Práctica de Básica de Recursividad de Pila en C#

Escuela de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

Algoritmos y Estructuras de Datos I

Profesores: M.Sc. Jeff Schmidt Peralta y M.Sc. Jason Leitón Jiménez y M.Sc. Leonardo Araya Martinez
Estudiante: Steven Aguilar Alvarez

Introducción

Este documento presenta una serie de ejercicios diseñados para fortalecer la comprensión y aplicación de la recursividad de pila en el lenguaje de programación C#. Los problemas se centran en la manipulación recursiva de números y operaciones básicas, permitiendo desarrollar las habilidades fundamentales en el manejo de la pila de ejecución mediante llamadas recursivas.

1. Ejercicios de Programación

1. **Extraer Unos**
   Desarrolle un método recursivo ExtractOnes que reciba un número entero largo y retorne un nuevo número formado únicamente por los dígitos que sean 1. Por ejemplo:
   - Entrada: 482401 $\rightarrow$ Salida: 1
   - Entrada: 182101 $\rightarrow$ Salida: 111
   - Entrada: 4 $\rightarrow$ Salida: 0
2. **Concatenación de Números**
   Implemente un método recursivo AppendDigits que encadene los dígitos de dos números dados. Por ejemplo:
   - AppendDigits(124, 56) $\rightarrow$ 12456
   - AppendDigits(2340, 0) $\rightarrow$ 23400
3. **Multiplicación Recursiva**
   Desarrolle un método recursivo RecursiveMultiply que implemente la multiplicación de dos números mediante sumas sucesivas:
   - RecursiveMultiply(8, 4) $\rightarrow$ 32
   - RecursiveMultiply(6.3, 3) $\rightarrow$ 18.9
4. **Suma de Dígitos**
   Implemente un método recursivo IsSumGreaterOrEqualTen que verifique si la suma de los dígitos de un número es mayor o igual a 10:
   - IsSumGreaterOrEqualTen(80642) $\rightarrow$ true
   - IsSumGreaterOrEqualTen(200412) $\rightarrow$ false
5. **Suma de Pares e Impares**
   Desarrolle un método recursivo SumEvenOddDigits que retorne una tupla con la suma de los dígitos pares e impares de un número:

- SumEvenOddDigits(482401) → (18, 1)
- SumEvenOddDigits(4) → (4, 0)

6. **Clasificación de Números**
   Implemente un método recursivo ClassifyNumber que determine si un número es deficiente, perfecto o abundante:
   - ClassifyNumber(28) → "Perfecto"
   - ClassifyNumber(30) → .ªbundante"

7. **Contención de Dígitos**
   Desarrolle un método recursivo AreDigitsContained que verifique si todos los dígitos del primer número están contenidos en el segundo:
   - AreDigitsContained(24, 42) → true
   - AreDigitsContained(333, 3) → true
   - AreDigitsContained(123, 632) → false

Observaciones Finales

Los ejercicios deben implementarse utilizando exclusivamente recursividad de pila en C#. Es fundamental mantener un control adecuado de la pila de ejecución mediante el uso apropiado de casos base y llamadas recursivas. La implementación debe seguir las convenciones de nomenclatura de C# y evitar el uso de estructuras iterativas. Los métodos auxiliares deben utilizarse cuando sea necesario para mantener la claridad y modularidad del código.