|  |
| --- |
| 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-08-18 | Валяев Д.А. |
| 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 целые числа. Составить программу, которая находит сумму последнего и предпоследнего элементов списка.

* 1. **Описание используемых структур данных**

Линейный однонаправленный список – это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на NULL. Элемент, на который нет указателя, является первым (головным) элементом списка.

class Node:  
 def \_\_init\_\_(self, value=None):  
 self.value = value  
 self.next = None  
  
  
class List:  
 def \_\_init\_\_(self):  
 self.head = Node()

* 1. **Пользовательский интерфейс**

Пользовательский интерфейс содержит: label – для вывода суммы 2-х последний элементов, list – для просмотра содержимого списка, entry – для ввода элементов, которые необходимо добавить, кнопка “Add” - для добавления нового элемента, кнопка “Sum” – для расчета суммы 2-х последних элементов.

* 1. **Описание алгоритма**

При нажатии на кнопку “Sum”, программа в цикле while будет присваивать временной переменной значение текущей ячейки, если следующий элемент не равен None, таким образом во временной переменной будет записан предпоследний элемент списка. Если следующий элемент списка равен None, то значение текущего элемент будет сохранено во второй временной переменной, сложение временных переменных на выходе даст сумму 2–х последний элементов списка.

* 1. **Тестирование**

На вход программы последовательно поступают числа: 1, 1, 6, 55, 2

Теоретический результат работы программы: Сумма: 57

Практический результат работы программы: Сумма: 57

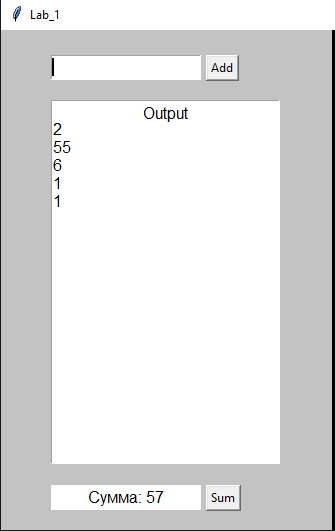


Рисунок 1.1 – Тестирование ЛОС

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

Составить процедуру проверки того, имеет ли очередь узел с информационным полем, содержащим целое число, задаваемое пользователем. При формировании очереди необходимо учесть, что она может содержать не более K элементов (переполнение очереди) и не может быть пустой.

* 1. **Описание используемых структур данных**

Очередь - структура данных, из которой удаляется первым тот элемент, который был первым в очередь добавлен. То есть очередь в программировании соответствует «бытовому» понятию очереди. Очередь также называют структурой типа FIFO (first in, first out – первым пришел, первым ушел).

class Queue:  
 def \_\_init\_\_(self):  
 self.item = []

* 1. **Пользовательский интерфейс**

Пользовательский интерфейс содержит: label – для вывода результата поиска, list – для просмотра содержимого очереди, entry – для ввода количества элементов K, entry – для ввода элементов, которые необходимо добавить, кнопка “Add” - для добавления нового элемента, кнопка “Search” – для поиска элемента в очереди, кнопка “Set” – для установки количества элементов K.

* 1. **Описание алгоритма**

При нажатии на кнопку “Search”, программа проверяет, не является ли очередь пустой и не является ли поле ввода пустым, далее программа с помощью метода “get” забирает каждый элемент из очереди, присваивает временной переменной, проверяет не равна ли временная переменная числу, находящемуся в поле ввода и добавляет временную переменную во вторую очередь. Поскольку при использовании метода “get” элемент очереди удаляется, по завершению поиска, элементы из второй очереди, добавляются обратно в первую, результат выводится в label.

* 1. **Тестирование**

На вход программы поступают случайные числа от 0 до 30.

Число для поиска: 4

Теоретический результат работы программы: Число найдено

Практический результат работы программы: Число найдено

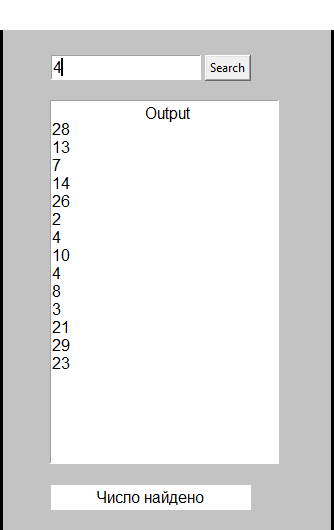


Рисунок 1.2 – тестирование очереди

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

Сформируйте исходный стек, элементами которого являются целые числа. Составить программу, которая добавляет в вершину стека S элемент, находящийся в его конце.

