

Практическая работа №4.

Тема: «Структуры данных «линейные списки».

Цель работы: изучить СД типа «линейный список», научиться их программно реализовывать и использовать.

Многочлен $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ с целыми коэффициентами можно представить в виде списка, причем если $a_i = 0$, то соответствующее звено не включать в список. Необходимо вывести на экран значения элементов многочлена при случайных значениях a .

Для реализации линейного списка сначала определим структуру данных для узла этого списка. Класс Node представлен на листинге 1.

Листинг 1. Класс Node.

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

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next

    def set_next(self, next):
        self.next = next
```

Функция для добавления элемента в начало списка представлена на листинге 2.

Листинг 2. Функция для добавления элемента в начало списка.

```
def push(self, value):
    temp = Node(value)
    temp.set_next(self.head)
```

					<i>AuСД.09.03.02.230000 ПР</i>									
Изм.	Лист	№ докум.	Подпись	Дат	Практическая работа №4 «Структуры данных «линейны списки».					Лит.	Лист	Листов		
Разраб.		Туманов Е.М.										2		
Провер.		Береза А.Н.								ИСОиП (филиал) ДГТУ в г.Шахты ИСТ-Тб21				
Реценз														
Н. Контр.														
Утверд.														

```
self.head = temp
```

Функция для вставки элемента в конец списка представлена на листинге 3. Диаграмма деятельности для этой функции представлена на рисунке 1.

Листинг 3. Функция для добавления элемента в конец списка.

