcpy(newDonor->location, location);

newDonor->next = NULL;

return newDonor;

}

// Function to insert donor at end

void insertDonor(char name[], char bloodGroup[], char location[]) {

Donor\* newDonor = createDonor(name, bloodGroup, location);

if (head == NULL) {

head = newDonor;

} else {

Donor\* temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newDonor;

}

}

// Function to display all donors

void displayDonors() {

Donor\* temp = head;

if (temp == NULL) {

printf("No donors available.\n");

return;

}

printf("\nList of Donors:\n");

while (temp != NULL) {

printf("Name: %s, Blood Group: %s, Location: %s\n", temp->name, temp->bloodGroup, temp->location);

temp = temp->next;

}

}

// Function to search donors by blood group and location

void searchDonor(char bloodGroup[], char location[]) {

Donor\* temp = head;

int found = 0;

printf("\nSearch Results:\n");

while (temp != NULL) {

if (strcmp(temp->bloodGroup, bloodGroup) == 0 && strcmp(temp->location, location) == 0) {

printf("Name: %s, Blood Group: %s, Location: %s\n", temp->name, temp->bloodGroup, temp->location);

found = 1;

}

temp = temp->next;

}

if (!found) {

printf("No matching donor found.\n");

}

}

// Function to load donors from file

void loadFromFile(char filename[]) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

printf("No previous data found.\n");

return;

}

char name[50], bloodGroup[5], location[50];

while (fscanf(file, "%s %s %s", name, bloodGroup, location) != EOF) {

insertDonor(name, bloodGroup, location);

}

fclose(file);

}

// Function to save donors to file

void saveToFile(char filename[]) {

FILE\* file = fopen(filename, "w");

if (file == NULL) {

printf("Error saving data.\n");

return;

}

Donor\* temp = head;

while (temp != NULL) {

fprintf(file, "%s %s %s\n", temp->name, temp->bloodGroup, temp->location);

temp = temp->next;

}

fclose(file);

}

// Function to remove newline character from fgets

void removeNewline(char str[]) {

str[strcspn(str, "\n")] = '\0';

}

int main() {

int choice;

char name[50], bloodGroup[5], location[50];

char filename[] = "donors.txt";

loadFromFile(filename);

do {

printf("\n--- Blood Bank Management System ---\n");

printf("1. Add Donor\n");

printf("2. Display All Donors\n");

printf("3. Search Donor by Blood Group and Location\n");

printf("4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

getchar(); // to clear the newline character after scanf

switch (choice) {

case 1:

printf("Enter Name: ");

fgets(name, sizeof(name), stdin);

removeNewline(name);

printf("Enter Blood Group (e.g., A+, O-): ");

fgets(bloodGroup, sizeof(bloodGroup), stdin);

removeNewline(bloodGroup);

printf("Enter Location: ");

fgets(location, sizeof(location), stdin);

removeNewline(location);

insertDonor(name, bloodGroup, location);

saveToFile(filename);

printf("Donor added successfully!\n");

break;

case 2:

displayDonors();

break;

case 3:

printf("Enter Blood Group to Search: ");

fgets(bloodGroup, sizeof(bloodGroup), stdin);

removeNewline(bloodGroup);

printf("Enter Location to Search: ");

fgets(location, sizeof(location), stdin);

removeNewline(location);

searchDonor(bloodGroup, location);

break;

case 4:

printf("Exiting... Saving data.\n");

saveToFile(filename);

break;

default:

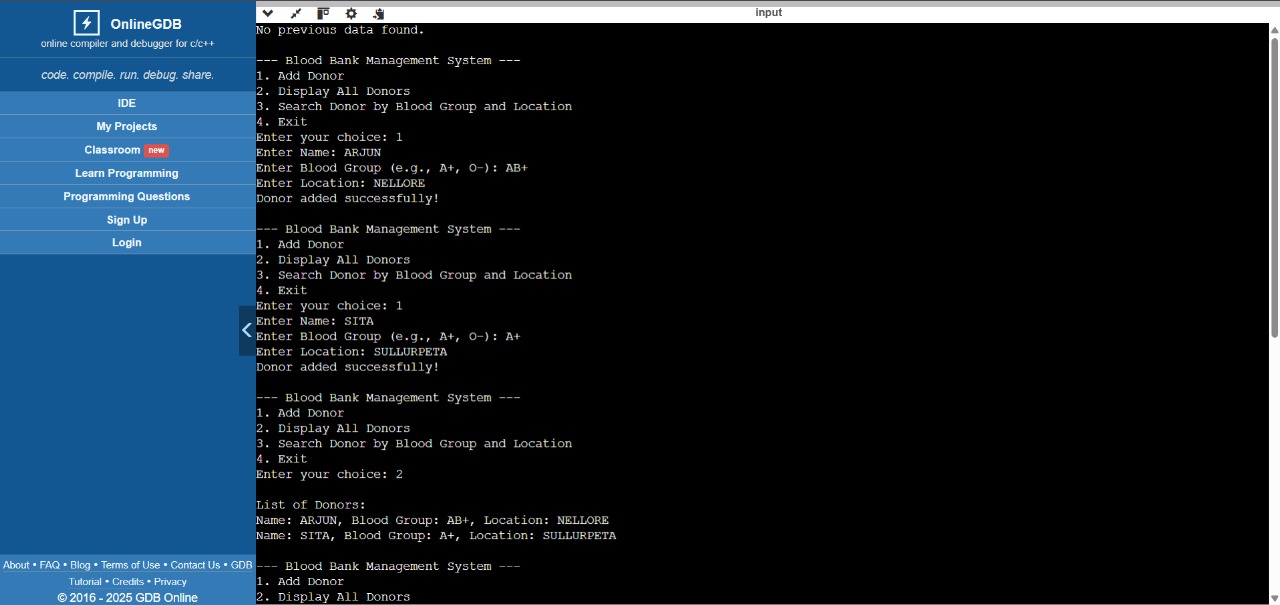
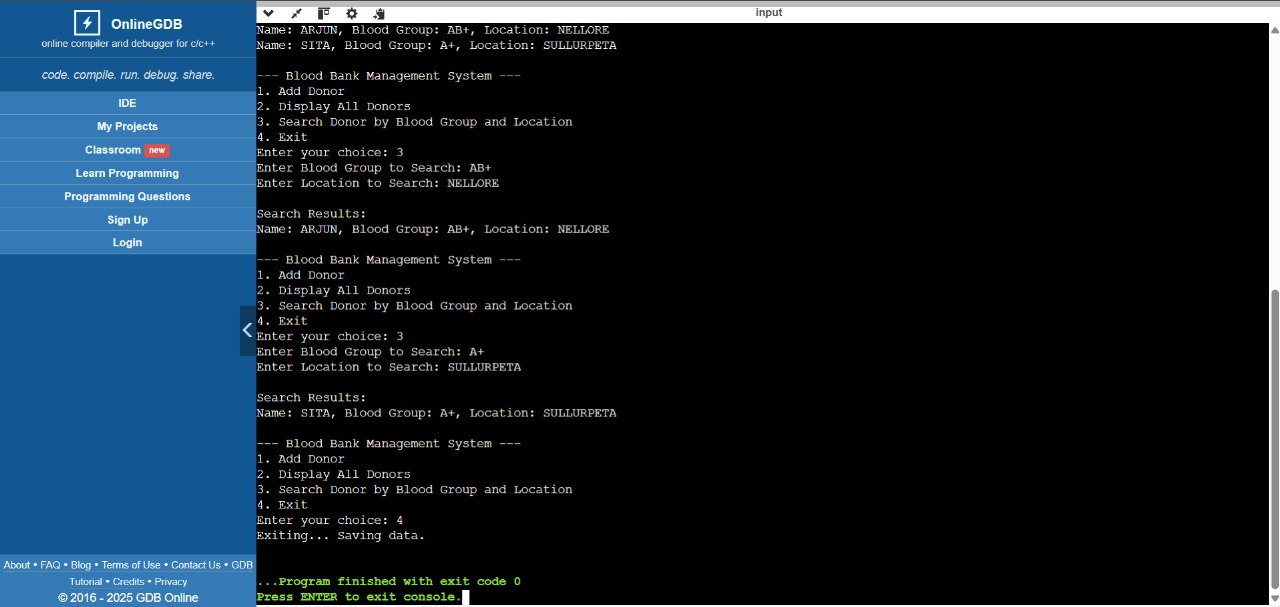
printf("Invalid choice! Please try again.\n");

}

} while (choice != 4);

return 0;

}

**OUTPUT**  

**TESTING AND VALIDATION**

**1. Unit Testing:**

Each function was tested individually to ensure it performs as expected. The following functions were tested:

createDonor(): Verified if a new donor node is created with correct data.

insertDonor(): Checked if donors are properly inserted into the linked list.

displayDonors(): Ensured all donor information was correctly displayed in the correct format.

searchDonor(): Tested for correct searching functionality based on blood group and location.

saveToFile() and loadFromFile(): Verified if donor data was correctly written to and read from the file.

**2. Functional Testing:**

The overall functionality of the program was tested by simulating various user interactions, such as:

Adding donors and ensuring the linked list is updated.

Searching for donors by blood group and location, ensuring accurate results.

Displaying donor data in the correct format.

Exiting the program and ensuring that data is saved correctly in the text file.

**3. Boundary Testing:**

Boundary cases were tested to ensure robustness:

Adding a donor with the maximum allowed name and location length.

Searching for donors with uncommon blood groups or locations.

Handling empty file scenarios when loading data.

**4. Input Validation:**

The system was tested for invalid inputs:

Non-alphabetic characters or special symbols in name, blood group, or location were handled by prompt re-entry.

