Gestión de lista de características (features)

Un equipo de desarrollo de software gestiona una lista de características (features) pendientes por implementar. Algunas deben agregarse al principio (si son urgentes), otras al final. Las características se implementan en orden, retirándolas del frente. Dada esta naturaleza dinámica, una lista enlazada es adecuada.

El equipo en reunión general, logran definir los requerimientos del programa utilizando la siguientes historias de usuario.

Historia de usuario Nro.	1	**Título:**	Gestionar lista de características (features)
Descripción	COMO:	Líder del equipo de desarrollo	
	QUIERO:	Agregar características al principio o al final de la lista y tomar la siguiente del principio	
	PARA:	Mantener un backlog de trabajo flexible	
Criterios de aceptación	▪ Se debe poder agregar un elemento al principio (pushFront). ▪ Se debe poder agregar un elemento al final (pushBack). ▪ Se debe poder quitar y obtener el primer elemento (popFront). ▪ Si se intenta hacer popFront en una lista vacía, debe mostrarse "LISTA VACIA".		

Usted es contratado por el equipo para construir un programa en Python que cumpla las funcionalidades requeridas por el equipo teniendo como referencia las historias de usuario presentadas previamente.

Entrada	Varias líneas, cada una con un comando: ▪ PUSH_FRONT <nombre> ▪ PUSH_BACK <nombre> ▪ POP_FRONT ▪ END (finaliza la entrada)
Salida	Por cada POP_FRONT, imprimir el valor eliminado o "LISTA VACIA" si no hay elementos. Los demás comandos no generan salida.

Instrucciones para la calificación automática

Antes de enviar la solución del reto, por favor tenga en cuenta los siguientes aspectos:

- Cada caso de prueba es una secuencia de comandos terminada en END.
- El código debe procesar uno por uno.
- Debe existir una clase RetoLista con un método run.
- Solo debe imprimirse salida para los POP_FRONT.

Casos de prueba (visibles)

Entradas de ejemplo	`POP_FRONT` `END`
Salida de ejemplo	`LISTA VACIA`

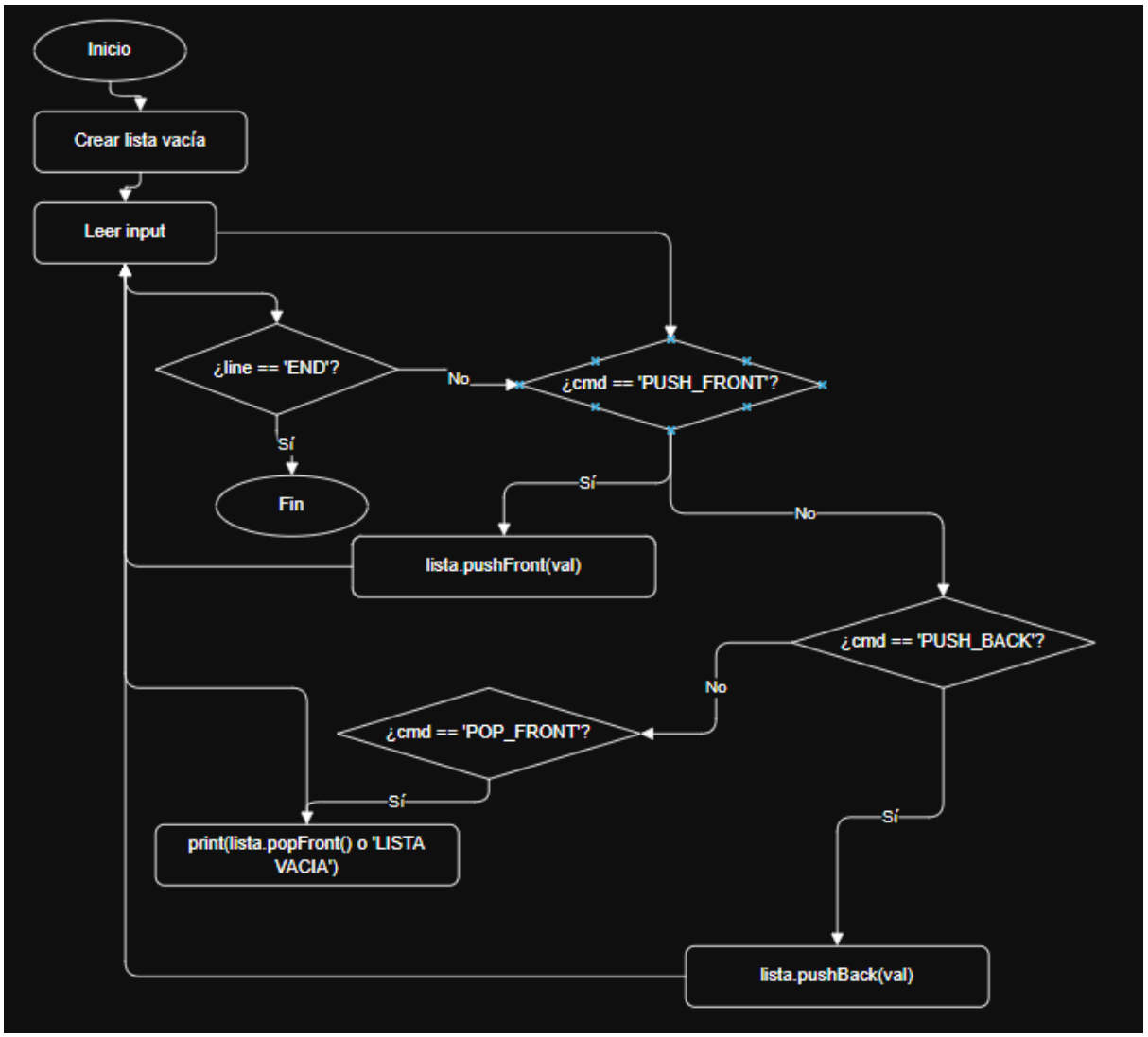
Casos de prueba (ocultos)

