Задача 4 (Односвязный список)

Односвязный список — классический пример структуры данных, для реализации которой нужно пользоваться памятью в куче. В этой задаче вам нужно реализовать шаблон класса LinkedList, который представляет собой односвязный список и обладает следующим интерфейсом:

```
1  template <typename T>
2  class LinkedList {
3  public:
4  struct Node {
5  T value;
6  Node* next = nullptr;
7  };

8  ~LinkedList();

9  void PushFront(const T& value);
10 void InsertAfter(Node* node, const T& value);
11 void RemoveAfter(Node* node);
12 void PopFront();

13 Node* GetHead() { return head; }
14 const Node* GetHead() const { return head; }

15 private:
16 Node* head = nullptr;
17 };
```

- Метод GetHead возвращает указатель на голову списка, он используется для перебора элементов списка (см. шаблон ToVector в заготовке решения)
- Метод PushFront добавляет новый элемент в голову списка.
- Метод InsertAfter вставляет новый элемент в список так, чтобы он шёл после узла node. Если node == nullptr, метод эквивалентен PushFront
- Метод PopFront удаляет элемент из головы списка и освобождает выделенную под него память. Если список пуст, метод завершается корректно. Если после выполнения метода список стал пустым, метод GetHead должен возвращать nullptr
- Метод RemoveAfter должен удалять из списка элемент, который следует за узлом node, и освобождать выделенную под него память. Если node == nullptr, метод эквивалентен PopFront. Если node->next == nullptr, то метод должен корректно завершаться.
- Все методы, перечисленные выше, должны работать за O(1)
- Деструктор класса LinkedList освобождает всю память, выделенную для хранения элементов списка.

Файл с заготовкой решения

Файл: linked_list.cpp