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

Институт Информационных технологий

Кафедра корпоративных информационных систем

**ЛАБОРАТОРНАЯ РАБОТА**

по дисциплине «Структуры и алгоритмы обработки данных»

**Тема лабораторной работы** Списки, очереди, стеки.

**Студент группы** ИКБО-12-17 Лисовой А.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Руководитель работы** Адрианова Е.Г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись руководителя)*

Работа представлена к защите «6» октября 2018 г.

Допущен к защите «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2018 г.

Москва, 2018

**Оглавление**

[Постановка задачи 2](#_Toc507327654)

[Теоретическое введение 3](#_Toc507327655)

[Алгоритм решения задачи 4](#_Toc507327656)

[Тестирование программы 5](#_Toc507327657)

[Кодирование алгоритма программы 8](#_Toc507327661)

# Постановка задачи

**Задание 1 (Вариант 17):**

Сформировать линейный двунаправленный список, элементами которого являются символы. Составить программу, которая проверяет, упорядочены ли

элементы списка по кодам ASCII.

**Задание 2 (Вариант 57):**

Сформировать кольцевой однонаправленный список L с

заглавным элементом. Составить программу определения, есть ли в

списке хотя бы один элемент, который равен следующему за ним

(по кругу) элементу.

**Задание 3 (Вариант 13):**

Составить программу построения очереди Q,

содержащей целые числа. Вычислить количество чисел, свободных

от квадратов и содержащихся в очереди. Число называется

свободным от квадратов, если оно не делится ни на один квадрат

простого числа.

Так же необходимо создать консольное меню для удобного доступа к выполнению заданий лаборторной работы.

**Теоретическое введение**

**Связный список:**

Связные списки применяются, когда требуется структура данных, размер которой заранее неизвестен. Она содрежит в себе элементы одного типа, где каждый элемент имеет указатель на следющий и/или предыдущий, а так же в неё можно добавлять и удалять из неё элементы уже после создания самого списка. Однако связный список требует дополнительных усилий для поддержания целостности при динамических операциях, так как при удалении элемента из списка обрывается связь между элементами, находящимися до и после него, и эти элементы необходимо снова связать между собой. Связные списки бывают односвязными, двусвязными и кольцевыми.

**Кольцевой связный список:**

Кольцевой список – это структура данных, представляющая собой последовательность элементов, последний элемент которой содержит указатель на первый элемент списка, а первый (в случае двунаправленного списка) – на последний. Основная особенность такой организации состоит в том, что в этом списке нет элементов, содержащих пустые указатели, и, следовательно, нельзя выделить крайние элементы. Для полного обхода такого списка достаточно иметь указатель на произвольный элемент, а не на первый, как в линейном связном списке. Понятие "первого" элемента здесь достаточно условно и не всегда требуется. Хотя иногда бывает полезно выделить некоторый элемент как "первый" путем установки на него специального указателя. Это требуется, например, для предотвращения "зацикливания" при просмотре списка.

**Очередь:**

Очередь – структура данных типа «список», позволяющая добавлять элементы лишь в конец списка, и извлекать их из его начала. Она функционирует по принципу FIFO (First In, First Out — «первым пришёл — первым вышел»), для которого характерно, что все элементы добавленные раньше элемента текущего, должны быть удалены прежде, чем будет удален текущий элемент. Также очередь может быть определена как частный случай односвязного списка, который обслуживает элементы в порядке их поступления. Как и в «живой» очереди, здесь первым будет обслужен тот, кто пришел первым.

**Описание алгоритма решения задачи**

Алгоритм:

1. Ознакомиться с заданиями лабораторной работы и принципами организации работы указанных структур данных

2. Разработать класс-модуль двусвязного списка, выполняющий условие первого задания

3. Разработать класс-модуль кольцевого списка, выполняющий условие второго задания

4. Разработать класс-модуль очереди, выполняющий условие третьего задания

5. Разбработать класс-модуль интерфейса, реализующий удобную работу с классами-модулями заданий работы.

6. Подключить все классы-модули к главному файлу лабораторной работы, отладить работу интерфейса, провести тест работы остальных классов-модулей лабораторной работы.