* 1. **Описание используемых структур данных**

Стек – структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека.

class Stack:  
 def \_\_init\_\_(self):  
 self.items = []

* 1. **Пользовательский интерфейс**

Пользовательский интерфейс содержит: list – для просмотра содержимого стека, entry – для ввода элементов, которые необходимо добавить, кнопка “Add” - для добавления нового элемента, кнопка “Replace” – для перемещения элемента из конца на вершину.

* 1. **Описание алгоритма**

При нажатии на кнопку “Replace”, программа проверяет не является ли стек пустым, если стек не пустой, программа присваивает каждый элемент стека временной переменной с помощью метода “pop” и добавляет ее во второй стек. Когда первый стек полностью перемещен во второй, с помощью метода “pop” берется верхний элемент и присваивается временной переменной. Далее второй стек с помощью метода “pop” перемещается в первый, когда работа по перемещению будет окончена, на вершину первого стека добавляется элемент, сохраненный во временной переменной.

* 1. **Тестирование**

На вход программы последовательно поступают числа: 45, 7, 7, 8, 23, 12, 9, 0, 65, 77, 6

Теоретический результат работы программы: 7, 7, 8, 23, 12, 9, 0, 65, 77, 6, 45

Практический результат работы программы: 7, 7, 8, 23, 12, 9, 0, 65, 77, 6, 45

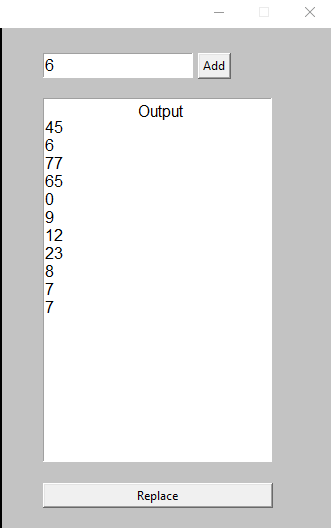


Рисунок 1.3 – тестирование стека

* 1. **Листинг программы**

import tkinter as tk  
import random  
  
count = 0  
count2 = 0  
temp = 0  
root = tk.Tk()  
background\_image = tk.PhotoImage(file='font.png')  
background\_label = tk.Label(root, image=background\_image)  
background\_label.place(relwidth=1, relheight=1)  
  
  
class Node:  
 def \_\_init\_\_(self, value=None):  
 self.value = value  
 self.next = None  
  
  
class List:  
 def \_\_init\_\_(self):  
 self.head = Node()  
  
 def add(self, value):  
 new\_Node = Node(value)  
 cur = self.head  
 while cur.next is not None:  
 cur = cur.next  
 cur.next = new\_Node  
  
 elem = []  
 cur\_node = self.head  
 while cur\_node.next is not None:  
 cur\_node = cur\_node.next  
 elem.append(cur\_node.value)  
  
 def summary(self):  
 cur = self.head  
 while cur.next is not None:  
 temp23 = cur.value  
 cur = cur.next  
 if cur.next is None:  
 temp32 = cur.value  
 u = int(temp23) + int(temp32)  
 return u  
  
  
class Queue:  
 def \_\_init\_\_(self):  
 self.item = []  
 for i in range(1, 15):  
 x = random.randint(0, 30)  
 self.push(x)  
 text\_queue.configure(state="normal")  
 text\_queue.insert(0.0, x)  
 text\_queue.insert(0.0, '\n')  
 text\_queue.configure(state="disabled")  
  
 def push(self, item):  
 self.item.append(item)  
  
 def pop(self):  
 return self.item.pop(0)  
  
 def check(self):  
 check = False  
 x = int(S\_queue.get())  
 for i in range(0, len(self.item)):  
 if self.item[i] == x:  
 check = True  
 return check  
  
  
class Stack:  
 def \_\_init\_\_(self):  
 self.items = []  
  
 def isEmpty(self):  
 return self.items == []  
  
 def push(self, item):  
 self.items.append(item)  
  
 def pop(self):  
 return self.items.pop()  
  
