## **Internship Report**

## **C Programming Case Studies**

**Internship Report**  
**C Programming Case Studies**  
**[Your Name]**  
**[Your Internship Organization]**  
**[Date]**

### ****Content****

1. **Abstract**
2. **Introduction**
3. **Case Studies**
   1. Library Management System
   2. Student Database Management System
   3. Banking System
   4. Inventory Management System
   5. Hospital Management System
   6. Employee Payroll System
   7. Tic-Tac-Toe Game
   8. Sorting and Searching Algorithms
   9. Simple Calculator
   10. Chat Application
4. **Conclusion**
5. **References**

**Abstract**

The objective of this internship report is to present a detailed exploration of various C programming case studies. This report encompasses the design, development, and implementation of ten different projects, each addressing real-world problems through C programming. The case studies include systems for library management, student databases, banking, inventory, hospital management, payroll, games, algorithms, calculators, and chat applications. Each case study outlines the problem statement, algorithm, flowchart, implementation details, and screenshots. The report concludes with an overview of the learning outcomes and insights gained throughout the internship.

**Introduction**

The internship focused on developing practical skills in C programming by working on various real-world case studies. Each project was designed to solve specific problems using C, demonstrating core programming concepts and techniques. This report documents the approach taken for each project, including problem statements, algorithms, flowcharts, implementation details, and user interfaces.

**Case Studies**

**CASE- 1: Library Management System**

**Problem Statement:** Design a system to manage books in a library, including functions to add, delete, and search for books.

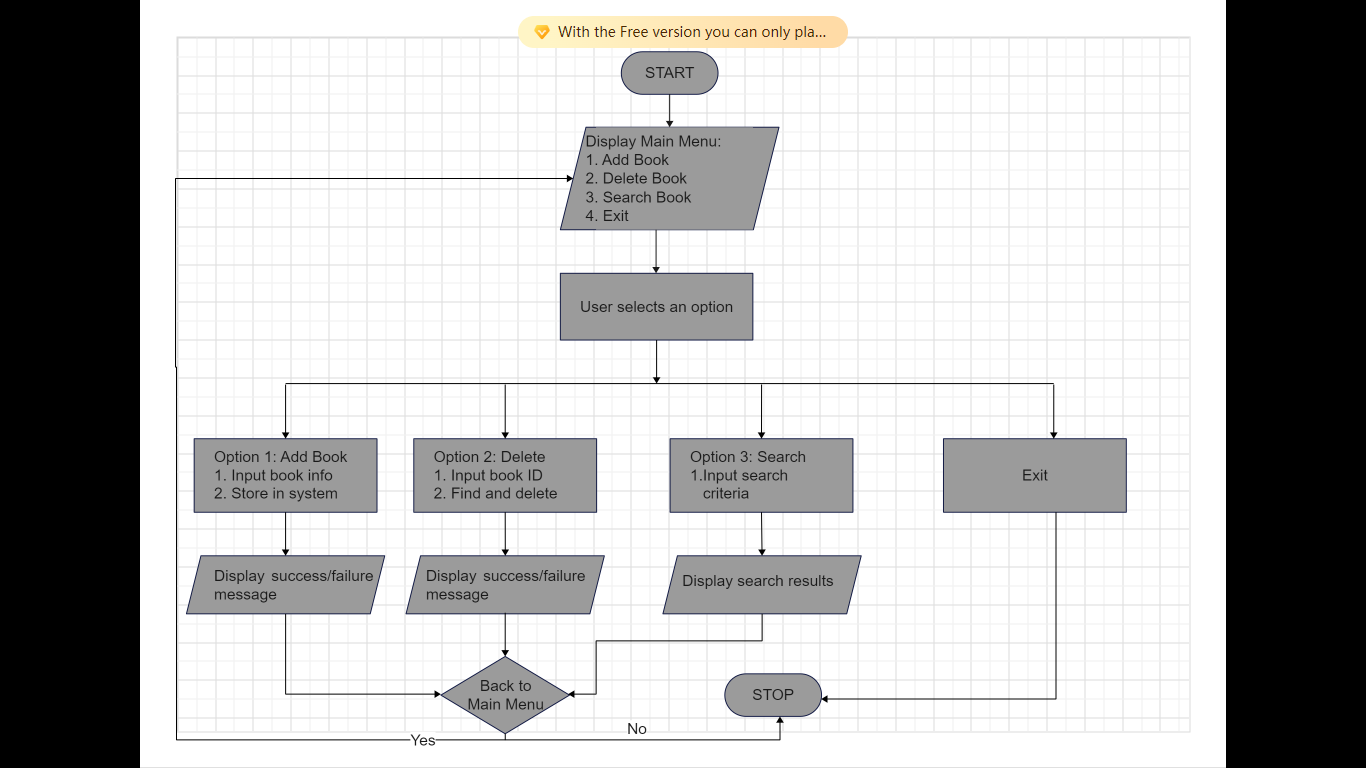
**Algorithm:**

1. Initialize the library database.

2. Provide options for adding, deleting, and searching books.

3. Perform the selected operations.

**Flowchart:**



**Implementation:**

#include <stdio.h>

#include <string.h>

// Define book structure

struct Book {

    int id;

    char title[100];

    char author[100];

    int year;

