

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №2**  
**по дисциплине «Информатика»**  
**Тема: Алгоритмы и структуры данных в Python.Тест**

Студент гр. 3341

Анисимов Д. А.

Преподаватель

Иванов Д.В.

Санкт-Петербург

2024

## **Цель работы**

Цель данного задания заключается в создании связанного однонаправленного списка через два взаимосвязанных класса: Node и LinkedList.

Задачи включают:

1. Создание класса Node, который описывает элемент списка с данными и ссылкой на следующий элемент, а также методами для инициализации, доступа к данным и удобного вывода информации об объекте.

2. Создание класса LinkedList, который представляет собой связанный однонаправленный список с головным элементом и количеством элементов. Необходимо реализовать методы инициализации, определения длины списка, вывода списка в строковом виде, добавления элемента в конец, очистки списка и удаления элемента по его порядковому номеру с конца списка.

## Задание

Node

Класс, который описывает элемент списка.

Он должен иметь 2 поля:

- o `data`    # Данные элемента списка, приватное поле.
  
- o `next`     # Ссылка на следующий элемент списка.

И следующие методы:

- o `__init__(self, data, next)` - конструктор, у которого значения по умолчанию для аргумента `next` равно `None`.
  
- o `get_data(self)` - метод возвращает значение поля `data` (это необходимо, потому что в идеале пользователь класса не должен трогать поля класса `Node`).
  
- o `__str__(self)` - перегрузка стандартного метода `__str__`, который преобразует объект в строковое представление. Для данной лабораторной необходимо реализовать следующий формат перевода объекта класса `Node` в строку:

“data: <node\_data>, next: <node\_next>”,

где <node\_data> - это значение поля `data` объекта `Node`, <node\_next> - это значение поля `next` объекта, на который мы ссылаемся, если он есть, иначе `None`.

Пример того, как должен выглядеть результат реализации `__str__` см. ниже.

Пример того, как должен выглядеть вывод объекта:

```
node = Node(1)
```

```
print(node) # data: 1, next: None
```

```
node.next = Node(2, None)
```

```
print(node) # data: 1, next: 2
```

## Linked List

Класс, который описывает связный однонаправленный список.

Он должен иметь 2 поля:

- o head      # Данные первого элемента списка.
- o length    # Количество элементов в списке.

И следующие методы:

- o `__init__(self, head)` - конструктор, у которого значения по умолчанию для аргумента head равно None.

- Если значение переменной head равно None, метод должен создавать пустой список.

- Если значение head не равно None, необходимо создать список из одного элемента.

- o `__len__(self)` - перегрузка метода `__len__`, он должен возвращать длину списка (этот стандартный метод, например, используется в функции `len`).

- о `append(self, element)` - добавление элемента в конец списка. Метод должен создать объект класса `Node`, у которого значение поля `data` будет равно `element` и добавить этот объект в конец списка.

- о `__str__(self)` - перегрузка стандартного метода `__str__`, который преобразует объект в строковое представление. Для данной лабораторной необходимо реализовать следующий формат перевода объекта класса однонаправленного списка в строку:

- Если список пустой, то строковое представление:

`"LinkedList[]"`

- Если не пустой, то формат представления следующий:

`"LinkedList[length = <len>, [data:<first_node>.data, next:  
<first_node>.data; data:<second_node>.data, next:<second_node>.data; ... ;  
data:<last_node>.data, next: <last_node>.data]"`,

где `<len>` - длина связного списка, `<first_node>`, `<second_node>`, `<third_node>`, ... , `<last_node>` - элементы однонаправленного списка.

Пример того, как должен выглядеть результат реализации см. ниже.

- о `pop(self)` - удаление последнего элемента. Метод должен выбрасывать исключение `IndexError` с сообщением `"LinkedList is empty!"`, если список пустой.

- о `clear(self)` - очищение списка.

o `delete_on_end(self, n)` - удаление n-того элемента с конца списка. Метод должен выбрасывать исключение `KeyError`, с сообщением "Element doesn't exist!", если количество элементов меньше n.

## **Основные теоретические положения**

Связанный список (LinkedList) представляет собой структуру данных, где каждый узел содержит данные и ссылку на следующий узел в списке. Основные характеристики LinkedList:

1. Динамичность: LinkedList может динамически изменяться при добавлении или удалении элементов благодаря ссылкам между узлами.

2. Быстрые операции добавления и удаления: Вставка и удаление элементов в LinkedList имеют постоянную сложность  $O(1)$ , что делает их эффективнее, чем в массиве.

