Structure : C code

Your Name

November 13, 2024

Contents

1 Digital Library System

Problem Statement: You need to implement a digital library system using C, which stores details of books in a library.

Example: The library contains books with the title, author, and price. You will input data for multiple books and display it.

C Code:

```c
#include <stdio.h>
#include <string.h>

// Define the structure for Book
struct Book {
    char title[50];
    char author[50];
    float price;
};

int main() {
    struct Book books[5];

    // Input details for 5 books
    printf("Enter␣details␣for␣5␣books:\n");
    for(int i = 0; i < 5; i++) {
        printf("Book␣%d:\n", i+1);
        printf("Title:␣");
        getchar(); // To consume newline character
        fgets(books[i].title, sizeof(books[i].title), stdin);
        books[i].title[strcspn(books[i].title, "\n")] = '\0'; //
            Remove newline character

        printf("Author:␣");
        fgets(books[i].author, sizeof(books[i].author), stdin);
        books[i].author[strcspn(books[i].author, "\n")] = '\0'; //
             Remove newline character

        printf("Price:␣");
        scanf("%f", &books[i].price);
    }

    // Display book details in tabular format
    printf("\nLibrary␣Database:\n");
    printf("
        ----------------------------------------------------------\
        n");
    printf("|␣%-20s␣|␣%-20s␣|␣%-5s␣|\n", "Title", "Author", "Price
```

```
        ");
    printf("
        ----------------------------------------------------------\
        n");

    for(int i = 0; i < 5; i++) {
        printf("| %-20s | %-20s | %-5.2f |\n", books[i].title,
            books[i].author, books[i].price);
    }

    printf("
        ----------------------------------------------------------\
        n");

    return 0;
}
```

2 Student Database Management System

Problem Statement: Create a system to manage student information such as name, age, and marks for a group of students.

Example: The program should take input for 5 students and display their details in a tabular format.

C Code:

```
#include <stdio.h>
#include <string.h>

// Define the structure for Student
struct Student {
    char name[30];
    int age;
    float marks;
};

int main() {
    struct Student students[5];

    // Input details for 5 students
    printf("Enter␣details␣for␣5␣students:\n");
    for(int i = 0; i < 5; i++) {
        printf("Student␣%d:\n", i+1);
        printf("Name:␣");
        getchar(); // To consume newline character
        fgets(students[i].name, sizeof(students[i].name), stdin);
        students[i].name[strcspn(students[i].name, "\n")] = '\0';
            // Remove newline character

        printf("Age:␣");
        scanf("%d", &students[i].age);

        printf("Marks:␣");
        scanf("%f", &students[i].marks);
    }

    // Display student details in tabular format
    printf("\nStudent␣Database:\n");
    printf("
        ----------------------------------------------------------\
        n");
    printf("|␣%-20s␣|␣%-5s␣|␣%-5s␣|\n", "Name", "Age", "Marks");
    printf("
        ----------------------------------------------------------\
```

```
        n");

    for(int i = 0; i < 5; i++) {
        printf("| %-20s | %-5d | %-5.2f |\n", students[i].name,
            students[i].age, students[i].marks);
    }

    printf("
        ------------------------------------------------------\
        n");

    return 0;
}
```

3 Rectangle Area Calculator

Problem Statement: Design a program that calculates the area of rectangles for multiple rectangles provided as input.

Example: For 5 rectangles, input their length and width, and output their area in a tabular format.

C Code:

```
#include <stdio.h>

// Define the structure for Rectangle
struct Rectangle {
    float length;
    float width;
};

int main() {
    struct Rectangle rectangles[5];

    // Input details for 5 rectangles
    printf("Enter␣details␣for␣5␣rectangles:\n");
    for(int i = 0; i < 5; i++) {
        printf("Rectangle␣%d:\n", i+1);
        printf("Length:␣");
        scanf("%f", &rectangles[i].length);

        printf("Width:␣");
        scanf("%f", &rectangles[i].width);
    }

    // Display rectangle areas in tabular format
    printf("\nRectangle␣Areas:\n");
    printf("-------------------------------------------------\n"
        );
    printf("|␣%-9s␣|␣%-6s␣|␣%-6s␣|␣%-5s␣|\n", "Rectangle", "Length
        ", "Width", "Area");
    printf("-------------------------------------------------\n"
        );

    for(int i = 0; i < 5; i++) {
        float area = rectangles[i].length * rectangles[i].width;
        printf("|␣%-9d␣|␣%-6.2f␣|␣%-6.2f␣|␣%-5.2f␣|\n", i+1,
            rectangles[i].length, rectangles[i].width, area);
    }

    printf("-------------------------------------------------\n"
```

```
        );

    return 0;
}
```

4 Employee Payroll System

Problem Statement: Implement an employee payroll system to store and display employee details including their name, ID, and salary.