```
def append(self, new_data):
    new_node = Node(new_data)
    if self.head is None:
        self.head = new_node
        return
    last = self.head
    while last.next:
        last = last.get_next()
    last.set_next(new_node)
```

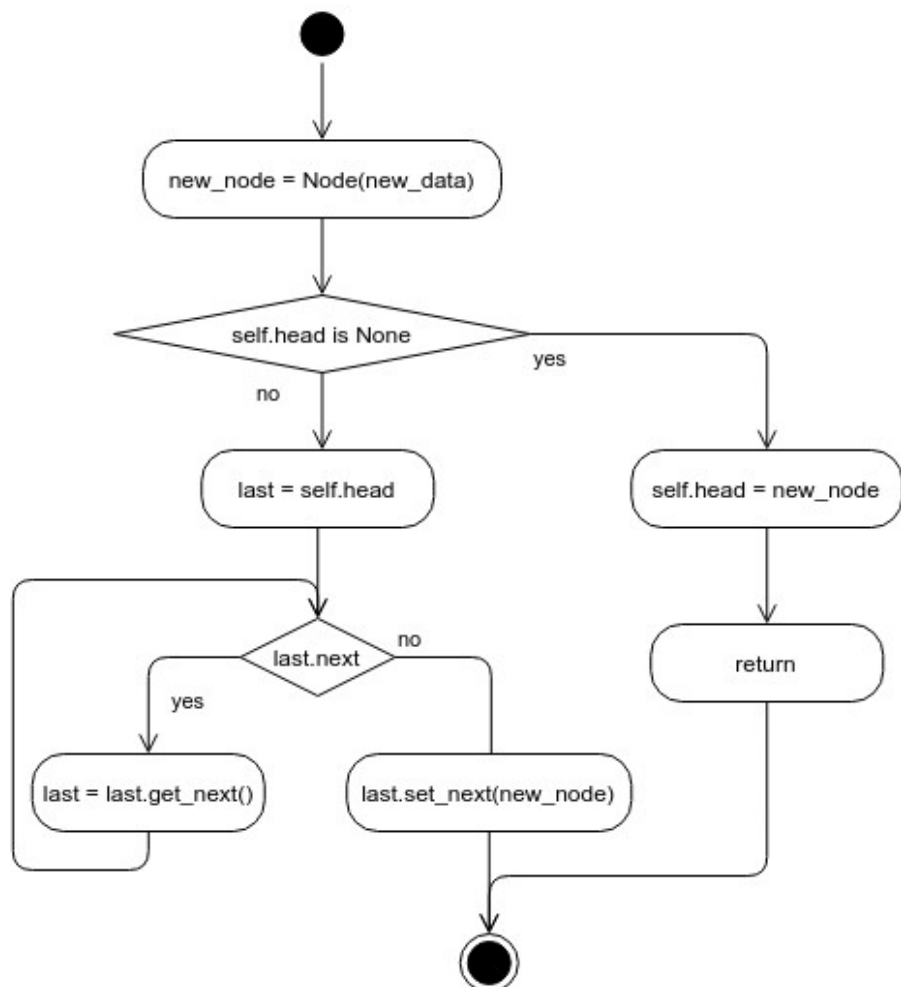


Рисунок 1. Диаграмма деятельности для функции добавления элемента в конец списка.

Изм.	Лист	№ докум.	Подпись	Дата

АИСД.09.03.02.230000 ПР

Функция для добавления элемента в произвольное место списка представлена на листинге 4. Диаграмма деятельности для этой функции представлена на рисунке 2.

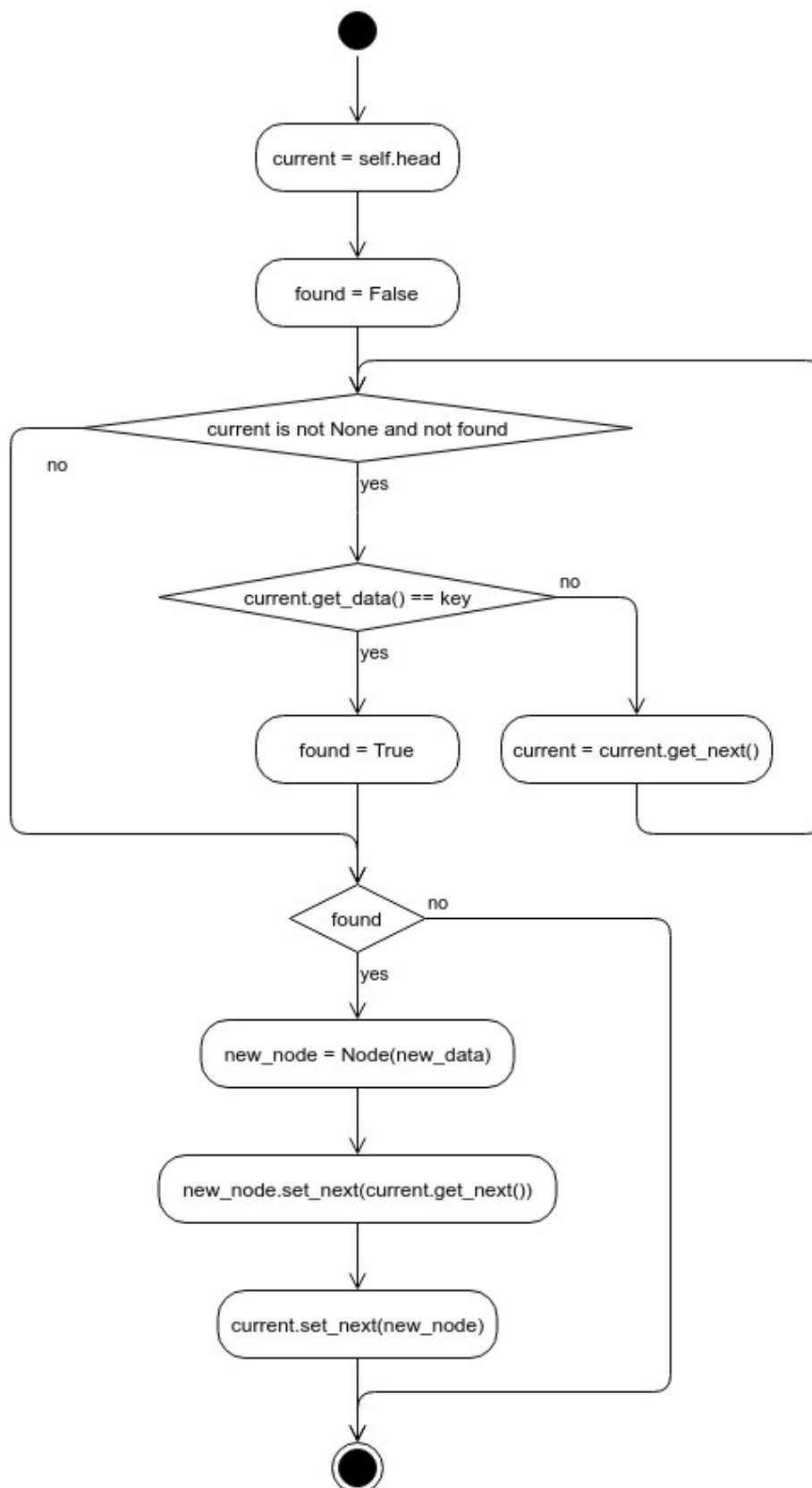


Рисунок 2. Диаграмма деятельности для функции добавления элемента в произвольное место списка.

Листинг 4. Функция добавления элемента в произвольное место списка.