Caso prueba	Entrada	Salida esperada
1	`POP_FRONT` `END`	`LISTA VACIA`

2	`PUSH_FRONT F1`
	`PUSH_BACK F2`
	`POP_FRONT`
	`POP_FRONT`
	`END`
		`F1`
		`F2`

3	`PUSH_BACK A`
	`PUSH_BACK B`
	`POP_FRONT`
	`PUSH_FRONT C`
	`POP_FRONT`
	`POP_FRONT`
	`END`
		`A`
		`C`
		`B`

Diagrama de flujo: diseño



Código Python :

```python
# Clase que representa un nodo individual en la lista enlazada simple.

class Node:

    def __init__(self, value):

        # Inicializa el nodo con un valor.

        self.value = value

        # Inicializa el puntero al siguiente nodo como None.

        self.next = None
```

```
# Clase que implementa una lista enlazada simple.
class SimpleLinkedList:
    def __init__(self):
        # Inicializa la cabeza (primer nodo) y la cola (último nodo) de la lista como None.
        self.head = None
        self.tail = None

    # Método para agregar un nodo al frente de la lista.
    def pushFront(self, value):
        # Crea un nuevo nodo con el valor dado.
        new_node = Node(value)
        # Establece el 'next' del nuevo nodo para que apunte a la cabeza actual.
        new_node.next = self.head
        # Actualiza la cabeza de la lista para que sea el nuevo nodo.
        self.head = new_node
           # Si la lista estaba vacía (la cola era None), la cola también se convierte en el nuevo nodo.
        if not self.tail:
            self.tail = new_node

    # Método para agregar un nodo al final de la lista.
    def pushBack(self, value):
        # Crea un nuevo nodo con el valor dado.
        new_node = Node(value)
```

```
        # Si la lista está vacía (la cola es None), el nuevo nodo se convierte en la cabeza y la cola.
        if not self.tail:
            self.head = new_node
            self.tail = new_node
        else:
            # Si la lista no está vacía, el 'next' de la cola actual apunta al nuevo nodo.
            self.tail.next = new_node
            # Actualiza la cola de la lista para que sea el nuevo nodo.
            self.tail = new_node

    # Método para eliminar y devolver el nodo del frente de la lista.
    def popFront(self):
        # Si la lista está vacía (la cabeza es None), devuelve None.
        if not self.head:
            return None
        # Guarda el valor del nodo de la cabeza antes de eliminarlo.
        value = self.head.value
        # Mueve la cabeza al siguiente nodo en la lista.
        self.head = self.head.next
        # Si después de mover la cabeza, la cabeza se vuelve None (la lista queda vacía), la cola también se establece a None.
        if not self.head:
            self.tail = None
        # Devuelve el valor del nodo que fue eliminado.
```

```
        return value


# Lógica principal para el "Reto Lista".
class RetoLista:
    def run(self):
        # Crea una instancia de la lista enlazada simple.
        lista = SimpleLinkedList()
        # Bucle principal para leer la entrada del usuario.
        while True:
            try:
                    # Lee una línea de entrada, elimina espacios en blanco al principio y al final.
                line = input().strip()
                # Si la línea es "END", sale del bucle.
                if line == "END":
                    break
                  # Si la línea comienza con "PUSH_FRONT", agrega un elemento al frente.
                if line.startswith("PUSH_FRONT"):
                    # Divide la línea para obtener el valor a agregar.
                    _, value = line.split(maxsplit=1)
                    lista.pushFront(value)
                  # Si la línea comienza con "PUSH_BACK", agrega un elemento al final.
                elif line.startswith("PUSH_BACK"):
                    # Divide la línea para obtener el valor a agregar.
```

```
                    _, value = line.split(maxsplit=1)
                    lista.pushBack(value)
                # Si la línea es "POP_FRONT", elimina y muestra el elemento del frente.
                elif line == "POP_FRONT":
                    # Llama al método popFront para obtener el resultado.
                    result = lista.popFront()
                    # Imprime el resultado o "LISTA VACIA" si no hay elementos.
                    print(result if result else "LISTA VACIA")
            # Maneja el error de fin de archivo (EOFError) para salir del bucle si la entrada termina inesperadamente.
            except EOFError:
                break

if __name__ == "__main__":
    RetoLista().run()
```

Ejecución de los casos de prueba

```
POP_FRONT
LISTA VACIA
END
```

```
if __name__ == "__main__":
    RetoLista().run()
```

```
PUSH_FRONT F1
PUSH_BACK F2
POP_FRONT
F1
POP_FRONT
F2
END
```

```
if __name__ == "__main__":
    RetoLista().run()
```

```
PUSH_BACK A
PUSH_BACK B
POP_FRONT
A
PUSH_FRONT C
POP_FRONT
C
POP_FRONT
B
end
```