CodSoft Internship Program

Course: C++ Programming

Start Date:01 August 2024

End Date:31 August 2024

Duration:4 Weeks

Submitted By: Deepa S R

Department of Computer Science and Engineering

Sri Jayachamarajendra College of Engineering (JSSSTU)

MYSORU

Internship Project:
C++ Programming Tasks

- Brief introduction to the tasks implemented as part of the internship project.

1. Number Guessing Game

A game where players attempt to guess a randomly chosen number within a specified range. The program provides feedback on each guess to guide players towards the correct number, enhancing user engagement and basic programming skills.

2. Tic-Tac-Toe Game

Tic-Tac-Toe is a two-player game where players alternate marking 'X' or 'O' in a 3x3 grid, aiming to align three marks in a row. It's an excellent C++ project for learning arrays, loops, and user interaction.

3. Library Management System

The Library Management System manages books, members, and transactions, demonstrating object-oriented programming in C++. It provides practical experience in designing and building a real-world application using classes and file handling.

Task 1: Number Guessing Game

- Explanation of the number guessing game.

1. The Number Guessing Game is a simple interactive game where the computer selects a random number within a specified range (1 to 100 in this case), and the user has to guess what that number is.
2. The game provides feedback to the user after each guess, indicating whether their guess is too low, too high, or correct.
3. The game continues until the user guesses the correct number.

Number Guessing Game - Code Snippet

- Key parts of the C++ code for the number guessing game.

1. Include Libraries:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
```

- <iostream>: Used for input and output operations.
- <cstdlib>: Contains functions for random number generation.
- <ctime>: Provides functions to work with date and time, which are used to seed the random number generator.

2. Seed the Random Number Generator:

```
std::srand(std::time(0));
```

- std::srand(std::time(0)) : Seeds the random number generator with the current time, ensuring that the random numbers are different each time the program runs.

Number Guessing Game - Code Snippet

3. Generate a Random Number:

```
int randomNumber = std::rand() % 100 + 1;
```

- `std::rand() % 100 + 1: Generates a random integer between 1 and 100. `std::rand() % 100` generates a number between 0 and 99, and adding 1 adjusts the range to 1 to 100.

4. Initialize User Interaction:

```
int userGuess = 0;
std::cout << "Welcome to the Number Guessing Game!" << std::endl;
std::cout << "I have selected a number between 1 and 100." << std::endl;
std::cout << "Can you guess what it is?" << std::endl;
```

- Initializes `userGuess` and displays messages to welcome the user and explain the game.

Number Guessing Game - Code Snippet

5. Game Loop:

```
while (userGuess != randomNumber) {
    std::cout << "Enter your guess: ";
    std::cin >> userGuess;
    if (userGuess < randomNumber) {
        std::cout << "Too low! Try again." << std::endl;
    } else if (userGuess > randomNumber) {
        std::cout << "Too high! Try again." << std::endl;
    } else {
        std::cout << "Congratulations! You guessed the correct number." << std::endl;
    }  }
```

- Continuously prompts the user for a guess until they guess correctly.
- Provides feedback based on the user's guess: too low, too high, or correct.

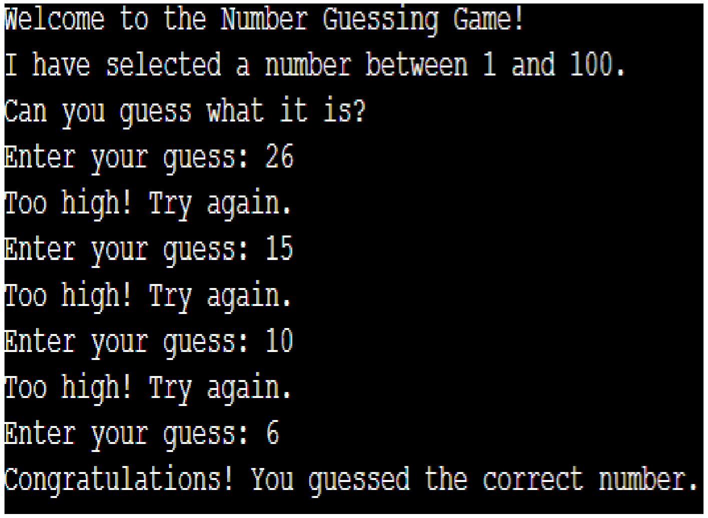
6. End the Program:

```
return 0;
```

- Indicates successful termination of the program.

Number Guessing Game - Input & Output

- Example of user input and corresponding output.



