LAPORAN PRAKTIKUM

MODUL 8

QUEUE

Disusun Oleh:

Rizaldy Aulia Rachman (2311104051)

S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

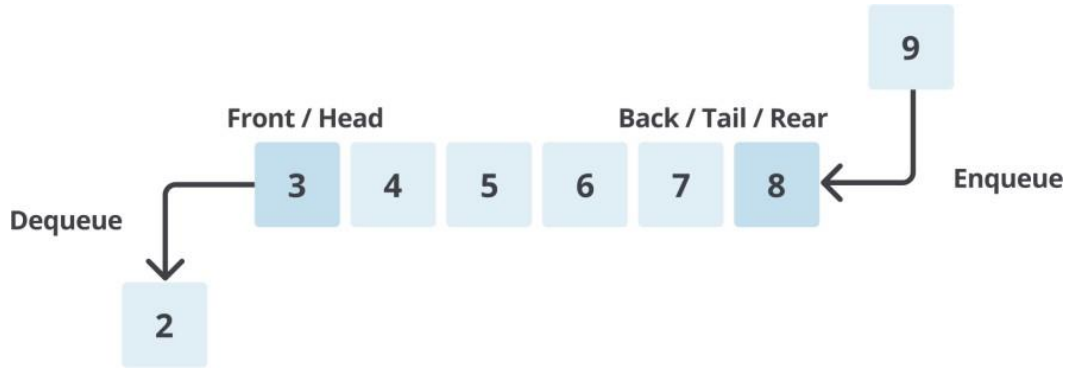
1. Mahasiswa mampu menjelaskan definisi dan konsep dari queue
2. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
3. Mahasiswa mampu menerapkan operasi tampil data pada queue

II. LANDASAN TEORI

2.1 Linked List dengan Pointer

Queue adalah struktur data yang digunakan untuk menyimpan data dengan metode **FIFO** (First-In First-Out). Data yang pertama dimasukkan ke dalam queue akan menjadidata 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.



FIRST IN FIRST OUT (FIFO)

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.

III. GUIDED

1. Guided1

Code:

```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;
}
```

Output:

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
PS C:\Praktikum Struktur data\pertemuan8>
```

2. Guided2

Code:

```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;
}
```

Output:

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
PS C:\Praktikum Struktur data\pertemuan8>
```

3. Guided3

Code:

```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;
}
```

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
PS C:\Praktikum Struktur data\pertemuan8>
```

IV. UNGUIDED

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

Jawaban:

Code:

```cpp
#include <iostream>
#include <string>

using namespace std;

// Struktur Node untuk Linked List
struct Node {
    string data;
    Node* next;
};

// Pointer untuk head dan tail queue
Node* head = nullptr;
Node* tail = nullptr;

// Fungsi mengecek apakah queue kosong
bool isEmpty() {
    return head == nullptr;
}

// Fungsi untuk menambahkan elemen ke queue
void enqueue(string data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;

    if (isEmpty()) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
    cout << "Data \"" << data << "\" berhasil ditambahkan ke antrian.\n";
}

// Fungsi untuk menghapus elemen dari queue
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
        return;
    }

    Node* temp = head;
    head = head->next;
    if (head == nullptr) {
        tail = nullptr;
    }
    cout << "Data \"" << temp->data << "\" telah keluar dari antrian.\n";
    delete temp;
}

// Fungsi untuk menampilkan elemen dalam queue
void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
        return;
    }

    Node* current = head;
    cout << "Isi antrian:\n";
    while (current != nullptr) {
        cout << "- " << current->data << endl;
        current = current->next;
    }
}

// Fungsi utama
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
PS C:\Praktikum Struktur data\pertemuan8>
```

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

Jawaban:

Code:

```cpp
#include <iostream>
#include <string>

using namespace std;

// Struktur Node untuk Linked List
struct Node {
    string nama;
    string nim; // Menggunakan string agar mendukung NIM 10 digit
    Node* next;
};

// Pointer untuk head dan tail queue
Node* head = nullptr;
Node* tail = nullptr;

// Fungsi mengecek apakah queue kosong
bool isEmpty() {
    return head == nullptr;
}

// Fungsi untuk menambahkan mahasiswa ke queue
void enqueue(string nama, string nim) {
    if (nim.length() != 10) {
        cout << "NIM harus berjumlah 10 digit.\n";
        return;
    }

    Node* newNode = new Node();
    newNode->nama = nama;
    newNode->nim = nim;
    newNode->next = nullptr;

    if (isEmpty()) {
        head = tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
    cout << "Mahasiswa " << nama << " dengan NIM " << nim << " berhasil ditambahkan ke antrian.\n";
}

// Fungsi untuk menghapus mahasiswa dari queue
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
        return;
    }

    Node* temp = head;
    head = head->next;
    if (head == nullptr) {
        tail = nullptr;
    }
    cout << "Mahasiswa " << temp->nama << " dengan NIM " << temp->nim << " telah keluar dari antrian.\n";
    delete temp;
}

// Fungsi untuk menampilkan isi antrian mahasiswa
void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
        return;
    }