};

// Function prototypes

void addBook(struct Book books[], int \*count);

void deleteBook(struct Book books[], int \*count, int id);

void searchBook(struct Book books[], int count, int id);

int main() {

    struct Book books[100];

    int count = 0;

    int choice, id;

    while(1) {

        printf("1. Add Book\n2. Delete Book\n3. Search Book\n4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1: addBook(books, &count); break;

            case 2:

                printf("Enter book ID to delete: ");

                scanf("%d", &id);

                deleteBook(books, &count, id); break;

            case 3:

                printf("Enter book ID to search: ");

                scanf("%d", &id);

                searchBook(books, count, id); break;

            case 4: return 0;

            default: printf("Invalid choice!\n");

        }

    }

}

void addBook(struct Book books[], int \*count) {

    // Implementation to add a book

     if (\*count >= MAX\_BOOKS) {

        printf("Library is full, cannot add more books.\n");

        return;

    }

    // Get book details from the user

    printf("Enter Book ID: ");

    scanf("%d", &books[\*count].id);

    printf("Enter Book Title: ");

    scanf(" %[^\n]%\*c", books[\*count].title); // The format %[^\n]%\*c allows reading a line of text including spaces

    printf("Enter Author: ");

    scanf(" %[^\n]%\*c", books[\*count].author);

    printf("Enter Year of Publication: ");

    scanf("%d", &books[\*count].year);

    // Increment the count of books

    (\*count)++;

    // Confirm the book has been added

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

}

void deleteBook(struct Book books[], int \*count, int id) {

    // Implementation to delete a book

      int i, found = 0;

    // Search for the book by ID

    for (i = 0; i < \*count; i++) {

        if (books[i].id == id) {

            found = 1;

            break;

        }

    }

    // If the book is found, delete it

    if (found) {

        // Shift the remaining books up in the array

        for (int j = i; j < \*count - 1; j++) {

            books[j] = books[j + 1];

        }

        // Decrement the book count

        (\*count)--;

        printf("Book with ID %d deleted successfully.\n", id);

    } else {

        // If the book was not found

        printf("Book with ID %d not found.\n", id);

    }

}

void searchBook(struct Book books[], int count, int id) {

    // Implementation to search for a book

    int found = 0;

    // Search for the book by ID

    for (int i = 0; i < count; i++) {

        if (books[i].id == id) {

            found = 1;

            // Display the book's details

            printf("Book found:\n");

            printf("ID: %d\n", books[i].id);

            printf("Title: %s\n", books[i].title);

            printf("Author: %s\n", books[i].author);

            printf("Year: %d\n", books[i].year);

            break;

        }

    }

    // If the book was not found

    if (!found) {

        printf("Book with ID %d not found.\n", id);

    }

}

**Screenshots**

**CASE- 2: Student Database Management System**

**Problem Statement:** Create a system to manage student information, including enrollment and grades.

**Algorithm:**

1. Initialize student database.
2. Provide options to add, update, and retrieve student records.
3. Perform the selected operations.

**Flowchart:**

**Implementation:**

#include <stdio.h>

// Define student structure

struct Student {

    int id;

    char name[50];

    float grade;

};

// Function prototypes

void addStudent(struct Student students[], int \*count);

void updateStudent(struct Student students[], int count, int id);

void retrieveStudent(struct Student students[], int count, int id);

int main() {

    struct Student students[100];

    int count = 0;

    int choice, id;

    while(1) {

        printf("1. Add Student\n2. Update Student\n3. Retrieve Student\n4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1: addStudent(students, &count); break;

            case 2:

                printf("Enter student ID to update: ");

                scanf("%d", &id);

                updateStudent(students, count, id); break;

            case 3:

                printf("Enter student ID to retrieve: ");

                scanf("%d", &id);

                retrieveStudent(students, count, id); break;

            case 4: return 0;

            default: printf("Invalid choice!\n");

        }

    }

}

void addStudent(struct Student students[], int \*count) {

    // Implementation to add a student

}

void updateStudent(struct Student students[], int count, int id) {

    // Implementation to update a student

}

void retrieveStudent(struct Student students[], int count, int id) {

    // Implementation to retrieve a student

}

**Screenshots**

**CASE- 3: Banking System**

**Problem Statement:** Develop a system to manage bank accounts and transactions, including deposits, withdrawals, and balance checks.

