**Advanced Data Structure**

**Course Project**

**Library Management System**

**DIV : AI – B**

**Group : 2**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#include <ctype.h>

#include <time.h>

enum Status { AVAILABLE, ISSUED };

int quantity = 0;

int generateBookID(char\* str);

int generateAlphabetID(char c);

int queue\_array[20];

struct BST\* rack[10][5];

struct User\* start = NULL;

struct Queue {

    int queue\_array[20];

    int top;

};

void initializeQueue(struct Queue \*queue) {

    queue->top = -1;

}

int isFull(struct Queue \*queue) {

    return queue->top == 19;

}

int isEmpty(struct Queue \*queue) {

    return queue->top == -1;

}

void enQueue(struct Queue \*queue, int value) {

    if (!isFull(queue)) {

        queue->top++;

        queue->queue\_array[queue->top] = value;

    } else {

        printf("Queue is full!\n");

    }

}

int deQueue(struct Queue \*queue) {

    int value = queue->queue\_array[0];

    for (int i = 0; i < queue->top; ++i) {

        queue->queue\_array[i] = queue->queue\_array[i + 1];

    }

    queue->top--;

    return value;

}

void resetQueue(struct Queue \*queue) {

    queue->top = -1;

}

int getTop(struct Queue \*queue) {

    return queue->top;

}

struct Book {

    char\* name;

    char\* author;

    int id, amount;

    int rack;

    struct Book\* next;

    enum Status status; // Add status field

    time\_t due\_date;

};

struct Book\* createBook(const char\* new\_name, const char\* new\_author, int new\_id, int new\_rack, int Quantity) {

    struct Book\* book = (struct Book\*)malloc(sizeof(struct Book));

    if (book != NULL) {

        // Allocate memory for name and author strings

        book->name = strdup(new\_name);     // Uses strdup to allocate and copy string

        book->author = strdup(new\_author); // Uses strdup to allocate and copy string

        book->id = new\_id;

        book->rack = new\_rack;

        book->amount = Quantity;

        book->status = AVAILABLE; // Initialize status

        book->next = NULL;

    } else {

        fprintf(stderr, "Memory allocation failed for Book\n");

    }

    return book;

}

#define ISSUED 1

struct BST {

    int id;

    struct BST\* left;

    struct BST\* right;

    struct BST\* next;

};

struct User {

    char name[100];

    struct Book\* book;

    struct User\* next;

    int code;

};

struct User\* createUser(char\* new\_name) {

    struct User\* user = (struct User\*)malloc(sizeof(struct User));

    strcpy(user->name, new\_name);

    user->next = NULL;

    user->book = NULL;

    return user;

}

struct BST\* searchBST(struct BST\* root, int id);

void displayIssuedBooks();

int returnBook(char\* user\_name, char\* book\_name,int unique\_cod);

struct Book\* searchPreviousBook(struct BST\* root, char\* name, char\* author\_name);

int searchBookcheck(struct BST\* root, char\* name, char\* author\_name);

int issueBook(char\* username, char\* bookname, char\* author\_name, int unique\_cod);

int deleteBook(char\* name, char\* author\_name);

struct Book\* searchBook(struct BST\* bst, char\* name, char\* author\_name);

int SearchingsearchBook(struct BST\* bst, char\* name, int rack, int shelve);

void generateLibraryData();

struct Book\* searchBookmain(struct BST\* root, char\* name, char\* author\_name);

void addBook(int shelve, char\* book\_name, char\* author\_name, int Quantity);

struct BST\* createBST(int id, struct BST\* root);

void displayLibrary(struct BST\* root);

struct Queue queue;

int main() {

    generateLibraryData(); // Load initial data

    int choice;

    while (1) {

        // Main menu: Choose if you are a student or an admin

        printf("\n++++++++++  WELCOME TO LIBRARY SYSTEM  ++++++++++\n");

        printf("1. Student\n");

        printf("2. Admin Member\n");

        printf("0. Exit\n");

        printf("\nSelect your role: ");

        scanf("%d", &choice);

        if (choice == 0) {

            break;