3. Медленный доступ к элементам: Для доступа к конкретному элементу в LinkedList необходимо пройти по всем предшествующим элементам, что делает доступ к элементам более медленным по сравнению с массивом.

4. Не непрерывное распределение памяти: Узлы LinkedList могут быть разбросаны по памяти, поэтому они не обязательно хранятся последовательно.

5. Применение: LinkedList хорошо подходит для частых операций добавления и удаления элементов, когда доступ по индексу не является основным критерием.

## Выполнение работы

1. Создается класс Node, который представляет узел в связанном списке. Узел содержит данные и ссылку на следующий узел.

2. Создается класс LinkedList, который представляет сам связанный список. В конструкторе инициализируется головной узел (head) и длина списка (length).

3. Метод append(element) добавляет новый узел с данными element в конец списка. Если список пустой, новый узел становится головным. Иначе происходит обход списка до последнего узла и новый узел добавляется в конец.

4. Метод str() возвращает строковое представление списка. Происходит обход списка и формируется строка, содержащая данные каждого узла и ссылку на следующий узел.

5. Метод pop() удаляет последний элемент из списка и возвращает его значение. Если список пустой, выбрасывается исключение IndexError.

6. Метод clear() очищает список, устанавливая головной узел в None и длину списка в 0.

7. Метод delete\_on\_end(n) удаляет n-й элемент с конца списка. Если n равно длине списка, удаляется головной узел. Иначе происходит обход списка до удаляемого элемента и устанавливается ссылка на следующий узел после него.

Для реализации бинарного поиска в линейном списке необходимо, чтобы элементы были отсортированы. Нельзя обратиться к элементу по индексу напрямую, как в обычном списке. Для доступа к элементам нужно итерироваться между ними. Реализация алгоритма будет подобна, но нужно добавить переменную для хранения информации о текущем элементе списка, который рассматривается.



## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<pre> node = Node(1)  print(node) # data: 1, next: None  node.next = Node(2, None) print(node) # data: 1, next: 2 print(node.get_data()) l_1 = LinkedList() print(l_1) print(len(l_1)) l_1.append(10) l_1.append(20) l_1.append(30) l_1.append(40) print(l_1)  print(len(l_1))  l_1.delete_on_end(3) print(l_1) </pre>	<pre> data: 1, next: None data: 1, next: 2 1 LinkedList[] 0 LinkedList[length = 4, [data: 10, next: 20; data: 20, next: 30; data: 30, next: 40; data: 40, next: None]] 4 LinkedList[length = 3, [data: 10, next: 30; data: 30, next: 40; data: 40, next: None]]1 </pre>	Проверка работы основных методов класса

## **Выводы**

После выполнения данного задания были реализованы два класса - Node и Linked List. Класс Node представляет элемент списка, с полями data и next, а также методами для получения данных и для преобразования объекта в строку. Класс Linked List представляет связный однонаправленный список с полями head и length, а также методами для добавления элемента, получения длины списка, удаления последнего элемента, очистки списка и удаления n-того элемента с конца списка. Оба класса были реализованы согласно указанным требованиям, что позволит эффективно работать с однонаправленным списком.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Node:

    def __init__(self, data, next=None):
        self.data = data
        self.next = next

    def get_data(self):
        return self.data

    def __str__(self):
        next_data = self.next.data if self.next else None
        return f"data: {self.data}, next: {next_data}"

class LinkedList:

    def __init__(self, head=None):
        self.head = head
        self.length = 0 if head is None else 1

    def __len__(self):
        return self.length

    def append(self, element):
        new_node = Node(element)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
        self.length += 1

    def __str__(self):
        if self.head is None:
            return "LinkedList[]"
        else:
            current = self.head
            result = "LinkedList[length = {}, {}".format(self.length)
            while current:
                result += f"data: {current.data}, next: "
                result += f"{current.next.data if current.next else None}; "
                current = current.next
            result = result[:-2] + "]"
            return result

    def pop(self):
        if self.head is None:
            raise IndexError("LinkedList is empty!")
        current = self.head
```

```

    if current.next is None:
        self.head = None
        self.length -= 1
        return current.data
    while current.next.next:
        current = current.next
    popped_data = current.next.data
    current.next = None
    self.length -= 1
    return popped_data

def clear(self):
    self.head = None
    self.length = 0

def delete_on_end(self, n):
    if (n < 1 or self.length < n):
        raise KeyError("<element> doesn't exist!")

    if (n == self.length):
        self.head = self.head.next
        self.length -= 1
        return

    current = self.head
    for i in range(self.length-n-1):
        current = current.next
    current.next = current.next.next
    self.length -= 1

```