

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

Высшего профессионального образования

**«Ижевский государственный технический университет имени
М.Т.Калашникова»**

(ФГБОУ ВПО ИжГТУ имени М.Т.Калашникова)

Лабораторная работа №2

по дисциплине «Алгоритмы и структуры данных»

Линейные структуры данных

Выполнил: студент гр. Б01-191-1зт

Шайхиев А.Ф.

Проверил(а): _____

Еланцев М.О.

Постановка задачи:

Реализовать две структуры данных на любом языке программирования, согласно варианту(2 вариант) (двунаправленный список и стек) .

Двунаправленный список:

```
#класс Node для определения элемента списка
class Node:
    def __init__(self, value = None, next = None, prev = None ):
        self.value = value
        self.next = next
        self.prev = prev

#класс Лист покупок
class ShopList:
    def __init__(self):
        self.head = None
        self.tail = None
        self.length = 0

    #вставить элемент в конец списка
    def add(self, x):
        self.length+=1
        if self.head == None:
            self.tail = self.head = Node(x)
        else:
            self.tail.next = self.tail = Node(x)

    #получить элемент по индексу
    def getElemByIndex(self, index):
        if index < self.length:
            checkBox = self.head
            count = 0
            while count <= index:
                if count == index:
                    print(checkBox.value)
                count += 1
                checkBox = checkBox.next

    #вставка элемента перед заданным индексом
    def addElemBeforeIndex(self, x, index):
        if self.head == None:
            self.tail = self.head = Node(x)
        if index < self.length:
            checkBox=self.head
            count = 0
            while count <= index:
                count += 1
                if count == index:
                    checkBox.next = Node(x, checkBox.next, checkBox.prev)
            checkBox = checkBox.next

    #удаление элемента по значению
    def delByName(self, x):
```

```
        checkBox = self.head
        if checkBox is not None:
            if checkBox.value == x:
                self.head = checkBox.next
                checkBox = None
                print('True')
                return
        while checkBox is not None:
            if checkBox.value == x:
                break
            last = checkBox
            checkBox = checkBox.next
        if checkBox == None:
            print('False')
            return
        last.next = checkBox.next
        checkBox = None
```

#Вывести список на печать

```
def printList(self):
    if self.head != None:
        current = self.head
        out = 'Список \n' + current.value + '\n'
        while current.next != None:
            current = current.next
            out += current.value + '\n'
        print(out)
```

#освобождение памяти

```
def clear(self):
    self.__init__()
```

```
newList = ShopList()
```

Описание работы программы:

Создадим список покупок, и выполним с ним следующие операции:

- 1) Вставка элемента в конец (целочисленное значение) Вставка должна корректно обрабатывать случай, когда список пуст (указатель на первый узел пуст)
- 2) Вставка элемента перед заданным индексом
- 3) Получение значения элемента по индексу
- 4) Удаление элемента по значению В случае, если элемент с ключом не найден, функция должна вернуть false, иначе - true
- 5) Печать всех элементов списка
- 6) Освобождение памяти от структуры данных

```
#класс Node для определения элемента списка
class Node:
    def __init__(self, value = None, next = None, prev = None ):
        self.value = value
        self.next = next
        self.prev = prev
```

Класс Node создает ячейку списка, "value" – ее значение, "prev" и "next" – индексы элементов перед ней и после, если перед или после ячейки не идет другой ячейки, то индекс = None.

```
#класс Лист покупок
class ShopList:
    def __init__(self):
        self.head = None
        self.tail = None
        self.length = 0
```

Класс "ShopList" – наш основной класс(сам список покупок), "head" и "tail" означают последний и первый элементы нашего списка(__init__ выступает в роли конструктора класса).

```
#вставить элемент в конец списка
def add(self, x):
    self.length+=1
    if self.head == None:
        self.tail = self.head = Node(x)
    else:
        self.tail.next = self.tail = Node(x)
```

Функция "add" принимает в аргумент значение, которое надо вставить в список, и ставит его на последнее место в этом списке.

```

#вставка элемента перед заданным индексом
def addElemBeforeIndex(self, x, index):
    if self.head == None:
        self.tail = self.head = Node(x)
    if index < self.length:
        checkBox=self.head
        count = 0
        while count <= index:
            count += 1
            if count == index:
                checkBox.next = Node(x, checkBox.next, checkBox.prev)
            checkBox = checkBox.next

```

Функция “addElemBeforeIndex” принимает в качестве аргументов 2 значения: значение, которое надо добавить и индекс. Если список пустой то добавленное значение будет первым и единственным.

```

#получить элемент по индексу
def getElemByIndex(self, index):
    if index < self.length:
        checkBox = self.head
        count = 0
        while count <= index:
            if count == index:
                print(checkBox.value)
            count += 1
            checkBox = checkBox.next

