LAPORAN PRAKTIKUM

Modul 08

“Queue”

Disusun Oleh:

Marvel Sanjaya Setiawan (2311104053)

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. **Tujuan**
    - Memahami konsep dasar queue.
    - Melakukan operasi tambah dan hapus pada queue.
    - Menampilkan data dari queue.
2. **Landasan Teori**

    Stack

    Queue adalah struktur data dengan metode FIFO (First-In First-Out). Data yang masuk pertama akan keluar pertama. Queue diimplementasikan menggunakan array atau linked list dan terdiri dari dua pointer: front (elemen pertama) dan rear (elemen terakhir).

    Perbedaan Stack dan Queue:

    - Stack: LIFO (Last In, First Out). Operasi dilakukan di satu ujung (top).
    - Queue: FIFO (First In, First Out). Operasi dilakukan di dua ujung (front dan rear).

    Operasi pada Queue:

    a. enqueue(): Menambahkan data ke belakang queue.

    b. dequeue(): Mengeluarkan data dari depan queue.

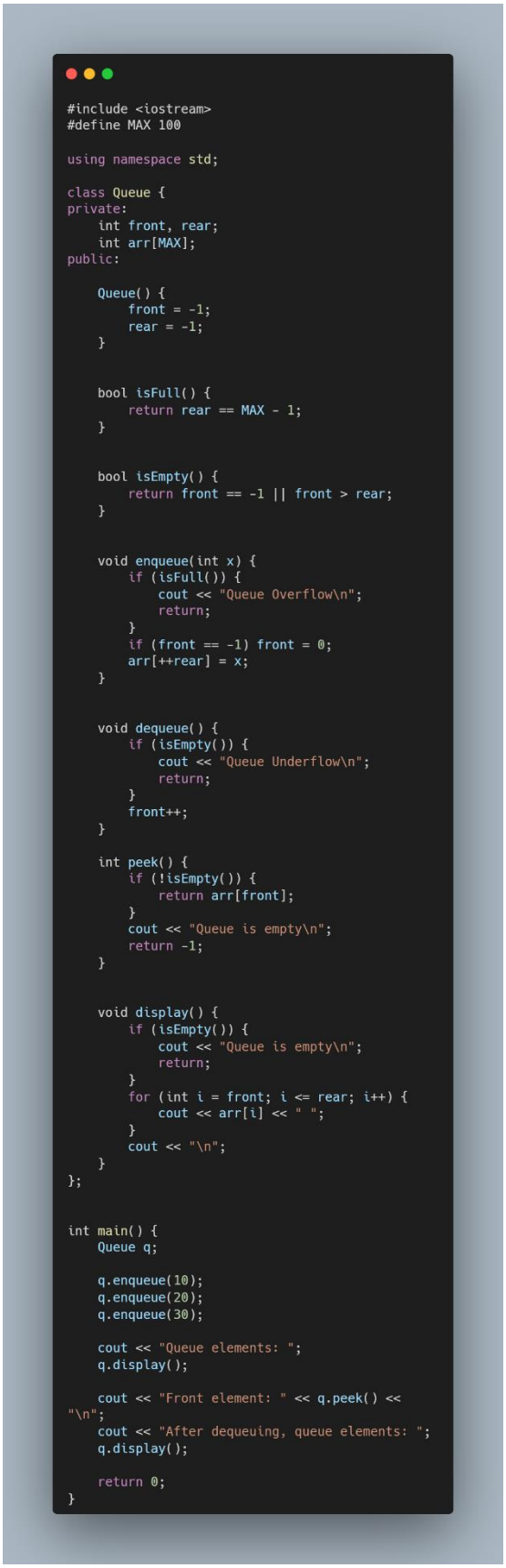
    c. peek(): Mengambil data dari depan tanpa menghapusnya.

    d. isEmpty(): Mengecek apakah queue kosong.

    e. isFull(): Mengecek apakah queue penuh.

    f. size(): Menghitung jumlah elemen dalam queue.

3. Guided



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

    return 0;
}
```

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
```

Cara Kerja:

1. Inisialisasi: Queue() menginisialisasi front dan rear ke -1.
2. Memeriksa Status: isFull() periksa rear, isEmpty() periksa front dan rear.
3. Menambahkan Elemen: enqueue(x) tambahkan elemen ke belakang queue.
4. Menghapus Elemen: dequeue() hapus elemen dari depan queue.
5. Melihat Elemen Depan: peek() kembalikan elemen depan tanpa menghapus.
6. Menampilkan Queue: display() tampilkan semua elemen queue.

