|  |
| --- |
| МИНОБРНАУКИ РОССИИ  Федеральное государственное  бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий

Кафедра корпоративных информационных систем

|  |  |  |
| --- | --- | --- |
| **ОТЧЕТ**  **по лабораторной работе №1** | | |
| **по дисциплине** | | |
| **«Структуры и алгоритмы обработки данных»**  **Тема лабораторной работы: «**Структуры данных: список, очередь, стек**»** | | |
| Студент группы | ИКБО-07-18 | Зейналов М.Г. |
| Принял | ассистент кафедры КИС | Габриелян Г.А. |
|  |  |  |
| Выполнено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись студента)* |
| Зачтено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись преподавателя)* |

Москва 2019

1. **Задача №1**
   1. **Постановка задачи**

Сформировать линейный однонаправленный список, элементами которого являются строки. Составить программу, подсчитывающую количество строк в списке, которые начинаются с того же символа, что и строка в последнем элементе списка.

* 1. **Описание используемых структур данных**

Линейный однонаправленный список – структура данных, состоящая из узлов, каждый из которых содержит как собственные значения, так и ссылку на следующий элемент.

* 1. **Пользовательский интерфейс**

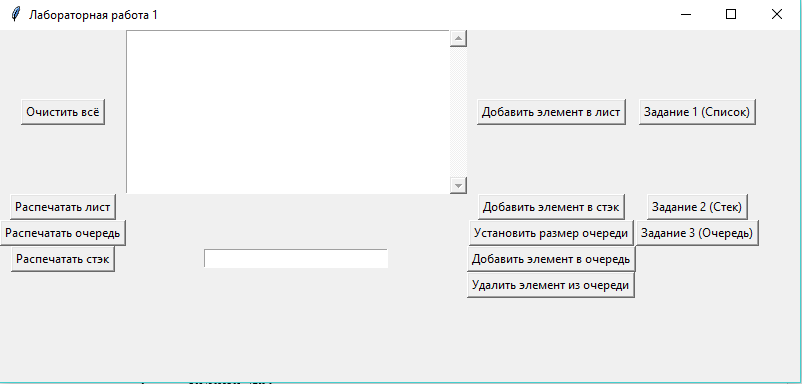


Рисунок 1 – «Пользовательский интерфейс»

* 1. **Описание алгоритма**

Создается первый класс Node, который имеет 2 поля: значение и ссылка на следующий элемент. Дальше создается класс List, в котором прописываем разные операции с помощью первого класса. Например, рассмотрим метод добавления append(). Сначала мы проверяем список на пустоту, в случае удовлетворения условию, присваиваем голове первый Node, если список не пустой, то в поле next присваиваем новый Node.

Удаление через pop() проверяем на длину, если меньше 1, то можем просто вернуть пустой список, иначе пока Node предпустой, проходим до конца. Как доходим до предпоследнего, обнуляем у него значение на следующий элемент.

Добавление из других структур данных осуществляется путем перебора элементов, и поэлементного добавления в лист.

Также реализованы системные классы (\_\_len\_\_(), \_\_getitem\_\_(), \_\_str\_\_) для удобства использования структуры.

* 1. **Тестирование**

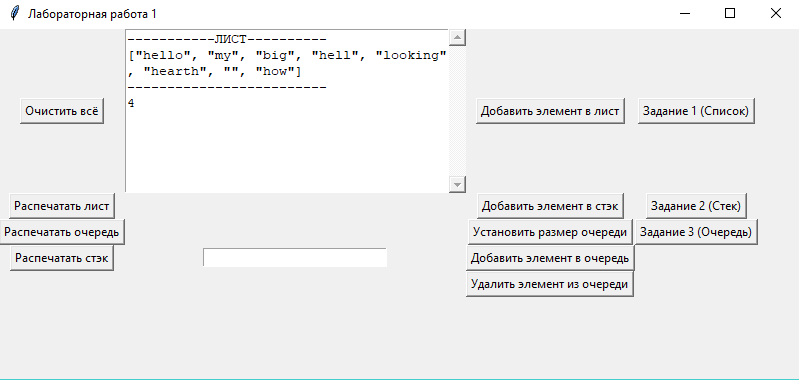


Рисунок 2 – «Тестирование задания с ЛОС»