Example: The program should take input for 5 employees and display their details in a formatted table.

C Code:

```
#include <stdio.h>

// Define the structure for Employee
struct Employee {
    char name[30];
    int id;
    float salary;
};

int main() {
    struct Employee employees[5];

    // Input details for 5 employees
    printf("Enter␣details␣for␣5␣employees:\n");
    for(int i = 0; i < 5; i++) {
        printf("Employee␣%d:\n", i+1);
        printf("Name:␣");
        getchar(); // To consume newline character
        fgets(employees[i].name, sizeof(employees[i].name), stdin)
            ;
        employees[i].name[strcspn(employees[i].name, "\n")] = '\0'
            ; // Remove newline character

        printf("ID:␣");
        scanf("%d", &employees[i].id);

        printf("Salary:␣");
        scanf("%f", &employees[i].salary);
    }

    // Display payroll information in tabular format
    printf("\nEmployee␣Payroll:\n");
    printf("
        ----------------------------------------------------------\
        n");
    printf("|␣%-20s␣|␣%-5s␣|␣%-10s␣|\n", "Name", "ID", "Salary");
    printf("
        ----------------------------------------------------------\
```

```
        n");

    for(int i = 0; i < 5; i++) {
        printf("| %-20s | %-5d | %-10.2f |\n", employees[i].name,
            employees[i].id, employees[i].salary);
    }

    printf("
        -------------------------------------------------------\
        n");

    return 0;
}
```

5 Student Marks Update System

Problem Statement: Create a system that takes student marks as input, updates the marks, and displays the updated information.

Example: The program should add 5 marks to the current marks of each student and display the updated marks.

C Code:

```
#include <stdio.h>

// Define the structure for Student
struct Student {
    char name[30];
    int rollNo;
    float marks;
};

int main() {
    struct Student students[5];

    // Input details for 5 students
    printf("Enter␣details␣for␣5␣students:\n");
    for(int i = 0; i < 5; i++) {
        printf("Student␣%d:\n", i+1);
        printf("Name:␣");
        getchar(); // To consume newline character
        fgets(students[i].name, sizeof(students[i].name), stdin);
        students[i].name[strcspn(students[i].name, "\n")] = '\0';
            // Remove newline character

        printf("Roll␣No:␣");
        scanf("%d", &students[i].rollNo);

        printf("Marks:␣");
        scanf("%f", &students[i].marks);
    }

    // Update marks and display in tabular format
    printf("\nUpdated␣Marks␣List:\n");
    printf("
        ----------------------------------------------------------\
        n");
    printf("|␣%-20s␣|␣%-5s␣|␣%-6s␣|\n", "Name", "Roll␣No", "Marks"
        );
    printf("
        ----------------------------------------------------------\
```

```
        n");

    for(int i = 0; i < 5; i++) {
        students[i].marks += 5.0; // Adding 5 marks to each
            student
        printf("| %-20s | %-5d | %-6.2f |\n", students[i].name,
            students[i].rollNo, students[i].marks);
    }

    printf("
        ------------------------------------------------------\
        n");

    return 0;
}
```

6 Student Report Generation System with Marks and Grade

Problem Statement: Create a Student Report Generation System where a structure `Student` contains: - `name` (string, up to 30 characters) - `marks1`, `marks2`, `marks3` (floats) - `grade` (string, up to 2 characters)

$$\text{If average marks } \geq 90, \quad \text{grade} = \text{"A"}$$

$$\text{If average marks } \geq 75 \text{ and } < 90, \quad \text{grade} = \text{"B"}$$

$$\text{If average marks } \geq 60 \text{ and } < 75, \quad \text{grade} = \text{"C"}$$

$$\text{If average marks } < 60, \quad \text{grade} = \text{"D"}$$

Create an array of 5 Student structures, input data for each student, and display their report.