        }

        if (choice == 1) { // Student menu

            while (1) {

                printf("\n++++++++++  STUDENT MENU  ++++++++++\n");

                printf("1. Issue Book\n");

                printf("2. Return Book\n");

                printf("3. Search Book\n");

                printf("4. Display All Books\n");

                printf("0. Back\n");

                printf("\nSelect an option: ");

                scanf("%d", &choice);

                if (choice == 0) {

                    break; // Go back to the main menu

                }

                switch (choice) {

                   case 1: {

        char book\_name[256];

        char author\_name[256];

        char username[256];

        int unique\_code;

        printf("\n\n\n++++++++++     I S S U E     ++++++++++\n");

        printf("\nEnter User Name : ");

        scanf(" %[^\n]s", username);

        printf("\nEnter Unique Code : ");  // Ask for unique code

        scanf("%d", &unique\_code);  // Read unique code from user

        printf("\nEnter Book Name : ");

        scanf(" %[^\n]s", book\_name);

        printf("\nEnter Author Name : ");

        scanf(" %[^\n]s", author\_name);

        // Call the issueBook function with the unique code parameter

        int issue = issueBook(username, book\_name, author\_name, unique\_code);

        if (issue) {

            printf("\n\nBook issued successfully!\n");

        }

        break;

    }

                    case 2: {

    char book\_name[256];

    char username[256];

    int code;

    printf("\n\n\n++++++++++     R E T U R N     ++++++++++\n");

    printf("\nEnter User Name : ");

    scanf(" %[^\n]s", username);

    printf("\nEnter Book Name : ");

    scanf(" %[^\n]s", book\_name);

    printf("\nEnter user Code : ");

    scanf("%d", &code);

    int returned = returnBook(username, book\_name, code);

    if (returned) {

        printf("\n\nBook returned successfully!\n");

    }

    break;

}

                    case 3: {

                char name[256];

                int found = 0;

                printf("\n\n\n++++++++++     S E A R C H     ++++++++++\n");

                printf("\nEnter Book Name : ");

                scanf(" %[^\n]s", name);

                int id = generateBookID(name);

                // For each rack

                for (int c = 0; c < 5; ++c) {

                    // For shelve

                    int shelve = id % 10;

                    // search BST

                    struct BST \*bst = searchBST(rack[shelve][c], id);

                    // If BST found

                    if (bst != NULL) {

                        found = SearchingsearchBook(bst, name, c, shelve);

                        if (found) {

                            break;

                        }

                    }

                }

                if (!found) {

                    printf("\n\nBook not found");

                }

                break;

            }

                    case 4: { // Display all books

                        printf("\nDisplaying all books:\n");

                        for (int c = 0; c < 5; ++c) {

                            for (int r = 0; r < 10; ++r) {

                                struct BST\* bst = rack[r][c];

                                displayLibrary(bst);

                            }

                        }

                        break;

                    }

                    default: {

                        printf("Invalid option.\n");

                        break;

                    }

                }

            }

        } else if (choice == 2) { // Admin menu

            while (1) {

                printf("\n++++++++++  ADMIN MENU  ++++++++++\n");

                printf("1. Insert Book\n");

                printf("2. Delete Book\n");

                printf("3. Display Issued Books\n");

                printf("4. Display All Books\n");

                printf("0. Back\n");

                printf("\nSelect an option: ");

                scanf("%d", &choice);

                if (choice == 0) {

                    break; // Go back to the main menu

                }

                switch (choice) {

                    case 1: { // Insert book

                        int option;

                        char book\_name[256];

                        char author\_name[256];

                        int Quantity;

                        printf("\nSelect Category:\n");

                        printf("1. Mathematics\n");

