LAPORAN PRAKTIKUM

MODUL 8

QUEUE

Disusun Oleh :

Farhan Kurniawan (2311104073)

Kelas:

SE-07-2

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

1. Mempelajari prinsip kerja *Queue* berdasarkan konsep FIFO (*First In, First Out*).
2. Mengembangkan pemahaman tentang perbedaan antara implementasi *Queue* menggunakan array dan *linked list.*
3. Melatih kemampuan untuk melakukan operasi *insert* (penambahan elemen) dan *delete* (penghapusan elemen) pada *Queue* berbasis *linked list.*
4. Menjelaskan cara kerja *Queue* dalam konteks struktur data *linked list*, baik *single linked list* maupun *double linked list.*
5. Mengaplikasikan teori *Queue* pada pemrograman nyata menggunakan bahasa pemrograman C.

II. LANDASAN TEORI

Queue (dibaca: kyu) merupakan struktur data yang dapat diumpamakan seperti sebuah antrian. Misalnya, antrian pada loket pembelian tiket kereta api. Orang yang pertama kali masuk dalam antrian akan mendapatkan pelayanan terlebih dahulu, sedangkan orang yang terakhir masuk akan mendapatkan layanan paling akhir. Dengan kata lain, prinsip dasar dari Queue adalah FIFO (*First In, First Out*), yaitu proses yang pertama masuk akan diakses terlebih dahulu.

Dalam pengimplementasian struktur Queue pada bahasa pemrograman C, dapat digunakan tipe data array maupun *linked list*. Namun, dalam praktikum ini, hanya akan dibahas pengimplementasian Queue dalam bentuk *linked list*.

Implementasi Queue menggunakan *linked list* sebenarnya tidak jauh berbeda dengan operasi pada list biasa, bahkan cenderung lebih sederhana. Hal ini sesuai dengan sifat FIFO, di mana proses *delete* hanya dilakukan pada bagian *Head* (depan list) dan proses *insert* selalu dilakukan pada bagian *Tail* (belakang list), atau sebaliknya, tergantung pada persepsi yang digunakan. Dalam penerapannya, Queue dapat diimplementasikan menggunakan *single linked list* maupun *double linked list*.

III. GUIDED

1. Guided 1

```cpp
#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";

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

    return 0;
}

```

Hasil run:

```
PS D:\Praktukum\C++\Modul8> cd "d:\Praktukum\C++\Modul8\" ;
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
PS D:\Praktukum\C++\Modul8>
```

2. Guided 2

```cpp
#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";

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

    return 0;
}

```

Hasil Run program:

```
PS D:\Praktukum\C++\Modul8> cd "d:\Praktukum\C++\Modul8\" ;
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
```

3. Guided 3

```cpp
#include<iostream>

using namespace std;

const int maksimalQueue = 5; // Maksimal antrian
int front = 0; // Penanda antrian
int back = 0; // Penanda
string queueTeller[5]; // Fungsi pengecekan

bool isFull() { // Pengecekan antrian penuh atau tidak
    if (back == maksimalQueue) {
        return true; // =1
    } else {
        return false;
    }
}

bool isEmpty() { // Antriannya kosong atau tidak
    if (back == 0) {
        return true;
    } else {
        return false;
    }
}

void enqueueAntrian(string data) { // Fungsi menambahkan antrian
    if (isFull()) {
        cout << "Antrian penuh" << endl;
    } else {
        if (isEmpty()) { // Kondisi ketika queue kosong
            queueTeller[0] = data;
            front++;
            back++;
        } else { // Antrianya ada isi
            queueTeller[back] = data;
            back++;
        }
    }
}

void dequeueAntrian() { // Fungsi mengurangi antrian
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        for (int i = 0; i < back; i++) {
            queueTeller[i] = queueTeller[i + 1];
        }
        back--;
    }
}

int countQueue() { // Fungsi menghitung banyak antrian
    return back;
}

void clearQueue() { // Fungsi menghapus semua antrian
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        for (int i = 0; i < back; i++) {
            queueTeller[i] = "";
        }
        back = 0;
        front = 0;
    }
}

void viewQueue() { // Fungsi melihat antrian
    cout << "Data antrian teller:" << endl;
    for (int i = 0; i < maksimalQueue; i++) {
        if (queueTeller[i] != "") {
            cout << i + 1 << ". " << queueTeller[i] << endl;
        } else {
            cout << i + 1 << ". (kosong)" << endl;
        }
    }
}

int main() {
    enqueueAntrian("Andi");
    enqueueAntrian("Maya");

    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;

    dequeueAntrian();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;

    clearQueue();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;

    return 0;
}
```

Hasil run:

```
PS D:\Praktukum\C++\Modul8> cd "d:\Praktukum\C++\Modul8\" ;
Data antrian teller:
1. Andi
2. Maya
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 2
Data antrian teller:
1. Maya
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 1
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
PS D:\Praktukum\C++\Modul8>
```

IV. UNGUIDED

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

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

// Mendefinisikan struktur node untuk data antrian
struct Node {
    string data;     // Data yang disimpan (misalnya nama mahasiswa atau data lainnya)
    Node* next;      // Pointer ke node berikutnya
};