C Code:

```c
#include <stdio.h>
#include <string.h>

// Define the structure for Student
struct Student {
    char name[30];
    float marks1, marks2, marks3;
    char grade[2];
};

int main() {
    struct Student students[5];

    // Input details for 5 students
    for(int i = 0; i < 5; i++) {
        printf("Student␣%d:\n", i+1);
        printf("Name:␣");
        getchar(); // To consume newline character
        fgets(students[i].name, sizeof(students[i].name), stdin);
        students[i].name[strcspn(students[i].name, "\n")] = '\0';
            // Remove newline character

        printf("Marks␣1:␣");
        scanf("%f", &students[i].marks1);
        printf("Marks␣2:␣");
        scanf("%f", &students[i].marks2);
        printf("Marks␣3:␣");
        scanf("%f", &students[i].marks3);
```

```
        // Calculate average and grade
        float average = (students[i].marks1 + students[i].marks2 +
             students[i].marks3) / 3;
        if(average >= 90) {
            strcpy(students[i].grade, "A");
        } else if(average >= 75) {
            strcpy(students[i].grade, "B");
        } else if(average >= 60) {
            strcpy(students[i].grade, "C");
        } else {
            strcpy(students[i].grade, "D");
        }
    }

    // Display student report
    printf("\nStudent␣Report:\n");
    printf("
        ------------------------------------------------------------\
        n");
    printf("|␣Name␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣Marks␣1␣|␣Marks␣2␣|␣Marks␣3␣|␣
        Average␣|␣Grade␣|\n");
    printf("
        ------------------------------------------------------------\
        n");

    for(int i = 0; i < 5; i++) {
        float average = (students[i].marks1 + students[i].marks2 +
             students[i].marks3) / 3;
        printf("|␣%-17s␣|␣%-7.2f␣|␣%-7.2f␣|␣%-7.2f␣|␣%-7.2f␣|␣%-5s
            ␣|\n", students[i].name, students[i].marks1, students[
            i].marks2, students[i].marks3, average, students[i].
            grade);
    }

    printf("
        ------------------------------------------------------------\
        n");

    return 0;
}
```

7 Date and Time Management System

Problem Statement: Design a Date and Time Management System with a structure `DateTime` that contains the following fields: - `day, month, year` (integer) - `hour, minute, second` (integer)

Create an array of 5 `DateTime` structures to store and display the date and time for 5 events. The program should then display all the entered date-time values in a tabular format.

C Code:

```
#include <stdio.h>

// Define the structure for DateTime
struct DateTime {
    int day, month, year;
    int hour, minute, second;
};

int main() {
    struct DateTime events[5];

    // Input details for 5 events
    for(int i = 0; i < 5; i++) {
        printf("Event␣%d:\n", i+1);
        printf("Day:␣");
        scanf("%d", &events[i].day);
        printf("Month:␣");
        scanf("%d", &events[i].month);
        printf("Year:␣");
        scanf("%d", &events[i].year);
        printf("Hour:␣");
        scanf("%d", &events[i].hour);
        printf("Minute:␣");
        scanf("%d", &events[i].minute);
        printf("Second:␣");
        scanf("%d", &events[i].second);
    }

    // Display the event date and time list
    printf("\nEvent␣Date␣and␣Time␣List:\n");
    printf("
        ------------------------------------------------------------\
        n");
    printf("|␣Event␣|␣Day␣|␣Month␣|␣Year␣|␣Hour␣|␣Minute␣|␣Second␣
        |\n");
    printf("
```

```
        ------------------------------------------------------------\
        n");

    for(int i = 0; i < 5; i++) {
        printf("| %-5d | %-3d | %-5d | %-4d | %-4d | %-6d | %-6d
            |\n", i+1, events[i].day, events[i].month, events[i].
            year, events[i].hour, events[i].minute, events[i].
            second);
    }

    printf("
        ------------------------------------------------------------\
        n");

    return 0;
}
```

8 Vehicle Registration System

Problem Statement: Create a Vehicle Registration System where a structure `Vehicle` contains: - `registrationNumber` (string, up to 20 characters) - `model` (string, up to 20 characters) - `ownerName` (string, up to 30 characters) - `yearOfManufacture` (integer)

Create an array of 5 `Vehicle` structures, input details for 5 vehicles, and then display all the information in a tabular format.

