Recursive Functions in C++ with Pointers and Structs

Your Name

April 7, 2025

1 Introduction

This document provides examples and exercises in C++ designed to help you familiarize yourself with recursive functions while incorporating pointers and structs. The examples are organized step-by-step for clarity, and the exercises are open-ended to encourage exploration and deeper understanding. This approach has been tailored for an AS/CR type learner, focusing both on logical sequencing and creative problem-solving.

2 Basic Examples

2.1 Recursive Factorial Function

A classic example of recursion is computing the factorial of a number. Although this example does not involve pointers or structs, it introduces the basic recursive mindset.

```cpp
#include <iostream>
using namespace std;

int factorial(int n) {
    if(n <= 1)
        return 1;  // Base case: factorial(0) = factorial(1) = 1
    return n * factorial(n - 1);
}

int main(){
    int number = 5;
    cout << "Factorial of " << number << " is " << factorial(number) << endl;
    return 0;
}
```

2.2 Recursive Linked List Length with Pointers and Structs

This example shows how to define a simple linked list using a struct and calculate its length recursively.

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

int listLength(Node* head) {
    if(head == nullptr)
        return 0;  // Base case: reached the end of the list
    return 1 + listLength(head->next);
}

int main(){
    // Creating a simple linked list: 1 -> 2 -> 3 -> nullptr
    Node n3 = {3, nullptr};
    Node n2 = {2, &n3};
    Node n1 = {1, &n2};

    cout << "Linked list length: " << listLength(&n1) <<
        endl;
    return 0;
}
```

3 Exercises

3.1 Exercise 1: Recursive Fibonacci Function

Task: Write a recursive function that computes the nth Fibonacci number.

Hints:

- Use the base cases: Fibonacci(0) = 0 and Fibonacci(1) = 1.
- For $n > 1$, define Fibonacci(n) as Fibonacci(n-1) + Fibonacci(n-2).
- Experiment with optimizations such as tail recursion or memoization if you feel adventurous.

3.2 Exercise 2: Print a Linked List in Reverse Order

Task: Extend the linked list example. Write a recursive function that prints the elements of a linked list in reverse order.

Hints:

- First, traverse to the end of the list recursively.
- Then, print each node's data on the way back up the recursive calls.
- This exercise reinforces both recursion and pointer manipulation.

3.3 Exercise 3: Binary Tree Height and Inorder Traversal

Task: Define a binary tree using structs and pointers. Implement two recursive functions:

1. One to compute the height of the tree.
2. One to perform an inorder traversal (left-root-right) of the tree.

Hints:

- For the height function, use the base case of an empty node (return 0) and compute the height as $1 + \max(\text{height of left, height of right})$.
- For inorder traversal, recursively visit the left subtree, print the current node's value, then visit the right subtree.
- This exercise helps you understand recursion in hierarchical data structures.

3.4 Exercise 4: Recursive Reversal of a Linked List

Task: Write a recursive function that reverses a singly linked list in place.

Hints:

- Consider the base case when the list is empty or contains only one node.
- Use pointer manipulation carefully to change the direction of the links.
- This exercise deepens your understanding of pointers and recursion.

4 Final Thoughts

These examples and exercises are structured to offer a logical progression and creative challenges. By working through these problems, you'll develop both a theoretical and practical understanding of recursion in C++ with pointers and structs.