                        printf("2. Computer\n");

                        printf("3. Physics\n");

                        printf("4. Islamiat\n");

                        printf("5. Others\n");

                        printf("\nSelect an option: ");

                        scanf("%d", &option);

                        if (option < 1 || option > 5) {

                            printf("Invalid category.\n");

                        } else {

                            printf("\nBook Name: ");

                            scanf(" %[^\n]s", book\_name);

                            printf("Author Name: ");

                            scanf(" %[^\n]s", author\_name);

                            printf("\nQuantity : ");

                            scanf("%d", &Quantity);

                            addBook(option - 1, book\_name, author\_name, Quantity);

                            printf("Book inserted successfully.\n");

                        }

                        break;

                    }

                    case 2: { // Delete book

                        char book\_name[256];

                        char author\_name[256];

                        printf("\nEnter Book Name: ");

                        scanf(" %[^\n]s", book\_name);

                        printf("Enter Author Name: ");

                        scanf(" %[^\n]s", author\_name);

                        int deleted = deleteBook(book\_name, author\_name);

                        if (deleted) {

                            printf("Book deleted successfully.\n");

                        } else {

                            printf("Book not found.\n");

                        }

                        break;

                    }

                    case 3: { // Display issued books

                        printf("\nDisplaying issued books:\n");

                        displayIssuedBooks();

                        break;

                    }

                    case 4: { // Display all books

                        printf("\n\n\n++++++++++     D I S P L A Y     ++++++++++\n");

                    for (int c = 0; c < 5; ++c) {

                        printf("\n+++++   R A C K - %d   +++++\n", c + 1);

                        for (int r = 0; r < 10; ++r) {

                            printf("\n+++ S H E L V E - %d +++\n", r + 1);

                            displayLibrary(rack[r][c]);

                        }

                    }

                        break;

                    }

                    default: {

                        printf("Invalid option.\n");

                        break;

                    }

                }

            }

        } else {

            printf("Invalid choice.\n");

        }

    }

    return 0;

}

int generateBookID(char \*str) {

    struct Queue queue;

    initializeQueue(&queue);

    int value = 0;

    // Check for each character in str

    for (int i = 0; i < strlen(str); ++i) {

        // If str[i] is a space

        if (str[i] == ' ') {

            enQueue(&queue, value);

            value = 0;

        } else {

            value += generateAlphabetID(str[i]);

        }

    }

    enQueue(&queue, value);

    value = 0;

    // Combining different parts

    char combining\_values[256];

    strcpy(combining\_values, "");

    while (!isEmpty(&queue)) {

        char num[5];

        sprintf(num, "%d", deQueue(&queue));

        strcat(combining\_values, num);

    }

    sscanf(combining\_values, "%d", &value);

    return value;

}

int generateAlphabetID(char c) {

    char alphabets[] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's',

                        't', 'u', 'v', 'w', 'x', 'y', 'z'};

    c = tolower(c);

    for (int i = 0; i < 26; ++i) {

        if (c == alphabets[i])

            return i + 1;

    }

    return -1;

}

struct BST \*searchBST(struct BST \*root, int id) {

    if (root == NULL) {

        return NULL;

    } else {

        while (root != NULL) {

            if (root->id > id) root = root->left;

            else if (root->id < id) root = root->right;

            else return root;

        }

        return NULL;

    }

}

void addBook(int shelve, char\* book\_name, char\* author\_name, int Quantity) {

    int id = generateBookID(book\_name);

    int row = id % 10;

    // Create a new book node

    struct Book\* new\_book = createBook(book\_name, author\_name, id, row, Quantity);

    // Checking if BST already exists then work on it

    if (rack[row][shelve] != NULL) {

        // Create it root and search if any BST already exists

        struct BST\* root = rack[row][shelve];

        struct BST\* searched = searchBST(root, id);

