LAPORAN PRATIKUM STRUKTUR DATA
IMPLEMENTASI DAN OPERASI DASAR STRUKTUR DATA QUEUE DALAM BAHASA JAVA

MATA KULIAH

DOSEN PENGAMPU : Dr. Wahyudi, S.T, M.T

OLEH:

AUFAN TAUFIQURRAHMAN

NIM 2411532011

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. PENDAHULUAN

Struktur data *Queue* (antrian) merupakan salah satu struktur data linier yang menerapkan prinsip FIFO (First In First Out). Elemen yang pertama kali dimasukkan akan menjadi elemen pertama yang keluar. Dalam Java, *Queue* dapat diimplementasikan dengan memanfaatkan interface Queue dan class LinkedList. Praktikum ini bertujuan untuk memahami cara kerja queue dan operasi dasar seperti penambahan, penghapusan, iterasi, pembalikan data, serta penggunaan queue untuk menyimpan input dari pengguna.

B. TUJUAN

- Mengimplementasikan queue menggunakan Java Collections (Queue, LinkedList).
- Memahami operasi dasar queue: add(), poll(), peek(), isEmpty().
- Menangani input dari pengguna menggunakan queue.
- Melakukan iterasi dan pembalikan isi queue menggunakan struktur data tambahan.

C. PROSEDUR

1. Class inputQueue.java

```
public class inputQueue {
    int front, rear, size;
    int capacity;
    int array[];
```

- int front, rear, size → Indeks depan, belakang, dan ukuran Queue.
- int capacity → Kapasitas maksimum.
- int array[] → Penyimpanan data.

```
public inputQueue (int capacity) {
    this.capacity = capacity;
    front = this.size = 0;
    rear = capacity - 1;
    array = new int [this.capacity];
}
```

- rear = capacity - 1 → Inisialisasi rear di indeks terakhir array (untuk circular queue).

```
boolean isFull (inputQueue queue) {
    return (queue.size == queue.capacity);
}

boolean isEmpty (inputQueue queue) {
    return (queue.size == 0);
}
```

- Cek kepenuhan/kekosongan berdasarkan size == capacity atau size == 0.

```
void enqueue (int item) {
    if (isFull(this))
    return;
    this.rear = (this.rear + 1) % this.capacity;
    this.array[this.rear] = item;
    this.size = this.size + 1;
    System.out.println(item + " enqueued to queue");
}
```

- rear = (rear + 1) % capacity → Update rear dengan operasi modulus (indeks circular).
- array[rear] = item → Simpan data.
- size++ → Tambah ukuran Queue.

```
int dequeue () {
    if (isEmpty(this))
    return Integer.MIN_VALUE;

    int item = this.array[this.front];
    this.front = (this.front + 1) % this.capacity;
    this.size = this.size - 1;
    return item;
}
```

- front = (front + 1) % capacity → Geser front ke indeks berikutnya (circular).
- size-- → Kurangi ukuran Queue.

```
int front(){
    if (isEmpty(this))
    return Integer.MIN_VALUE;

    return this.array[this.front];
}
int rear () {
    if (isEmpty(this))
    return Integer.MIN_VALUE;

    return this.array[this.rear];
}
```

- Ambil data dari indeks front atau rear.
- Jika kosong, kembalikan Integer.MIN_VALUE.

2. Class TestQueue.java

```
public static void main(String[] args) {
    inputQueue queue = new inputQueue(capacity:1000);
    queue.enqueue(item:10);
    queue.enqueue(item:20);
    queue.enqueue(item:30);
    queue.enqueue(item:40);
    System.out.println("Front item is " + queue.front());
    System.out.println("Rear item is " + queue.rear());
    System.out.println(queue.dequeue() + " dequeue from queue");
    System.out.println("Front item is " + queue.front());
    System.out.println("Rear item is " + queue.rear());
}
```

- inputQueue queue = new inputQueue(1000) → Instansiasi Queue dari program inputQueue,java yang sudah dibuat dengan kapasitas 1000.
- queue.enqueue(10), queue.enqueue(20), dll. → Menambahkan data ke Queue.
- queue.front() → Ambil elemen terdepan (10).
- queue.rear() → Ambil elemen terbelakang (40).
- queue.dequeue() → Hapus elemen terdepan (10).
- Cetak hasil operasi.