```
def insert_after(self, key, new_data):
    current = self.head
    found = False
    while current is not None and not found:
        if current.get_data() == key:
            found = True
        else:
            current = current.get_next()
    if found:
        new_node = Node(new_data)
        new_node.set_next(current.get_next())
        current.set_next(new_node)
```

Функция для получения длины списка представлена на листинге 5.

Листинг 5. Функция для получения длины списка.

```
def length(self):
    current = self.head
    count = 0
    while current is not None:
        count += 1
        current = current.get_next()
    return count
```

Функция для поиска элемента в списке представлена на листинге 6.

Диаграмма деятельности для этой функции представлена на рисунке 3.

Листинг 6. Функция поиска элемента в списке.

```
def search(self, key):
    current = self.head
    found = False
    while current is not None and not found:
        if current.get_data() == key:
            found = True
        else :
            current = current.get_next()
    return found
```

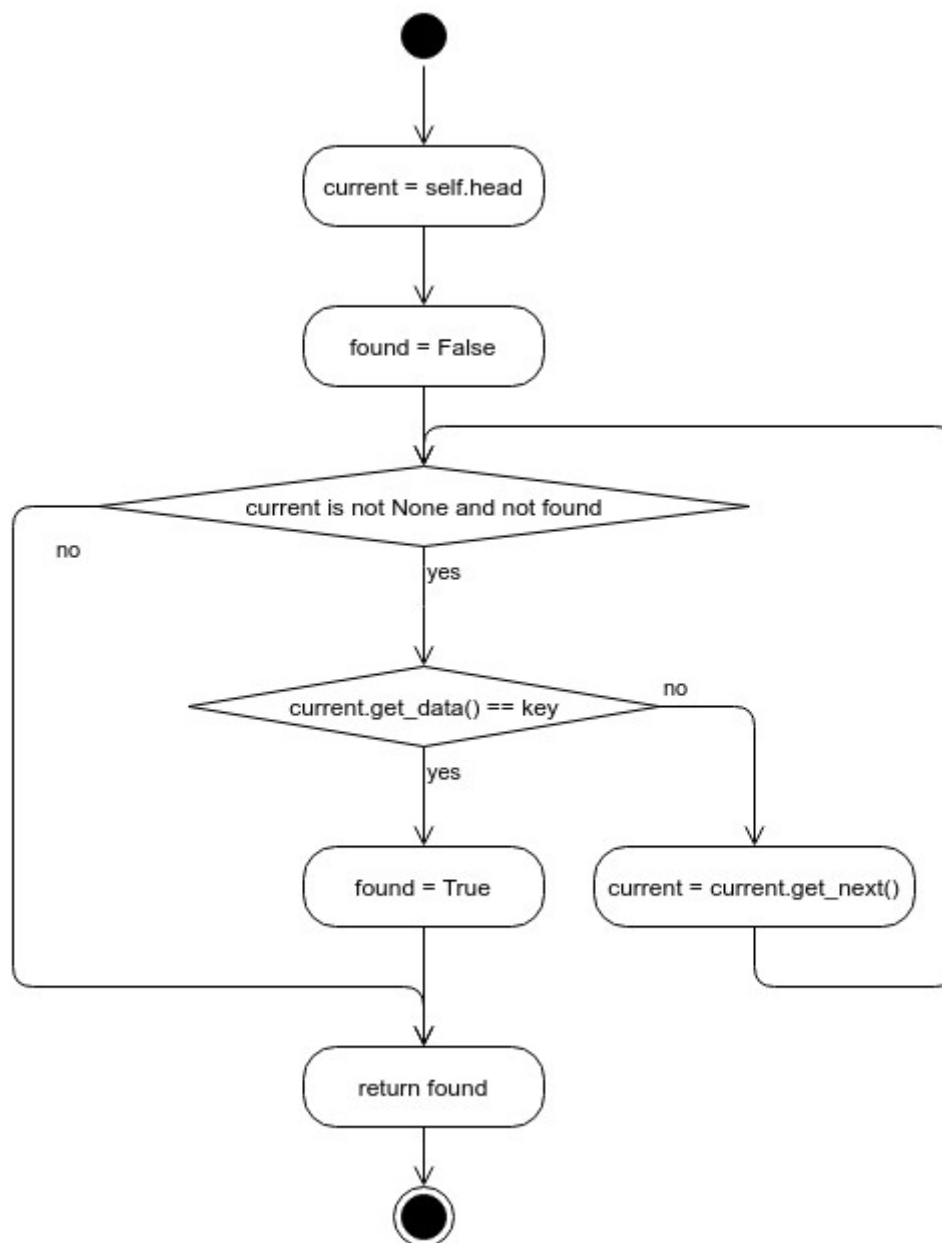


Рисунок 3. Диаграмма деятельности для функции поиска элемента в списке.

Функция для удаления элемента списка представлена на листинге 7. Диаграмма деятельности для этой функции представлена на рисунке 4.

Листинг 7. Функция для удаления элемента списка.