  
# region Functions  
def add\_num(entry):  
 if len(entry) != 0:  
 list\_1.add(entry)  
 text\_list.configure(state="normal")  
 text\_list.insert(0.0, entry)  
 text\_list.insert(0.0, '\n')  
 text\_list.configure(state="disabled")  
  
  
def Sum():  
 y = list\_1.summary()  
 label\_list.configure(text="Сумма: " + str(y))  
  
  
def Search(entry):  
 if len(entry) != 0:  
 if queue\_1.check() == True:  
 label\_queue.configure(text="Число найдено")  
 else:  
 label\_queue.configure(text="Число не найдено")  
  
  
def FillStack(entry):  
 global count2  
 if len(entry) != 0:  
 stack\_1.push(entry)  
 text\_stack.configure(state="normal")  
 text\_stack.insert(0.0, entry)  
 text\_stack.insert(0.0, '\n')  
 text\_stack.configure(state="disabled")  
 count2 += 1  
  
  
def Replace():  
 global temp  
 if not stack\_1.isEmpty():  
 text\_stack.config(state="normal")  
 text\_stack.delete(1.0, tk.END)  
 while not stack\_1.isEmpty():  
 u = stack\_1.pop()  
 stack\_2.push(u)  
 temp = stack\_2.pop()  
 while not stack\_2.isEmpty():  
 u = stack\_2.pop()  
 stack\_1.push(u)  
 text\_stack.insert(0.0, u)  
 text\_stack.insert(0.0, '\n')  
 stack\_1.push(temp)  
 text\_stack.insert(0.0, temp)  
 text\_stack.insert(0.0, '\n')  
 text\_stack.config(state="disabled")  
  
  
# endregion  
  
# region UI list  
text\_list = tk.Text(root, font=12, width=25, height=20)  
text\_list.configure(state="disabled")  
text\_list.place(relx=0.05, rely=0.14)  
  
label\_name = tk.Label(root, font=12, text="Output", fg='black', bg='white')  
label\_name.place(relx=0.09, rely=0.141, relwidth=0.15, relheight=0.05)  
  
label\_list = tk.Label(root, font=12, fg='black', bg='white')  
label\_list.place(relx=0.05, rely=0.91, relwidth=0.15, relheight=0.05)  
  
entry\_list = tk.Entry(root, font=12)  
entry\_list.place(relx=0.05, rely=0.05, relwidth=0.15, relheight=0.05)  
  
button\_add = tk.Button(root, text="Add", command=lambda: add\_num(entry\_list.get()))  
button\_add.place(relx=0.205, rely=0.05)  
  
button\_check = tk.Button(root, text="Sum", command=lambda: Sum())  
button\_check.place(relx=0.205, rely=0.91)  
# endregion  
# region UI queue  
  
text\_queue = tk.Text(root, font=12, width=25, height=20)  
text\_queue.configure(state="disabled")  
text\_queue.place(relx=0.383, rely=0.14)  
  
label\_name\_q = tk.Label(root, font=12, text="Output", fg='black', bg='white')  
label\_name\_q.place(relx=0.426, rely=0.141, relwidth=0.15, relheight=0.05)  
  
S\_queue = tk.Entry(root, font=12)  
S\_queue.place(relx=0.384, rely=0.05, relwidth=0.15, relheight=0.05)  
  
label\_queue = tk.Label(root, font=12, fg='black', bg='white')  
label\_queue.place(relx=0.384, rely=0.91, relwidth=0.2, relheight=0.05)  
  
button\_Search = tk.Button(root, text="Search", command=lambda: Search(S\_queue.get()))  
button\_Search.place(relx=0.538, rely=0.05)  
# endregion  
# region UI stack  
entry\_stack = tk.Entry(root, font=12)  
entry\_stack.place(relx=0.710, rely=0.05, relwidth=0.15, relheight=0.05)  
  
button\_stack = tk.Button(root, text="Add", command=lambda: FillStack(entry\_stack.get()))  
button\_stack.place(relx=0.865, rely=0.05)  
  
text\_stack = tk.Text(root, font=12, width=25, height=20)  
text\_stack.configure(state="disabled")  
text\_stack.place(relx=0.710, rely=0.14)  
  
label\_name\_s = tk.Label(root, font=12, text="Output", fg='black', bg='white')  
label\_name\_s.place(relx=0.753, rely=0.141, relwidth=0.15, relheight=0.05)  
  
button\_Replace = tk.Button(root, text="Replace", command=lambda: Replace())  
button\_Replace.place(relx=0.710, rely=0.91, relwidth=0.23, relheight=0.05)  
# endregion  
queue\_1 = Queue()  
list\_1 = List()  
stack\_1 = Stack()  
stack\_2 = Stack()  
if \_\_name\_\_ == "\_\_main\_\_":  
 root.title("Lab\_1")  
 root.geometry("1000x500")  
 root.resizable(False, False)  
 root.mainloop()