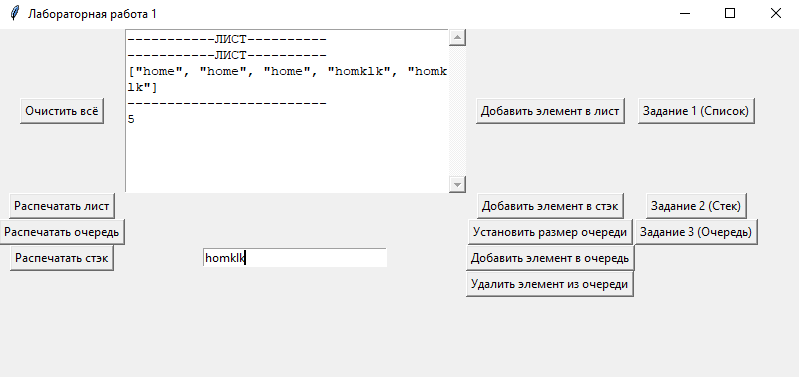


Рисунок 3 – «Тестирование действий с ЛОС»

* 1. **Листинг программы**

class Node:  
 def \_\_init\_\_(self, dat):  
 self.value = dat  
 self.next = None  
  
  
class List:  
 def \_\_init\_\_(self):  
 self.head = None  
  
 def append(self, node):  
 if self.head is None: # if seq is empty  
 self.head = Node(node)  
 else: # if is not => next = new item  
 cur = self.head  
 while cur.next:  
 cur = cur.next  
 cur.next = Node(node)  
  
 def append\_first(self, value):  
 node = Node(value)  
 node.next = self.head  
 self.head = node  
  
 def length(self):  
 cur = self.head  
 count = 0  
 while cur is not None:  
 count += 1  
 cur = cur.next  
 return count  
  
 def pop(self): # deleting. for future in lab  
 cur = self.head  
 if List.length(self) == 1:  
 cur = None  
 return  
 while cur.next.next is not None:  
 cur = cur.next  
 cur.next = None  
  
 def append\_from(self, sequence): # append from others sequences (list, ...)  
 for item in sequence:  
 List.append(self, item)  
  
 def \_\_str\_\_(self): # system func for printing list  
 l = "["  
 cur = self.head  
 l += '"' + str(cur.value) + '", '  
 cur = cur.next  
 while cur is not None:  
 l += '"' + str(cur.value) + '", '  
 cur = cur.next  
 l = l[0:-2] # deleting ", " in the end  
 l += "]"  
 return l  
  
 def \_\_len\_\_(self): # for item in sequence does not work without this func:(. updЖ does not work with this thing too  
 return List.length(self)  
  
 def \_\_getitem\_\_(self, key): # system class for accessing to item. for ex. "a[1]"  
 length = 0  
 cur1 = None  
 cur = self.head  
 while key != length and cur.next is not None:  
 cur = cur.next  
 length += 1  
 if key == length:  
 cur1 = cur.value  
 return cur1

def strings\_count():  
 count = 0  
 if len(arr[len(arr) - 1]) != 0:  
 char = arr[len(arr) - 1][0]  
 else:  
 return count  
  
 for i in range(len(arr)):  
 # description: we add one if our element is not an !!! empty string(lambda) !!!  
 # or return the same count if the condition is not met  
 # lambda: we check is string empty, cause otherwise it can fall  
 count = count + 1 if char == (lambda mas: mas[0] if len(mas) != 0 else "")(arr[i]) else count # IT WORKS AAAAAA  
  
 return count

1. **Задача №2**
   1. **Постановка задачи**

Сформируйте исходный стек, элементами которого являются целые числа. Составить программу, которая находит сумму элементов, находящихся в стеке

* 1. **Описание используемых структур данных**

Стек (от англ. stack — стопка) — структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. Притом первым из стека удаляется элемент, который был помещен туда последним, то есть в стеке реализуется стратегия «последним вошел — первым вышел» (last-in, first-out — LIFO). Примером стека в реальной жизни может являться стопка тарелок: когда мы хотим вытащить тарелку, мы должны снять все тарелки выше.