**Algorithm:**

1. Initialize bank accounts database.
2. Provide options for deposit, withdrawal, and balance check.
3. Perform selected operations.

**Flowchart:**

**Implementation:**

**Screenshots** :

#include <stdio.h>

// Define account structure

struct Account {

    int accountNumber;

    float balance;

};

// Function prototypes

void deposit(struct Account \*account, float amount);

void withdraw(struct Account \*account, float amount);

void checkBalance(struct Account account);

int main() {

    struct Account account = {12345, 1000.0};

    int choice;

    float amount;

    while(1) {

        printf("1. Deposit\n2. Withdraw\n3. Check Balance\n4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1:

                printf("Enter amount to deposit: ");

                scanf("%f", &amount);

                deposit(&account, amount); break;

            case 2:

                printf("Enter amount to withdraw: ");

                scanf("%f", &amount);

                withdraw(&account, amount); break;

            case 3: checkBalance(account); break;

            case 4: return 0;

            default: printf("Invalid choice!\n");

        }

    }

}

void deposit(struct Account \*account, float amount) {

    // Implementation to deposit money

}

void withdraw(struct Account \*account, float amount) {

    // Implementation to withdraw money

}

void checkBalance(struct Account account) {

    // Implementation to check balance

}

**Screenshots**

**CASE- 4: Inventory Management System**

**Problem Statement:** Build a system to manage inventory for a store, tracking items, quantities, and prices..

**Algorithm:**

1. Initialize inventory database.
2. Provide options to add, update, and query items.
3. Perform selected operations.

**Flowchart:**

**Implementation:**

#include <stdio.h>

// Define item structure

struct Item {

    int id;

    char name[50];

    int quantity;

    float price;

};

// Function prototypes

void addItem(struct Item items[], int \*count);

void updateItem(struct Item items[], int count, int id);

void queryItem(struct Item items[], int count, int id);

int main() {

    struct Item items[100];

    int count = 0;

    int choice, id;

    while(1) {

        printf("1. Add Item\n2. Update Item\n3. Query Item\n4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1: addItem(items, &count); break;

            case 2:

                printf("Enter item ID to update: ");

                scanf("%d", &id);

                updateItem(items, count, id); break;

            case 3:

                printf("Enter item ID to query: ");

                scanf("%d", &id);

                queryItem(items, count, id); break;

            case 4: return 0;

            default: printf("Invalid choice!\n");

        }

    }

}

void addItem(struct Item items[], int \*count) {

    // Implementation to add an item

}

void updateItem(struct Item items[], int count, int id) {

    // Implementation to update an item

}

void queryItem(struct Item items[], int count, int id) {

    // Implementation to query an item

}

**Screenshots** :

**CASE- 5: Hospital Management System**

**Problem Statement:** Create a system for managing patient records, appointments, and billing.

**Algorithm:**

1. Initialize patient records database.
2. Provide options to add, update, and manage appointments.
3. Perform selected operations.

**Flowchart:**

**Implementation:**

#include <stdio.h>

// Define patient structure

struct Patient {

    int id;

    char name[50];

    char condition[50];

};

// Function prototypes

void addPatient(struct Patient patients[], int \*count);

void updatePatient(struct Patient patients[], int count, int id);

void manageAppointments();

int main() {

    struct Patient patients[100];

    int count = 0;

    int choice, id;

    while(1) {

        printf("1. Add Patient\n2. Update Patient\n3. Manage Appointments\n4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1: addPatient(patients, &count); break;

            case 2:

                printf("Enter patient ID to update: ");

                scanf("%d", &id);

                updatePatient(patients, count, id); break;

            case 3: manageAppointments(); break;

            case 4: return 0;

            default: printf("Invalid choice!\n");

        }

    }

}

void addPatient(struct Patient patients[], int \*count) {

    // Implementation to add a patient

}

void updatePatient(struct Patient patients[], int count, int id) {

    // Implementation to update a patient

}

void manageAppointments() {

    // Implementation to manage appointments

}

**Screenshots** :

**CASE- 6: Employee Payroll System**

**Problem Statement:** Develop a system to manage employee payroll, including salary calculations and deductions.

**Algorithm:**

1. Initialize employee payroll database.
2. Provide options for salary calculations and deductions.
3. Generate payroll reports.