3. Class ContonQueue2.java

```
import java.util.LinkedList;
import java.util.Queue;
```

- import java.util.LinkedList dan import java.util.Queue → Untuk menggunakan Queue dari Java Collections.

```
public static void main(String[] args) {
    Queue<Integer> q = new LinkedList<>();
    for (int i = 0; i < 6; i++) {
        q.add(i);
    }
    System.out.println("Elemen Antrian " + q);

    int hapus = q.remove();
    System.out.println("Hapus elemen = " + hapus);
    System.out.println(q);

    int depan = q.peek();
    System.out.println("Kepala antrian = " + depan);

    int banyak = q.size();
    System.out.println("size antrian = " + banyak);
}
```

- Queue<Integer> q = new LinkedList<>() → Instansiasi Queue menggunakan LinkedList.
- Loop for (int i = 0; i < 6; i++) → Tambahkan 0-5 ke Queue.
- q.remove() → Hapus elemen terdepan (0).
- q.peek() → Ambil elemen terdepan tanpa menghapus.
- q.size() → Hitung jumlah elemen dalam Queue.

4. Class ReverseData.java

```
public static void main(String[] args) {
    Queue<Integer> q = new LinkedList<Integer>();
    q.add(e:1);
    q.add(e:2);
    q.add(e:3);
    System.out.println("sebelum reverse " + q);
    Stack<Integer> s = new Stack<Integer>();
```

- Queue<Integer> q = new LinkedList<>() → Instansiasi Queue.
- q.add(1), q.add(2), q.add(3) → Isi Queue dengan data 1, 2, 3.
- Stack<Integer> s = new Stack<>() → Instansiasi Stack untuk membalik data.

```
while (!q.isEmpty()) {
    s.push(q.remove());
}
while (!s.isEmpty()) {
    q.add(s.pop());
}
System.out.println("sesudah reverse = " + q);
```

- s.push(q.remove()) → Pindahkan semua elemen Queue ke Stack (Stack: [1,2,3]).
- q.add(s.pop()) → Kembalikan elemen dari Stack ke Queue (Queue: [3,2,1]).
- Cetak hasil sebelum dan sesudah reverse.

5. Class IterasiQueue.java

```java
public static void main(String[] args) {
    Queue<String> q = new LinkedList<>();

    q.add(e:"Praktikum");
    q.add(e:"Struktur");
    q.add(e:"Data");
    q.add(e:"Dan");
    q.add(e:"Algoritma");
```

- Queue<String> q = new LinkedList<>() → Instansiasi Queue.
- q.add("Praktikum"), dll. → Tambahkan 5 string ke Queue.

```java
Iterator<String> iterator = q.iterator();
while (iterator.hasNext()) {
    System.out.print(iterator.next() + " ");
}
```

- Iterator<String> iterator = q.iterator() → Ambil iterator dari Queue.
- while (iterator.hasNext()) → Loop untuk iterasi elemen.
- iterator.next() → Akses elemen tanpa menghapus.

D. KESIMPULAN

Implementasi Queue dapat dilakukan dengan dua cara: menggunakan array manual atau memanfaatkan Java Collections. Queue manual menggunakan array dengan teknik indeks berputar (circular) dan operasi modulus (%) untuk mengatur penambahan/pengurangan elemen. Sementara itu, Java Collections (contoh: LinkedList) menyediakan antrian dinamis yang mudah digunakan dengan fungsi bawaan seperti add() dan remove().