```

def delete_node(self, key):
    current = self.head
    previous = None
    found = False
    while not found:
        if current.get_data() == key:
            found = True
        else:

```

```

        previous = current
        current = current.get_next()
    if previous is None:
        self.head = current.get_next()
    else:
        previous.set_next(current.get_next())

```

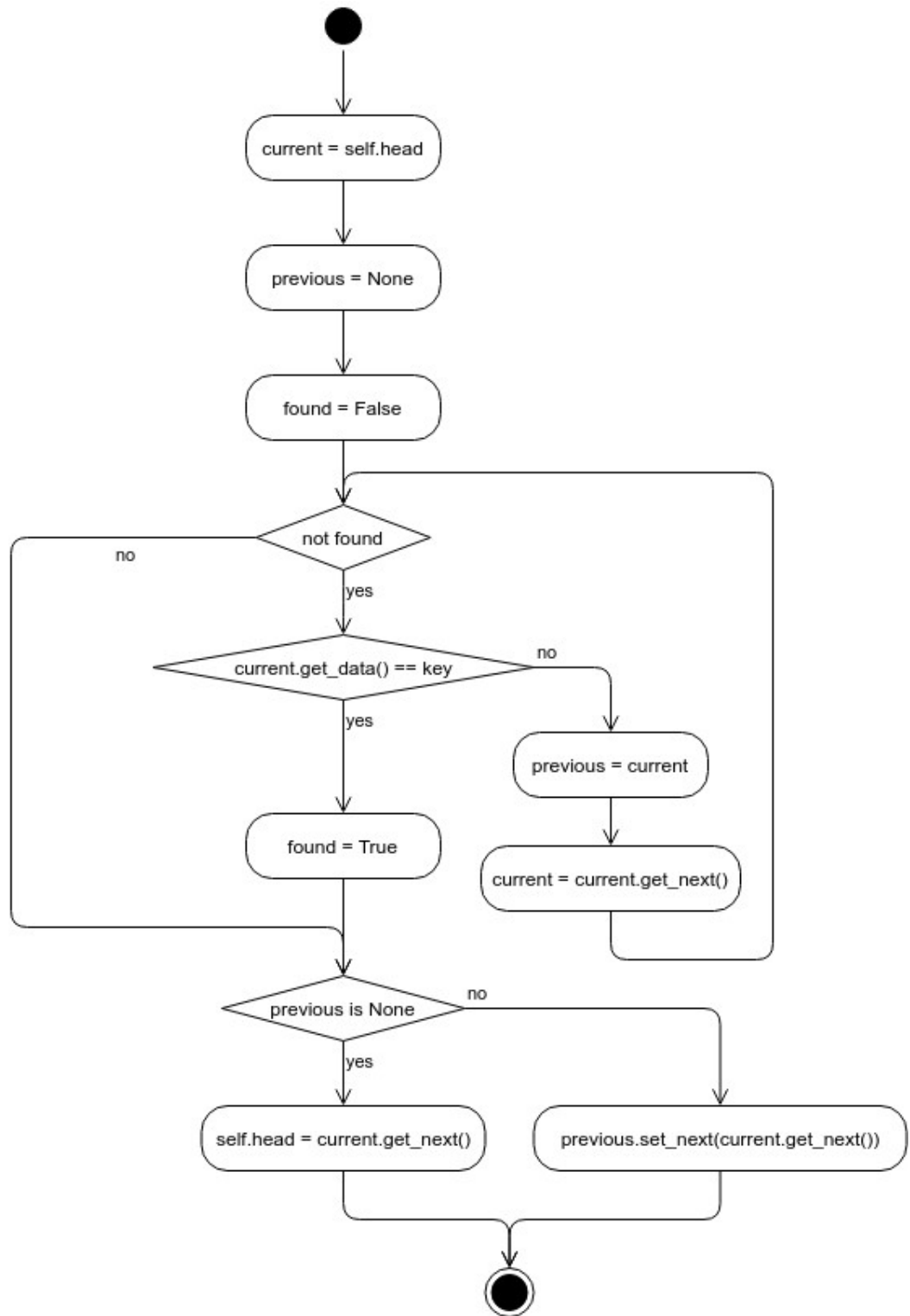


Рисунок 4. Диаграмма деятельности для функции удаления элемента списка.

Полностью исходный код для класса MyLinkedList представлены на листинге 8.

Листинг 8. Полный исходный код класса MyLinkedList.