Checked for invalid blood group format (e.g., "A+" or "O-").

Tested for incorrect menu selections, ensuring that the user is prompted again until a valid option is entered.

**5. File Integrity:**

Verified that donor data was correctly saved to a file and loaded back during program execution, ensuring persistence across sessions.

Checked for correct data formatting in the text file after saving.

**6. Performance Testing:**

While the program was not optimized for large datasets, it was tested with a reasonable number of records to confirm that the linked list functions (insertion, deletion, traversal) performed efficiently without errors.

**7. User Interface Testing:**

The menu-driven interface was tested to ensure it was intuitive and user-friendly.

The flow between adding, searching, and displaying donors was evaluated to ensure smooth interaction.

**LIMITATIONS**

**Limited Search Functionality:**

The search functionality is limited to blood group and location. Advanced search features, such as by donor name or blood type compatibility, are not implemented.

**Data Storage Format:**

The system uses a plain text file (donors.txt) for storing donor information. This format lacks the flexibility and security of more advanced storage systems like databases. Large datasets or complex queries could result in performance issues.

**Single User System:**

The application is designed for single-user operation. It does not support multi-user environments or concurrent access to the data.

**No Data Validation for Blood Group Format:**

The blood group format is not strictly validated (e.g., ensuring only valid blood groups like "A+", "O-", etc., are entered).

**No GUI (Graphical User Interface):**

The system uses a text-based interface, making it less visually appealing and user-friendly compared to a graphical user interface (GUI) solution.

**Limited Error Handling:**

The system does not include extensive error handling for unexpected events like file corruption or memory allocation failures.

**Scalability Issues:**

The system may face performance issues with large numbers of donor records, as it uses a linked list to store and traverse data. It could be inefficient for managing thousands of donor records.

**Lack of Security:**

The application does not include any form of authentication or encryption for sensitive donor data, which could be a concern in real-world applications.

**FUTURE ENHANCEMENT**

**Database Integration:**

Instead of using a text file, a relational database (like MySQL or SQLite) can be integrated for better data management, scalability, and complex queries. This would also allow for easier retrieval and manipulation of donor data.

**Graphical User Interface (GUI):**

A GUI using frameworks like GTK or Qt could be developed to enhance user experience. This would allow users to interact with the system through a more intuitive interface, including forms for adding, searching, and displaying donor information.

**Multi-user Support:**

Implementing multi-user support with role-based access (admin, user) would allow multiple users to interact with the system simultaneously, providing enhanced functionality for larger organizations.

**Advanced Search and Filtering:**

More advanced search features, such as searching by name, blood compatibility, or availability of specific blood types, could be added. Users could filter the list of donors based on multiple criteria.

**Data Validation and Error Handling:**

Implement more robust data validation (e.g., ensuring valid blood groups and proper formatting) and better error handling mechanisms to handle file access issues, invalid inputs, and memory allocation failures.

**Mobile Application:**

A mobile version of the Blood Bank Management System could be developed to make donor information more accessible on the go. This could be built for both Android and iOS platforms, allowing people to check donor availability or register as donors directly from their phones.

**Automated Notification System:**

The system could include automated notifications (via email or SMS) to alert donors or users when blood donations are needed or when new donors are added to the database in a specific area.

**Data Encryption and Security:**

For enhanced security, sensitive donor information could be encrypted, and user authentication could be implemented to restrict unauthorized access to the system.

**Reporting and Analytics:**

Adding reporting features, such as generating donor statistics, blood group distribution, and donation trends, would provide valuable insights to blood bank administrators and health organizations.

**Cloud Integration:**

Storing the donor data on a cloud platform could enhance accessibility and allow for real-time updates, ensuring that the system is accessible from multiple locations and devices.

**CONCLUSION**

In conclusion, the Blood Bank Management System successfully addresses the challenges of managing donor information for blood banks. By utilizing the C programming language and key concepts such as linked lists and file handling, the system ensures efficient management, searchability, and persistence of donor data. The project provides a simple, user-friendly interface that allows for easy addition, display, and searching of donor records.

Although the system has certain limitations, such as the lack of a graphical interface and scalability concerns, it serves as a solid foundation for more advanced developments. The system’s modular design, dynamic memory handling, and file-based data storage demonstrate essential concepts in software development. Overall, the project has achieved its goal of providing a functional and educational solution for managing blood donor information, with potential for future enhancements to improve usability, performance, and security.

**REFERENCES**

1. Algorithms and Data Structures: The Basic Toolbox by Kurt Mehlhorn and Peter

Sanders

2. C Data Structures and Algorithms by Alfred V. Aho, Jeffrey D. Ullman, and John E.

Hopcroft

3. Problem Solving with Algorithms and Data Structures" by Brad Miller and David

Ranum

4. Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L.

Rivest, and Clifford Stein

5 . Websites: https://www.onlinegdb.com/