Assignment: Peano Arithmetic Synthesizer

Inductive Math Modeling via Classes and Recursion

Overview

This project challenges you to model natural numbers from scratch using Peano's axioms. Designed for learners who value logical rigor (Abstract Sequential) and creative problem-solving (Concrete Random), this assignment will guide you through:

- Analyzing underlying principles of inductive reasoning.
- Constructing recursive data structures.
- Implementing arithmetic operations using clear, logical steps.
- Exploring creative performance enhancements.

1 Underlying Principles and Compelling Reasons

Why this project?

- **Inductive Reasoning:** Understand how the Peano axioms underpin natural number arithmetic.
- **Logical Analysis:** Break down complex ideas into base cases and inductive steps.
- **Creative Exploration:** Experiment with alternative memory management strategies and performance benchmarking.
- **Practical Application:** Bridge theoretical concepts with hands-on C++ coding.

2 Tasks

Task 1: Define the Peano Axioms in English

- **Objective:** Write down the Peano axioms for $\mathbb{N}$.
- **Instructions:**
  - Identify which statements serve as the base case(s) and which are inductive.

  - Provide logical reasoning for the structure of each axiom.

- **Goal:** Solidify your understanding of how recursive structures form the basis of natural number arithmetic.

Task 2: Define a Recursive Struct for Natural Numbers

- **Objective:** Create a basic `PeanoNumber` structure.
- **Requirements:**
  - A boolean flag `isZero` to indicate the base case.
  - A pointer to another `PeanoNumber` representing the successor.
- **Implementation:**
  - Write a constructor that initializes either a zero or a successor type.
  - Implement a `print()` method to display the integer value.
- **Key Concepts:** Recursive data structures, pointer ownership, and proper object construction.

Task 3: Write a Recursive Addition Function

- **Function Signature:** `PeanoNumber* add(PeanoNumber* a, PeanoNumber* b)`
- **Recursive Definition:**

$$\begin{aligned} \texttt{add}(a, 0) &= a, \\ \texttt{add}(a, S(b)) &= S(\texttt{add}(a, b)) \end{aligned}$$

- **Test Case:** Add 2 and 3; use your `print()` method to verify the result.

Task 4: Implement Multiplication

- **Function Signature:** `PeanoNumber* multiply(PeanoNumber* a, PeanoNumber* b)`
- **Recursive Definition:**

$$\begin{aligned} \texttt{multiply}(a, 0) &= 0, \\ \texttt{multiply}(a, S(b)) &= \texttt{add}(a, \texttt{multiply}(a, b)) \end{aligned}$$

- **Test Case:** Multiply 2 and 3 to produce 6.

Task 5: Wrap in a Class with Operator Overloads

- **Class Name:** `PeanoInt`
- **Requirements:**
  - Encapsulate an internal pointer to your recursive structure.
  - Implement a constructor that accepts an `int` and builds the corresponding Peano representation.
  - Provide a `toInt()` method to convert back to a native integer.
  - Overload operators: `+`, `*`, `==`, and the stream output operator (<<).
- **Design Tips:**
  - Use smart pointers (e.g., `std::shared_ptr`) or implement a custom destructor to ensure proper memory management.
  - Guard against memory leaks to ensure robust performance.

Task 6: Test Your Class

- **Implementation:** Write a `main()` function.
- **Test Cases:**
  - Validate that `2 + 3 = 5` and `2 * 3 = 6`.
  - Output results using `std::cout`.

Bonus Task: Compile-Time Peano Arithmetic

- **Objective:** Explore template metaprogramming to encode Peano numbers at compile-time.
- **Instructions:**
  - Define `struct Zero {};` and `template <typename N> struct Succ {};`.
  - Implement `Add<A, B>::result` using `typedef`.
- **Goal:** Gain insight into how compile-time logic can represent inductive reasoning, blending theory with practice.

Black Diamond: Performance / Systems Challenge

- **Benchmarking:** Compare the performance of your Peano arithmetic operations with native integer operations.
- **Optimization:**

- Implement a basic allocator for `PeanoNode` to reduce heap fragmentation.
- Optionally, integrate a counter in each node to measure recursion depth and further analyze performance.