|  |
| --- |
| 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-10-18 | Минаев А. И. |
| 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, next = None):

self.value = value

self.next = next

class LOS:

def \_\_init\_\_(self):

self.first = None

self.last = None

self.length = 0

def add(self, x):

self.length += 1

if self.first == None:

self.first = self.last = Node(x, None)

else:

self.last.next = self.last = Node(x, None)

* 1. **Пользовательский интерфейс**

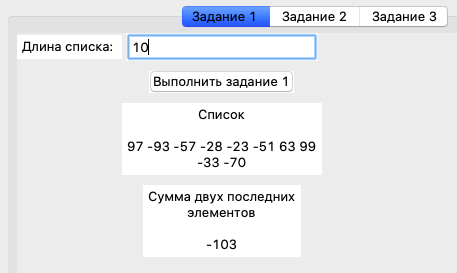


Рис. 1 – графический интерфейс для задания 1

* 1. **Описание алгоритма**

Алгоритм состоит в том, чтобы в цикле for пройти по всем элементам списка, и сложить значения последнего и предпоследнего элементов

for i in range(zdohnet.length):

if i == zdohnet.length - 2 or i == zdohnet.length - 1:

s += current.value

b += str(current.value) + ' '

current = current.next

* 1. **Тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый результат | Результат программы | Отметка о прохождении |
| 1. | 1 2 3 4 5 | 9 | 9 | Пройден |
| 2. | 83 93 2 | 95 | 95 | Пройден |
| 3. | 55 | 55 | 55 | Пройден |

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

Сформируйте исходный стек, элементами которого являются целые числа. Составить программу, которая добавляет в конец стека элемент, находящийся в его вершине.

* 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 Node:

def \_\_init\_\_(self, value = None, next = None):

self.value = value

self.next = next

class Stack:

def \_\_init\_\_(self):

self.head = None

def push(self, a):

if self.head == None:

self.head = Node(a)

else:

self.head = Node(a, self.head)

def pop(self):

if self.head == None:

return None

else:

f = self.head.value

self.head = self.head.next

return f

def peek(self):

if self.head == None:

return None

else:

return self.head.value

* 1. **Пользовательский интерфейс**

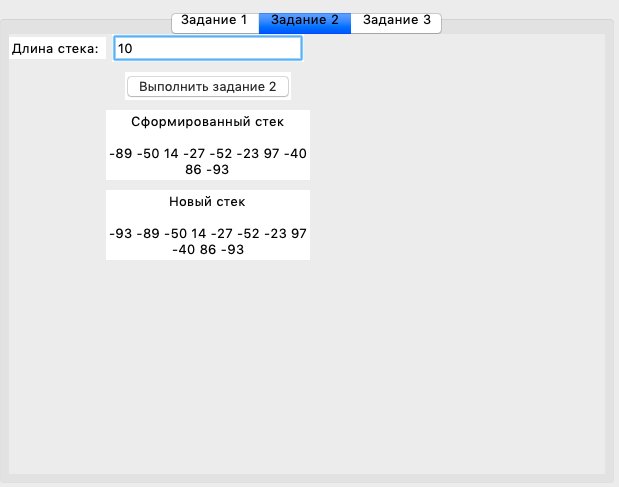


Рис. 2 – графический интерфейс для задания 2

* 1. **Описание алгоритма**

Верхний элемент стека записывается в переменную и создается еще один стек, куда переносятся все элементы стека задом-наперед. После этого старый стек создается заново пустым и туда переносятся все элементы нового стека, в результате чего получается старый стек с добавленным в конец верхним элементом.

st1 = Stack()

f = st.peek()

for i in range(n):

st1.push(st.pop())

st = Stack()

st.push(f)

s = str(st.peek()) + ' '

for i in range(n):

st.push(st1.pop())

s += str(st.peek()) + ' '

s = s[:-1]

del st1

* 1. **Тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый результат | Результат программы | Отметка о прохождении |
| 1. | 1 2 3 4 5 | 5 1 2 3 4 5 | 5 1 2 3 4 5 | Пройден |
| 2. | 78 60 3 28 | 28 78 60 3 28 | 28 78 60 3 28 | Пройден |
| 3. | 4 | 4 4 | 4 4 | Пройден |