Telkom University Purwokerto

```cpp
#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;
}
```

Telkom University Purwokerto

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
```

Cara Kerja:

1. Inisialisasi: Queue() set front dan rear ke nullptr.
2. Periksa Status: isEmpty() cek front.
3. Tambah Elemen: enqueue(x) buat node baru, update rear.
4. Hapus Elemen: dequeue() hapus node depan, update front.
5. Lihat Depan: peek() kembalikan data depan.
6. Tampilkan Queue: display() cetak semua elemen.

Telkom University Purwokerto

```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;
}
```

```
Data antrian teller:
1. Andi
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
Antrian kosong
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
```

Cara Kerja:

1. Inisialisasi: Queue() set front dan rear ke nullptr.
2. Periksa Status: isEmpty() cek front.
3. Tambah Elemen: enqueue(x) buat node baru, update rear.
4. Hapus Elemen: dequeue() hapus node depan, update front.
5. Lihat Depan: peek() kembalikan data depan.
6. Tampilkan Queue: display() cetak semua elemen.

4. Unguided

1. Mengubah Queue menjadi Linked List.

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

struct Node {
    string data;
    Node* next;
};

class Queue {
private:
    Node* front;
    Node* back;
    int size;

public:
    Queue() {
        front = nullptr;
        back = nullptr;
        size = 0;
    }

    bool isFull() {
        return false; // Linked list tidak memiliki batas ukuran tetap
    }

    bool isEmpty() {
        return size == 0;
    }

    void enqueue(string data) {
        Node* newNode = new Node();
        newNode->data = data;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            back->next = newNode;
            back = newNode;
        }
        size++;
    }

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

    int count() {
        return size;
    }

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

    void view() {
        cout << "Isi queue: ";
        Node* current = front;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    Queue queue;

    // Input data oleh user
    int jumlahData;
    cout << "Masukkan jumlah nama: ";
    cin >> jumlahData;
    cin.ignore(); // Membersihkan buffer newline

    for (int i = 0; i < jumlahData; i++) {
        string nama;
        cout << "Masukkan nama ke-" << i + 1 << ": ";
        getline(cin, nama);
        queue.enqueue(nama);
    }

    cout << "\nData dalam queue setelah input:\n";
    queue.view();

    return 0;
}
```

```
Masukkan jumlah nama: 3
Masukkan nama ke-1: Marvel
Masukkan nama ke-2: Ikhsan
Masukkan nama ke-3: Rizal

Data dalam queue setelah input:
Isi queue: Marvel Ikhsan Rizal
```

Cara Kerja:

1. Inisialisasi: Queue() set front dan rear ke nullptr.
2. Periksa Status: isEmpty() cek front.
3. Tambah Elemen: enqueue(x) buat node baru, update rear.
4. Hapus Elemen: dequeue() hapus node depan, update front.
5. Lihat Depan: peek() kembalikan data depan.
6. Tampilkan Queue: display() cetak semua elemen.

2. Membuat pembalikan terhadap kalimat.

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

struct Mahasiswa {
    string nama;
    long long nim;
};

struct Node {
    Mahasiswa data;
    Node* next;
};

class Queue {
private:
    Node* front;
    Node* back;
    int size;

public:
    Queue() {
        front = nullptr;
        back = nullptr;
        size = 0;
    }

    bool isFull() {
        return false; // Linked list tidak memiliki batas ukuran tetap
    }

    bool isEmpty() {
        return size == 0;
    }

    void enqueue(string nama, long long nim) {
        Node* newNode = new Node();
        newNode->data.nama = nama;
        newNode->data.nim = nim;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            back->next = newNode;
            back = newNode;
        }
        size++;
    }

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

    int count() {
        return size;
    }

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

    void view() {
        cout << "\nData mahasiswa dalam queue:\n" << endl;
        Node* current = front;
        int index = 1;
        while (current != nullptr) {
            cout << index << ". Nama: " << current->data.nama << ", NIM: " << current->data.nim <<
endl;       current = current->next;
            index++;
        }
    }
};

int main() {
    Queue queue;

    // Input data mahasiswa oleh user
    int jumlahMahasiswa;
    cout << "Masukkan jumlah mahasiswa: ";
    cin >> jumlahMahasiswa;
    cin.ignore(); // Membersihkan buffer newline

    for (int i = 0; i < jumlahMahasiswa; i++) {
        string nama;
        long long nim;
        cout << "Masukkan Nama Mahasiswa: ";
        getline(cin, nama);
        cout << "Masukkan NIM Mahasiswa: ";
        cin >> nim;
        cin.ignore(); // Membersihkan buffer newline

        queue.enqueue(nama, nim);
    }

    queue.view();

    cout << "\nJumlah antrian = " << queue.count() << endl;
    return 0;
}
```

```
Masukkan jumlah mahasiswa: 2
Masukkan Nama Mahasiswa: Marvel
Masukkan NIM Mahasiswa: 2311104053
Masukkan Nama Mahasiswa: Rizaldy
Masukkan NIM Mahasiswa: 2311104051

Data mahasiswa dalam queue:

