Implementación de una pila LIFO con listas enlazadas

Introducción

En este laboratorio, exploraremos el concepto de listas enlazadas y su aplicación en la implementación de una pila LIFO (Last-In, First-Out**)**. Las listas enlazadas son estructuras de datos fundamentales en programación que nos permiten organizar y manipular datos de manera eficiente, especialmente cuando se requiere inserción y eliminación dinámica de elementos.

Listas Enlazadas

Las listas enlazadas son estructuras de datos dinámicas que consisten en nodos enlazados mediante punteros. Cada nodo contiene un elemento de datos y un puntero que apunta al siguiente nodo en la lista. Esto permite que los elementos se enlacen de manera no contigua en memoria, lo que facilita la inserción y eliminación de elementos en cualquier posición de la lista con un tiempo de ejecución O(n).

Programación Orientada a Objetos y Clases

En programación orientada a objetos, las clases son plantillas para crear objetos que comparten características y comportamiento

Implementación de una Pila LIFO

Una pila LIFO es una estructura de datos donde el último elemento en entrar es el primero en salir. Utilizando una lista enlazada, podemos implementar fácilmente esta estructura mediante operaciones como push para añadir elementos en la parte superior de la pila y pop para eliminar el elemento superior.

Representación de la Información de los Libros

Para gestionar la información de los libros, proponemos definir una estructura de datos compuesta que contenga los atributos necesarios, como:

1. **Código**: Un identificador único para cada libro.
2. **Nombre**: El título del libro.
3. **Descripción**: Una breve sinopsis o descripción del contenido del libro.
4. **Año de Lanzamiento**: El año en que se publicó el libro.

Estos atributos se modelarían como variables dentro de una clase específica para los libros. Al crear instancias de esta clase, podemos asociar datos a cada libro de manera organizada.

Listas Enlazadas y Pila LIFO

Utilizaremos listas enlazadas para representar nuestra pila LIFO (Last-In, First-Out). Mediante esta estructura, podemos apilar libros y desapilarlos según este principio. Además, podemos consultar el libro en la parte superior de la pila sin modificar la pila misma.

Código

```python
from uuid import uuid4

class Book:
    def __init__(self, nombre, descripcion, codigo=None, año=0):
        self.nombre = nombre
        self.descripcion = descripcion
        self.codigo = uuid4()
        self.año = año

    def getNombre(self):
        return self.nombre

    def getDescripcion(self):
        return self.descripcion

    def getCodigo(self):
        return self.codigo

    def getAño(self):
        return self.año

    def setNombre(self, nombre):
        self.nombre = nombre

    def setDescripcion(self, descripcion):
        self.descripcion = descripcion

    def setCodigo(self, codigo):
        self.codigo = codigo

    def setAño(self, año):
        self.año = año

    def __str__(self):
        return f"Nombre: {self.nombre}\nDescripcion:
{self.descripcion}\nCodigo: {self.codigo}\nAño: {self.año}\n"

class Node:
    def __init__(self, value=None):
        self.value = value
        self.next = None
        self.prev = None #doble enlace
        self.setValue(value)

    def setValue(self, value):
        self.value = value

    def getValue(self):
        return self.value

    def getNext(self):
```

```
52          return self.next
53
54      def setNext(self, next):
55          if not (isinstance(next, Node) or next is None):
56              raise Exception("El tipo del atributo next no es del tipo de
57 dato esperado.")
58          self.next = next
59
60      def getPrev(self):
61          return self.prev
62      def setPrev(self, prev):
63          if not isinstance(prev, Node) or prev is None:
64              raise Exception("El tipo del nodo anterior no es del tipo de
65 dato esperado.")
66          self.prev = prev
67
68      def __str__(self):
69          return "{} - {}".format(self.value, self.next if self.next else
70 None)
71
72
73 class LinkedList:
74      def __init__(self, elements=[]):
75          self.head = None
76          self.tail = None
77          self.len = 0
78          for e in elements:
79              self.append(e)
80
81      def __len__(self):
82          return self.len
83
84      def setHead(self, node):
85          if not isinstance(node, Node):
86              raise Exception("El tipo del atributo head no es del tipo de
87 dato esperado.")
88          self.head = node
89      def setTail(self, node):
90          if not isinstance(node, Node):
91              raise Exception("El tipo del atributo tail no es del tipo de
92 dato esperado.")
93          self.tail = node
94          if self.tail is not None:
95              self.tail.setNext(None)
96
97      def append(self, value):
98          new_node = Node(value)
99          if self.isEmpty():
100             self.setHead(new_node)
101             self.setTail(new_node)
102         else:
103             tail = self.tail
```

```
104             tail.setNext(new_node)  # doble enlace
105             new_node.setPrev(tail)  # doble enlace
106             self.setTail(new_node)
107         self.len += 1
108
109     def find(self, value):
110         node_result = self.head
111         while node_result is not None and node_result.getValue() !=
112 value:
113             node_result = node_result.getNext()
114         return node_result
115
116     def update(self, old_value, new_value):
117         node_to_update = self.find(old_value)
118         if node_to_update is not None:
119             node_to_update.setValue(new_value)
120
121     def isEmpty(self):
122         return self.head is None
123
124     def __str__(self):
125         current = self.head
126         book_list = ""
127         while current:
128             book_list += str(current.value) + "\n"
129             current = current.getNext()
130         return book_list
131
132     def eliminar(self):
133         if self.isEmpty():
134             return None
135         current = self.head
136         prev = None
137         while current.getNext():
138             prev = current
139             current = current.getNext()
140         if prev:
141             prev.setNext(None)  # Elimina el enlace del penúltimo nodo al
142 último nodo
143         else:
144             self.head = None  # Si solo hay un nodo, elimina la cabeza
145         self.len -= 1
146         return self
147
148     def __str__(self):
149         current = self.head
150         book_list = ""
151         while current:
152             book_list += str(current.value) + "\n"
153             current = current.getNext()
154         return book_list
155
```

```
156 def main():
157     book1 = Book("El Señor de los Anillos", "Una aventura épica en la
158 Tierra Media", "", 1954)
159     book2 = Book("Orgullo y Prejuicio", "Una historia de amor y sociedad
160 en la Inglaterra del siglo XIX", "", 1813)
161     book3 = Book("Cien años de soledad", "Una saga familiar que abarca
162 varias generaciones en Macondo", "", 1967)
163     book4 = Book("El Principito", "Un relato poético sobre la amistad, el
164 amor y la vida", "", 1943)
165     book5 = Book("1984", "Una novela distópica sobre un futuro controlado
166 por el Gran Hermano", "", 1949)

        book_list = LinkedList()
        book_list.append(book1)
        book_list.append(book2)
        book_list.append(book3)
        book_list.append(book4)
        book_list.append(book5)
        book_list.eliminar()

        print("Lista de libros:")
        print(book_list)

    main()
```