**Тестирование программы**

**Задание 1:**

Тест №1: Проверить, упорядочены ли следующие элементы списка [‘2’, ‘3’, ‘F’, ‘a’, ‘z’, ‘Х’, ‘л’]:

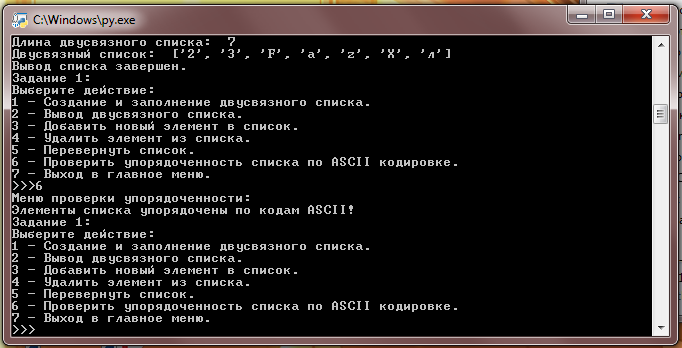


Рис. 1 Подтверждает корректную работу, так как заданные символы имеют следующие ASCII коды: ‘2’ – 50, ‘3’ – 51, ‘F’ – 70, ‘a’ – 97, ‘z’ – 122, ‘Х’ – 213, ‘л’ – 235.

Тест №2: Проверить, упорядочены ли следующие элементы списка [‘F’, ‘2’, ‘z’, ‘5’]:

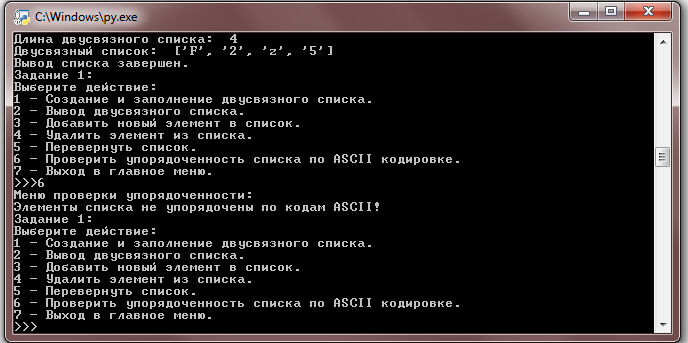


Рис. 2 Подтверждает корректную работу программы, так как заданные символы имеют следующие ASCII коды: ‘F’ – 70, ‘2’ – 50, ‘z’ – 122, ‘5’ -53.

**Задание 2:**

Тест №3: Определить есть ли в кольцевом списке L = [‘Red’, ‘Blue’, ‘White’, ‘White’, ‘Yellow’] элемент, равный следующему по кругу:

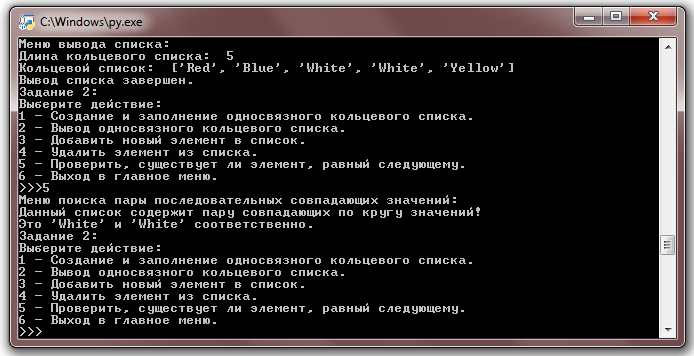


Рис. 3 Подтверждает корректную работу программы, так как ‘White’ и ‘White’ действительно последовательно действуют друг за другом.

Тест №4: Определить есть ли в кольцевом списке L = [‘море’, ‘без’, ‘волн’, ‘-’, ‘не’, ‘море] элемент, равный следующему по кругу:

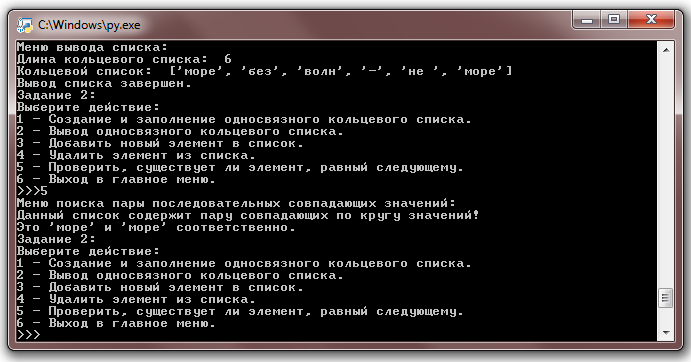


Рис. 4 Подтверждает корректную работу программы, так как последний элемент кольцевого списка - ‘море’ указывает на первый элемент списка - ‘море’ соответственно.

