¿Qué son estructuras lineales?

Comenzaremos nuestro estudio de las estructuras de datos considerando cuatro conceptos sencillos pero muy poderosos. Las pilas, las colas, las colas dobles y las listas son ejemplos de colecciones de datos cuyos ítems se ordenan dependiendo de cómo se agregan o eliminan. Una vez se agrega un ítem, éste se mantiene en la misma posición relativa respecto a los otros ítems previos y posteriores a él. Colecciones como éstas se denominan a menudo **estructuras de datos lineales**.

Se puede pensar que las estructuras lineales tienen dos extremos. A veces, estos extremos se denominan “izquierda” y “derecha” o en algunos casos “frente” y “final”. También se les puede llamar “tope” y “fondo”. Los nombres dados a los extremos no son significativos. Lo que distingue una estructura lineal de otra es la forma en que los ítems se agregan y eliminan, en particular el lugar donde se producen estas adiciones y remociones. Por ejemplo, una estructura podría permitir que se agreguen nuevos ítems en un solo extremo. Algunas estructuras podrían permitir que los elementos se eliminen de cualquiera de los extremos.

Estas variaciones dan lugar a algunas de las estructuras de datos más útiles en ciencias de la computación. Aparecen en muchos algoritmos y pueden ser utilizadas para resolver una variedad de problemas importantes.

Ejercicios de Recursión y manejo de Archivos

En el primero se resuelven seis ejercicios que utilizan recursión de pila. La mayoría de ejercicios son de algoritmos numéricos, pero se incluye uno de manejo de listas:

1. Imprimir los números pares desde 2 hasta un entero positivo N.
2. Imprimir los números impares entre dos enteros M y N con M < N.
3. Sumar todos los números pares de una lista de enteros.
4. Contar la cantidad de dígitos de un número entero.
5. Sumar los dígitos de un número entero.
6. Invertir el orden de los dígitos de un número entero.

Resultado

1. Imprimir los números pares desde 2 hasta un entero positivo N.

```
def imprimirParesHastaN(n):
    if type(n) != int or n < 1:
        raise Exception("n debe ser entero positivo.")
    n -= n % 2
    imprimirParesHastaNAux(n)
    print()

def imprimirParesHastaNAux(n):
    if n == 0:
        return
    else:
        imprimirParesHastaNAux(n - 2)
        print(n, end = " ")
```

2 Imprimir los números impares entre dos enteros M y N con M < N.

```
def imprimirImparesEntreMyN(m, n):
    if type(m) != int:
        raise Exception("m debe ser entero positivo.")
    if type(n) != int or n <= m:
        raise Exception("n debe ser entero mayor que m.")
    m = m + 1 if m % 2 == 0 else m
    n = n - 1 if n % 2 == 0 else n
    imprimirImparesEntreMyNAux(m, n)

def imprimirImparesEntreMyNAux(m, n):
    if m > n:
        print()
    else:
        print(m, end = " ")
        imprimirImparesEntreMyNAux(m + 2, n)
```

3 Sumar todos los números pares de una lista de enteros.

```
def sumarPares(n):
    if type(n) != int or n < 3:
        raise Exception("n debe ser entero mayor que 2.")
    n -= n % 2
    return sumarParesAux(n)

def sumarParesAux(n):
    if n == 0:
        return 0
    else:
        return sumarParesAux(n - 2) + n
```

4 Contar la cantidad de dígitos de un número entero.

```
def contarDigitos(n):
    if type(n) != int or n < 0:
        raise Exception("n debe ser entero no negativo.")
    return contarDigitosAux(n)

def contarDigitosAux(n):
    if n < 10:
        return 1
    else:
        return contarDigitosAux(n // 10) + 1
```

5 Sumar los dígitos de un número entero.

```python
def sumarDigitos(n):
    if type(n) != int or n < 0:
        raise Exception("n debe ser entero no negativo.")
    return sumarDigitosAux(n)

def sumarDigitosAux(n):
    if n < 10:
        return n
    else:
        return sumarDigitosAux(n // 10) + n % 10
```

6 Invertir el orden de los dígitos de un número entero.

```python
def invertirEntero(n):
    if type(n) != int or n < 1:
        raise Exception("n debe ser entero positivo.")
    return invertirEnteroAux(n)

def invertirEnteroAux(n):
    if n < 10:
        return n
    else:
        return (n % 10) * 10**contarDigitos(n // 10)\
               + invertirEnteroAux(n // 10)
```