Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

по курсу «Логика и основы алгоритмизации

в инженерных задачах»

на тему «Динамические списки»

Выполнил: ст. гр. 24ВВВ1

Цыбузин Д. В.

Тусков А. А.

Принял:

к.т.н, доцент Юрова О. В.

к.т.н., доцент Деев М.В.

Пенза 2025

**Цель работы:**

Познакомиться с динамическими списками и реализовать предложенные типы данных.

**Лабораторное задание:**

1. Реализовать приоритетную очередь, путём добавления элемента в список в

соответствии с приоритетом объекта (т.е. объект с большим приоритетом

становится перед объектом с меньшим приоритетом).

2. \*На основе приведенного кода реализуйте структуру данных Очередь.

3. \*На основе приведенного кода реализуйте структуру данных Стек.

**Ход работы:**

1. **Приоритетная очередь** представляет собой абстрактную структуру данных, где порядок обработки элементов определяется не временем добавления, а их «важностью», то есть приоритетом. У приоритетной очереди, как и у обычной, имеется начало и конец, а элементы извлекаются от начала. Но у приоритетной очереди элементы упорядочиваются по ключу, так что элемент с наименьшим (в некоторых реализациях — наибольшим) значением ключа всегда находится в начале. Новые элементы вставляются в позициях, сохраняющих порядок сортировки.

Для реализации было объявлено два класса: *Node* – элемент очереди, хранящий данные и свой приоритет, и *PriorityQueue* – непосредственно реализация приоритетной очереди Было реализовано 8 методов:

1) \_\_init\_\_(self) - Инициализирует пустую приоритетную очередь, устанавливая указатель head в None.

2) is\_empty(self) - Возвращает True, если очередь пуста (head = None), иначе False.

3) leng(self) – возвращает длину приоритетной очереди;

4) push(self, data, priority) - Создает новый узел с данными и приоритетом. Вставляет элемент в начало, если очередь пуста или приоритет выше текущего head в нужное место, если приоритет ниже — ищет позицию с помощью цикла

5) pop(self) – удаляет и возвращает элемент с высшим приоритетом(из головоы списка);

6)peek(self) – возвращает данные, хранящиеся в наиболее приоритетном элементе очереди, не извлекая сам элемент;

7)display(self) – выводит все элементы в формате: (данные, приоритет). Элементы отображаются в порядке убывания приоритета. Если очередь пуста — выводит соответствующее сообщение.

8) dop(self, nazv) – Дополнительное задание. Это метод, который позволяет удалить из приоритетной очереди определённый элемент по значению.

Код реализации приоритетной очереди представлен в листинге.

2. **Очередь** представляет собой структуру данных, добавление и удаление элементов из которой осуществляется по принципу «первым вошел — первым вышел» (англ. *first-in, first-out — FIFO*). У очереди имеется голова и хвост. Когда элемент ставится в очередь, он занимает место в её хвосте. Из очереди всегда выводится элемент, который находится в ее голове.

Для реализации был объявлен класс *Queue* было реализовано 8 методов:

1) \_\_init\_\_(self) - Инициализирует пустую очередь, устанавливая указатель head и tail в None.

2) is\_empty(self) – возвращает *True*, если очередь не имеет ни одного элемента, и *False* в противном случае;

3) leng(self) – возвращает текущий размер очереди;

4) enqueue(self, data)– создаёт и вставляет в конец новый элементб если очередь пуста устанавливает новый узел, как head и tail, если не пуста устанавливает элемент после tail и смещает указатель tail;

5) dequeue(self) – возвращает последний добавленный элемент очереди и удаляет его из очереди;

6)peek(self) – возвращает первый добавленный элемент очереди;

7) display(self) - выводит на экран все элементы очереди через «->»;

8) dop(self, nazv) – Дополнительное задание. Это метод, который позволяет удалить из очереди определённый элемент по значению.

Код реализации очереди представлен в листинге.

3. **Стек** представляет собой упорядоченный набор элементов, в котором 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).