**Задание 3:**

Тест №5: Определить, есть ли в очереди Q = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20] свободные от квадратов элементы, а также подсчитать их количество:

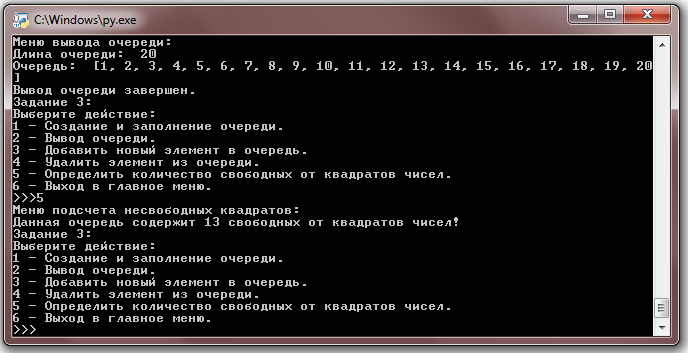


Рис. 5 Подтверждает корректную работу программы, для большей наглядности ниже будет приведена таблица OEIS(A005117) свободных от квадратов чисел:

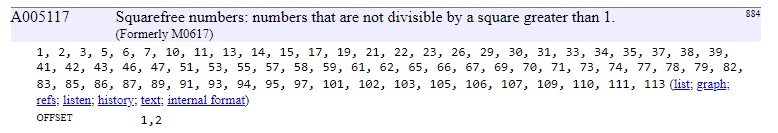


Рис. 6 Таблица свободных от квадратов чисел. Из нее видно, что в диапазоне [1, 20] лежит ровно 13 свободных от квадратов чисел, что наглядно подтверждает правильность программы.

**Кодирование алгоритма программы**

**Модуль LR1\_Python.py:**

print ("Лабораторная работа №1: Лисовой А.А. ИКБО-12-17")

print ("Варианты: 1)17 2)57 3)13 \n")

#Импорт кастомного двусвязного списка:

from DoubleLinkedList import DoubleLinkedList

#Импорт кастомного кольцевого односвязного списка:

from CircularLinkedList import CircularLinkedList

#Импорт кастомной очереди:

from Queue import Queue

#Импорт класса-меню:

from Interface import Interface

#Консольное меню:

Menu = Interface()

Menu.Interface\_Main()

while True:

Step = Menu.Input\_Controller()

###################################################################################

#Первое задание:

if Step == 1:

Menu.Interface\_Task1()

#Создание объекта списка:

List = DoubleLinkedList()

while True:

Step = Menu.Input\_Controller()

#Заполнение:

if Step == 1:

print ("Меню заполнения списка:")

print ("Введите изначальную длину двусвязного списка:")

Length = Menu.Input\_Controller()

#Создание объекта списка:

List = DoubleLinkedList()

for i in range(1, Length + 1):

print ("Введите ", i, " элемент списка:")

Data = str(input(">>>"))

Data = Data[0:1]

List.Append(Data)

print ("Ввод списка завершен.")

Menu.Interface\_Task1()

#Вывод двусвязного списка:

elif Step == 2:

print ("Меню вывода списка:")

if List.length == 0:

print("Список пуст!")

else:

print("Длина двусвязного списка: ", List.length)

print("Двусвязный список: ", List.\_\_repr\_\_())

print ("Вывод списка завершен.")

Menu.Interface\_Task1()

#Добавление элементов в список:

elif Step == 3:

Menu.Interface\_Task1\_Add()

while True:

Step = Menu.Input\_Controller()

#Добавить в начало:

if Step == 1:

print ("Введите добавляемый элемент списка:")

Data = str(input(">>>"))

Data = Data[0:1]

List.Prepend(Data)

Menu.Interface\_Task1\_Add()

#Добавить в конец:

elif Step == 2:

print ("Введите добавляемый элемент списка:")

Data = str(input(">>>"))

Data = Data[0:1]

List.Append(Data)

Menu.Interface\_Task1\_Add()

#Возврат в подменю:

elif Step == 3:

Menu.Interface\_Task1()

break

#Остальные случаи:

else:

print("Ошибка ввода!")