```

Функция “getElemByIndex” принимает в качестве аргумента индекс, и перебирает весь список, пока не дойдет до нужного индекса, тогда и выводит значение.

```

#удаление элемента по значению
def delByName(self, x):
    checkBox = self.head
    if checkBox is not None:
        if checkBox.value == x:
            self.head = checkBox.next
            checkBox = None
            print('True')
            return
    while checkBox is not None:
        if checkBox.value == x:
            break
        last = checkBox
        checkBox = checkBox.next
    if checkBox == None:
        print('False')
        return
    last.next = checkBox.next
    checkBox = None

```

Функция “delByName” принимает в качестве аргумента значение, которое нужно удалить. Если в списке присутствует это значение, то оно удаляется и в качестве параметра next для значения которое шло перед ним встает значение, которое шло за удаленным, а значение после удаленного параметра “prev” стает равно значению которое шло до удаленного.

```
#вывести список на печать
def printList(self):
    if self.head != None:
        current = self.head
        out = 'Список \n' + current.value + '\n'
        while current.next != None:
            current = current.next
            out += current.value + '\n'
        print(out)
```

Функция “printList” выводит на печать все элементы списка

```
#освобождение памяти
def clear(self):
    self.__init__()
```

Функция “clear” удаляет список

Пример работы программы:

```
print("создаем список для школы\n")
newList = ShopList()#создаем пустой список продуктов
newList.add('pen')# добавим в него ручку
newList.add('lastik')# добавим в него ластик
newList.add('note')# добавим в него тетрадь
newList.printList()#Выведем наш список на печать

print("добавим на 2 место в списке линейку\n")
newList.addElemBeforeIndex('ruler', 2)#добавим перед 2 элементом в списке линейку
newList.printList()#Выведем наш список на печать

print("выведем элемент с индексом 1\n")
newList.getElemByIndex(1)

print("Удалим ручку и линейку из списка\n")
newList.delByName('pen')
newList.delByName('ruler')
newList.printList()#Выведем наш список на печать

print("Попробуем удалить несуществующий элемент\n")
newList.delByName('apple')
```

```
print("Полностью очистим наш список и добавим новый элемент\n")
newList.clear()#удаляем список
newList.add('potato')
newList.printList()
```

```
создаем список для школы

pen
lastik
note

добавим на 2 место в списке линейку

pen
lastik
ruler
note

выведем элемент с индексом 1

lastik

Удалим ручку и линейку из списка

True

lastik
note

Попробуем удалить несуществующий элемент

False

Полностью очистим наш список и добавим новый элемент

potato
```


Стек:

Стек - абстрактный тип данных, представляющий собой список элементов, организованных по принципу *LIFO* (англ. *last in — first out*, «последним пришёл — первым вышел»).

В Python стек можно реализовать с помощью списков.

```
stack = []#инициализация стека

#вставка элементов на вершину стека
stack.append(1)
stack.append(2)
stack.append(3)

print(stack)

#клонирование списка

sec_stack = stack[:]

#извлечение элементов с вершины стека
length = len(stack)
while length > -2:
    if length <= 0:
        print("false\n")
        break
    else:
        stack.pop()
        print("true\n")
    length -= 1

print(sec_stack)

#удаление стека
sec_stack.clear()
print(sec_stack)
```

Задаем пустой стек “stack” добавляем в него элементы, с помощью `.append` (append добавляет элемент в конец списка), как мы знаем в стек элементы могут добавляться только в конец стека.

Клонируем стек путем присваивания его в другую переменную (если бы мы не указали `[:]`, то переменная бы просто содержала адрес ячейки памяти, где хранится наш первый стек, в случае изменения основного изменился бы и тот, который мы присвоили в переменную).

Элементы из стека мы можем извлекать только сверху вниз, для этого там идеально подойдет функция `.pop()`, она удаляет последний элемент и присваивает его в другую переменную, если та задана. В коде в условие цикла стоит пока длина стека меньше чем `-2`, для того чтоб показать, что если в стеке не осталось элементов то выйдет `false` и цикл закончится.

Удаление стека происходит с помощью `.clear()`.

Результат работы программы:

```
[1, 2, 3]
true

true

true

false

[1, 2, 3]
[]
```

Мы записали числа в стек, затем скопировали его в другую переменную, из первого стека по очереди убрали все элементы, когда все элементы были убраны - сообщение `false`. В стеке, который мы присвоили в другую переменную, ничего не удалилось, затем мы удаляем все из второго стека тоже.