// Mendefinisikan kelas Antrian dengan linked list
class Queue {
private:
    Node* front;     // Menunjuk ke elemen pertama dalam antrian
    Node* back;      // Menunjuk ke elemen terakhir dalam antrian

public:
    // Konstruktor untuk inisialisasi antrian
    Queue() {
        front = nullptr;
        back = nullptr;
    }

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

    // Fungsi untuk menambahkan data ke antrian (enqueue)
    void enqueue(string data) {
        Node* newNode = new Node; // Membuat node baru
        newNode->data = data;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode;  // Jika antrian kosong, node baru menjadi front dan back
            back = newNode;
        } else {
            back->next = newNode;  // Menambahkan node baru di belakang
            back = newNode;        // Update back ke node baru
        }
    }

    // Fungsi untuk mengeluarkan data dari antrian (dequeue)
    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Node* temp = front;
            front = front->next;  // Menggeser front ke node berikutnya
            delete temp;          // Menghapus node lama
            if (front == nullptr) {
                back = nullptr;   // Jika antrian kosong, update back menjadi null
            }
        }
    }

    // Fungsi untuk melihat data dalam antrian
    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Node* current = front;
            cout << "Data antrian:" << endl;
            int position = 1;
            while (current != nullptr) {
                cout << position << ". " << current->data << endl;
                current = current->next;
                position++;
            }
        }
    }

    // Fungsi untuk menghitung jumlah data dalam antrian
    int countQueue() {
        int count = 0;
        Node* current = front;
        while (current != nullptr) {
            count++;
            current = current->next;
        }
        return count;
    }

    // Fungsi untuk menghapus semua data dalam antrian
    void clearQueue() {
        while (!isEmpty()) {
            dequeue();  // Menghapus data satu per satu
        }
        cout << "Antrian telah dikosongkan." << endl;
    }
};

int main() {
    Queue q;
    int pilihan;
    string data;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore(); // Untuk membersihkan newline setelah input pilihan menu

        switch (pilihan) {
            case 1:
                cout << "Masukkan data: ";
                getline(cin, data);
                q.enqueue(data);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.viewQueue();
                break;
            case 4:
                q.clearQueue();
                break;
            case 5:
                cout << "Jumlah antrian: " << q.countQueue() << endl;
                break;
            case 6:
                cout << "Terima kasih!" << endl;
                return 0;
            default:
                cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
        }
    }
}
```

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

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

// Mendefinisikan struktur node untuk data antrian mahasiswa
struct Mahasiswa {
    string nama;     // Nama mahasiswa
    string nim;      // NIM mahasiswa
    Mahasiswa* next; // Pointer ke node berikutnya
};

// Mendefinisikan kelas Antrian dengan linked list
class Queue {
private:
    Mahasiswa* front;  // Menunjuk ke elemen pertama dalam antrian
    Mahasiswa* back;   // Menunjuk ke elemen terakhir dalam antrian

public:
    // Konstruktor untuk inisialisasi antrian
    Queue() {
        front = nullptr;
        back = nullptr;
    }

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

    // Fungsi untuk menambahkan data ke antrian (enqueue) dengan prioritas berdasarkan NIM
    void enqueue(string nama, string nim) {
        Mahasiswa* newNode = new Mahasiswa; // Membuat node baru
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

        if (isEmpty()) {
            // Jika antrian kosong, node baru menjadi front dan back
            front = newNode;
            back = newNode;
        } else {
            // Jika antrian tidak kosong, kita cari posisi yang sesuai berdasarkan NIM
            if (stoi(nim) < stoi(front->nim)) {
                // Jika NIM lebih kecil dari NIM di depan, insert di depan
                newNode->next = front;
                front = newNode;
            } else {
                // Mencari posisi yang sesuai di antrian
                Mahasiswa* current = front;
                while (current->next != nullptr && stoi(nim) > stoi(current->next->nim)) {
                    current = current->next;
                }
                // Menyisipkan node baru setelah node current
                newNode->next = current->next;
                current->next = newNode;

                // Jika node yang ditambahkan berada di belakang, update back
                if (newNode->next == nullptr) {
                    back = newNode;
                }
            }
        }
    }

    // Fungsi untuk mengeluarkan data dari antrian (dequeue)
    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* temp = front;
            front = front->next;  // Menggeser front ke node berikutnya
            delete temp;          // Menghapus node lama
            if (front == nullptr) {
                back = nullptr;   // Jika antrian kosong, update back menjadi null
            }
        }
    }

    // Fungsi untuk melihat data dalam antrian
    void viewQueue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Mahasiswa* current = front;
            cout << "Data antrian mahasiswa:" << endl;
            int position = 1;
            while (current != nullptr) {
                cout << position << ". Nama: " << current->nama << ", NIM: " << current->nim << endl;
                current = current->next;
                position++;
            }
        }
    }

    // Fungsi untuk menghitung jumlah data dalam antrian
    int countQueue() {
        int count = 0;
        Mahasiswa* current = front;
        while (current != nullptr) {
            count++;
            current = current->next;
        }
        return count;
    }

    // Fungsi untuk menghapus semua data dalam antrian
    void clearQueue() {
        while (!isEmpty()) {
            dequeue();  // Menghapus data satu per satu
        }
        cout << "Antrian telah dikosongkan." << endl;
    }
};

int main() {
    Queue q;
    int pilihan;
    string nama, nim;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore(); // Untuk membersihkan newline setelah input pilihan menu

        switch (pilihan) {
            case 1:
                cout << "Masukkan Nama Mahasiswa: ";
                getline(cin, nama);
                cout << "Masukkan NIM Mahasiswa: ";
                getline(cin, nim);
                q.enqueue(nama, nim);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.viewQueue();
                break;
            case 4:
                q.clearQueue();
                break;
            case 5:
                cout << "Jumlah antrian: " << q.countQueue() << endl;
                break;
            case 6:
                cout << "Terima kasih!" << endl;
                return 0;
            default:
                cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
        }
    }
}
```