```
class MyLinkedList:
    def __init__(self):
        self.head = None

    def __str__(self):
        if self.head is not None:
            current = self.head
            out = "[" + str(current.get_data())
            while current.get_next() is not None:
                current = current.get_next()
                out += "," + " " + str(current.get_data())
            return out + "]"

    def push(self, value):
        temp = Node(value)
        temp.set_next(self.head)
        self.head = temp

    def append(self, new_data):
        new_node = Node(new_data)
        if self.head is None:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.get_next()
        last.set_next(new_node)

    def insert_after(self, key, new_data):
        current = self.head
        found = False
        while current is not None and not found:
```

```

        if current.get_data() == key:
            found = True
        else:
            current = current.get_next()
    if found:
        new_node = Node(new_data)
        new_node.set_next(current.get_next())
        current.set_next(new_node)

def length(self):
    current = self.head
    count = 0
    while current is not None:
        count += 1
        current = current.get_next()
    return count

def search(self, key):
    current = self.head
    found = False
    while current is not None and not found:
        if current.get_data() == key:
            found = True
        else :
            current = current.get_next()
    return found

def delete_node(self, key):
    current = self.head
    previous = None
    found = False
    while not found:
        if current.get_data() == key:
            found = True
        else:
            previous = current

```



```

        current = current.get_next()
    if previous is None:
        self.head = current.get_next()
    else:
        previous.set_next(current.get_next())

```

Исходный код программы для вывода значений элементов многочлена представлен на листинге 9.

Листинг 9. Исходный код программы для выполнения задания.

```

from Linked_list.linked_list import MyLinkedList
import random as rd

polynomial = MyLinkedList()

def polynom(x, n):
    for i in range(n, -1, -1):
        p = rd.randint(0, 10) * x ** i
        if p != 0:
            polynomial.append(p)
    return polynomial

print(polynom(2, 10))

```

Вывод: в ходе выполнения данной практической работы была реализована структура данных линейный список, и выполнено индивидуальное задание.