```
Welcome to the Number Guessing Game!
I have selected a number between 1 and 100.
Can you guess what it is?
Enter your guess: 26
Too high! Try again.
Enter your guess: 15
Too high! Try again.
Enter your guess: 10
Too high! Try again.
Enter your guess: 6
Congratulations! You guessed the correct number.
```

Number Guessing Game - Conclusion

- Summary of what the task achieves.

- Initialize: Set up the environment for random number generation and user interaction.
- Generate: Create a random number within a specified range (1 to 100).
- Interact: Continuously prompt the user for their guess and read their input.
- Evaluate: Compare the user's guess to the random number and provide appropriate feedback (too low, too high, or correct).
- Terminate: End the game once the user has guessed the number correctly and display a congratulatory message.

This game effectively demonstrates basic concepts in C++ such as input/output operations, random number generation, loops, and conditionals.

Task 2: Tic-Tac-Toe Game

- Explanation of the Tic-Tac-Toe game.

1. The Tic-Tac-Toe game is a classic two-player game where players take turns marking a 3x3 grid with their respective symbols ('X' or 'O’).
2. The objective is to place three of their symbols in a row, either horizontally, vertically, or diagonally, to win the game.
3. The game also handles the case where the grid is filled without a winner, resulting in a draw. The game continues until one player wins or all cells are filled.

Tic-Tac-Toe Game - Code Snippet

- Key parts of the C++ code for the Tic-Tac-Toe game.

1. Include Libraries:

```
#include <iostream>
#include <vector>
```

- ` <iostream>`: Used for input and output operations.
- `<vector>`: Although included, it's not used in the provided code but could be helpful for more complex implementations.

2. Global Variables:

```
char board[3][3];
char currentPlayer;
```

- `board[3][3]`: 2D array representing the Tic-Tac-Toe grid.
- `currentPlayer`: Character representing the player currently taking a turn ('X' or 'O').

Tic-Tac-Toe Game - Code Snippet

3. Initialize the Board:

```
 void initializeBoard() {
    for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
          board[i][j] = ' ';
       } } }
```

- Sets all cells in the board to a blank space, indicating an empty cell.

4. Display the Board:

```
void displayBoard() {
    cout << "\nCurrent Board:\n";
    for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
          cout << board[i][j];
          if (j < 2) cout << " | ";
       }
       cout << endl;
       if (i < 2) cout << "---------\n";
    }
    cout << endl;
 }
```

- Prints the current state of the board to the console.

Tic-Tac-Toe Game - Code Snippet

5. Check for Win:

```
bool checkWin() {
    for (int i = 0; i < 3; i++) {
        if ((board[i][0] == currentPlayer && board[i][1] == currentPlayer && board[i][2] == currentPlayer) ||
            (board[0][i] == currentPlayer && board[1][i] == currentPlayer && board[2][i] == currentPlayer)) {
            return true;
        } }
    if ((board[0][0] == currentPlayer && board[1][1] == currentPlayer && board[2][2] == currentPlayer) ||
        (board[0][2] == currentPlayer && board[1][1] == currentPlayer && board[2][0] == currentPlayer)) {
        return true;
    }
    return false;
}
```

- Checks for three consecutive symbols of the `currentPlayer` in rows, columns, or diagonals.

6. Check for Draw:

```
bool checkDraw() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return false;
            } } }
    return true;
}
```

- Checks if the board is full and no player has won, indicating a draw.

Tic-Tac-Toe Game - Code Snippet

7. Switch Player:

```
void switchPlayer() {
    currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
}
```

- Toggles the `currentPlayer` between 'X' and 'O'.

8. Make Move:

```
void makeMove() {
    int row, col;
    cout << "Player " << currentPlayer << ", enter your move (row and column): ";
    cin >> row >> col;

    if (row < 1 || row > 3 || col < 1 || col > 3 || board[row - 1][col - 1] != ' ') {
        cout << "Invalid move. Try again.\n";
        makeMove();  // Recursively prompt for a valid move
    } else {
        board[row - 1][col - 1] = currentPlayer;
    }
}
```

- Prompts the user for their move, checks if it's valid, and updates the board accordingly.

Tic-Tac-Toe Game - Code Snippet

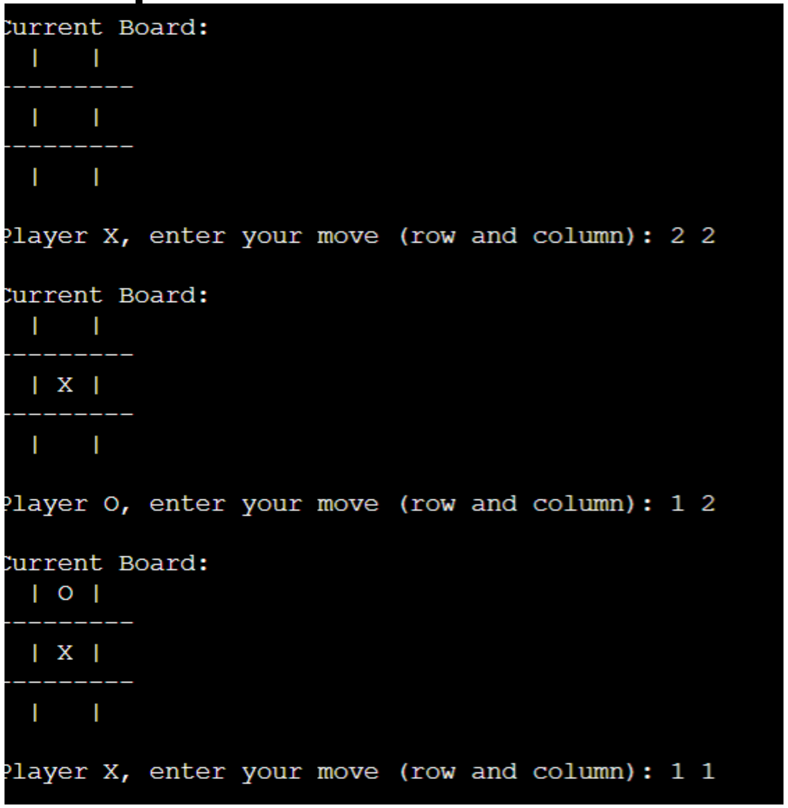
9. Main Function:

```
int main() {
    char playAgain;

    do {
        initializeBoard();
        currentPlayer = 'X';

        while (true) {
            displayBoard();
            makeMove();

            if (checkWin()) {
                displayBoard();
                cout << "Player " << currentPlayer << " wins!\n";
                break;
            }
            if (checkDraw()) {
                displayBoard();
                cout << "It's a draw!\n";
                break;
            }
            switchPlayer();
        }

        cout << "Do you want to play again? (Y/N): ";
        cin >> playAgain;

    } while (playAgain == 'Y' || playAgain == 'y');