        // If no BST with such ID exists

        if (searched == NULL) {

            struct BST\* new\_BST = createBST(id, root);

            // Adding new book to BST

            new\_BST->next = (struct BST\*)new\_book;

        }

        // If BST exists with the same ID

        else {

            // Search if book with the same name exists

            struct Book\* searched\_book = searchBook(searched, book\_name, author\_name);

            // If the same book is already present

            if (searched\_book != NULL)

                searched\_book->amount++;

            else {

                new\_book->next = (struct Book\*)searched->next;

                searched->next = (struct BST\*)new\_book;

            }

        }

    }

    // If BST does not exist in the shelve array

    else {

        rack[row][shelve] = createBST(id, NULL);

        rack[row][shelve]->next = (struct BST\*)new\_book;

    }

}

struct BST\* createBST(int id, struct BST\* root){

    struct BST\* current = (struct BST\*)malloc(sizeof(struct BST));

    current->id = id;

    current->left = NULL;

    current->right = NULL;

    current->next = NULL;

    struct BST\* p = root;

    struct BST\* k = p;

    if (p == NULL) {

        root = current;

    } else {

        while (p != NULL) {

            k = p;

            if (p->id > id) p = p->left;

            else p = p->right;

        }

        if (k->id > id) k->left = current;

        else k->right = current;

    }

    return current;

}

void displayLibrary(struct BST\* root) {

    if (root != NULL) {

        displayLibrary(root->left);

        // Displaying book linked list

        struct Book\* book = (struct Book\*)root->next;

        while (book != NULL) {

            printf("Book ID = %-10d Amount = %-10d Book Name = %-30s Author Name = %10s\n", book->id, book->amount,

                   book->name, book->author);

            book = book->next;

        }

        displayLibrary(root->right);

    }

}

struct Book\* searchBook(struct BST\* bst, char\* name, char\* author\_name) {

    int id = generateBookID(name);

    struct Book\* book = (struct Book\*)bst->next;

    while (book != NULL) {

        if (strcmp(book->name, name) == 0 && strcmp(book->author, author\_name) == 0){

            return book;

        }

        book = book->next;

    }

    return NULL;

}

int SearchingsearchBook(struct BST\* bst, char\* name, int rack, int shelve) {

    int id = generateBookID(name);

    struct Book\* book = (struct Book\*)bst->next;

    int found = 0;

    while (book != NULL) {

        if (strcmp(book->name, name) == 0) {

            found = 1;

            printf("\nBook name : %s       ID : %d   Rack: %d  Shelve: %d  Author Name: %s",

                   book->name, book->id, rack + 1, shelve + 1, book->author);

        }

        book = book->next;

    }

    return found;

}

int deleteBook(char\* name, char\* author\_name) {

    int id = generateBookID(name);

    // For each rack

    for (int c = 0; c < 5; ++c) {

        // For shelve

        int shelve = id % 10;

        // search BST

        struct BST\* bst = searchBST(rack[shelve][c], id);

        // If BST found

        if (bst != NULL) {

            // Searching book

            if (searchBookcheck(bst, name, author\_name)) {

                struct Book\* book = searchBookmain(bst, name, author\_name);

                struct Book\* prevbook = searchPreviousBook(bst, name, author\_name);

                if (strcmp(book->name, name) == 0 && strcmp(book->author, author\_name) == 0) {

                    if (book->amount > 1) {

                        book->amount--;

                    } else {

                        if (book->next == NULL) {

                            bst->next = NULL;

                            free(book);

                        } else {

                            bst->next = (struct BST\*)book->next;

                            free(book);

                        }

                    }

                    return 1;

                } else if (prevbook != NULL) {

                    if (strcmp(book->next->name, name) == 0 && strcmp(book->next->author, author\_name) == 0) {

                        // If amount is greater than 1

                        if (book->next->amount > 1) {

                            book->next->amount--;

                            return 1;