* 1. **Пользовательский интерфейс**

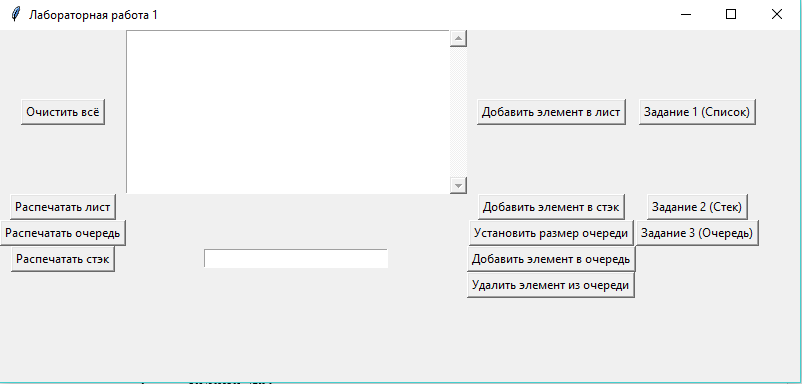


Рисунок 4 – «Пользовательский интерфейс»

* 1. **Описание алгоритма**

Стек основан на линейном списке. Дописаны методы push() и pop().

Для выполнения задания необходимо пройти по стеку и сложить все значения.

* 1. **Тестирование**

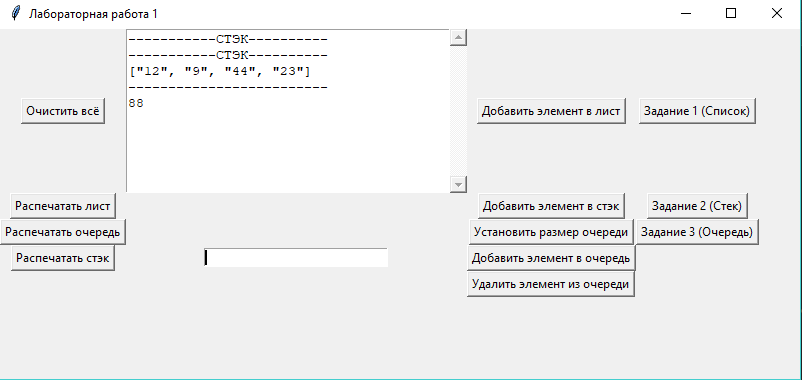


Рисунок 5 – «Тестирование стека»

* 1. **Листинг программы**

from LinkedList import List as ls  
  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.val = ls()  
  
 def push(self, value):  
 self.val.append(value)  
  
 def pop(self):  
 return self.val.pop()  
  
 def \_\_str\_\_(self):  
 return self.val.\_\_str\_\_()  
  
 def \_\_len\_\_(self):  
 return self.val.length()  
  
 def \_\_getitem\_\_(self, key):  
 return self.val.\_\_getitem\_\_(key)

def stack\_sum():  
 sum = 0  
 for i in range(len(st)):  
 sum += int(st[i])  
 return sum

1. **Задача №3**
   1. **Постановка задачи**

Составить процедуру формирования очереди, при этом необходимо учесть, что очередь может содержать не более K элементов (переполнение очереди) и не может быть пустой.