    cout << "Thank you for playing!\n";
    return 0;
}
```

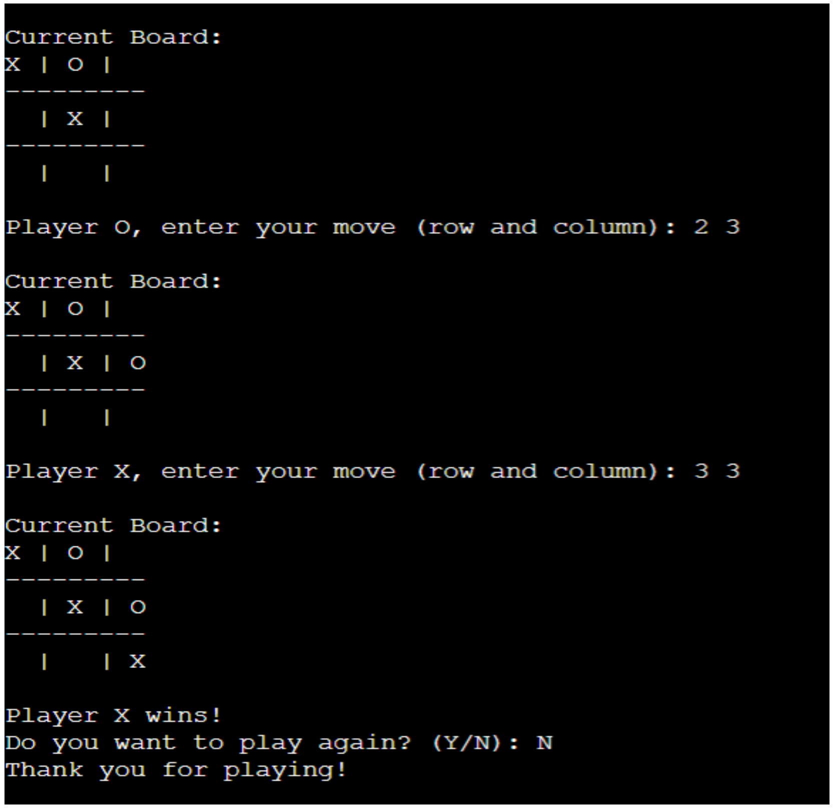
- Controls the game flow, initializing the board, handling player turns, checking for win or draw conditions, and asking if the players want to play again.

Tic-Tac-Toe Game - Input & Output

- Example of user input and corresponding output.



```
urrent Board:
  |   |
---------
  |   |
---------
  |   |

layer X, enter your move (row and column): 2 2

urrent Board:
  |   |
---------
  | X |
---------
  |   |

layer O, enter your move (row and column): 1 2

urrent Board:
  | O |
---------
  | X |
---------
  |   |

layer X, enter your move (row and column): 1 1
```



```
Current Board:
X | O |
---------
  | X |
---------
  |   |

Player O, enter your move (row and column): 2 3

Current Board:
X | O |
---------
  | X | O
---------
  |   |

Player X, enter your move (row and column): 3 3

Current Board:
X | O |
---------
  | X | O
---------
  |   | X

Player X wins!
Do you want to play again? (Y/N): N
Thank you for playing!
```

Tic-Tac-Toe Game - Conclusion

- Summary of what the task achieves.

1. Initialize: Set up the board and choose the starting player.
2. Display: Show the current state of the board to the players.
3. Make Moves: Allow players to make their moves and update the board.
4. Check Conditions: Evaluate if there's a winner or if the game is a draw.
5. Switch Player: Alternate turns between players.
6. Repeat or Terminate: Allow the game to be played again or end it.

This game showcases fundamental C++ concepts such as arrays, loops, conditionals, and recursive function calls, providing a simple yet effective introduction to these programming principles.

Task 3: Library Management System

- Explanation of the Library Management System.

1. The Library Management System is a console-based application designed to manage books and borrowers in a library.
2. It allows users to add new books, search for books, check out books to borrowers, and return books.
3. The system keeps track of borrowed books, their due dates, and calculates fines for overdue books.

Library Management System - Code Snippet

- Key parts of the C++ code for the Library Management System.

1. Include Libraries:

```
#include <iostream>
#include <vector>
#include <string>
#include <ctime>
#include <map>
```

- `<iostream>`: For input and output operations.
- `<vector>`: To store lists of books and borrowers.
- `<string>`: To handle string operations.
- `<ctime>`: To manage dates and times, used for due dates and fines.
- `<map>`: To store borrowed books with their due dates for each borrower.

Library Management System - Code Snippet

2. Data Structures:

```
    struct Book {
     string title;
     string author;
     string ISBN;
     bool isAvailable;