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 - 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 Node:

def \_\_init\_\_(self, value = None, next = None):

self.value = value

self.prev = next

class Queue:

def \_\_init\_\_(self):

self.first = None

self.last = None

def enqueue(self, item):

if self.first == None:

self.first = Node(item)

elif self.first.prev == None:

self.last = Node(item)

self.first.prev = self.last

else:

t = Node(item)

self.last.prev = t

self.last = t

del t

def dequeue(self):

if self.first == None:

return None

elif self.first.prev == None:

res = self.first.value

self.first = None

self.last = None

return res

else:

res = self.first.value

i = self.first

while (i != None and i.prev != None):

i.value = i.prev.value

if i.prev.prev == None:

i.prev = None

i = i.prev

return res

def top(self):

return self.first.value

def size(self):

i = self.first

res = 0

while (i != None and i.prev != None):

i = i.prev

res += 1

if self.last != None:

res += 1

return res

def isEmpty(self):

if self.first == None:

return True

else:

return False

def \_\_str\_\_(self):

s = ""

i = self.first

while (i != None and i.prev != None):

s += str(i.value) + ' '

i = i.prev

if i != None:

s += str(i.value)

return s

* 1. **Пользовательский интерфейс**

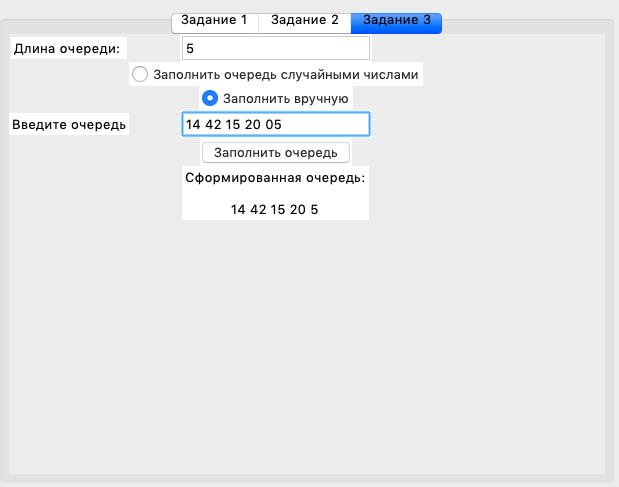


Рис. 3 – графический интерфейс для задания 3

* 1. **Описание алгоритма**

Пользователю на выбор дается два метода заполнения очереди: случайными числами и вручную. Пользователь вводит длину очереди, и если при ручном вводе длина введенной им очереди не соответствует заданной, то выводится ошибка. Если ошибки нет, то введенная пользователем строка с очередью разбивается на массив подстрок, из которого, в свою очередь, составляется очередь.

def task3():

try:

q = Queue()

n = int(txt31.get())

if n == 0:

raise Exception("Очередь не может быть пустой")

if var.get() == 0:

for i in range(n):

q.enqueue(random.randrange(-99, 100))

if var.get() == 1:

s = str(txt32.get())

arr = list(map(int, s.split()))

if len(arr) == 0:

raise Exception("Очередь не может быть пустой")

elif len(arr) > n:

raise Exception("Переполнение очереди")

for i in range(n):

q.enqueue(arr[i])

lbl33.grid()

lbl33.config(text = ("Сформированная очередь:\r\n{0}").format(str(q)))

except Exception as e:

lbl33.grid()

lbl33.config(text = "ошибка " + str(e))

* 1. **Тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый результат | Результат программы | Отметка о прохождении |
| 1. | 5, 5 12 14 85 02 | 5 12 14 85 2 | 5 12 14 85 2 | Пройден |
| 2. | 5, 78 60 3 28 | Вывод ошибки | Вывод ошибки | Пройден |
| 3. | 6, 1 2 3 4 5 6 | 1 2 3 4 5 6 | 1 2 3 4 5 6 | Пройден |

**Листинг программы**

from tkinter import \*

import tkinter.ttk as ttk

from LAB1t1 import Node, LOS

from LAB1t2 import Stack

from LAB1t3 import Queue

import random

def task1():

try:

n = int(txt1.get())

if n == 0:

raise Exception

zdohnet = LOS()

s = 0

b = ""

for i in range(n):

zdohnet.add(random.randrange(-99, 100))

current = zdohnet.first

for i in range(zdohnet.length):

if i == zdohnet.length - 2 or i == zdohnet.length - 1:

s += current.value

b += str(current.value) + ' '

current = current.next

lbl12.grid()

lbl13.grid()

lbl12.config(text = ("Список\r\n{0}".format(b[:-1])))

lbl13.config(text = ("Сумма двух последних элементов\r\n{0}".format(s)))

except Exception:

txt1.delete(0, END)

lbl12.grid()

lbl13.grid\_remove()

lbl12.config(text = "ВВЕДЕНА НЕПРАВИЛЬНАЯ ДЛИНА СПИСКА")

lbl13.config(text = '')

def task2():

try:

n = int(txt2.get())

if n == 0:

raise Exception

st = Stack()