C Code:

```
#include <stdio.h>
#include <string.h>

// Define the structure for Vehicle
struct Vehicle {
    char registrationNumber[20];
    char model[20];
    char ownerName[30];
    int yearOfManufacture;
};

int main() {
    struct Vehicle vehicles[5];

    // Input details for 5 vehicles
    for(int i = 0; i < 5; i++) {
        printf("Vehicle␣%d:\n", i+1);
        printf("Registration␣Number:␣");
        getchar(); // To consume newline character
        fgets(vehicles[i].registrationNumber, sizeof(vehicles[i].
            registrationNumber), stdin);
        vehicles[i].registrationNumber[strcspn(vehicles[i].
            registrationNumber, "\n")] = '\0';

        printf("Model:␣");
        fgets(vehicles[i].model, sizeof(vehicles[i].model), stdin)
            ;
        vehicles[i].model[strcspn(vehicles[i].model, "\n")] = '\0'
            ;

        printf("Owner␣Name:␣");
        fgets(vehicles[i].ownerName, sizeof(vehicles[i].ownerName)
            , stdin);
        vehicles[i].ownerName[strcspn(vehicles[i].ownerName, "\n")
            ] = '\0';
```

```
        printf("Year␣of␣Manufacture:␣");
        scanf("%d", &vehicles[i].yearOfManufacture);
    }

    // Display the vehicle registration list
    printf("\nVehicle␣Registration␣List:\n");
    printf("
        ----------------------------------------------------------\
        n");
    printf("|␣Registration␣Number␣|␣Model␣␣␣␣␣␣␣␣␣␣␣|␣Owner␣␣␣␣␣␣␣
        ␣|␣Year␣of␣Manufacture␣|\n");
    printf("
        ----------------------------------------------------------\
        n");

    for(int i = 0; i < 5; i++) {
        printf("|␣%-20s␣|␣%-15s␣|␣%-12s␣|␣%-19d␣|\n", vehicles[i].
            registrationNumber, vehicles[i].model, vehicles[i].
            ownerName, vehicles[i].yearOfManufacture);
    }

    printf("
        ----------------------------------------------------------\
        n");

    return 0;
}
```

9 Course Registration System

Problem Statement: In a Course Registration System, define a structure `Course` that holds: - `courseName` (string, up to 40 characters) - `courseCode` (string, up to 10 characters) - `creditHours` (integer)

Create an array of 5 `Course` structures. Allow users to input data for 5 courses, and then display the course list with their respective credit hours.

C Code:

```c
#include <stdio.h>
#include <string.h>

// Define the structure for Course
struct Course {
    char courseName[40];
    char courseCode[10];
    int creditHours;
};

int main() {
    struct Course courses[5];

    // Input details for 5 courses
    for(int i = 0; i < 5; i++) {
        printf("Course %d:\n", i+1);
        printf("Course Name: ");
        getchar(); // To consume newline character
        fgets(courses[i].courseName, sizeof(courses[i].courseName)
            , stdin);
        courses[i].courseName[strcspn(courses[i].courseName, "\n")
            ] = '\0';

        printf("Course Code: ");
        fgets(courses[i].courseCode, sizeof(courses[i].courseCode)
            , stdin);
        courses[i].courseCode[strcspn(courses[i].courseCode, "\n")
            ] = '\0';

        printf("Credit Hours: ");
        scanf("%d", &courses[i].creditHours);
    }

    // Display the course registration list
    printf("\nCourse Registration List:\n");
    printf("
        ----------------------------------------------------------\
```

```
        n");
    printf("| Course Name          | Course Code | Credit Hours |\n
        ");
    printf("
        ----------------------------------------------------------\
        n");

    for(int i = 0; i < 5; i++) {
        printf("| %-19s | %-11s | %-12d |\n", courses[i].
            courseName, courses[i].courseCode, courses[i].
            creditHours);
    }

    printf("
        ----------------------------------------------------------\
        n");

    return 0;
}
```

10 Employee Salary Management System

Problem Statement: Design a Salary Management System using a structure `Employee` that contains: - `name` (string, up to 30 characters) - `employeeID` (integer) - `basicSalary` (float) - `bonus` (float) - `totalSalary` (float)

Write a program to input the details for 5 employees, calculate their total salary (basic salary + bonus), and display the employee details.

