C++ Programming Assignment

Assignment Title: Exploring Functions in C++

Objectives:

1. Understand the use of predefined functions and learn how to create and work with user-defined functions.

2. Develop skills in using value-returning functions, reference parameters, and void functions.

3. Familiarize with the syntax and flow of function prototypes, parameter lists, and return statements.

4. Explore advanced topics such as function overloading, default parameters, scope, memory allocation, and debugging with drivers and stubs.

Instructions:

1. Code Structure: Organize your code by breaking down tasks into separate functions. Write a main function and ensure each task is defined within a separate user-defined function, using value-returning functions where needed.

2. Documentation: Add comments to explain each function's purpose, parameter requirements, and the return type (if applicable).

3. Compilation: Ensure that the program compiles without errors. Test each function separately before integrating them into the main program.

4. Formatting: Use consistent and clear formatting with indentation and naming conventions (e.g., camelCase or snake_case for variable and function names).

5. Submission: Submit the program source code in a single `.cpp` file.

Task Requirements

Task 1: Basic Calculator using Predefined Functions

Create a menu-driven program with predefined mathematical functions like `sqrt`, `pow`, and `abs`. Prompt the user to:

- Choose an operation: square root, power, or absolute value.

- Input the required numbers based on the selected operation.

- Display the result to the user.

Task 2: Custom Functions for Temperature Conversion

Define two user-defined value-returning functions:

1. `fahrenheitToCelsius(double fahrenheit)`: Converts Fahrenheit to Celsius.

2. `celsiusToFahrenheit(double celsius)`: Converts Celsius to Fahrenheit.

Each function should:

- Take a temperature value as an input parameter.

- Return the converted temperature.

- Use a `return` statement and proper value-returning function syntax.

Task 3: Area and Perimeter Calculations with Reference Parameters

Create a function `calculateRectangle(double length, double width, double &area, double &perimeter)` that:

- Takes `length` and `width` as value parameters.

- Calculates and updates the values of `area` and `perimeter` using reference parameters.

- Display the results in `main` by calling `calculateRectangle` and passing the required parameters.

Task 4: Scope and Static Variables in Counter Function

Create a function `incrementCounter()` that:

- Uses a static variable to maintain its count across multiple calls.

- Prints the current count on each call.

- Demonstrates the concept of static and automatic variables by calling `incrementCounter()` multiple times within `main`.

Task 5: Function Overloading for Shape Area Calculation

Implement overloaded functions `calculateArea` to handle different shapes:

1. `calculateArea(double radius)`: Calculates the area of a circle.

2. `calculateArea(double length, double width)`: Calculates the area of a rectangle.

3. `calculateArea(double base, double height, bool isTriangle)`: Calculates the area of a triangle when `isTriangle` is set to `true`.

Prompt the user to choose a shape and provide the relevant inputs. Display the calculated area.

Task 6: Default Parameters in Greeting Function

Write a function `greetUser(string name, string title = "Mr./Ms.")`:

- Accepts the user's name as a parameter and an optional title.

- Prints a greeting like "Hello, [title] [name]!".

- Demonstrate this by calling `greetUser` with and without the `title` argument in `main`.

Additional Guidelines

- Function Prototype: Use function prototypes for all functions before the `main` function.

- Parameter Passing: Experiment with both value and reference parameters, understanding how memory allocation differs.

- Global Variables: Avoid using global variables unless necessary. Test the effect of global variables on side effects.

- Debugging: Test each function individually by creating simple drivers or stub functions to verify functionality before integration.

Expected Output Example

```
Choose an operation:
1. Basic Calculator
2. Temperature Conversion
3. Rectangle Area and Perimeter
4. Counter Function
5. Shape Area Calculation
6. Greet User
7. Exit
Enter your choice: 1
Basic Calculator:
1. Square Root
2. Power
3. Absolute Value
Enter your choice: 1
Enter number: 25
Square Root: 5
```

```
Choose an operation:
1. Basic Calculator
2. Temperature Conversion
3. Rectangle Area and Perimeter
4. Counter Function
5. Shape Area Calculation
6. Greet User
7. Exit
Enter your choice: 2
Temperature Conversion:
1. Fahrenheit to Celsius
2. Celsius to Fahrenheit
Enter your choice: 2
Enter temperature in Celsius: 20
Fahrenheit: 68

Choose an operation:
1. Basic Calculator
2. Temperature Conversion
3. Rectangle Area and Perimeter
4. Counter Function
5. Shape Area Calculation
6. Greet User
7. Exit
Enter your choice: 5
Shape Area Calculation:
```

```
3. Rectangle Area and Perimeter
4. Counter Function
5. Shape Area Calculation
6. Greet User
7. Exit
Enter your choice: 5
Shape Area Calculation:
1. Circle
2. Rectangle
3. Triangle
Enter your choice: 1
Enter radius: 5
Circle Area: 78.5398

Choose an operation:
1. Basic Calculator
2. Temperature Conversion
3. Rectangle Area and Perimeter
4. Counter Function
5. Shape Area Calculation
6. Greet User
7. Exit
Enter your choice: 7
Exiting the program. Goodbye!
```

```
Choose an operation:
1. Basic Calculator
2. Temperature Conversion
3. Rectangle Area and Perimeter
4. Counter Function
5. Shape Area Calculation
6. Greet User
7. Exit
Enter your choice: 6
Enter name: Sumanth
Enter title (or leave blank for default): Mr
Hello, Mr Sumanth!

Choose an operation:
1. Basic Calculator
2. Temperature Conversion
3. Rectangle Area and Perimeter
4. Counter Function
5. Shape Area Calculation
6. Greet User
7. Exit
Enter your choice: 7
Exiting the program. Goodbye!
```

Grading Criteria

- Functionality: Does the program perform as expected?

- Code Quality: Are naming conventions, formatting, and documentation clear and consistent?

- Use of Concepts: Did you apply function prototypes, overloading, reference parameters, and other required topics effectively?

- Error Handling: Does the program handle invalid input gracefully?