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

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

// Node untuk menyimpan data mahasiswa
struct Node {
    string namaMahasiswa;
    string nimMahasiswa;
    Node* next;
};

// Pointer untuk mengelola queue
Node* front = nullptr; // Awal antrian

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

void enqueueAntrian(string nama, string nim) {
    // Fungsi menambahkan data dengan prioritas NIM
    Node* newNode = new Node();
    newNode->namaMahasiswa = nama;
    newNode->nimMahasiswa = nim;
    newNode->next = nullptr;

    if (isEmpty() || nim < front->nimMahasiswa) {
        // Jika queue kosong atau prioritas lebih tinggi
        newNode->next = front;
        front = newNode;
    } else {
        // Sisipkan di posisi yang sesuai
        Node* current = front;
        while (current->next != nullptr && current->next->nimMahasiswa < nim) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
    cout << "Mahasiswa (" << nama << ", " << nim << ") berhasil ditambahkan ke antrian." << endl;
}

void dequeueAntrian() { // Fungsi menghapus data dari queue
    if (isEmpty()) {
        cout << "Antrian kosong, tidak ada yang dapat dihapus." << endl;
    } else {
        Node* temp = front;
        front = front->next;
        cout << "Mahasiswa (" << temp->namaMahasiswa << ", " << temp->nimMahasiswa << ") telah keluar dari antrian." << endl;
        delete temp;
    }
}

void viewQueue() { // Fungsi untuk melihat data pada queue
    if (isEmpty()) {
        cout << "Antrian kosong." << endl;
    } else {
        cout << "Data antrian berdasarkan prioritas NIM:" << endl;
        Node* temp = front;
        int nomor = 1;
        while (temp != nullptr) {
            cout << nomor++ << ". Nama: " << temp->namaMahasiswa
                 << ", NIM: " << temp->nimMahasiswa << endl;
            temp = temp->next;
        }
    }
}

void clearQueue() { // Fungsi untuk menghapus semua data pada queue
    while (!isEmpty()) {
        dequeueAntrian();
    }
    cout << "Semua data dalam antrian telah dihapus." << endl;
}

int main() {
    int pilihan;
    do {
        cout << "\nMenu Antrian Mahasiswa dengan Prioritas NIM:\n";
        cout << "1. Tambah Antrian (Enqueue)\n";
        cout << "2. Hapus Antrian (Dequeue)\n";
        cout << "3. Lihat Antrian\n";
        cout << "4. Kosongkan Antrian\n";
        cout << "5. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore(); // Membersihkan input buffer

        switch (pilihan) {
            case 1: {
                string nama, nim;
                cout << "Masukkan nama mahasiswa: ";
                getline(cin, nama);
                cout << "Masukkan NIM mahasiswa: ";
                getline(cin, nim);
                enqueueAntrian(nama, nim);
                break;
            }
            case 2:
                dequeueAntrian();
                break;
            case 3:
                viewQueue();
                break;
            case 4:
                clearQueue();
                break;
            case 5:
                cout << "Program selesai." << endl;
                break;
            default:
                cout << "Pilihan tidak valid, silakan coba lagi." << endl;
        }
    } while (pilihan != 5);

    return 0;
}
```

V. KESIMPULAN

Queue adalah salah satu struktur data yang merepresentasikan konsep antrian, di mana elemen yang pertama masuk akan menjadi elemen pertama yang diproses, sesuai dengan prinsip FIFO (First In, First Out). Konsep ini mirip dengan antrian pada loket tiket, di mana pelanggan yang datang lebih awal dilayani terlebih dahulu. Dalam pemrograman C, Queue dapat diimplementasikan menggunakan array atau linked list. Namun, implementasi dengan linked list memiliki keunggulan karena lebih fleksibel dan efisien dalam memanfaatkan memori. Operasi dasar Queue, seperti memasukkan data (enqueue) dan menghapus data (dequeue), dilakukan pada bagian belakang dan depan list, sehingga mempermudah pengelolaan data. Penggunaan linked list dalam Queue memungkinkan pengelolaan data secara dinamis, baik melalui single linked list maupun double linked list, menjadikannya pilihan yang sederhana namun sangat fungsional untuk berbagai kebutuhan praktis.