                        }

                        if (book->next->next != NULL) {

                            book->next = book->next->next;

                            free(book->next);

                        } else {

                            book->next = NULL;

                            free(book->next);

                        }

                        return 1;

                    }

                }

            }

        }

    }

    return 0;

}

struct Book\* searchPreviousBook(struct BST\* root, char\* name, char\* author\_name) {

    int id = generateBookID(name);

    struct Book\* book = (struct Book\*)root->next;

    struct Book\* prev\_book = book;

    while (book != NULL) {

        if (strcmp(book->name, name) == 0 && strcmp(book->author, author\_name) == 0)

            return prev\_book;

        prev\_book = book;

        book = book->next;

    }

    return NULL;

}

struct Book\* searchBookmain(struct BST\* root, char\* name, char\* author\_name) {

    int id = generateBookID(name);

    struct Book\* book = (struct Book\*)root->next;

    while (book != NULL) {

        if (strcmp(book->name, name) == 0 && strcmp(book->author, author\_name) == 0)

            return book;

        book = book->next;

    }

    return NULL;

}

int searchBookcheck(struct BST\* root, char\* name, char\* author\_name) {

    int id = generateBookID(name);

    void\* book = root->next;

    while (book != NULL) {

        struct Book\* current\_book = (struct Book\*)book;

        if (strcmp(current\_book->name, name) == 0 && strcmp(current\_book->author, author\_name) == 0)

            return 1;

        book = current\_book->next;

    }

    return 0;

}

void generateLibraryData() {

    FILE\* inFile;

    inFile = fopen("C:\\Users\\HP\\Desktop\\Dny\_CP\\CP\\datafile.txt", "r");

    if (!inFile) {

        printf("Unable to open file datafile.txt");

        exit(EXIT\_FAILURE);   // call system to stop

    }

    // Variables that will collect data

    int shelve;

    char shelve\_str[512];

    char book\_name[512];

    char author\_name[512];

    while (fgets(shelve\_str, sizeof(shelve\_str), inFile) != NULL) {

        // converting shelve\_str to int

        shelve = atoi(shelve\_str);

        if (fgets(book\_name, sizeof(book\_name), inFile) == NULL) {

            printf("Invalid file format: missing book name");

            exit(EXIT\_FAILURE);

        }

        book\_name[strcspn(book\_name, "\n")] = 0;  // Remove newline character

        if (fgets(author\_name, sizeof(author\_name), inFile) == NULL) {

            printf("Invalid file format: missing author name");

            exit(EXIT\_FAILURE);

        }

        author\_name[strcspn(author\_name, "\n")] = 0;  // Remove newline character

        addBook(shelve, book\_name, author\_name, quantity);

    }

    fclose(inFile);

}

int issueBook(char\* user\_name, char\* book\_name, char\* author\_name, int unique\_code) {

    // Search if the user already exists

    struct User\* user = start;

    while (user != NULL) {

        if (strcmp(user->name, user\_name) == 0 && user->code == unique\_code) {

            printf("\nUser with unique code already exists. Cannot issue another book.");

            return 0;

        }

        user = user->next;

    }

    // Search for the book

    int id = generateBookID(book\_name);

    int shelve = id % 10;

    struct Book\* book = NULL;

    for (int i = 0; i < 5; ++i) {

        struct BST\* bst = searchBST(rack[shelve][i], id);

        if (bst != NULL) {

            book = searchBook(bst, book\_name, author\_name);

            if (book != NULL) {

                // Allocate memory for the new user

                struct User\* new\_user = createUser(user\_name);

                new\_user->code = unique\_code;

                // Create a new book node for the user

                struct Book\* new\_book = createBook(book\_name, author\_name, id, shelve, quantity);

                // Calculate due date (10 days from the current date)

                time\_t now = time(NULL);

                time\_t due\_date = now + (10 \* 24 \* 60 \* 60); // 10 days in seconds

                new\_book->due\_date = due\_date;