#Удалить элемент из списка:

elif Step == 4:

Menu.Interface\_Task1\_Del()

while True:

Step = Menu.Input\_Controller()

#Удалить первый элемент:

if Step == 1:

List.DeleteFirst()

Menu.Interface\_Task1\_Del()

#Удалить последний элемент:

elif Step == 2:

List.DeleteLast()

Menu.Interface\_Task1\_Del()

#Удалить первое вхождение введенного элемента:

elif Step == 3:

print ("Введите удаляемый элемент списка:")

Data = str(input())

Data = Data[0:1]

List.Remove(Data)

Menu.Interface\_Task1\_Del()

#Возврат в подменю:

elif Step == 4:

Menu.Interface\_Task1()

break

#Остальные случаи:

else:

print("Ошибка ввода!")

#Перевернуть список:

elif Step == 5:

List.Reverse()

Menu.Interface\_Task1()

#Проверить упорядоченность списка по ASCII кодировке:

elif Step == 6:

Flag = List.ASCII\_Check()

Menu.Interface\_Task1()

#Выход в главное меню:

elif Step == 7:

Menu.Interface\_Main()

break

#Остальные случаи:

elif str(Step).isdigit():

print("Ошибка ввода!")

###################################################################################

#Второе задание:

elif Step == 2:

Menu.Interface\_Task2()

#Создание объекта кольцевого списка:

L = CircularLinkedList()

while True:

Step = Menu.Input\_Controller()

#Создание и заполнение кольцевого списка:

if Step == 1:

print ("Меню заполнения кольцевого списка:")

print ("Введите изначальную длину кольцевого списка:")

Length = Menu.Input\_Controller()

#Создание объекта списка:

L = CircularLinkedList()

for i in range(1, Length + 1):

print ("Введите ", i, " элемент списка:")

Data = str(input(">>>"))

L.Append(Data)

print ("Ввод списка завершен.")

Menu.Interface\_Task2()

#Вывод кольцевого списка:

elif Step == 2:

print ("Меню вывода списка:")

if L.length == 0:

print("Список пуст!")

else:

print("Длина кольцевого списка: ", L.length)

print("Кольцевой список: ", L.\_\_repr\_\_())

print ("Вывод списка завершен.")

Menu.Interface\_Task2()

#Добавление элементов в кольцевой список:

elif Step == 3:

Menu.Interface\_Task2\_Add()

while True:

Step = Menu.Input\_Controller()

#Добавить в начало:

if Step == 1:

print ("Введите добавляемый элемент списка:")

Data = str(input(">>>"))

L.Prepend(Data)

Menu.Interface\_Task2\_Add()

#Добавить в конец:

elif Step == 2:

print ("Введите добавляемый элемент списка:")

Data = str(input(">>>"))

L.Append(Data)

Menu.Interface\_Task2\_Add()

#Добавить в указанную позицию:

elif Step == 3:

print ("Введите добавляемый элемент списка:")

Data = str(input(">>>"))

print ("Введите позицию для вставки:")

Position = Menu.Input\_Controller()

if Position <= L.length + 1:

L.Insert(Data, Position)

else:

print ("Указана недоступная для вставки позиция!")

Menu.Interface\_Task2\_Add()

#Возврат в подменю:

elif Step == 4:

Menu.Interface\_Task2()

break

#Остальные случаи:

else:

print("Ошибка ввода!")

#Удалить элемент из кольцевого списка:

elif Step == 4:

Menu.Interface\_Task2\_Del()

while True:

Step = Menu.Input\_Controller()

#Удалить первый элемент:

if Step == 1:

L.DeleteFirst()

Menu.Interface\_Task2\_Del()

#Удалить последний элемент:

elif Step == 2:

L.DeleteLast()

Menu.Interface\_Task2\_Del()

#Возврат в подменю:

elif Step == 3:

Menu.Interface\_Task2()

break

#Остальные случаи:

else:

print("Ошибка ввода!")

