LAPORAN PRAKTIKUM
Modul 8
QUEUE

Disusun Oleh:

Muhammad Shafiq Rasuna - 2311104043

Kelas :

S1SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd, M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

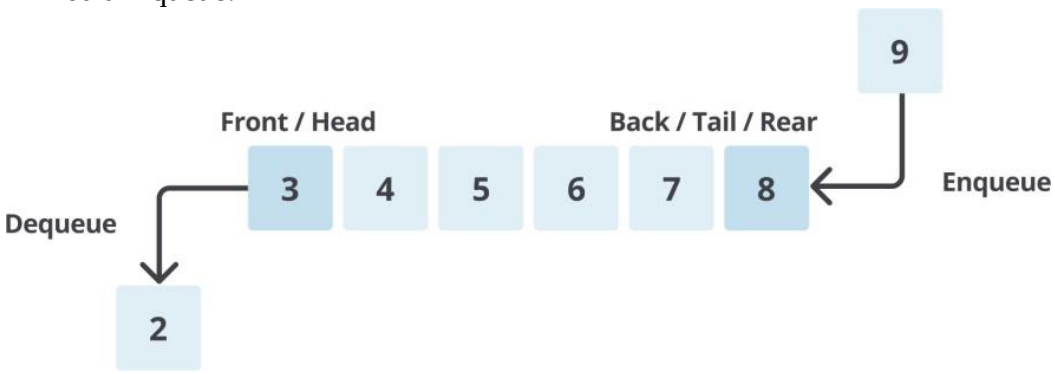
1. Tujuan

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. Dasar 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.



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.

3. Guided 3.1

Kode programnya:

```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;
}
```

Hasil runnya:

```
c:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan8>cd
emuan8\" && g++ guided1.cpp -o guided1 && "c:\Users\ASUS\OneDrive\Dokumen\tugas smt
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
```

Guided 3.2

```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;
    }

```

```cpp

    // 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;
}
```

Hasil runnya:

```
C:\Users\ASUS\OneDrive\Dokumen\tugas smt  3\Pemograman Struktur Data 3\pertemuan8>cd
emuan8\" && g++ guided2.cpp -o guided2 && "c:\Users\ASUS\OneDrive\Dokumen\tugas smt
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
```

Guided 3.3

```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;
}
```

```cpp
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 runnya:

```
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
```

4. Unguided

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

Kode program :

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

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

Node* front = nullptr;
Node* back = nullptr;

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

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

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

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

        if (front == nullptr) {
            back = nullptr;
        }
    }
}

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

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

```cpp
void viewQueue() {
    cout << "Data antrian teller:" << endl;
    if (isEmpty()) {
        cout << "(Antrian kosong)" << endl;
    } else {
        Node* current = front;
        int index = 1;
        while (current != nullptr) {
            cout << index++ << ". " << current->data << endl;
            current = current->next;
        }
    }
}

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 outputnya :

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

Penjelasan Program ;

1. Node:
   Setiap elemen di queue direpresentasikan sebagai sebuah node dengan dua atribut: data (isi) dan next (pointer ke node berikutnya).
2. Front & Back:
   front menunjuk ke elemen pertama di queue, sedangkan back menunjuk ke elemen terakhir.
3. Fungsi Enqueue:
   Membuat node baru dan menambahkannya di akhir queue.
4. Fungsi Dequeue:
   Menghapus node di awal queue dan mengatur front ke node berikutnya.
5. Fungsi Tambahan:
   countQueue menghitung jumlah node.
   clearQueue menghapus semua node.
   viewQueue menampilkan semua elemen di queue.

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

Kode programnya:

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

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

Node* front = nullptr;
Node* back = nullptr;

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

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

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

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

        if (front == nullptr) {
            back = nullptr;
        }
    }
}

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

```cpp
void clearQueue() {
    while (!isEmpty()) {
        dequeueAntrian();
    }
}

void viewQueue() {
    cout << "Data antrian mahasiswa:" << endl;
    if (isEmpty()) {
        cout << "(Antrian kosong)" << endl;
    } else {
        Node* current = front;
        int index = 1;
        while (current != nullptr) {
            cout << index++ << ". Nama: " << current->nama << ", NIM: " << current->nim << endl;
            current = current->next;
        }
    }
}

int main() {
    enqueueAntrian("Andi", "12345678");
    enqueueAntrian("Maya", "87654321");
    enqueueAntrian("Budi", "13579111");

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

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

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

    return 0;
}

```

Output nya :

```
Data antrian mahasiswa:
1. Nama: Andi, NIM: 12345678
2. Nama: Maya, NIM: 87654321
3. Nama: Budi, NIM: 13579111
Jumlah antrian = 3
Data antrian mahasiswa:
1. Nama: Maya, NIM: 87654321
2. Nama: Budi, NIM: 13579111
Jumlah antrian = 2
Data antrian mahasiswa:
(Antrian kosong)
Jumlah antrian = 0
```

Penjelasan

1. Struktur Node:
   Ditambah atribut nama untuk menyimpan nama mahasiswa.
   Ditambah atribut nim untuk menyimpan NIM mahasiswa.
2. Fungsi enqueueAntrian:
   Menambahkan node baru dengan atribut nama dan nim.
3. Fungsi viewQueue:
   Menampilkan data mahasiswa (nama dan NIM) dalam antrian.
4. Fungsi lainnya:
   Tidak berubah dari implementasi sebelumnya, hanya disesuaikan untuk mendukung struktur data baru.

