**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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Корниенко А. Е. |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Изучить алгоритмы и структуры данных и их реализацию на языке Python. С их помощью написать программу, создающую однонаправленный список.

## Задание

Node

Класс, который описывает элемент списка.

Он должен иметь 2 поля:

* data # Данные элемента списка, приватное поле.
* next # Ссылка на следующий элемент списка.

И следующие методы:

1) \_\_init\_\_(self, data, next) - конструктор, у которого значения по умолчанию для аргумента next равно None.

2) get\_data(self) - метод возвращает значение поля data (это необходимо, потому что в идеале пользователь класса не должен трогать поля класса Node).

3) \_\_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.

Linked List

Класс, который описывает связный однонаправленный список.

Он должен иметь 2 поля:

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

И следующие методы:

1) \_\_init\_\_(self, head) - конструктор, у которого значения по умолчанию для аргумента head равно None.

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

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

2) \_\_len\_\_(self) - перегрузка метода \_\_len\_\_, он должен возвращать длину списка (этот стандартный метод, например, используется в функции len).

3) 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.

4) \_\_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.

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

6) clear(self) - очищение списка.

7) delete\_on\_start(self, n) - удаление n-того элемента с НАЧАЛА списка. Метод должен выбрасывать исключение KeyError, с сообщением "Element doesn't exist!", если количество элементов меньше n.

## Выполнение работы

Node:

1) \_\_init\_\_ - Создается два поля экземпляра self.\_\_data - приватное поле, хранившее данные; self.next - ссылка на следующий элемент класса Node.

2) get\_data – геттер для self.\_\_data.

3) \_\_str\_\_ - Вывод строкового представления экземпляра в соответсвии с шаблоном.

LinkedList:

1) \_\_init\_\_ - констуктор, если передаётся Node элемент, то он становится головой, иначе создаёт пустой LinkedList.

2) \_\_len\_\_ - Вывод длины списка.

3) append - Добавление нового элемента класса Node.

4) \_\_str\_\_ - Строковое представление списка в соответсвии с шаблоном. Циклом происходит проход по элементам и добавление к итоговое строке описание каждого элемента.

5) pop - Удаление последнего элемента из списка.

6) delete\_on\_start - По аналогии с pop. Только идет удаление по значению элемента.

7) clear - Очищение списка, используется pop().

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |
| --- | --- | --- |
| № п/п | Входные данные | Выходные данные |
| 1. | linked\_list = LinkedList()  print(linked\_list)  print(len(linked\_list))  linked\_list.append(10)  print(linked\_list)  print(len(linked\_list))  linked\_list.append(20)  print(linked\_list)  print(len(linked\_list))  linked\_list.pop()  print(linked\_list)  print(len(linked\_list)) | LinkedList[]  0  LinkedList[length = 1, [data: 10, next: None]]  1  LinkedList[length = 2, [data: 10, next:20; data: 20, next: None]]  2  LinkedList[length = 1, [data: 10, next: None]]  1 |

## Выводы

Была разработана программа, содержащая классы элемента однонаправленного списка и сам список. Написаны методы для каждого из них и протестирована их работа.

# 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):

if self.next is None:

return f"data: {self.\_\_data}, next: None"

else:

return f"data: {self.\_\_data}, next: {self.next.get\_data()}"

class LinkedList:

def \_\_init\_\_(self, head=None):

self.head = None

if head is not None:

self.length = 1

self.head = Node(head)

else:

self.length = 0

def \_\_len\_\_(self):

return self.length

def append(self, element):

if self.length != 0:

self.length += 1

node = Node(element)

current = self.head

while current.next is not None:

current = current.next

current.next = node

else:

self.length += 1

self.head = Node(element)

def \_\_str\_\_(self):

if self.length == 0:

return "LinkedList[]"

else:

string = f"LinkedList[length = {self.length}, ["

current = self.head

while current is not None:

if current.next is None:

string += f"data: {current.get\_data()}, next: None; "

else:

string += f"data: {current.get\_data()}, next: {current.next.get\_data()}; "

current = current.next

string = string[:len(string) - 2]

string += "]]"

return string

def pop(self):

if self.length == 0:

raise IndexError("LinkedList is empty!")

else:

if self.head.next is not None:

self.length -= 1

current = self.head

while current.next.next is not None:

current = current.next

current.next = None

else:

self.head = None

self.length = 0

def delete\_on\_start(self, n):

if self.length < n or n <= 0:

raise KeyError("Element doesn't exist!")

elif n == self.length:

self.pop()

elif n == 1:

self.length -= 1

self.head = self.head.next

else:

self.length -= 1

current = self.head

for i in range(n - 2):

current = current.next

current.next = current.next.next

def clear(self):

while self.length != 0:

self.pop()