#Поиск пары совпадений:

elif Step == 5:

print("Меню поиска пары последовательных совпадающих значений:")

L.SearchEqualPair()

Menu.Interface\_Task2()

#Выход в главное меню:

elif Step == 6:

Menu.Interface\_Main()

break

#Остальные случаи:

elif str(Step).isdigit():

print("Ошибка ввода!")

###################################################################################

#Третье задание:

elif Step == 3:

#Третье задание:

Menu.Interface\_Task3()

#Создание объекта очереди:

Q = Queue()

while True:

Step = Menu.Input\_Controller()

#Создание и заполнение очереди:

if Step == 1:

print ("Меню заполнения очереди:")

print ("Введите изначальную длину очереди:")

Length = Menu.Input\_Controller()

#Создание объекта очереди:

Q = Queue()

for i in range(1, Length + 1):

print ("Введите ", i, " элемент очереди:")

Data = Menu.Int\_Controller()

Q.Enqueue(Data)

print ("Элемент успешно помещен в очередь!")

print ("Ввод списка завершен.")

Menu.Interface\_Task3()

#Вывод очереди:

elif Step == 2:

print ("Меню вывода очереди:")

if Q.length == 0:

print("Очередь пуста!")

else:

print("Длина очереди: ", Q.length)

print("Очередь: ", Q.\_\_repr\_\_())

print ("Вывод очереди завершен.")

Menu.Interface\_Task3()

#Добавление элемента в очередь:

elif Step == 3:

print ("Меню добавление элемента в очередь:")

print ("Введите добавляемый элемент очереди:")

Data = Menu.Int\_Controller()

Q.Enqueue(Data)

print ("Элемент успешно помещен в очередь!")

Menu.Interface\_Task3()

#Удаление элемента из очереди:

elif Step == 4:

print ("Меню удаления элемента из очереди:")

Q.Dequeue()

print ("Элемент успешно удален из очереди!")

Menu.Interface\_Task3()

#Подсчет свободных от квадратов чисел:

elif Step == 5:

print ("Меню подсчета несвободных квадратов:")

Q.NQVD\_Counter()

Menu.Interface\_Task3()

#Выход в главное меню:

elif Step == 6:

Menu.Interface\_Main()

break

#Остальные случаи:

elif str(Step).isdigit():

print("Ошибка ввода!")

###################################################################################

#Выход из программы

elif Step == 4:

print("Выход из программы осуществлен!")

break

#Остальные случаи:

elif str(Step).isdigit():

print("Ошибка ввода!")

**Модуль Interface.py:**

class Interface:

#Конструктор:

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

pass

#Главное меню:

def Interface\_Main(self):

print ("Главное меню:")

print ("Выберите действие:")

print ("1 - Задание с двусвязным списком.")

print ("2 - Задание с кольцевым списком.")

print ("3 - Задание с очередью.")

print ("4 - Выход из программы.")

pass

#Подменю первого задания:

def Interface\_Task1(self):

print ("Задание 1:")

print ("Выберите действие:")

print ("1 - Создание и заполнение двусвязного списка.")

print ("2 - Вывод двусвязного списка.")

print ("3 - Добавить новый элемент в список.")

print ("4 - Удалить элемент из списка.")

print ("5 - Перевернуть список.")

print ("6 - Проверить упорядоченность списка по ASCII кодировке.")

print ("7 - Выход в главное меню.")

pass

#Меню выбора способа добавления элемента первого задания:

def Interface\_Task1\_Add(self):

print ("Меню добавления элемента в список:")

print ("Выберите действие:")

print ("1 - Добавить элемент в начало.")

print ("2 - Добавить элемент в конец.")

print ("3 - Возврат в подменю.")

pass

#Меню удаления элемента в первом задании:

def Interface\_Task1\_Del(self):

print ("Меню удаления элемента из списка:")

print ("Выберите действие:")

print ("1 - Удалить первый элемент.")

print ("2 - Удалить последний элемент.")

print ("3 - Удалить первое вхождение введенного элемента.")

print ("4 - Возврат в подменю.")

pass

#Подменю второго задания:

def Interface\_Task2(self):

print ("Задание 2:")

print ("Выберите действие:")

print ("1 - Создание и заполнение односвязного кольцевого списка.")

print ("2 - Вывод односвязного кольцевого списка.")

print ("3 - Добавить новый элемент в список.")

print ("4 - Удалить элемент из списка.")

