LAPORAN PRAKTIKUM

PERTEMUAN 8

Disusun Oleh:

Naura Aisha Zahira (2311104078)

S1SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

1) Mahasiswa mampu menjelaskan definisi dan konsep dari queue
2) Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
3) Mahasiswa mampu menerapkan operasi tampil data pada queue

2. Landasan Teori

Queue adalah struktur data yang digunakan untuk menyimpan data dengan metode FIFO (First-In First-Out). Data yang pertama dimasukkan ke dalam queue akan menjadi data yang pertama pula untuk dikeluarkan dari queue. Queue mirip dengan konsep antrian pada kehidupan sehari-hari, dimana konsumen yang datang lebih dulu akan dilayani terlebih dahulu.

Implementasi queue dapat dilakukan dengan menggunakan array atau linked list.Struktur data queue terdiri dari dua pointer yaitu front dan rear. Front/head adalah pointer ke elemen pertama dalam queue dan rear/tail/back adalah pointer ke elemen terakhir dalam queue.

Perbedaan antara *stack* dan *queue* terletak pada aturan penambahan dan penghapusan elemen. Pada *stack*, operasi penambahan dan penghapusan elemen dilakukan di satu ujung yang disebut *top* (ujung atas). Elemen yang terakhir kali dimasukkan ke dalam *stack* akan berada di posisi paling atas dan akan menjadi elemen pertama yang dihapus. Sifat ini dikenal dengan istilah *LIFO* (Last In, First Out). Contoh analogi sederhana dari *stack* adalah tumpukan piring, di mana piring terakhir yang ditambahkan berada di posisi paling atas dan akan diambil atau dihapus terlebih dahulu.

Sebaliknya, pada *queue*, operasi penambahan dan penghapusan elemen dilakukan di dua ujung yang berbeda. Elemen baru ditambahkan di ujung belakang (*rear* atau *tail*), dan elemen dihapus dari ujung depan (*front* atau *head*). Proses ini mengikuti prinsip *FIFO* (First In, First Out), yang berarti elemen pertama yang dimasukkan ke dalam *queue* akan menjadi elemen pertama yang dikeluarkan. Dalam konteks *queue*, operasi penambahan elemen dikenal sebagai *Enqueue*, dan operasi penghapusan elemen disebut *Dequeue*.
Pada *Enqueue*, elemen ditambahkan di belakang *queue* setelah elemen terakhir yang ada, sementara pada *Dequeue*, elemen paling depan (*head*) dihapus, dan posisi *head* akan bergeser ke elemen berikutnya. Contoh penggunaan *queue* dalam kehidupan sehari-hari adalah antrean di kasir, di mana orang pertama yang datang adalah yang pertama dilayani.

Operasi pada Queue

- enqueue() : menambahkan data ke dalam queue.
- dequeue() : mengeluarkan data dari queue.
- peek() : mengambil data dari queue tanpa menghapusnya.
- isEmpty() : mengecek apakah queue kosong atau tidak.
- isFull() : mengecek apakah queue penuh atau tidak.
- size() : menghitung jumlah elemen dalam queue.

3. Guided

1. Guided 1

Sourcecode:

```
#include <iostream>
#define MAX 100

using namespace std;

class Queue {
private:
    int front, rear;
    int arr[MAX];
public:

    Queue() {
        front = -1;
        rear = -1;
    }



    bool isFull() {
        return rear == MAX - 1;
    }
```

```
bool isEmpty() {
    return front == -1 || front > rear;
}


void enqueue(int x) {
    if (isFull()) {
        cout << "Queue Overflow\n";
        return;
    }
    if (front == -1) front = 0;
    arr[++rear] = x;
}


void dequeue() {
    if (isEmpty()) {
        cout << "Queue Underflow\n";
        return;
    }
    front++;
}

int peek() {
    if (!isEmpty()) {
        return arr[front];
    }
    cout << "Queue is empty\n";
    return -1;
}


void display() {
    if (isEmpty()) {
        cout << "Queue is empty\n";
```

```
            return;
        }
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << "\n";
    }
};


int main() {
    Queue q;

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    cout << "Queue elements: ";
    q.display();

    cout << "Front element: " << q.peek() << "\n";

    cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

2. Guided 2

Sourcecode:

```
#include <iostream>

using namespace std;
```

```
// Node untuk setiap elemen Queue
class Node {
public:
    int data;      // Data elemen
    Node* next;    // Pointer ke node berikutnya

    // Konstruktor untuk Node
    Node(int value) {
        data = value;
        next = nullptr;
    }
};

// Kelas Queue menggunakan linked list
class Queue {
private:
    Node* front; // Pointer ke elemen depan Queue
    Node* rear;  // Pointer ke elemen belakang Queue

public:
    // Konstruktor Queue
    Queue() {
        front = rear = nullptr;
    }

    // Mengecek apakah Queue kosong
    bool isEmpty() {
        return front == nullptr;
    }

    // Menambahkan elemen ke Queue
    void enqueue(int x) {
        Node* newNode = new Node(x);
        if (isEmpty()) {
            front = rear = newNode; // Jika Queue kosong
```

```
        return;
    }
    rear->next = newNode; // Tambahkan node baru ke belakang
    rear = newNode;       // Perbarui rear
}

// Menghapus elemen dari depan Queue
void dequeue() {
    if (isEmpty()) {
        cout << "Queue Underflow\n";
        return;
    }
    Node* temp = front;      // Simpan node depan untuk dihapus
    front = front->next;     // Pindahkan front ke node berikutnya
    delete temp;             // Hapus node lama
    if (front == nullptr)    // Jika Queue kosong, rear juga harus null
        rear = nullptr;
}

// Mengembalikan elemen depan Queue tanpa menghapusnya
int peek() {
    if (!isEmpty()) {
        return front->data;
    }
   cout << "Queue is empty\n";
    return -1; // Nilai sentinel
}

// Menampilkan semua elemen di Queue
void display() {
    if (isEmpty()) {
        cout << "Queue is empty\n";
        return;
    }
    Node* current = front; // Mulai dari depan
```

```
        while (current) {      // Iterasi sampai akhir
            cout << current->data << " ";
            current = current->next;
        }
        cout << "\n";
    }
};

// Fungsi utama untuk menguji Queue
int main() {
    Queue q;

    // Menambahkan elemen ke Queue
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    // Menampilkan elemen di Queue
    cout << "Queue elements: ";
    q.display();

    // Menampilkan elemen depan
    cout << "Front element: " << q.peek() << "\n";

    // Menghapus elemen dari depan Queue
    q.dequeue();
    cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

4. Unguided

1. Ubahlah penerapan konsep queue pada bagian guided dari array menjadi linked list.

Sourcecode:

```
#include <iostream>
#include <string>
using namespace std;

struct Mahasiswa {
    string nama;
    int nim;
    Mahasiswa* next;
};

struct Queue {
    Mahasiswa* front;
    Mahasiswa* rear;

    Queue() {
        front = nullptr;
        rear = nullptr;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    void enqueue(string nama, int nim) {
        Mahasiswa* newMahasiswa = new Mahasiswa{nama, nim, nullptr};
        if (isEmpty()) {
            front = rear = newMahasiswa;
        } else {
            rear->next = newMahasiswa;
            rear = newMahasiswa;
        }
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* temp = front;
            front = front->next;
            delete temp;
            if (front == nullptr) {
                rear = nullptr;
            }
        }
```

```
    }

    int countQueue() {
        int count = 0;
        Mahasiswa* temp = front;
        while (temp != nullptr) {
            count++;
            temp = temp->next;
        }
        return count;
    }

    void clearQueue() {
        while (!isEmpty()) {
            dequeue();
        }
    }

    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* temp = front;
            cout << "Data antrian mahasiswa:" << endl;
            while (temp != nullptr) {
                cout << "Nama: " << temp->nama << ", NIM: " << temp->nim <<
endl;
                temp = temp->next;
            }
        }
    }
};

int main() {
    Queue antrian;
    int pilihan;
    string nama;
    int nim;

    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Mahasiswa\n";
        cout << "2. Hapus Mahasiswa\n";
        cout << "3. Lihat Antrian\n";
        cout << "4. Kosongkan Antrian\n";
```

```
        cout << "0. Keluar\n";
        cout << "Pilih: ";
        cin >> pilihan;

        switch (pilihan) {
        case 1:
            cout << "Masukkan Nama Mahasiswa: ";
            cin.ignore(); // Clear input buffer
            getline(cin, nama);
            cout << "Masukkan NIM Mahasiswa: ";
            cin >> nim;
            antrian.enqueue(nama, nim);
            break;
        case 2:
            antrian.dequeue();
            break;
        case 3:
            antrian.viewQueue();
            break;
        case 4:
            antrian.clearQueue();
            break;
        case 0:
            cout << "Keluar..." << endl;
            break;
        default:
            cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != 0);

    return 0;
}
```

