LAPORAN PRAKTIKUM

MODUL 8

QUEUE

Disusun Oleh:

Dhiemas Tulus Ikhsan 2311104046

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

I. TUJUAN

a. Mahasiswa mampu menjelaskan definisi dan konsep dari queue

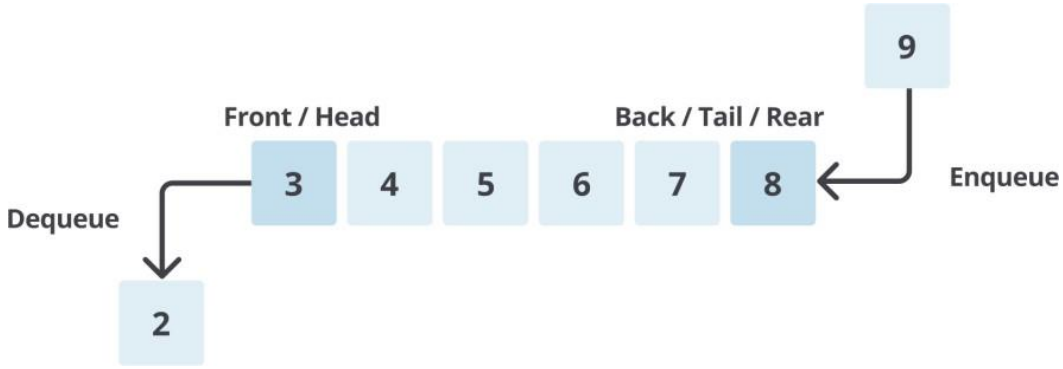
b. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue

c. Mahasiswa mampi menerapkan operasi tampil data pada queue

II. LANDASAN TEORI

Queue adalah struktur data yang menerapkan prinsip FIFO (First-In First-Out), di mana data yang pertama kali masuk akan menjadi data pertama yang keluar. Konsep ini mirip dengan antrian di kehidupan nyata, di mana orang yang datang lebih dulu akan dilayani lebih 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 dalam menambahkan dan menghapus elemen. Pada stack, penambahan dan penghapusan elemen dilakukan di satu sisi yang disebut top (ujung atas). Elemen terakhir yang masuk ke dalam stack berada di posisi atas dan menjadi elemen pertama yang dihapus. Sifat ini dikenal dengan LIFO (Last In, First Out). Analogi sederhana dari stack adalah tumpukan piring, di mana piring yang ditambahkan terakhir berada di atas dan diambil terlebih dahulu.

Sebaliknya, pada queue, elemen ditambahkan di bagian belakang (rear atau tail) dan dihapus dari bagian depan (front atau head). Struktur ini menggunakan prinsip FIFO (First In, First Out), artinya elemen yang masuk lebih dulu akan dikeluarkan lebih dulu. Operasi penambahan elemen disebut Enqueue, sedangkan penghapusan elemen dinamakan 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.

III. GUIDED

1. guided_1

```
#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;
}
```

Output:

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

2. guided_2

```
#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;
}
```

Output:

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

3. guided_3

```
#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;
}
```

Output:

```
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
```

IV. UNGUIDED

1. task_1

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

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

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

Node* front = nullptr;
Node* rear = nullptr;

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

void enqueue(string data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    if (isEmpty()) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << "Data \"" << data << "\" berhasil ditambahkan ke antrian." << endl;
}

void dequeue() {
    if (isEmpty()) {
        cout << "Queue is empty." << endl;
    } else {
```

```
        Node* temp = front;
        cout << "Data \"" << temp->data << "\" telah keluar dari antrian." << endl;
        front = front->next;
        delete temp;
        if (front == nullptr) rear = nullptr;
    }
}

void viewQueue() {
    if (isEmpty()) {
        cout << "Isi antrian: [Kosong]" << endl;
    } else {
        cout << "Isi antrian:\n";
        Node* temp = front;
        while (temp != nullptr) {
            cout << "- " << temp->data << endl;
            temp = temp->next;
        }
    }
}

int main() {
    enqueue("Andi");
    enqueue("Maya");
    viewQueue();
    dequeue();
    viewQueue();
    return 0;
}
```

Output:

```
Data "Andi" berhasil ditambahkan ke antrian.
Data "Maya" berhasil ditambahkan ke antrian.
Isi antrian:
- Andi
- Maya
Data "Andi" telah keluar dari antrian.
Isi antrian:
- Maya
```

2. task_2

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

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

// Struktur Node untuk menyimpan data mahasiswa
struct Node {
    string nama;  // Nama mahasiswa
    string nim;   // NIM mahasiswa
    Node* next;   // Pointer ke node berikutnya
};

// Pointer untuk mengelola antrian
Node* front = nullptr; // Node paling depan
Node* rear = nullptr;  // Node paling belakang

// Fungsi untuk memeriksa apakah antrian kosong
bool isEmpty() {
    return front == nullptr;
}
```

```
// Fungsi untuk menambahkan mahasiswa ke antrian
void enqueue(string nama, string nim) {
    Node* newNode = new Node();
    newNode->nama = nama;
    newNode->nim = nim;
    newNode->next = nullptr;

    if (isEmpty()) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << "Mahasiswa bernama \"" << nama << "\" dengan NIM \"" << nim
         << "\" berhasil ditambahkan ke antrian." << endl;
}

// Fungsi untuk menghapus mahasiswa dari antrian
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong, tidak ada mahasiswa yang dapat dikeluarkan." <<
endl;
        return;
    }

    Node* temp = front;
    cout << "Mahasiswa bernama \"" << temp->nama << "\" dengan NIM \"" << temp-
>nim
         << "\" telah keluar dari antrian." << endl;
    front = front->next;

    if (front == nullptr) {
        rear = nullptr; // Jika antrian menjadi kosong
    }
    delete temp;
}

// Fungsi untuk menampilkan isi antrian
void viewQueue() {
    if (isEmpty()) {
        cout << "Isi antrian: [Kosong]" << endl;
        return;
    }

    cout << "Daftar mahasiswa dalam antrian:\n";
    Node* temp = front;
    while (temp != nullptr) {
        cout << "- Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
        temp = temp->next;
    }
}

// Fungsi utama untuk menjalankan program
int main() {
    enqueue("Andi", "2311101234");
    enqueue("Maya", "2023105678");
    enqueue("Siti", "2311140456");
    viewQueue();
    dequeue();
    viewQueue();
    return 0;
}
```

Output:

```
Mahasiswa bernama "Andi" dengan NIM "2311101234" berhasil ditambahkan ke antrian.
Mahasiswa bernama "Maya" dengan NIM "2023105678" berhasil ditambahkan ke antrian.
Mahasiswa bernama "Siti" dengan NIM "2311140456" berhasil ditambahkan ke antrian.
Daftar mahasiswa dalam antrian:
- Nama: Andi, NIM: 2311101234
- Nama: Maya, NIM: 2023105678
- Nama: Siti, NIM: 2311140456
Mahasiswa bernama "Andi" dengan NIM "2311101234" telah keluar dari antrian.
Daftar mahasiswa dalam antrian:
- Nama: Maya, NIM: 2023105678
- Nama: Siti, NIM: 2311140456
```

3. task_3

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

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm> // Tambahkan header ini untuk sort
using namespace std;

struct Node {
    string nama;
    string nim;
    Node* next;
};

Node* front = nullptr;
Node* rear = nullptr;

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

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

    if (isEmpty()) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }

    cout << "Mahasiswa dengan Nama: " << nama << " dan NIM: " << nim << " berhasil
ditambahkan ke antrian." << endl;
}

void dequeue() {
    if (isEmpty()) {
        cout << "Queue is empty. Tidak ada mahasiswa yang dapat dikeluarkan." << endl;
    } else {
        Node* temp = front;
        cout << "Mahasiswa dengan Nama: " << temp->nama << " dan NIM: " << temp->nim
<< " telah dikeluarkan dari antrian." << endl;
        front = front->next;
        delete temp;
        if (front == nullptr) rear = nullptr; // Jika queue menjadi kosong
    }
}
```