* 1. **Описание используемых структур данных**

Очередь (англ. queue) - это структура данных, добавление и удаление элементов в которой происходит путём операций push и pop соответственно. Притом первым из очереди удаляется элемент, который был помещен туда первым, то есть в очереди реализуется принцип «первым вошел — первым вышел» (англ. first-in, first-out — FIFO). У очереди имеется голова (англ. head) и хвост (англ. tail). Когда элемент ставится в очередь, он занимает место в её хвосте. Из очереди всегда выводится элемент, который находится в ее голове.

* 1. **Пользовательский интерфейс**

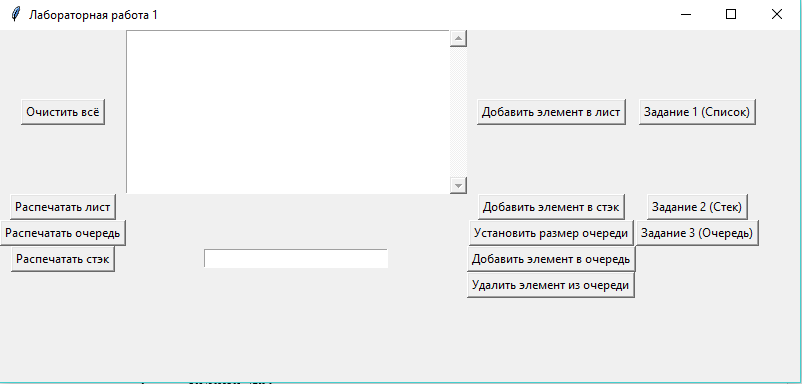


Рисунок 6 – «Пользовательский интерфейс»

* 1. **Описание алгоритма**

Очередь основана на списке. Для инициализации необходимо задать размер, в случае некорректных значений будет выдано исключение. При добавлении мы проверяем длину очереди, если она больше заданной, то будет переполнение, т.е. вытеснено правое значение. При удалении мы также проверяем длину, но в данном случае, длина должна быть больше 1.

* 1. **Тестирование**

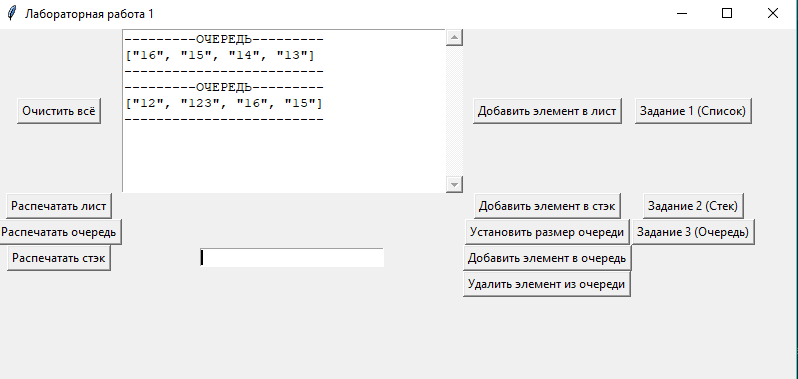


Рисунок 7 – «Тестирование очереди»

* 1. **Листинг программы**

1. from LinkedList import List as ls  
     
     
   class Queue:  
    def \_\_init\_\_(self, item, nec\_size):  
    self.queue = ls()  
    if nec\_size < 1:  
    raise ValueError("k should be >= 1")  
    self.nec\_size = nec\_size # k  
    self.queue.append(item)  
     
    def enqueue(self, item):  
    length = self.queue.length()  
    if length >= self.nec\_size:  
    self.queue.append\_first(item)  
    self.queue.pop()  
    else:  
    self.queue.append\_first(item)  
     
    def dequeue(self):  
    if self.queue.length() - 1 <= 0:  
    raise ValueError("Queue can not be empty")  
    return self.queue.pop()  
     
    def \_\_len\_\_(self):  
    return self.queue.length()  
     
    def \_\_str\_\_(self):  
    return self.queue.\_\_str\_\_()