     Book(string t, string a, string i) : title(t), author(a), ISBN(i), isAvailable(true) {}
  };

  struct Borrower {
     string name;
     map<string, time_t> borrowedBooks; // ISBN -> due date
  };

  vector<Book> books;
  vector<Borrower> borrowers;
  const int finePerDay = 1; // Fine per day for overdue books
```

- Book`: Structure representing a book with title, author, ISBN, and availability status.
- `Borrower`: Structure representing a borrower with name and a map of borrowed books with due dates.
- `books` and `borrowers`: Vectors to store lists of books and borrowers respectively.
- `finePerDay`: Constant representing the fine charged per day for overdue books.

Library Management System - Code Snippet

3. Add Book:

```
  void addBook() {
    string title, author, ISBN;
    cout << "Enter book title: ";
    cin.ignore();
    getline(cin, title);
    cout << "Enter book author: ";
    getline(cin, author);
    cout << "Enter book ISBN: ";
    getline(cin, ISBN);
    books.push_back(Book(title, author, ISBN));
    cout << "Book added successfully." << endl;
 }
```

- Prompts for book details and adds a new `Book` object to the `books` vector.

4. Search Book:

```
    void searchBook() {
    int choice;
    string query;
    cout << "Search by: 1. Title 2. Author 3. ISBN\nEnter choice: ";
    cin >> choice;
    cout << "Enter query: ";
    cin.ignore();
    getline(cin, query);
    for (const auto& book : books) {
       if ((choice == 1 && book.title.find(query) != string::npos) ||
          (choice == 2 && book.author.find(query) != string::npos) ||
          (choice == 3 && book.ISBN.find(query) != string::npos)) {
          cout << "Title: " << book.title << ", Author: " << book.author
              << ", ISBN: " << book.ISBN << ", Available: "
              << (book.isAvailable ? "Yes" : "No") << endl;
       }} }
```

- Searches for books based on title, author, or ISBN and displays matching books.

Library Management System - Code Snippet

5. Checkout Book:

```
void checkoutBook() {
 string borrowerName, ISBN;
 cout << "Enter borrower's name: ";
 cin.ignore();
 getline(cin, borrowerName);
 cout << "Enter book ISBN: ";
 getline(cin, ISBN);
 for (auto& book : books) {
     if (book.ISBN == ISBN && book.isAvailable) {
         book.isAvailable = false;
         time_t now = time(0);
         time_t dueDate = now + 14 * 24 * 60 * 60; // 2 weeks from now
         for (auto& borrower : borrowers) {
             if (borrower.name == borrowerName) {
                 borrower.borrowedBooks[ISBN] = dueDate;
                 cout << "Book checked out successfully. Due date: " << ctime(&dueDate);
                 return;
             } }
         Borrower newBorrower;
         newBorrower.name = borrowerName;
         newBorrower.borrowedBooks[ISBN] = dueDate;
         borrowers.push_back(newBorrower);
         cout << "Book checked out successfully. Due date: " << ctime(&dueDate);
         return;
     } }
 cout << "Book is not available or ISBN not found." << endl;
}
```

- Checks out a book to a borrower, updates the book's availability, and sets the due date.

Library Management System - Code Snippet

6. Return Book:

```
    void returnBook() {
    string borrowerName, ISBN;
    cout << "Enter borrower's name: ";
    cin.ignore();
    getline(cin, borrowerName);
    cout << "Enter book ISBN: ";
    getline(cin, ISBN);
    for (auto& borrower : borrowers) {
      if (borrower.name == borrowerName) {
        auto it = borrower.borrowedBooks.find(ISBN);
        if (it != borrower.borrowedBooks.end()) {
          time_t now = time(0);
          if (now > it->second) {
            int overdueDays = (now - it->second) / (24 * 60 * 60);
            int fine = overdueDays * finePerDay;
            cout << "Book is overdue. Fine: " << fine << " units." << endl;
          } else {
            cout << "Book returned on time. No fine." << endl;
          }
          borrower.borrowedBooks.erase(it);
          for (auto& book : books) {
            if (book.ISBN == ISBN) {
              book.isAvailable = true;
              break;
            } }
          return;
      } } }
    cout << "Borrower or ISBN not found." << endl;
  }
```

- Processes the return of a book, calculates any overdue fines, and updates the book's availability.

Library Management System - Code Snippet

7. User Interface:

```
void userInterface() {
 int choice;
 do {
    cout << "\nLibrary Management System" << endl;
    cout << "1. Add Book" << endl;
    cout << "2. Search Book" << endl;
    cout << "3. Checkout Book" << endl;
    cout << "4. Return Book" << endl;
    cout << "5. Exit" << endl;
    cout << "Enter your choice: ";
    cin >> choice;
    switch (choice) {
        case 1:
            addBook();
            break;
        case 2:
            searchBook();
            break;
        case 3:
            checkoutBook();
            break;
        case 4:
            returnBook();
            break;
        case 5:
            cout << "Exiting..." << endl;
            break;
        default:
            cout << "Invalid choice. Please try again." << endl;
    } } while (choice != 5);
}
```

- Provides a menu-driven interface for users to interact with the library system.

Library Management System - Code Snippet

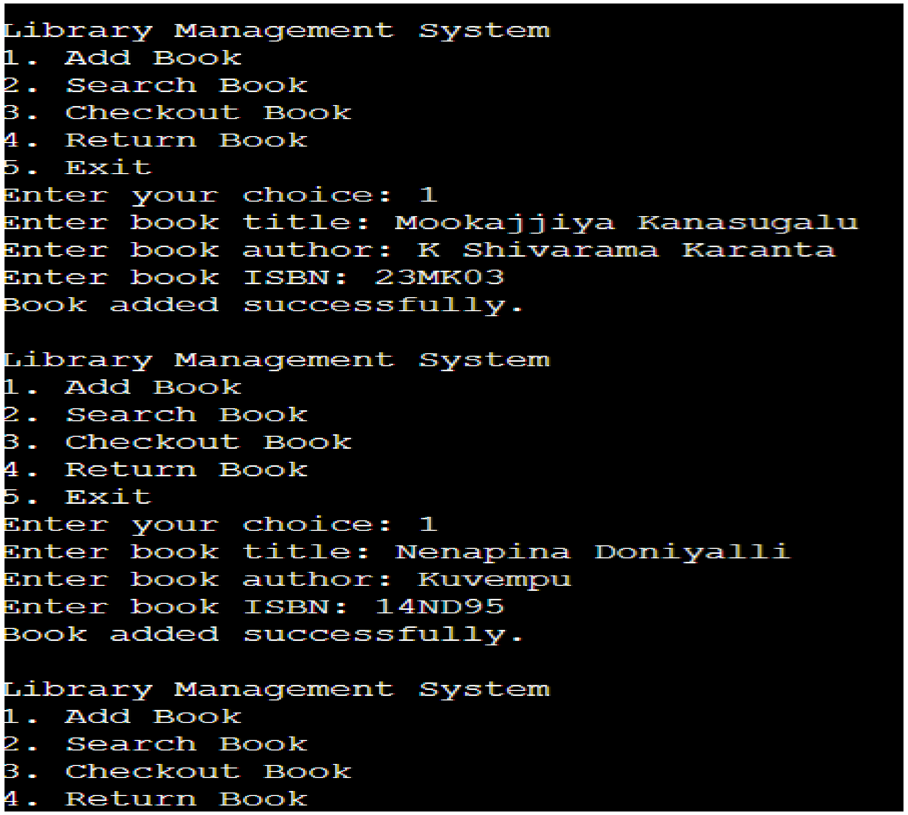
8. Main Function:

```
int main() {
    userInterface();
    return 0;
  }
```

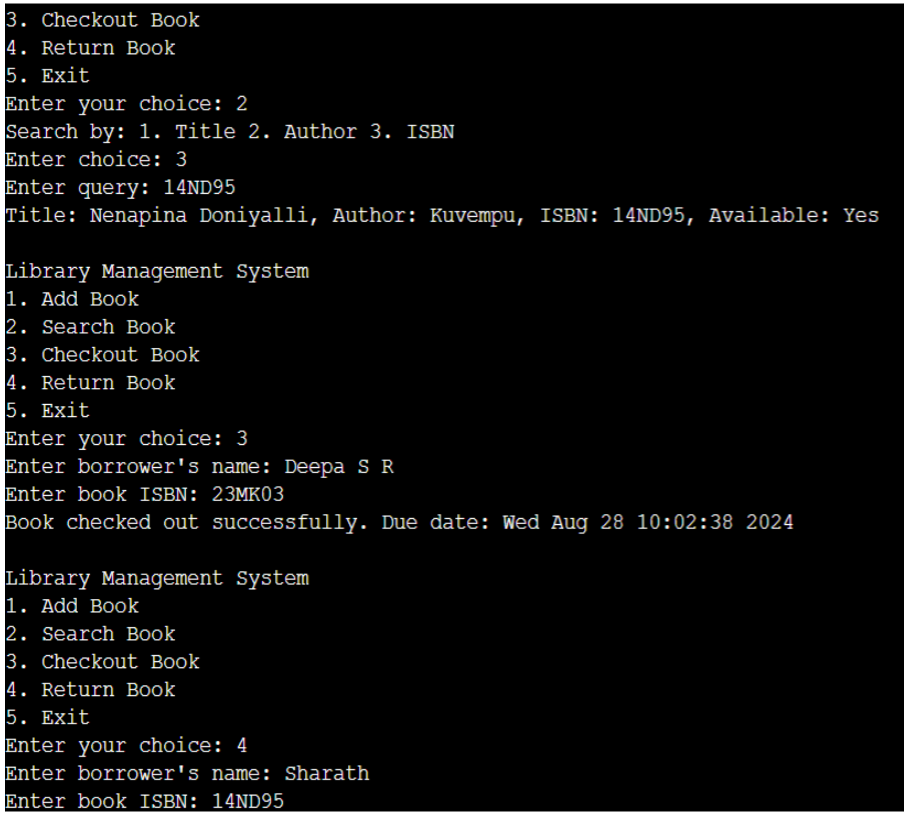
- Calls the `userInterface` function to start the library management system.

Library Management System - Input & Output

- Example of user input and corresponding output.



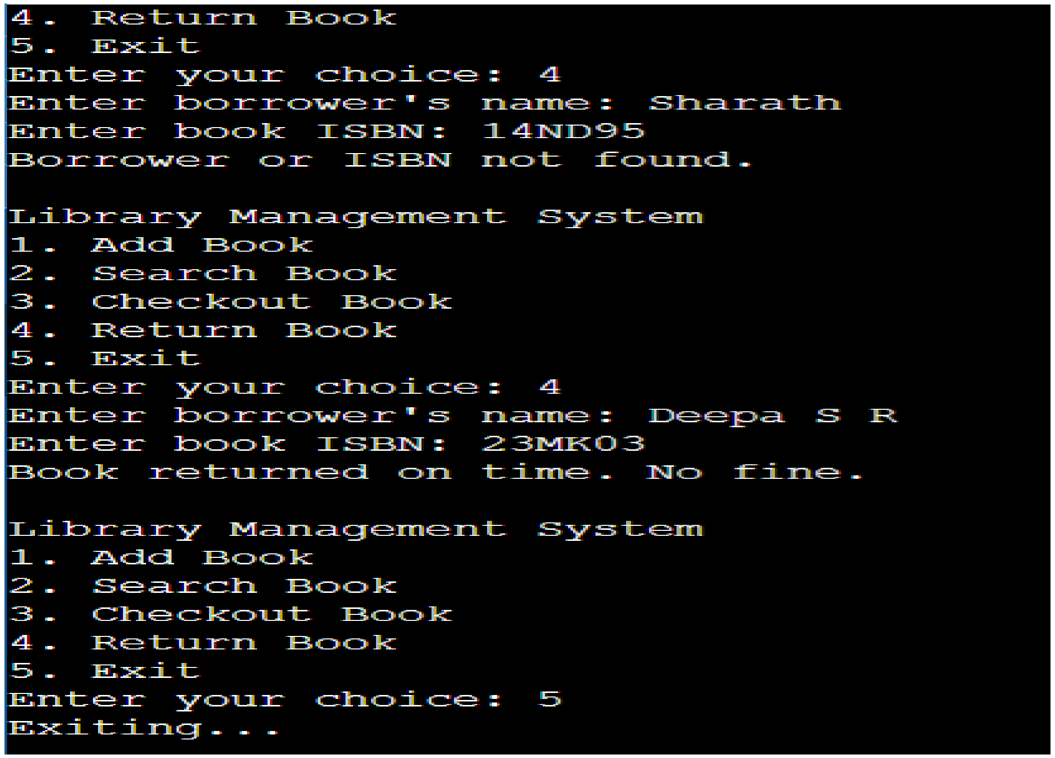
```
Library Management System
1. Add Book
2. Search Book
3. Checkout Book
4. Return Book
5. Exit
Enter your choice: 1
Enter book title: Mookajjiya Kanasugalu
Enter book author: K Shivarama Karanta
Enter book ISBN: 23MK03
Book added successfully.