print ("5 - Проверить, существует ли элемент, равный следующему.")

print ("6 - Выход в главное меню.")

pass

#Меню вставки элемента второго задания:

def Interface\_Task2\_Add(self):

print ("Меню добавления элемента в список:")

print ("Выберите действие:")

print ("1 - Добавить в начало списка.")

print ("2 - Добавить в конец списка.")

print ("3 - Добавить в указанную позицию.")

print ("4 - Возврат в подменю.")

pass

#Меню удаления элемента во втором задании:

def Interface\_Task2\_Del(self):

print ("Меню удаления элемента из списка:")

print ("Выберите действие:")

print ("1 - Удалить первый элемент.")

print ("2 - Удалить последний элемент.")

print ("3 - Возврат в подменю.")

pass

#Подменю третьего задания:

def Interface\_Task3(self):

print ("Задание 3:")

print ("Выберите действие:")

print ("1 - Создание и заполнение очереди.")

print ("2 - Вывод очереди.")

print ("3 - Добавить новый элемент в очередь.")

print ("4 - Удалить элемент из очереди.")

print ("5 - Определить количество свободных от квадратов чисел.")

print ("6 - Выход в главное меню.")

pass

#Защита от некорректного ввода:

def Input\_Controller(self):

while True:

try:

Value = int(input(">>>"))

if Value > 0:

return Value

else:

print("Ошибка ввода!")

except ValueError:

print("Ошибка ввода!")

#Защита от ввода всего, что не являет собой чисел:

def Int\_Controller(self):

while True:

try:

Value = int(input(">>>"))

return Value

except ValueError:

print("Ошибка ввода!")

**Модуль Node.py**

#Универсальный класс-узел:

class Node(object):

#Конструтор:

def \_\_init\_\_(self, data = None, \_next = None, \_prev = None):

self.\_next = \_next

self.data = data

self.\_prev = \_prev

#Привидение к строке:

def \_\_repr\_\_(self):

return repr(self.data)

**Модуль LQS.py**

from Node import Node

#Класс-шаблон для создания списков, очередей и стеков:

class LQS(object):

#Конструктор:

def \_\_init\_\_(self, length = 0):

self.\_head = None

self.length = 0

#Преобразование к строке:

def \_\_repr\_\_(self):

Nodes = []

Current = self.\_head

if not self.\_head:

return '[' + ', '.join(Nodes) + ']'

if not Current.\_next:

Nodes.append(repr(Current))

else:

Current = self.\_head

while Current:

Nodes.append(repr(Current))

Current = Current.\_next

return '[' + ', '.join(Nodes) + ']'

#Удаление объекта:

def Clear(self):

self.\_head = None

**Модуль DoubleLinkedList.py**

#Импорт класс узла:

from Node import Node

#Импорт класса-шаблона:

from LQS import LQS

#Класс двусвязного списка:

class DoubleLinkedList(LQS):

#Вставить элемент в конец списка:

def Append(self, data):

if not self.\_head:

self.\_head = self.\_tail = Node(data = data)

self.length += 1

print ("Элемент был добавлен в конец списка.")

return

Current = self.\_head

while Current.\_next:

Current = Current.\_next

Current.\_next = self.\_tail = Node(data = data, \_prev = Current)

self.length += 1

print ("Элемент был добавлен в конец списка.")

#Вставить элемент в начало списка:

def Prepend(self, data):

new\_head = Node(data = data, \_next = self.\_head)

if not self.\_head:

self.\_head = self.\_tail = Node(data = data)

self.length += 1

print ("Элемент был добавлен в начало списка.")

elif self.\_head:

self.\_head.\_prev = new\_head

self.\_head = new\_head

self.length += 1

print ("Элемент был добавлен в начало списка.")