Для реализации был объявлен класс *Stack*, хранящий данные о текущем размере (*size*), верхнем элементе (*top*).Было реализовано 8 методов:

1) \_\_init\_\_(self) *-* Инициализирует пустой стек, устанавливая указатель top в None*.*

2) is\_empty(self) – возвращает *True*, если стек пуст, и *False* в противном случае;

3) leng(self) – возвращает текущий размер стека;

4) push(self, data) – добавление нового элемента в стек;

5)peek(self)– возвращает «верхушку», т.е. последний добавленный элемент стека;

6) display(self) — выводит на экран все элементы стек в столбик;

7) pop(self) – осуществляет извлечение верхнего элемента стека;

8) dop(self, nazv) – дополнительное задание. Это метод, который позволяет удалить из стека определённый элемент по значению.

Код реализации представлен в листинге.

**Вывод:** Реализованы три ключевые структуры данных с разными принципами доступа: стек (LIFO), очередь (FIFO) и приоритетная очередь. Цель работы достигнута, принципы работы структур изучены.

**Листинг программы:**

class Node:

    """Узел связного списка"""

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

        self.data = data

        self.priority = priority

        self.next = None

class PriorityQueue:

    def \_\_init\_\_(self):

        self.head = None

    def is\_empty(self):

        return self.head is None

    def leng(self):

        cnt = 0

        current = self.head

        while current is not None:

            cnt+=1

            current = current.next

        return cnt

    def push(self, data, priority):

        """Добавление элемента с приоритетом"""

        new\_node = Node(data, priority)

        # Если очередь пуста или новый элемент имеет высший приоритет чем голова

        if self.is\_empty() or priority > self.head.priority:

            new\_node.next = self.head

            self.head = new\_node

        else:

            # Ищем место для вставки

            current = self.head

            while (current.next is not None and

                   current.next.priority >= priority):

                current = current.next

            new\_node.next = current.next

            current.next = new\_node

    def pop(self):

        """Извлечение элемента с наивысшим приоритетом"""

        if self.is\_empty():

            raise IndexError("Приоритетная очередь пуста")

        data = self.head.data

        "сделать, чтобы можгно было из всео выбратьопределённый элемент изменить или удалить"

        if self.head.next is None:

            self.head = None

        else:

            self.head = self.head.next

        return data

    def peek(self):

        """Просмотр элемента с наивысшим приоритетом без извлечения"""

        if self.is\_empty():

            raise IndexError("Приоритетная очередь пуста")

        return self.head.data

    def display(self):

        """Вывод содержимого очереди"""

        if self.is\_empty():

            print("Приоритетная очередь пуста")

            return

        current = self.head

        elements = []

        while current:

            elements.append(f"({current.data}, приоритет: {current.priority})")

            current = current.next

        print(" -> ".join(elements))

    def dop(self, nazv):

        if self.is\_empty():

            raise IndexError("Приоритетная очередь пуста")

        if self.head.data == nazv:

            self.head = self.head.next

            return

        current  = self.head

        while current.next is not None:

            if current.next.data == nazv:

                current.next = current.next.next

                return

            current = current.next

        raise ValueError(f"Элемент с данными '{nazv}' не найден")

class Queue:

    """Очередь (FIFO - First In First Out)"""

    def \_\_init\_\_(self):

        self.head = None

        self.tail = None

    def is\_empty(self):

        """Проверка на пустоту"""

        return self.head is None

    def leng(self):

        cnt = 0

        current = self.head

        while current is not None:

            cnt+=1

            current = current.next

        return cnt

    def enqueue(self, data):

        new\_node = Node(data)

        if self.is\_empty():

            self.head = self.tail = new\_node

        else:

            self.tail.next = new\_node

            self.tail = new\_node

    def dequeue(self):

        if self.is\_empty():

            raise IndexError("Очередь пуста")

        data = self.head.data

        self.head = self.head.next

        # Если после извлечения очередь стала пустой

        if self.head is None:

            self.tail = None

        return data

    def peek(self):

        if self.is\_empty():

            raise IndexError("Очередь пуста")

        return self.head.data

    def display(self):

        if self.is\_empty():

            print("Очередь пуста")

            return

        current = self.head

        elements = []

        while current:

            elements.append(str(current.data))

            current = current.next

        print(" -> ".join(elements))

    def dop(self, nazv):

        if self.is\_empty():

            raise IndexError("очередь пуста")

        if self.head.data == nazv:

            self.head = self.head.next

            return

        current  = self.head

        while current.next is not None:

            if current.next.data == nazv:

                current.next = current.next.next

                return

            current = current.next

        raise ValueError(f"Элемент с данными '{nazv}' не найден")

class Stack:

    def \_\_init\_\_(self):

        self.top = None

    def leng(self):

        cnt = 0

        current = self.head

        while current is not None:

            cnt+=1

            current = current.next

        return cnt

    def is\_empty(self):

        """Проверка на пустоту"""

        return self.top is None

    def push(self, data):

        """Добавление элемента на вершину стека"""

        new\_node = Node(data)

        new\_node.next = self.top

        self.top = new\_node

    def pop(self):

        """Извлечение элемента с вершины стека"""

        if self.is\_empty():

            raise IndexError("Стек пуст")

        data = self.top.data

        self.top = self.top.next

        return data

    def peek(self):

        """Просмотр вершины стека без извлечения"""

        if self.is\_empty():

            raise IndexError("Стек пуст")

        return self.top.data

    def display(self):

        """Вывод содержимого стека"""

        if self.is\_empty():

            print("Стек пуст")

            return

        current = self.top

        elements = []

        while current:

            elements.append(str(current.data))

            current = current.next

        print("""\n  |

  V

""".join(elements))

    def dop(self, nazv):

        if self.is\_empty():

            raise IndexError("стек пуст")

        if self.top.data == nazv:

            self.top = self.top.next

            return

        current  = self.top

        while current.next is not None:

            if current.next.data == nazv:

                current.next = current.next.next

                return

            current = current.next

        raise ValueError(f"Элемент с данными '{nazv}' не найден")

def vvod(start, end):

    while True:

        try:

            print(f"Введите число от {start} до {end}")

            chis = int(input())

            if chis>end or chis<start:

                print(f"ЧИСЛО ОТ {start} до {end}")

            else: return chis

        except:

            print("Введите число")

# Демонстрация работы всех структур

if \_\_name\_\_ == "\_\_main\_\_":

    print("=== ДЕМОНСТРАЦИЯ РАБОТЫ СТРУКТУР ДАННЫХ ===\n")

    var = vvod(1, 3)

    if var==1:

        print("1.ПРИОРИТЕТНАЯ ОЧЕРЕДЬ:")

        pq = PriorityQueue()

        for i in range(1, 10):

            pq.push(f"Задача{i}", i)

            print("Содержимое после добавления:")

            pq.display()

        pq.pop()

        print("Содержимое после удаления элемента с высшим приоритетом:")

        pq.display()

        pq.dop("Задача3")

        print("Содержимое после удаления элемента с определённым значением:")

        pq.display()

    if var ==2:

        print("2.ОЧЕРЕДЬ:")

        q = Queue()

        for i in range(1, 10):

            q.enqueue(f"Задача{i}")

            print("Содержимое после добавления:")

            q.display()

        q.dequeue()

        print("Содержимое после удаления первого элемента:")

        q.display()

        q.dop("Задача3")

        print("Содержимое после удаления элемента с определённым значением:")

        q.display()

    if var == 3:

        print("3.CТЕК:")

        st = Stack()

        for i in range(1, 10):

            st.push(f"Задача{i}")

            print("Содержимое после добавления:")

            st.display()

        st.pop()

        print("Содержимое после верхнего элемента:")

        st.display()

        st.dop("Задача3")

        print("Содержимое после удаления элемента с определённым значением:")

        st.display()