```
void viewQueue() {
    if (isEmpty()) {
        cout << "Queue is empty. Tidak ada mahasiswa dalam antrian." << endl;
    } else {
        cout << "Daftar mahasiswa dalam antrian (prioritas berdasarkan NIM):\n";

        // Salin elemen-elemen queue ke dalam vektor untuk diurutkan
        Node* temp = front;
        vector<pair<string, string>> sortedQueue;
        while (temp != nullptr) {
            sortedQueue.push_back({temp->nim, temp->nama});
            temp = temp->next;
        }

        // Urutkan berdasarkan NIM menggunakan fungsi sort
        sort(sortedQueue.begin(),  sortedQueue.end(),  [](pair<string,  string>  a,
pair<string, string> b) {
            return a.first < b.first; // Urutkan berdasarkan NIM
        });

        // Tampilkan elemen yang sudah diurutkan
        for (auto& item : sortedQueue) {
            cout << "Nama: " << item.second << ", NIM: " << item.first << endl;
        }
    }
}

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

    do {
        cout << "\nMenu Queue:\n";
        cout << "1. Tambah mahasiswa (enqueue)\n";
        cout << "2. Hapus mahasiswa (dequeue)\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;

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

    return 0;
}
```

Output:

```
Menu Queue:
1. Tambah mahasiswa (enqueue)
2. Hapus mahasiswa (dequeue)
3. Lihat antrian
4. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: Dhiemas Tulus Ikhsan
Masukkan NIM Mahasiswa: 2311104046
Mahasiswa dengan Nama: Dhiemas Tulus Ikhsan dan NIM: 2311104046 berhasil ditambahkan ke antrian.

Menu Queue:
1. Tambah mahasiswa (enqueue)
2. Hapus mahasiswa (dequeue)
3. Lihat antrian
4. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: Barbara Millicent
Masukkan NIM Mahasiswa: 2311104095
Mahasiswa dengan Nama: Barbara Millicent dan NIM: 2311104095 berhasil ditambahkan ke antrian.
```

```
Menu Queue:
1. Tambah mahasiswa (enqueue)
2. Hapus mahasiswa (dequeue)
3. Lihat antrian
4. Keluar
Pilih menu: 2
Mahasiswa dengan Nama: Dhiemas Tulus Ikhsan dan NIM: 2311104046 telah dikeluarkan dari antrian.

Menu Queue:
1. Tambah mahasiswa (enqueue)
2. Hapus mahasiswa (dequeue)
3. Lihat antrian
4. Keluar
Pilih menu: 3
Daftar mahasiswa dalam antrian (prioritas berdasarkan NIM):
Nama: Barbara Millicent, NIM: 2311104095

Menu Queue:
1. Tambah mahasiswa (enqueue)
2. Hapus mahasiswa (dequeue)
3. Lihat antrian
4. Keluar
Pilih menu: 4
Keluar dari program.
```

V. KESIMPULAN

Kesimpulan dari praktikum queue dalam C++ adalah bahwa queue merupakan struktur data yang bekerja berdasarkan prinsip FIFO (First In, First Out), di mana elemen yang dimasukkan terlebih dahulu akan menjadi elemen pertama yang dikeluarkan. Dalam implementasi queue, terdapat dua operasi utama, yaitu **Enqueue**, yang digunakan untuk menambahkan elemen ke bagian belakang antrian, dan **Dequeue**, yang digunakan untuk menghapus elemen dari bagian depan. Struktur data ini sangat berguna dalam berbagai aplikasi, seperti pengelolaan antrian pelanggan, penjadwalan proses dalam sistem operasi, atau pengiriman data dalam jaringan.

Melalui praktikum ini, pemahaman mengenai konsep dasar queue, penerapannya dalam kode C++, dan perbedaan dengan struktur data lain seperti stack menjadi lebih jelas. Selain itu, praktikum ini juga melatih keterampilan dalam mengimplementasikan logika algoritma yang melibatkan manipulasi data secara dinamis. Hal ini menunjukkan bahwa queue tidak hanya penting secara teoritis, tetapi juga sangat aplikatif dalam pemecahan masalah sehari-hari yang memerlukan pengelolaan data secara berurutan.