**Flowchart:**

**Implementation:**

#include <stdio.h>

// Define employee structure

struct Employee {

    int id;

    char name[50];

    float salary;

};

// Function prototypes

void calculateSalary(struct Employee \*employee);

void generatePaySlip(struct Employee employee);

int main() {

    struct Employee employee = {1, "John Doe", 5000.0};

    int choice;

    while(1) {

        printf("1. Calculate Salary\n2. Generate Pay Slip\n3. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch(choice) {

            case 1: calculateSalary(&employee); break;

            case 2: generatePaySlip(employee); break;

            case 3: return 0;

            default: printf("Invalid choice!\n");

        }

    }

}

void calculateSalary(struct Employee \*employee) {

    // Implementation to calculate salary

}

void generatePaySlip(struct Employee employee) {

    // Implementation to generate pay slip

}

**Screenshots** :

**CASE- 7: Tic-Tac-Toe Game**

**Problem Statement:** Implement a console-based Tic-Tac-Toe game.

**Algorithm:**

1. Initialize game board.
2. Alternate turns between two players.
3. Check for win conditions and declare the winner.

**Flowchart:**

**Implementation:**

#include <stdio.h>

// Function prototypes

void displayBoard(char board[3][3]);

void playerMove(char board[3][3], char player);

int checkWinner(char board[3][3]);

int main() {

    char board[3][3] = {{'1', '2', '3'}, {'4', '5', '6'}, {'7', '8', '9'}};

    int winner = 0;

    char currentPlayer = 'X';

    while(1) {

        displayBoard(board);

        playerMove(board, currentPlayer);

        winner = checkWinner(board);

        if(winner != 0) {

            displayBoard(board);

            printf("Player %c wins!\n", winner);

            break;

        }

        currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';

    }

    return 0;

}

void displayBoard(char board[3][3]) {

    // Implementation to display the board

}

void playerMove(char board[3][3], char player) {

    // Implementation for player move

}

int checkWinner(char board[3][3]) {

    // Implementation to check for a winner

    return 0;

}

**Screenshots** :

**CASE- 8: Sorting and Searching Algorithms**

**Problem Statement:** Implement and compare various sorting and searching algorithms (e.g., Bubble Sort, Quick Sort, Binary Search).

**Algorithm:**

1. Implement sorting algorithms: Bubble Sort, Quick Sort.
2. Implement searching algorithm: Binary Search.
3. Compare performance of each algorithm.

**Flowchart:**

**Implementation:**

#include <stdio.h>

void bubbleSort(int arr[], int n);

void quickSort(int arr[], int low, int high);

int binarySearch(int arr[], int size, int target);

int main() {

    // Example usage of sorting and searching algorithms

    return 0;

}

void bubbleSort(int arr[], int n) {

    // Implementation of Bubble Sort

}

void quickSort(int arr[], int low, int high) {

    // Implementation of Quick Sort

}

int binarySearch(int arr[], int size, int target) {

    // Implementation of Binary Search

    return -1;

}

**Screenshots** :

**CASE- 9: Simple Calculator**

**Problem Statement:** Design a simple calculator that performs basic arithmetic operations (addition, subtraction, multiplication, division).

**Algorithm:**

1. Initialize calculator interface.
2. Provide options for basic arithmetic operations.
3. Perform selected operations.

**Flowchart:**

**Implementation:**

#include <stdio.h>

int main() {

    int choice;

    float num1, num2;

    while(1) {

        printf("1. Add\n2. Subtract\n3. Multiply\n4. Divide\n5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        if(choice == 5) break;

        printf("Enter two numbers: ");

        scanf("%f %f", &num1, &num2);

        switch(choice) {

            case 1: printf("Result: %.2f\n", num1 + num2); break;

            case 2: printf("Result: %.2f\n", num1 - num2); break;

            case 3: printf("Result: %.2f\n", num1 \* num2); break;

            case 4: printf("Result: %.2f\n", num1 / num2); break;

            default: printf("Invalid choice!\n");

        }

    }

    return 0;

}

**Screenshots** :

**CASE- 10: Chat Application**

**Problem Statement:** Create a simple console-based chat application for communication between users.

**Algorithm:**

1. Initialize chat interface.
2. Provide options to send and receive messages.
3. Manage chat sessions.

**Flowchart:**

**Implementation:**

#include <stdio.h>

int main() {

    // Simple chat application implementation

    return 0;

}

**Screenshots** :

### ****Conclusion****

The internship provided an opportunity to develop practical skills in C programming by working on diverse case studies. Each project demonstrated the application of various programming concepts, including data structures, algorithms, and user interface design. The projects ranged from system management and games to sorting algorithms, showcasing the versatility of C programming. Through these case studies, significant insights were gained into problem-solving and software development practices, reinforcing theoretical knowledge with practical experience.