LAPORAN PRAKTIKUM

PERTEMUAN 8

Nama :

Razhendriya Vania Ramadhan Suganjarsarwat (2311104048)

Dosen :

WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

\• Memahami definisi dan konsep queue.
- Menerapkan operasi tambah, hapus, dan menampilkan data pada queue.
- Mengembangkan implementasi queue menggunakan array atau linked list.

II. TOOL

VScode

III. DASAR TEORI

Queue adalah struktur data yang menggunakan metode FIFO (First-In, First-Out). Artinya, data yang pertama masuk akan menjadi data pertama yang keluar. Queue sering digunakan dalam berbagai aplikasi seperti sistem antrean, proses multitasking, atau layanan jaringan.

Pada queue, terdapat dua operasi utama:

- **Enqueue: Menambahkan elemen di akhir queue.**
- **Dequeue: Menghapus elemen dari awal queue.**

Perbedaan antara Stack dan Queue

- **Stack menggunakan prinsip LIFO (Last-In, First-Out), di mana elemen terakhir yang masuk adalah elemen pertama yang keluar.**
- **Queue menggunakan prinsip FIFO, di mana elemen pertama yang masuk adalah elemen pertama yang keluar.**

Operasi tambahan lainnya pada queue:

- **Peek: Melihat elemen di awal queue tanpa menghapusnya.**
- **isEmpty: Mengecek apakah queue kosong.**
- **isFull: Mengecek apakah queue penuh.**
- **Size: Menghitung jumlah elemen di dalam queue.**

IV. 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;
}
```

Menggunakan array dengan ukuran tetap (MAX = 100).
Operasi enqueue menambahkan elemen di belakang, dequeue menghapus elemen dari depan.
Sederhana dan cepat, tetapi ukuran queue terbatas (statis).

```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;
             }
                                  // Tambahkan node baru ke
        rear->next = newNode; 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;
 }
```

Menggunakan node yang terhubung (linked list), sehingga ukuran queue fleksibel.
enqueue menambahkan elemen di belakang, dequeue menghapus elemen dari depan.
Memori lebih efisien untuk data besar, tapi implementasinya lebih kompleks.

```
#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;
 }
```

Array kecil dengan penanda front dan back, serta fitur tambahan seperti menghitung elemen (countQueue) dan membersihkan queue (clearQueue). Operasi dequeue memerlukan pergeseran elemen, sehingga kurang efisien dibanding metode lain.

V. UNGUIDED

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

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

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

// Enqueue: Menambahkan elemen ke dalam queue
void enqueue(string value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;
    if (rear == nullptr) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
       }
 }

// Dequeue: Menghapus elemen dari depan queue
void dequeue() {
    if (front == nullptr) {
        cout << "Queue kosong.\n";
        return;
       }
    Node* temp = front;
    front = front->next;
    if (front == nullptr) {
        rear = nullptr;
       }
    delete temp;
 }

// Menampilkan elemen queue
void displayQueue() {
    Node* temp = front;
    cout << "Isi queue:\n";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
       }
    cout << endl;
 }

int main() {
    enqueue("Data1");
    enqueue("Data2");
    displayQueue();
    dequeue();
    displayQueue();
    return 0;
 }
```

- **Fungsi Utama**: Mengimplementasikan queue untuk menyimpan data string.
- **Operasi**:
    - **Enqueue**: Menambahkan elemen di belakang queue.
    - **Dequeue**: Menghapus elemen dari depan queue.
    - **Display**: Menampilkan semua elemen di queue.
- **Ciri**: Tidak ada prioritas, elemen diproses dalam urutan FIFO (*First In, First Out*).

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

struct Mahasiswa {
    string nama;
    int nim;
    Mahasiswa* next;
};

Mahasiswa* front = nullptr;
Mahasiswa* rear = nullptr;

// Enqueue
void enqueue(string nama, int nim) {
    Mahasiswa* baru = new Mahasiswa();
    baru->nama = nama;
    baru->nim = nim;
    baru->next = nullptr;
    if (rear == nullptr) {
        front = rear = baru;
    } else {
        rear->next = baru;
        rear = baru;
    }
}

// Dequeue
void dequeue() {
    if (front == nullptr) {
        cout << "Queue kosong.\n";
        return;
    }
    Mahasiswa* temp = front;
    front = front->next;
    if (front == nullptr) {
        rear = nullptr;
    }
    delete temp;
}

// Menampilkan Queue
void displayQueue() {
    Mahasiswa* temp = front;
    cout << "Isi queue mahasiswa:\n";
    while (temp != nullptr) {
        cout << temp->nama << " (" << temp->nim << ")\n";
        temp = temp->next;
    }
}

int main() {
    enqueue("Andi", 12345);
    enqueue("Maya", 54321);
    displayQueue();
    dequeue();
    displayQueue();
    return 0;
}
```

- **Fungsi Utama**: Queue untuk menyimpan informasi mahasiswa (nama dan NIM).
- **Operasi**:
    - **Enqueue**: Menambahkan mahasiswa ke belakang queue.
    - **Dequeue**: Menghapus mahasiswa dari depan queue.
    - **Display**: Menampilkan semua mahasiswa dalam queue.
- **Ciri**: Seperti kode pertama, elemen diproses dalam urutan FIFO.

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

struct Mahasiswa {
    string nama;
    int nim;
    Mahasiswa* next;
};

Mahasiswa* front = nullptr;

// Enqueue dengan prioritas
void enqueue(string nama, int nim) {
    Mahasiswa* baru = new Mahasiswa();
    baru->nama = nama;
    baru->nim = nim;
    baru->next = nullptr;

    if (front == nullptr || nim < front->nim) {
        baru->next = front;
        front = baru;
    } else {
        Mahasiswa* temp = front;
        while (temp->next != nullptr && temp->next->nim < nim) {
            temp = temp->next;
        }
        baru->next = temp->next;
        temp->next = baru;
    }
}

// Dequeue
void dequeue() {
    if (front == nullptr) {
        cout << "Queue kosong.\n";
        return;
    }
    Mahasiswa* temp = front;
    front = front->next;
    delete temp;
}

// Menampilkan Queue
void displayQueue() {
    Mahasiswa* temp = front;
    cout << "Isi queue mahasiswa:\n";
    while (temp != nullptr) {
        cout << temp->nama << " (" << temp->nim << ")\n";
        temp = temp->next;
    }
}

int main() {
    enqueue("Andi", 12345);
    enqueue("Maya", 54321);
    enqueue("Budi", 11111);
    displayQueue();
    dequeue();
    displayQueue();
    return 0;
}
```

Fungsi Utama: Queue dengan prioritas berdasarkan nilai NIM (semakin kecil NIM, semakin tinggi prioritas).

Operasi:

- **Enqueue**: Menambahkan mahasiswa ke posisi yang sesuai dengan prioritas.
- **Dequeue**: Menghapus elemen dengan prioritas tertinggi (NIM terkecil).
- **Display**: Menampilkan semua mahasiswa dalam queue.

Ciri: Tidak menganut FIFO, melainkan berdasarkan urutan prioritas (NIM).

VI. KESIMPULAN

Dalam laporan ini, mahasiswa berhasil memahami konsep dasar queue dengan implementasi FIFO, baik menggunakan array maupun linked list. Selain itu, modifikasi prioritas berdasarkan atribut NIM memberikan gambaran nyata tentang penerapan konsep queue dalam kehidupan nyata.