1. Nama: Marvel, NIM: 2311104053
2. Nama: Rizaldy, NIM: 2311104051

Jumlah antrian = 2
```

Cara Kerja:

1. Inisialisasi: Queue() set front dan back ke nullptr, size ke 0.
2. Cek Status: isFull() selalu false, isEmpty() cek size.
3. Tambah Elemen: enqueue(nama, nim) tambah node baru di back, update size.
4. Hapus Elemen: dequeue() hapus node front, kurangi size.
5. Hitung Elemen: count() kembalikan size.
6. Hapus Semua: clear() panggil dequeue() sampai kosong.
7. Tampilkan Elemen: view() cetak elemen dari front ke back.
8. Fungsi Utama: Input nama dan NIM, enqueue(), view(), count()..

Telkom University Purwokerto

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

struct Mahasiswa {
    string nama;
    long long nim;
};

struct Node {
    Mahasiswa data;
    Node* next;
};

class Queue {
private:
    Node* front;
    Node* back;
    int size;

public:
    Queue() {
        front = nullptr;
        back = nullptr;
        size = 0;
    }

    bool isFull() {
        return false; // Linked list tidak memiliki batas ukuran tetap
    }

    bool isEmpty() {
        return size == 0;
    }

    void enqueue(string nama, long long nim) {
        Node* newNode = new Node();
        newNode->data.nama = nama;
        newNode->data.nim = nim;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            if (newNode->data.nim < front->data.nim) {
                newNode->next = front;
                front = newNode;
            } else {
                Node* current = front;
                while (current->next != nullptr && current->next->data.nim < newNode->data.nim) {
                    current = current->next;
                }
                newNode->next = current->next;
                current->next = newNode;
                if (newNode->next == nullptr) {
                    back = newNode;
                }
            }
        }
        size++;
    }

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

    int count() {
        return size;
    }

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

    void view() {
        cout << "Data mahasiswa dalam queue:" << endl;
        Node* current = front;
        int index = 1;
        while (current != nullptr) {
            cout << index << ". Nama: " << current->data.nama << ", NIM: " << current->data.nim <<
endl;        current = current->next;
            index++;
        }
    }
};

int main() {
    Queue queue;

    // Input data mahasiswa oleh user
    int jumlahMahasiswa;
    cout << "Masukkan jumlah mahasiswa: ";
    cin >> jumlahMahasiswa;
    cin.ignore(); // Membersihkan buffer newline

    for (int i = 0; i < jumlahMahasiswa; i++) {
        string nama;
        long long nim;
        cout << "Masukkan Nama Mahasiswa: ";
        getline(cin, nama);
        cout << "Masukkan NIM Mahasiswa: ";
        cin >> nim;
        cin.ignore(); // Membersihkan buffer newline

        queue.enqueue(nama, nim);
    }

    cout << "\nData mahasiswa setelah diurutkan berdasarkan NIM:\n";
    queue.view();

    cout << "\nJumlah antrian = " << queue.count() << endl;
    return 0;
}
```

```
Masukkan jumlah mahasiswa: 2
Masukkan Nama Mahasiswa: Rizaldy
Masukkan NIM Mahasiswa: 2311104051
Masukkan Nama Mahasiswa: Marvel
Masukkan NIM Mahasiswa: 2311104053

Data mahasiswa setelah diurutkan berdasarkan NIM:
Data mahasiswa dalam queue:
1. Nama: Rizaldy, NIM: 2311104051
2. Nama: Marvel, NIM: 2311104053

Jumlah antrian = 2
```

Cara Kerja:

1. Inisialisasi: Queue() set front dan back ke nullptr, size ke 0.
2. Cek Status: isFull() selalu false, isEmpty() cek size.
3. Tambah Elemen: enqueue(nama, nim) buat node baru dengan nama dan nim. Jika kosong, node baru jadi front dan back. Jika tidak, tambahkan node sesuai urutan NIM.
4. Hapus Elemen: dequeue() hapus node depan, update front, kurangi size.
5. Hitung Elemen: count() kembalikan size.
6. Hapus Semua: clear() panggil dequeue() sampai queue kosong.
7. Tampilkan Elemen: view() cetak elemen dari front ke back.
8. Fungsi Utama: Input nama dan NIM, enqueue(), view(), count().

5. Kesimpulan

Laporan ini menjelaskan konsep dasar queue, perbedaannya dengan stack, serta operasi-operasi dasar pada queue seperti enqueue, dequeue, peek, isEmpty, isFull, dan size. Melalui contoh yang diberikan, laporan ini menggambarkan bagaimana queue dapat diimplementasikan menggunakan array dan linked list, serta menjelaskan cara kerja masing-masing implementasi. Kesimpulan menunjukkan pentingnya memahami dan mengaplikasikan operasi-operasi dasar pada queue untuk manajemen data yang efisien dalam berbagai aplikasi.