#Найти элемент в списке:

def \_Find(self, key):

Current = self.\_head

while Current and Current.data != key:

Current = Current.\_next

return Current

#Удалить узел из списка:

def \_Delete(self, Node):

if Node.\_prev:

Node.\_prev.\_next = Node.\_next

if Node.\_next:

Node.\_next.\_prev = Node.\_prev

if Node is self.\_head:

self.\_head = Node.\_next

if Node is self.\_tail:

self.\_tail = Node.\_prev

Node.\_prev = None

Node.\_next = None

#Удалить первое вхождение элемента из списка:

def Remove(self, key):

if self.length == 0:

print("Список пуст!")

else:

Element = self.\_Find(key)

if not Element:

print ("Такого элемента в списке нет!")

return

self.\_Delete(Element)

print ("Первое вхождение введенного элемента удалено!")

self.length -= 1

#Удалить первый элемент списка:

def DeleteFirst(self):

if self.\_head != None:

self.\_Delete(self.\_head)

self.length -= 1

print ("Элемент успешно удален!")

else:

print("Список пуст!")

return

#Удалить последний элемент списка:

def DeleteLast(self):

if self.length == 0:

print("Список пуст!")

elif self.length == 1:

self.\_Delete(self.\_head)

self.length -= 1

elif self.\_tail:

self.\_Delete(self.\_tail)

self.length -= 1

print ("Элемент успешно удален!")

#Перевернуть список:

def Reverse(self):

print("Меню переворачивания списка:")

if self.length == 0:

print("Список пуст!")

elif self.length == 1:

print("Список был успешно перевернут!")

else:

self.\_tail = self.\_head

Current = self.\_head

Previous\_Node = None

while Current:

Previous\_Node = Current.\_prev

Current.\_prev = Current.\_next

Current.\_next = Previous\_Node

Current = Current.\_prev

self.\_head = Previous\_Node.\_prev

print("Список был успешно перевернут!")

#Проверка упорядоченности по ASCII коду:

def ASCII\_Check(self):

print("Меню проверки упорядоченности:")

if self.length == 0:

print("Список пуст!")

else:

Current = self.\_head

Plus = 0

Minus = 0

#Цикл прохода по списку попарно:

while Current and Current.\_next:

Old\_Current\_Data = Current.data

Current\_Data = Current.\_next.data

if ord(Old\_Current\_Data) < ord(Current\_Data):

Plus += 1

elif ord(Old\_Current\_Data) == ord(Current\_Data):

Plus += 1

Minus += 1

elif ord(Old\_Current\_Data) > ord(Current\_Data):

Minus += 1

Current = Current.\_next

if Plus == self.length - 1:

print ("Элементы списка упорядочены по кодам ASCII!")

elif Minus == self.length - 1:

print ("Элементы списка упорядочены по кодам ASCII!")

else:

print ("Элементы списка не упорядочены по кодам ASCII!")

**Модуль CircularLinkedList.py**

#Импорт класса-узла:

from Node import Node

#Импорт класса-шаблона:

from LQS import LQS

#Класс кольцевого односвязного списка:

class CircularLinkedList(LQS):

#Переопределение преобразования к строке для кольцевого списка:

def \_\_repr\_\_(self):

Nodes = []

Current = self.\_head

if not self.\_head:

return '[' + ', '.join(Nodes) + ']'

if not Current.\_next:

Nodes.append(repr(Current))

else:

Nodes.append(repr(Current))

Current = self.\_head.\_next

while Current != self.\_head:

Nodes.append(repr(Current))

Current = Current.\_next

return '[' + ', '.join(Nodes) + ']'

#Вставить в конец списка:

def Append(self, data):

if not self.\_head:

self.\_head = Node(data)

self.\_head.\_next = self.\_head

self.length += 1

else:

Current = self.\_head

while Current.\_next != self.\_head:

Current = Current.\_next

Current.\_next = Node(data)

Current.\_next.\_next = self.\_head

self.length += 1

print ("Элемент был добавлен в конец списка!")

#Вставить в начало списка:

def Prepend(self, data):

if not self.\_head:

self.\_head = Node(data)

self.\_head.\_next = self.\_head

self.length += 1

else:

Current = self.\_head

while Current.\_next != self.\_head:

Current = Current.\_next

New\_Node = Node(data)

New\_Node.\_next = self.\_head

self.\_head = New\_Node

Current.\_next = self.\_head

self.length += 1

print ("Элемент был добавлен в начало списка!")

#Вставить после указанной позиции:

def Insert(self, data, position):

if not self.\_head:

self.\_head = Node(data)

self.\_head.\_next = self.\_head

self.length += 1

else:

if position == self.length + 1:

self.Append(data)

elif position == 1:

self.Prepend(data)

else:

Current = self.\_head

Current\_Position = 0

while Current\_Position < position - 2:

Current = Current.\_next

Current\_Position += 1

New\_Node = Node(data)

New\_Node.\_next = Current.\_next

Current.\_next = New\_Node

self.length += 1

print ("Элемент был успешно добавлен в указанную позицию!")