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

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

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

Node* front = nullptr;

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

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

    if (isEmpty() || front->nim > nim) {
        newNode->next = front;
        front = newNode;
    } else {
        Node* current = front;
        while (current->next != nullptr && current->next->nim <= nim) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
}

```

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

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

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

void viewQueue() {
    cout << "Data antrian mahasiswa berdasarkan prioritas (NIM terkecil di depan):" << endl;
    if (isEmpty()) {
        cout << "(Antrian kosong)" << endl;
    } else {
        Node* current = front;
        int index = 1;
        while (current != nullptr) {
            cout << index++ << ". Nama: " << current->nama << ", NIM: " << current->nim << endl;
            current = current->next;
        }
    }
}
```

```
int main() {
    int choice;
    do {
        cout << "\nMenu Antrian Mahasiswa dengan Prioritas:" << endl;
        cout << "1. Tambah Mahasiswa ke Antrian" << endl;
        cout << "2. Hapus Mahasiswa dari Antrian" << endl;
        cout << "3. Lihat Antrian" << endl;
        cout << "4. Hapus Semua Antrian" << endl;
        cout << "5. Keluar" << endl;
        cout << "Pilih menu: ";
        cin >> choice;
        cin.ignore();

        switch (choice) {
            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();
                cout << "Semua antrian telah dihapus." << endl;
                break;
            case 5:
                cout << "Keluar dari program." << endl;
                break;
            default:
                cout << "Pilihan tidak valid!" << endl;
        }
    } while (choice != 5);

    return 0;
}
```

Output nya:

```
Menu Antrian Mahasiswa dengan Prioritas:
1. Tambah Mahasiswa ke Antrian
2. Hapus Mahasiswa dari Antrian
3. Lihat Antrian
4. Hapus Semua Antrian
5. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: memet
Masukkan NIM Mahasiswa: 23126569

Menu Antrian Mahasiswa dengan Prioritas:
1. Tambah Mahasiswa ke Antrian
2. Hapus Mahasiswa dari Antrian
3. Lihat Antrian
4. Hapus Semua Antrian
5. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: udin
Masukkan NIM Mahasiswa: 69231265

Menu Antrian Mahasiswa dengan Prioritas:
1. Tambah Mahasiswa ke Antrian
2. Hapus Mahasiswa dari Antrian
3. Lihat Antrian
4. Hapus Semua Antrian
5. Keluar
Pilih menu: 3
Data antrian mahasiswa berdasarkan prioritas (NIM terkecil di depan):
1. Nama: memet, NIM: 23126569
2. Nama: udin, NIM: 69231265
```

Penjelasan

1. Prioritas Berdasarkan NIM:
   Saat menambahkan mahasiswa ke antrian (enqueueAntrian), program memeriksa posisi berdasarkan nilai nim (dalam urutan menaik).
   Elemen dimasukkan di tempat yang tepat sehingga NIM terkecil selalu berada di depan.
2. Struktur Node:
   Tidak berubah, hanya digunakan untuk menyimpan nama, nim, dan next.
3. Proses Enqueue:
   Jika antrian kosong atau NIM dari mahasiswa baru lebih kecil daripada mahasiswa di depan, node baru dimasukkan di depan.
   Jika tidak, program mencari posisi yang tepat dan menyisipkan node di tempat tersebut.
4. Fungsi Lainnya:
   dequeueAntrian: Menghapus elemen dengan NIM terkecil (elemen di depan).
   viewQueue: Menampilkan daftar mahasiswa dalam urutan prioritas.

.

5. Kesimpulan

Kesimpulan dari laporan praktikum di atas adalah bahwa struktur data queue merupakan konsep dasar dalam pemrograman yang menggunakan prinsip FIFO (First-In, First-Out), di mana elemen pertama yang masuk akan menjadi elemen pertama yang keluar. Queue dapat diimplementasikan menggunakan array maupun linked list, dengan masing-masing memiliki kelebihan dan kekurangan. Dalam praktiknya, operasi pada queue meliputi *enqueue* untuk menambah elemen di belakang, *dequeue* untuk menghapus elemen di depan, serta fungsi tambahan seperti *peek*, *isEmpty*, dan *isFull*. Dengan mengubah implementasi dari array ke linked list, fleksibilitas queue dapat ditingkatkan, seperti memungkinkan manipulasi yang lebih dinamis tanpa batas