C Code:

```c
#include <stdio.h>
#include <string.h>

// Define the structure for Employee
struct Employee {
    char name[30];
    int employeeID;
    float basicSalary;
    float bonus;
    float totalSalary;
};

int main() {
    struct Employee employees[5];

    // Input details for 5 employees
    for(int i = 0; i < 5; i++) {
        printf("Employee %d:\n", i+1);
        printf("Name: ");
        getchar(); // To consume newline character
        fgets(employees[i].name, sizeof(employees[i].name), stdin)
            ;
        employees[i].name[strcspn(employees[i].name, "\n")] = '\0'
            ;

        printf("Employee ID: ");
        scanf("%d", &employees[i].employeeID);

        printf("Basic Salary: ");
        scanf("%f", &employees[i].basicSalary);

        printf("Bonus: ");
        scanf("%f", &employees[i].bonus);

        // Calculate total salary
        employees[i].totalSalary = employees[i].basicSalary +
            employees[i].bonus;
```

```
    }

    // Display the employee salary details
    printf("\nEmployee␣Salary␣Details:\n");
    printf("
        --------------------------------------------------------------\
        n");
    printf("|␣Name␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣Employee␣ID␣|␣Basic␣Salary␣|␣
        Bonus␣|␣Total␣Salary␣|\n");
    printf("
        --------------------------------------------------------------\
        n");

    for(int i = 0; i < 5; i++) {
        printf("|␣%-17s␣|␣%-11d␣|␣%-12.2f␣|␣%-5.2f␣|␣%-12.2f␣|\n",
             employees[i].name, employees[i].employeeID, employees
            [i].basicSalary, employees[i].bonus, employees[i].
            totalSalary);
    }

    printf("
        --------------------------------------------------------------\
        n");

    return 0;
}
```

11 Edge Case for Book Input System

Edge Case 1: Empty Book Title or Author

Explanation: The system should handle cases where a user tries to add a book with an empty title or author. Empty titles or author names are invalid and should prompt the user to provide valid input.

Example Input:

- Title = ""
- Author = "John Doe"
- Year = 2020

Expected Behavior:

- The system should reject the empty title and ask the user to input a valid title.
- The system should reject the empty author and ask the user to input a valid author name.

Expected Output:

```
Book 1:
Title:
Title cannot be empty. Please enter a valid title.
Title: Programming in C
Author: John Doe
Price: 29.99
Year of Publication: 2020
```

Edge Case 2: Invalid Year of Publication (Year in the Future)

Explanation: A book cannot have a publication year in the future. The system should reject a year greater than the current year.

Example Input:

- Title = "Advanced Programming"
- Author = "Alice"
- Year = 2050

Expected Behavior:

- The system should reject the future year and ask the user to input a valid year.

Expected Output:

```
Book 2:
Title: Advanced Programming
Author: Alice
Price: 39.99
Year of Publication: 2050
Invalid year. Please enter a valid year of publication (not in the future).
Year of Publication: 2023
```

Edge Case 3: Maximum Length for Title and Author

Explanation: The system should handle cases where the title or author exceeds the maximum allowed character length (100 characters). The user should be notified if their input is too long.

Example Input:

- Title = "A very long book title that exceeds usual limits of normal titles to check boundary condition"
- Author = "A very long author name for testing purposes"
- Year = 2000

Expected Behavior:

- The system should reject titles and authors that exceed the maximum length (100 characters).
- The system should prompt the user to enter a shorter title or author.

Expected Output:

```
Book 3:
Title: A very long book title that exceeds usual limits of normal titles to check boundary c
Title is too long. Please enter a title with a maximum of 100 characters.
Title: Introduction to Algorithms and Data Structures in Computer Science
Author: A very long author name for testing purposes
Author name is too long. Please enter an author name with a maximum of 100 characters.
Author: Robert C. Martin
Price: 49.95
Year of Publication: 2005
```

12 Edge Cases for Student Database Management System

The following are edge cases for the **Student Database Management System** (Question 2). These cases ensure that the system handles various types of inputs and conditions gracefully, including invalid or extreme values.

Edge Case 1. Empty or Missing Input

Case: If the user does not provide any input for the student fields, such as name, marks, or grade, the program should handle this case gracefully.

Example:

- **Input**:

```
Name: (empty)
Marks 1: 85
Marks 2: 90
Marks 3: 88
```

- **Expected Behavior**: The program should either prompt the user to enter a valid name or handle the empty input by setting a default value or skipping the record.

Edge Case 2. Negative Marks

Case: The program might encounter negative marks as input, which are not valid in this context (marks should always be non-negative).

