Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации

в инженерных задачах»

на тему «Динамические списки»

Выполнил: ст. гр. 24ВВВ1

Будников А.С.

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, хранящая максимальную вместимость (*capacity*), текущий размер очереди (*size*), имитацию двоичной кучи (*heap*) для упрощенного доступа к элементам и методы для работы с ней. Было реализовано 10 методов:

1) void up(size\_t index) - поднимает новый элемент «вверх» по очереди, если его приоритет больше, чем у вышестоящих;

2) void down(size\_t index) – производит сортировку элементов очереди после удаления самого приоритетного элемента;

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

4) bool is\_full() const – метод обратный *is\_empty*;

5) size\_t get\_size() const – возвращает текущий размер очереди;

6) size\_t get\_capacity() const – возвращает вместимость очереди;

7) const E& top() const – возвращает данные, хранящиеся в наиболее приоритетном элементе очереди, не извлекая сам элемент;

8) int top\_priority() const – возвращает приоритет «верхнего» элемента очереди;

9) void push(E&& value, int priority) – осуществляет вставку элемента в очередь с последующей сортировкой по приоритету;

10) E pop() – осуществляет извлечение элемента, находящегося на верхней границе очереди.

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*, хранящий данные о вместимости очереди (*capacity*), текущем размере очереди (*size*), голове и хвосте очереди (в данном случае, *front* и *rear* соответственно) и массив элементов (*heap*). Было реализовано 8 методов, исключая перегрузку *push*:

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

2) bool is\_full() const – метод обратный *is\_empty*;

3) size\_t get\_size() const – возвращает текущий размер очереди;

4) size\_t get\_capacity() const – возвращает вместимость очереди;

5) const E& peek\_back() const – возвращает последний добавленный элемент очереди;

6) const E& peek\_front() const – возвращает первый добавленный элемент очереди;

7) void push(…) - осуществляет вставку элемента в очередь;

8) E pop() – осуществляет извлечение элемента, находящегося на верхней границе очереди.

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).