2. Dari nomor 1 buatlah konsep antri dengan atribut Nama mahasiswa dan NIM Mahasiswa.
   Sourcecode:

```
void enqueue(string nama, int nim) {
    Mahasiswa* newMahasiswa = new Mahasiswa{nama, nim, nullptr};

    if (isEmpty()) {
        front = rear = newMahasiswa;
    } else if (nim < front->nim) {
        newMahasiswa->next = front;
        front = newMahasiswa;
    } else {
```

```
        Mahasiswa* temp = front;
        while (temp->next != nullptr && temp->next->nim < nim) {
            temp = temp->next;
        }
        newMahasiswa->next = temp->next;
        temp->next = newMahasiswa;

        if (newMahasiswa->next == nullptr) {
            rear = newMahasiswa;
        }
    }
}
```

3. Modifikasi program pada soal 1 sehingga mahasiswa dapat diprioritaskan berdasarkan NIM (NIM yang lebih kecil didahulukan pada saat output).
   Sourcecode:

```
#include <iostream>
#include <string>
using namespace std;

struct Mahasiswa {
    string nama;
    int nim;
    Mahasiswa* next;
};

struct Queue {
    Mahasiswa* front;
    Mahasiswa* rear;

    Queue() {
        front = nullptr;
        rear = nullptr;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    void enqueue(string nama, int nim) {
        Mahasiswa* newMahasiswa = new Mahasiswa{nama, nim, nullptr};
        if (isEmpty()) {
            front = rear = newMahasiswa;
        } else if (nim < front->nim) {
            newMahasiswa->next = front;
```

```
            front = newMahasiswa;
        } else {
            Mahasiswa* temp = front;
            while (temp->next != nullptr && temp->next->nim < nim) {
                temp = temp->next;
            }
            newMahasiswa->next = temp->next;
            temp->next = newMahasiswa;
            if (newMahasiswa->next == nullptr) {
                rear = newMahasiswa;
            }
        }
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* temp = front;
            front = front->next;
            delete temp;
            if (front == nullptr) {
                rear = nullptr;
            }
        }
    }

    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* temp = front;
            cout << "Data antrian mahasiswa:" << endl;
            while (temp != nullptr) {
                cout << "Nama: " << temp->nama << ", NIM: " << temp->nim <<
endl;
                temp = temp->next;
            }
        }
    }

    void clearQueue() {
        while (!isEmpty()) {
            dequeue();
        }
```

```
    }
};

int main() {
    Queue antrian;
    int pilihan;
    string nama;
    int nim;

    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Mahasiswa\n";
        cout << "2. Hapus Mahasiswa\n";
        cout << "3. Lihat Antrian\n";
        cout << "4. Kosongkan Antrian\n";
        cout << "0. Keluar\n";
        cout << "Pilih: ";
        cin >> pilihan;

        switch (pilihan) {
        case 1:
            cout << "Masukkan Nama Mahasiswa: ";
            cin.ignore();
            getline(cin, nama);
            cout << "Masukkan NIM Mahasiswa: ";
            cin >> nim;
            antrian.enqueue(nama, nim);
            break;
        case 2:
            antrian.dequeue();
            break;
        case 3:
            antrian.viewQueue();
            break;
        case 4:
            antrian.clearQueue();
            break;
        case 0:
            cout << "Keluar..." << endl;
            break;
        default:
            cout << "Pilihan tidak valid!" << endl;
        }
    } while (pilihan != 0);
```

```
    return 0;
}
```

Noted : Untuk data mahasiswa dan nim dimasukan oleh user

5. Kesimpulan

Queue adalah struktur data FIFO yang memungkinkan operasi penambahan (enqueue) dan penghapusan **(**dequeue**)** secara efisien. Implementasi queue dapat dilakukan dengan array atau linked list, dengan linked list memberikan fleksibilitas lebih untuk mengelola data dinamis. Modifikasi pada queue berhasil menerapkan prioritas berdasarkan NIM, di mana elemen dengan NIM terkecil diprioritaskan.