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

// Fungsi utama
int main() {
    enqueue("Rizaldy Aulia Rachman", "2311104051");
    enqueue("Kenma", "2023105678");
    viewQueue();
    dequeue();
    viewQueue();
    return 0;
}
```

Output:

```
Mahasiswa Rizaldy Aulia Rachman dengan NIM 2311104051 berhasil ditambahkan ke antrian.
Mahasiswa Kenma dengan NIM 2023105678 berhasil ditambahkan ke antrian.
Isi antrian mahasiswa:
Nama: Rizaldy Aulia Rachman, NIM: 2311104051
Nama: Kenma, NIM: 2023105678
Mahasiswa Rizaldy Aulia Rachman dengan NIM 2311104051 telah keluar dari antrian.
Isi antrian mahasiswa:
Nama: Kenma, NIM: 2023105678
PS C:\Praktikum Struktur data\pertemuan8>
```

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

   Noted : Untuk data mahasiswa dan nim dimasukan oleh user

Jawaban:

Code:

```cpp
#include <iostream>
#include <string>

using namespace std;

// Struktur Node untuk Linked List
struct Node {
    string nama;
    string nim;
    Node* next;
};

// Pointer untuk head queue
Node* head = nullptr;

// Fungsi mengecek apakah queue kosong
bool isEmpty() {
    return head == nullptr;
}

// Fungsi untuk menambahkan mahasiswa ke queue dengan prioritas NIM
void enqueue(string nama, string nim) {
    if (nim.length() != 10) {
        cout << "NIM harus berjumlah 10 digit.\n";
        return;
    }

    Node* newNode = new Node();
    newNode->nama = nama;
    newNode->nim = nim;
    newNode->next = nullptr;

    if (isEmpty() || nim < head->nim) {
        newNode->next = head;
        head = newNode;
    } else {
        Node* current = head;
        while (current->next != nullptr && current->next->nim <= nim) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
    cout << "Mahasiswa " << nama << " dengan NIM " << nim << " berhasil ditambahkan ke antrian.\n";
}

// Fungsi untuk menghapus mahasiswa dari queue
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
        return;
    }

    Node* temp = head;
    head = head->next;
    cout << "Mahasiswa " << temp->nama << " dengan NIM " << temp->nim << " telah keluar dari antrian.\n";
    delete temp;
}

// Fungsi untuk menampilkan isi antrian
void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
        return;
    }

    Node* current = head;
    cout << "Isi antrian mahasiswa (prioritas berdasarkan NIM):\n";
    while (current != nullptr) {
        cout << "Nama: " << current->nama << ", NIM: " << current->nim << endl;
        current = current->next;
    }
}

// Fungsi utama
int main() {
    int choice;
    string nama, nim;

    do {
        cout << "\nMenu Queue Mahasiswa:\n";
        cout << "1. Tambah Mahasiswa (Enqueue)\n";
        cout << "2. Hapus Mahasiswa (Dequeue)\n";
        cout << "3. Tampilkan Antrian\n";
        cout << "4. Keluar\n";
        cout << "Pilihan: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan Nama Mahasiswa: ";
                cin.ignore();
                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.\n";
                break;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (choice != 4);

    return 0;
}
```

Output:

```
Menu Queue Mahasiswa:
1. Tambah Mahasiswa (Enqueue)
2. Hapus Mahasiswa (Dequeue)
3. Tampilkan Antrian
4. Keluar
Pilihan: 1
Masukkan Nama Mahasiswa: Rizaldy Aulia Rachman
Masukkan NIM Mahasiswa: 2311104051
Mahasiswa Rizaldy Aulia Rachman dengan NIM 2311104051 berhasil ditambahkan ke antrian.

Menu Queue Mahasiswa:
1. Tambah Mahasiswa (Enqueue)
2. Hapus Mahasiswa (Dequeue)
3. Tampilkan Antrian
4. Keluar
Pilihan: 1
Masukkan Nama Mahasiswa: Kurama
Masukkan NIM Mahasiswa: 2543671127
Mahasiswa Kurama dengan NIM 2543671127 berhasil ditambahkan ke antrian.

Menu Queue Mahasiswa:
1. Tambah Mahasiswa (Enqueue)
2. Hapus Mahasiswa (Dequeue)
3. Tampilkan Antrian
4. Keluar
Pilihan: 2
Mahasiswa Rizaldy Aulia Rachman dengan NIM 2311104051 telah keluar dari antrian.
```

```
Menu Queue Mahasiswa:
1. Tambah Mahasiswa (Enqueue)
2. Hapus Mahasiswa (Dequeue)
3. Tampilkan Antrian
4. Keluar
Pilihan: 3
Isi antrian mahasiswa (prioritas berdasarkan NIM):
Nama: Kurama, NIM: 2543671127

Menu Queue Mahasiswa:
1. Tambah Mahasiswa (Enqueue)
2. Hapus Mahasiswa (Dequeue)
3. Tampilkan Antrian
4. Keluar
Pilihan: 4
Keluar dari program.
PS C:\Praktikum Struktur data\pertemuan8>
```