Практическая работа №4

Тема: «Структуры данных «линейные списки».

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

Многочлен $P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ с целыми коэффициентами можно представить в виде списка, причем если $a_i = 0$, то соответствующее звено не включать в список. Определить логическую функцию РАВНО (p, q), проверяющие на равенство многочлены p, q.

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
```

					АиСД.09.03.02.050000 ПР		
Изм.	*Лист*	*№ докум.*	*Подпись*	*Дат*			
Разраб.		*Благородов И.*			Практическая работа №4 «Структуры данных «линейны списки».	*Лит.*	*Лист* 2 / *Листов*
Провер.		*Береза А.Н.*					
Реценз						*ИСОиП (филиал) ДГТУ в г.Шахты ИСТ-Tb21*	
Н. Контр.							
Утверд.							

Класс LinkedList представлен на листинге 2.

Листинг 2. Класс LinkedList.

```
class LinkedList:
    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 + "]"
```

Функция для добавления элемента в начало списка представлена на листинге 3.

Листинг 3. Функция для добавления элемента в начало списка.

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

Функция для вставки элемента в конец списка представлена на листинге 4.

Листинг 4. Функция для вставки 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)
```

Функция для добавления элемента в произвольное место списка представлена на листинге 5.

Листинг 5. 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 6.

Листинг 6. 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 7.

Листинг 7. 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
```

Функция для удаления элемента списка представлена на листинге 8.

Листинг 8. 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())
```

Полностью исходный код программы представлен на листинге 9.

Листинг 9. Полный исходный программы.

```
import random as rd


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


class LinkedList:
    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())


list1 = LinkedList()
list2 = LinkedList()


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


def compare(list1, list2):
    compare = []
    clist1 = list1.head
    clist2 = list2.head
    if list1.length() == list2.length():
        while clist1.get_next() is not None:
            if clist1.get_data() == clist2.get_data():
                compare.append(True)
            else:
                compare.append(False)
            clist1 = clist1.get_next()
            clist2 = clist2.get_next()
        return compare
    else:
        return False


polynom(2, 10, list1)
polynom(2, 10, list2)
```

```
print(list1)
print(list2)
print(compare(list1, list2))
```

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