s = ""

for i in range(n):

st.push(random.randrange(-99, 100))

s += str(st.peek()) + ' '

s = s[:-1]

lbl22.grid()

lbl22.config(text = ("Сформированный стек\r\n{0}".format(s)))

st1 = Stack()

f = st.peek()

for i in range(n):

st1.push(st.pop())

st = Stack()

st.push(f)

s = str(st.peek()) + ' '

for i in range(n):

st.push(st1.pop())

s += str(st.peek()) + ' '

s = s[:-1]

del st1

lbl23.grid()

lbl23.config(text = ("Новый стек\r\n{0}".format(s)))

except Exception:

txt2.delete(0, END)

lbl22.grid()

lbl23.grid\_remove()

lbl22.config(text = "ВВЕДЕНА НЕПРАВИЛЬНАЯ ДЛИНА СТЕКА")

lbl23.config(text = '')

def radio():

if var.get() == 0:

lbl32.grid\_remove()

txt32.grid\_remove()

if var.get() == 1:

lbl32.grid()

txt32.grid()

lbl33.grid\_remove()

def task3():

try:

q = Queue()

n = int(txt31.get())

if n == 0:

raise Exception("Очередь не может быть пустой")

if var.get() == 0:

for i in range(n):

q.enqueue(random.randrange(-99, 100))

if var.get() == 1:

s = str(txt32.get())

arr = list(map(int, s.split()))

if len(arr) == 0:

raise Exception("Очередь не может быть пустой")

elif len(arr) > n:

raise Exception("Переполнение очереди")

for i in range(n):

q.enqueue(arr[i])

lbl33.grid()

lbl33.config(text = ("Сформированная очередь:\r\n{0}").format(str(q)))

except Exception as e:

lbl33.grid()

lbl33.config(text = "ошибка " + str(e))

window = Tk()

window.title("Лабораторная работа №1")

window.geometry("650x500+350+100")

tab\_control = ttk.Notebook()

tab1 = ttk.Frame(tab\_control)

tab2 = ttk.Frame(tab\_control)

tab3 = ttk.Frame(tab\_control)

tab\_control.add(tab1, text = "Задание 1")

tab\_control.add(tab2, text = "Задание 2")

tab\_control.add(tab3, text = "Задание 3")

btn1 = Button(tab1, text = "Выполнить задание 1", command = task1)

lbl11 = Label(tab1, text = "Длина списка: ")

lbl12 = Label(tab1, wraplength = 200)

lbl13 = Label(tab1, wraplength = 200)

txt1 = Entry(tab1)

lbl11.grid(column = 0, row = 0)

txt1.grid(column = 1, row = 0)

btn1.grid(column = 1, row = 1, pady = 10)

lbl12.grid(column = 1, row = 2)

lbl13.grid(column = 1, row = 3, pady = 10)

lbl12.grid\_remove()

lbl13.grid\_remove()

btn2 = Button(tab2, text = "Выполнить задание 2", command = task2)

lbl21 = Label(tab2, text = "Длина стека: ")

lbl22 = Label(tab2, wraplength = 200)

lbl23 = Label(tab2, wraplength = 200)

txt2 = Entry(tab2)

lbl21.grid(column = 0, row = 0)

txt2.grid(column = 1, row = 0)

btn2.grid(column = 1, row = 1, pady = 10)

lbl22.grid(column = 1, row = 2)

lbl23.grid(column = 1, row = 3, pady = 10)

lbl22.grid\_remove()

lbl23.grid\_remove()

var = BooleanVar()

var.set(0)

lbl31 = Label(tab3, text = "Длина очереди: ")

txt31 = Entry(tab3)

r1 = Radiobutton(tab3, text = "Заполнить очередь случайными числами", variable = var, value = 0, command = radio)

r2 = Radiobutton(tab3, text = "Заполнить вручную", variable = var, value = 1, command = radio)

lbl32 = Label(tab3, text = "Введите очередь")

txt32 = Entry(tab3)

btn3 = Button(tab3, text = "Заполнить очередь", command = task3)

lbl33 = Label(tab3, wraplength = 200)

lbl31.grid(column = 0, row = 0)

txt31.grid(column = 1, row = 0)

r1.grid(column = 1, row = 1)

r2.grid(column = 1, row = 2)

lbl32.grid(column = 0, row = 3)

lbl32.grid\_remove()

txt32.grid(column = 1, row = 3)

txt32.grid\_remove()

btn3.grid(column = 1, row = 4)

lbl33.grid(column = 1, row = 5)

lbl33.grid\_remove

tab\_control.pack(expand = 1, fill = "both")

window.mainloop()