                // Print the due date

                struct tm\* due\_date\_tm = localtime(&due\_date);

                printf("\nDue Date: %d-%02d-%02d", due\_date\_tm->tm\_year + 1900, due\_date\_tm->tm\_mon + 1, due\_date\_tm->tm\_mday);

                new\_user->book = new\_book;

                // Add the user to the user list

                if (start == NULL) {

                    start = new\_user;

                } else {

                    struct User\* temp = start;

                    while (temp->next != NULL) {

                        temp = temp->next;

                    }

                    temp->next = new\_user;

                }

                // Update the book status to reflect that it is issued

                book->status = ISSUED;

                 deleteBook(book\_name, author\_name);

                return 1;

            }

        }

    }

    printf("Book Not Found");

    return 0;

}

int returnBook(char\* user\_name, char\* book\_name, int unique\_code) {

    // searching username in list

    struct User\* user = start;

    struct User\* prev\_user = start;

    while (user != NULL) {

        prev\_user = user;

        if (strcmp(user->name, user\_name) == 0 && user->code == unique\_code) break;

        user = user->next;

    }

    // If user exists

    if (user != NULL) {

        // Searching book

        struct Book\* book = user->book;

        // If book exists

        if (book != NULL && strcmp(book->name, book\_name) == 0) {

            // If there is only one book in user

            if (user->book == book) {

                addBook(book->rack, book\_name, book->author,quantity);

                user->book = NULL;

                free(book);

                // If the user is the root

                if (user == start) {

                    if (start->next != NULL) {

                        start = start->next;

                    } else {

                        start = NULL;

                    }

                } else {

                    prev\_user->next = user->next;

                    free(user);

                }

                return 1;

            }

        } else {

            printf("\nBook not Found");

        }

    } else {

        printf("User Doesn't Exist");

    }

    return 0;

}

void displayIssuedBooks() {

    struct User\* user = start;

    while (user != NULL) {

        struct Book\* book = user->book;

        // Books

        while (book != NULL) {

            printf("Username : %-15s User code : %-10d Book name : %-15s Author Name : %-15s \n", user->name, user->code,book->name, book->author);

            book = book->next;

        }

        user = user->next;

    }

}

**Output:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

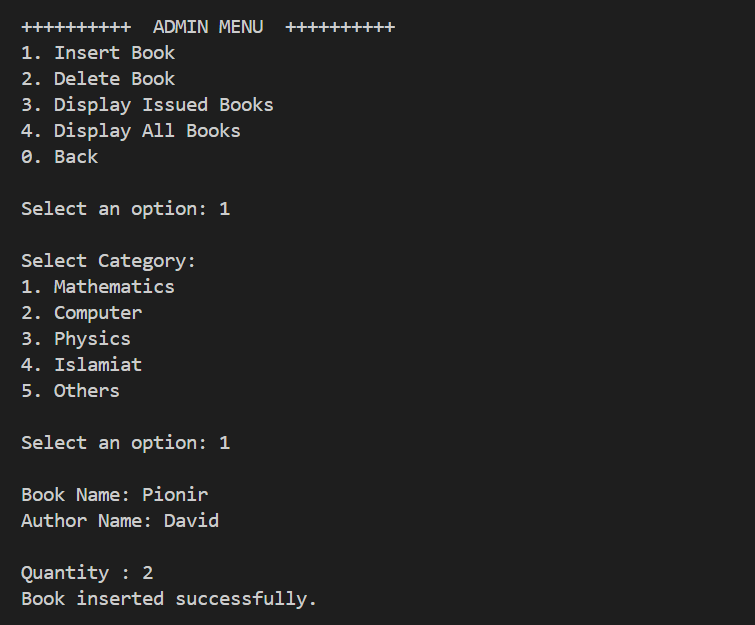
Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated  
  
  
A screenshot of a computer

Description automatically generated