Example:

- **Input**:

```
Name: John
Marks 1: -10
Marks 2: 90
Marks 3: 80
```

- **Expected Behavior**: The program should validate the marks input and either reject the negative value or set it to 0 or ask for a valid positive number.

Edge Case 3. Marks Exceeding Maximum Value (Above 100)

Case: Marks exceeding the maximum value of 100 might be entered. This is invalid as marks usually range from 0 to 100.

Example:

- **Input**:

```
Name: Alice
Marks 1: 105
Marks 2: 90
Marks 3: 92
```

- **Expected Behavior**: The program should handle this by either rejecting the input or correcting the value (e.g., capping it to 100 or prompting the user for a valid value).

Edge Case 4. Boundary Conditions for Grades (90, 75, 60)

Case: Students whose average marks are exactly on the boundary between two grades (e.g., 90, 75, and 60).

Examples:

- **Input**:

```
Name: Bob
Marks 1: 90
Marks 2: 90
Marks 3: 90
```

- **Expected Output**: Grade = "A" (since the average marks are exactly 90, which is the boundary for an "A").
- **Input**:

```
Name: Carol
Marks 1: 75
Marks 2: 75
Marks 3: 75
```

- **Expected Output**: Grade = "B" (average is exactly 75, so it should still fall in the "B" category).
- **Input**:

```
Name: David
Marks 1: 60
Marks 2: 60
Marks 3: 60
```

- **Expected Output**: Grade = "C" (average is exactly 60, so it falls within the "C" range).

Edge Case 5. Non-Numeric Input for Marks

Case: If the user enters non-numeric characters when providing marks (e.g., letters, special characters), the program should handle the error gracefully.

Example:

- **Input**:

```
Name: Alice
Marks 1: abc
Marks 2: 85
Marks 3: 90
```

- **Expected Behavior**: The program should either prompt the user to enter a valid numeric value or handle the invalid input by providing a default value (e.g., 0) or showing an error message.

Edge Case 6. Maximum Number of Students Exceeded

Case: If the user tries to input more than the expected number of students (e.g., more than 5 students in this case), the program should either prevent this or gracefully handle the additional input.

Example:

- **Input**:

```
Name: Alice
Marks 1: 80
Marks 2: 90
Marks 3: 88
Name: Bob
Marks 1: 70
Marks 2: 75
Marks 3: 80
...
(Additional student data)
```

- **Expected Behavior**: The program might be designed to handle exactly 5 students, so it should either reject additional students or store only the first 5. Alternatively, it might allow for dynamic input beyond the limit.

Edge Case 7. Floating Point Precision Issues

Case: Floating-point precision errors can occur when calculating averages or handling decimal marks, which might result in slightly inaccurate averages or grades.

Example:

- **Input**:

```
Name: John
Marks 1: 89.9999
Marks 2: 90.0001
Marks 3: 90.0
```

- **Expected Behavior**: The program should correctly round or display the average with an appropriate number of decimal places to avoid floating-point inaccuracies.

Edge Case 8. Very Large or Very Small Inputs for Marks

Case: If the marks entered are extremely large or small (such as beyond the expected range), the program should handle it gracefully.

Example:

- **Input**:

```
Name: Charlie
Marks 1: 1000000
Marks 2: 999999
Marks 3: 1000000
```

- **Expected Behavior**: The program should detect that the marks are out of a reasonable range (typically between 0 and 100) and either reject them or limit the input to a valid range (e.g., $0 \leq marks \leq 100$).

Edge Case 9. Whitespace and Special Characters in Name

Case: The program should correctly handle special characters or extra spaces in the student names (e.g., names with spaces or punctuation).

Example:

- **Input**:

```
Name: Alice O'Connor
Marks 1: 80
Marks 2: 75
Marks 3: 88
```

- **Expected Behavior**: The program should correctly accept and display names with spaces or special characters.

13 Edge Cases for Rectangle Area Calculator

Edge Case 1: Negative or Zero Values for Length and Width

- Input values for length or width should not be negative or zero, as it doesn't make sense to have negative or zero dimensions for a rectangle.
- The program should reject such input and prompt the user for valid positive values.

Expected Output:

```
Enter length: -5
Length and width must be positive values. Please enter a valid length.
Enter width: 4
Area cannot be calculated with negative dimensions.
```

Edge Case 2: Extremely Large Values

- The program should handle very large inputs (e.g., dimensions close to or exceeding the maximum allowable values for integers).
- The program should either limit input size or handle overflow errors.