Library Management System
1. Add Book
2. Search Book
3. Checkout Book
4. Return Book
5. Exit
Enter your choice: 1
Enter book title: Nenapina Doniyalli
Enter book author: Kuvempu
Enter book ISBN: 14ND95
Book added successfully.

Library Management System
1. Add Book
2. Search Book
3. Checkout Book
4. Return Book
```



```
3. Checkout Book
4. Return Book
5. Exit
Enter your choice: 2
Search by: 1. Title 2. Author 3. ISBN
Enter choice: 3
Enter query: 14ND95
Title: Nenapina Doniyalli, Author: Kuvempu, ISBN: 14ND95, Available: Yes

Library Management System
1. Add Book
2. Search Book
3. Checkout Book
4. Return Book
5. Exit
Enter your choice: 3
Enter borrower's name: Deepa S R
Enter book ISBN: 23MK03
Book checked out successfully. Due date: Wed Aug 28 10:02:38 2024

Library Management System
1. Add Book
2. Search Book
3. Checkout Book
4. Return Book
5. Exit
Enter your choice: 4
Enter borrower's name: Sharath
Enter book ISBN: 14ND95
```

Library Management System - Input & Output

- Example of user input and corresponding output.



```
4. Return Book
5. Exit
Enter your choice: 4
Enter borrower's name: Sharath
Enter book ISBN: 14ND95
Borrower or ISBN not found.

Library Management System
1. Add Book
2. Search Book
3. Checkout Book
4. Return Book
5. Exit
Enter your choice: 4
Enter borrower's name: Deepa S R
Enter book ISBN: 23MK03
Book returned on time. No fine.

Library Management System
1. Add Book
2. Search Book
3. Checkout Book
4. Return Book
5. Exit
Enter your choice: 5
Exiting...
```

Library Management System - Conclusion

- Summary of what the task achieves.

1. Initialize: Set up the environment by defining data structures and constants.
2. Add Book: Add new books to the library's collection.
3. Search Book: Search for books based on title, author, or ISBN.
4. Checkout Book: Check out books to borrowers, set due dates, and add new borrowers if necessary.
5. Return Book: Process book returns, calculate fines for overdue books, and update book availability.
6. User Interface: Provide an interactive menu for users to perform various library management tasks.

This program demonstrates key C++ concepts such as structures, vectors, maps, time handling, and user interaction through a console-based interface.

Conclusion

- Summary of the tasks completed and skills learned during the internship project.

1. During this internship project, I developed a solid understanding of C++ programming by working on three key projects: the Number Guessing Game, Tic-Tac-Toe Game, and Library Management System.
2. Each project offered valuable hands-on experience, from basic programming concepts like loops and conditionals to more advanced topics such as object-oriented design and file handling.
3. This journey has significantly enhanced my problem-solving skills and prepared me for further challenges in software development.

THANK YOU