#Удалить первый элемент списка:

def DeleteFirst(self):

if self.length == 0:

print("Список пуст!")

elif self.\_head.\_next == self.\_head:

self.\_head = None

self.length -= 1

print ("Первый элемент успешно удален!")

else:

Current = Head\_Buffer = self.\_head

while Current.\_next != self.\_head:

Current = Current.\_next

self.\_head = Head\_Buffer.\_next

Current.\_next = self.\_head

self.length -= 1

print ("Первый элемент успешно удален!")

#Удалить последний элемент списка:

def DeleteLast(self):

if self.length == 0:

print("Список пуст!")

elif self.\_head.\_next == self.\_head:

self.\_head = None

self.length -= 1

print ("Последний элемент успешно удален!")

else:

Current = self.\_head.\_next

while Current.\_next.\_next != self.\_head:

Current = Current.\_next

Current.\_next = self.\_head

self.length -= 1

print ("Последний элемент успешно удален!")

#Определить, есть ли в списке хотя бы один элемент, который равен следующему:

def SearchEqualPair(self):

if self.length < 2:

print ("Данный список не содержит в себе пары значений!")

else:

Current = self.\_head

while Current.\_next != self.\_head:

if Current.data == Current.\_next.data:

print("Данный список содержит пару совпадающих по кругу значений!")

print("Это \'" + Current.data + "\' и \'" + Current.\_next.data + "\' соответственно.")

return

Current = Current.\_next

if Current.data == Current.\_next.data:

print("Данный список содержит пару совпадающих по кругу значений!")

print("Это \'" + Current.data + "\' и \'" + Current.\_next.data + "\' соответственно.")

return

print("Данный список не содержит пару совпадающих по кругу значений!")

**Модуль Queue.py**

#Импорт класс узла:

from Node import Node

#Импорт класса-шаблона:

from LQS import LQS

#Класс очереди:

class Queue(LQS):

#Поместить в очередь:

def Enqueue(self, data):

if not self.\_head:

self.\_head = Node(data)

self.length += 1

return

else:

Current = self.\_head

while Current.\_next:

Current = Current.\_next

Current.\_next = Node(data)

self.length += 1

#Удалить из очереди:

def Dequeue(self):

if self.length == 0:

print ("Очередь пуста!")

else:

if not self.\_head.\_next:

Head\_Buffer = self.\_head

self.\_head = self.\_head.\_next

self.length -= 1

return Head\_Buffer.data

else:

Current = Head\_Buffer = self.\_head

while Current:

Current = Current.\_next

self.\_head = Head\_Buffer.\_next

Current = self.\_head

self.length -= 1

return Head\_Buffer.data

#Получить элемент из очереди:

def Peek(self):

if self.\_head:

return self.\_head.data

else:

print ("Очередь пуста!")

#Вычислить количество чисел, свободных от квадратов:

def NQVD\_Counter(self):

if self.length == 0:

print ("Очередь пуста!")

else:

Count = 0

Current = self.\_head

while Current:

if self.\_NonQuadValuesDetector(Current.data):

Count += 1

Current = Current.\_next

if Count > 0:

print ("Данная очередь содержит " + str(Count) + " свободных от квадратов чисел!")

else:

print ("Данная очередь не содержит свободных от квадратов чисел!")

#Функция опеределния свободы от квадртов

def \_NonQuadValuesDetector(self, data):

#Очередь, хранящяя квадраты простых чисел:

QuadPrimeQueue = Queue()

if data > 0:

for value in range(2, data):

if value\*value <= data and self.\_PrimeDetector(value):

QuadPrimeQueue.Enqueue(value\*value)

#Цикл проверки элемента на делимость квадратами:

for value in range(1, QuadPrimeQueue.length + 1):

if data % QuadPrimeQueue.Dequeue() == 0:

return False

return True

#Детектор простых чисел:

def \_PrimeDetector(self, data):

for value in range(2, 11):

check = False

if value == data:

continue

if data % value != 0:

check = True

if check == False or data == 1 or data == 0:

return False

return True