Expected Output:

```
Enter length: 1000000000
Enter width: 1000000000
Area calculated: 1000000000000000000
```

14 Edge Cases for Employee Payroll System

Edge Case 1: Negative Salary or Hours Worked

- The program should not accept negative values for salary or hours worked.
- The program should prompt the user to input valid positive values.

Expected Output:

```
Enter hourly wage: -25
Hourly wage must be a positive value. Please enter a valid amount.
Enter hours worked: 40
Payroll cannot be calculated with negative salary.
```

Edge Case 2: Invalid Employee ID

- The system should reject invalid employee IDs (e.g., empty or non-numeric IDs).
- The system should request a valid numeric employee ID.

Expected Output:

```
Enter employee ID:
Employee ID cannot be empty. Please enter a valid ID.
```

15 Edge Cases for Student Marks Update System

Edge Case 1: Marks Outside the Valid Range (0-100)

- The program should ensure that student marks are between 0 and 100. Any input outside this range should be rejected.

Expected Output:

```
Enter marks for student 1: 120
Marks should be between 0 and 100. Please enter a valid mark.
```

Edge Case 2: Non-Numeric Input for Marks

- The system should reject non-numeric input for marks.

Expected Output:

```
Enter marks for student 2: abc
Invalid input. Marks must be a numeric value.
```

16 Edge Cases for Student Report Generation System

Edge Case 1: Division by Zero in Grade Calculation

- The system should handle division by zero errors when calculating grades (e.g., total marks being zero).
- The program should ensure that marks are appropriately handled to avoid zero division.

Expected Output:

```
Total Marks: 0
Grade cannot be calculated due to zero total marks.
```

Edge Case 2: Invalid Grade Scale

- The grade scale should be predefined (e.g., A for 90+, B for 80-89, etc.). If the marks don't fit into the predefined grade scale, the system should flag it.

Expected Output:

```
Enter marks: 150
Invalid marks. Please enter marks within the valid range.
```

17 Edge Cases for Date and Time Management System

Edge Case 1: Invalid Date Format

- The system should handle invalid date formats (e.g., entering "2022/13/45").
- It should reject incorrect formats and prompt the user to re-enter a valid date.

Expected Output:

```
Enter date (DD/MM/YYYY): 31/02/2022
Invalid date. Please enter a valid date.
```

Edge Case 2: Time Zone Mismatches

- The system should handle time zone mismatches and ensure that the correct time zone is used.

Expected Output:

```
Enter time: 15:30
Enter time zone: UTC+2
Time zone mismatch detected. Please enter a valid time zone.
```

18 Edge Cases for Vehicle Registration System

Edge Case 1: Invalid License Plate Format

- The system should handle cases where the user inputs an incorrect license plate format (e.g., "1234-ABC" instead of "ABC-1234").
- The system should reject invalid formats and prompt the user to enter the correct format.

Expected Output:

```
Enter license plate: 1234-ABC
Invalid license plate format. Please enter in the format ABC-1234.
```

Edge Case 2: Duplicate License Plate

- The system should prevent duplicate license plate entries.
- If a license plate is already registered, the system should notify the user and reject the entry.

Expected Output:

```
License plate already registered. Please enter a new license plate.
```

19 Edge Cases for Course Registration System

Edge Case 1: Invalid Course Code

- The system should handle invalid course codes (e.g., incorrect length or format).
- It should reject such codes and prompt the user for a valid course code.

Expected Output:

```
Enter course code: CS10101
Invalid course code. Please enter a valid 6-digit course code.
```

Edge Case 2: Exceeding Maximum Course Registration Limit

- The system should ensure that students do not register for more courses than the allowed limit.
- The system should reject the registration attempt if the student exceeds the maximum allowed course load.

Expected Output:

```
Course limit exceeded. You can register for up to 5 courses only.
```

20 Edge Cases for Employee Salary Management System

Edge Case 1: Negative Salary Value

- The system should not accept negative salary values.
- It should prompt the user to enter a valid positive salary value.

Expected Output:

```
Enter salary: -1000
Salary cannot be negative. Please enter a valid salary.
```

Edge Case 2: Invalid Bonus Percentage

- The program should handle invalid bonus percentages (e.g., greater than 100
- It should ask the user to enter a valid bonus percentage between 0 and 100.

Expected Output:

```
Enter bonus percentage: 120
Invalid bonus percentage. Please enter a value between 0 and 100.
```