LAPORAN PRAKTIKUM

Modul 08

“QUEUE”

Disusun Oleh:

Faishal Arif Setiawan -2311104066

Kelas:

SE-07-B

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. **Landasan Teori**

   Queue adalah struktur data yang menerapkan prinsip FIFO (First-In, First-Out), di mana elemen pertama yang dimasukkan ke dalam antrian adalah elemen pertama yang dikeluarkan. Konsep ini mirip dengan antrian di kehidupan sehari-hari, seperti antrean di kasir, di mana orang pertama yang datang akan dilayani terlebih dahulu. Operasi dasar dalam queue meliputi enqueue, yang menambahkan elemen ke bagian belakang antrian (rear), dan dequeue, yang menghapus elemen dari bagian depan antrian (front). Queue dapat diimplementasikan menggunakan array atau linked list, masing-masing dengan kelebihan dan kekurangannya.

   Berbeda dengan stack yang menerapkan prinsip LIFO (Last-In, First-Out), di mana elemen terakhir yang dimasukkan adalah yang pertama keluar, queue memproses data secara berurutan sesuai urutan kedatangan. Queue digunakan dalam berbagai aplikasi sehari-hari, seperti sistem antrean, penjadwalan tugas, dan pengolahan data. Struktur data ini sangat berguna untuk memproses data yang membutuhkan urutan eksekusi berdasarkan waktu atau kedatangan, seperti dalam pengolahan permintaan jaringan atau antrian printer.

3. **Guided**

   1.

```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;
}
```

Penjelasan:

-Konstruktor queue digunakan untuk menginisialisasi objek antrian.Saat objek baru dibuat,front dan rear di set ke-1,menandakan bahwa kosong.

-Fungsi isFull() untuk memeriksa apakah antrian sudah penuh.

-Fungsi isEmpty() memeriksa apakah antrian kosong.

-fungsi enqueue menambahkan elemen ke dalam antrian

-Fungsi dequeue di gunakan untuk menghapus elemen pertama dari antrian

-Fungsi peek() mengembalikan elemen pertama di antrian tanpa menghapusnya

-Fungsi display() menampilkan semua elemen dalam antrian

Output:

```
PS D:\struktur data pemograman> cd 'd:\struktur data pemograman\modul8\output'
PS D:\struktur data pemograman\modul8\output> & .\'guided1.exe'
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
```

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;
    }

    // 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;
}
```

Penjelasan:

Fungsi isEmpty():

Mengecek apakah antrian kosong atau tidak. Jika front adalah nullptr, berarti antrian kosong.

Mengembalikan true jika antrian kosong dan false jika tidak.

Fungsi enqueue():

Menambahkan elemen baru ke dalam antrian. Fungsi ini membuat node baru yang menyimpan nilai x.

Jika antrian kosong (baik front dan rear adalah nullptr), maka front dan rear diatur untuk menunjuk ke node baru.

Jika antrian tidak kosong, maka elemen baru ditambahkan di belakang antrian dengan menghubungkannya ke rear->next, dan kemudian memperbarui rear untuk menunjuk ke node baru.

Fungsi dequeue():

Menghapus elemen dari depan antrian (node pertama).

Jika antrian kosong, akan mencetak "Queue Underflow".

Jika antrian tidak kosong, elemen di depan dihapus dengan memindahkan pointer front ke node berikutnya. Jika setelah operasi tersebut antrian menjadi kosong, maka rear juga diatur ke nullptr.

Fungsi peek():

Mengembalikan nilai elemen di depan antrian tanpa menghapusnya.

Jika antrian kosong, fungsi ini akan mencetak "Queue is empty" dan mengembalikan nilai -1.

Fungsi display():
Menampilkan semua elemen dalam antrian, dimulai dari elemen depan hingga elemen belakang.

Jika antrian kosong, fungsi ini mencetak "Queue is empty".

Fungsi ini mengiterasi melalui setiap node, menampilkan data, dan beralih ke node berikutnya.

Output:

```
PS D:\struktur data pemograman\modul8\output> cd 'd:\struktur data pemograman\modul8\output'
PS D:\struktur data pemograman\modul8\output> & .\'guided2.exe'
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
```

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;
}

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;
}
```

Penjelasan:

Fungsi isFull():

Fungsi ini memeriksa apakah antrian sudah penuh dengan cara membandingkan nilai back dengan kapasitas maksimal (maksimalQueue).

Jika back == maksimalQueue, antrian dianggap penuh dan fungsi mengembalikan true, sebaliknya mengembalikan false.

Fungsi isEmpty():

Fungsi ini memeriksa apakah antrian kosong. Jika back == 0, berarti tidak ada elemen dalam antrian, sehingga fungsi ini mengembalikan true, dan jika tidak, mengembalikan false.

Fungsi enqueueAntrian(String data):

Fungsi ini menambahkan elemen baru ke dalam antrian. Jika antrian tidak penuh, data akan ditambahkan pada posisi back, dan kemudian back akan ditingkatkan.

Jika antrian kosong (penanda front == 0), data pertama kali akan dimasukkan ke posisi queueTeller[0] dan penanda front dan back diubah.

Fungsi dequeueAntrian():
Fungsi ini menghapus elemen pertama dalam antrian, yaitu elemen yang ada di posisi front.

Setelah elemen pertama dihapus, semua elemen lainnya akan digeser ke depan, mengurangi indeks antrian, dan mengubah nilai back.

Jika antrian kosong, fungsi ini akan mencetak pesan "Antrian kosong".

Fungsi countQueue():

- Fungsi ini mengembalikan jumlah elemen dalam antrian dengan mengembalikan nilai back, yang merupakan jumlah elemen yang ada.

Fungsi clearQueue():

Fungsi ini menghapus semua elemen dalam antrian. Jika antrian tidak kosong, semua elemen dalam queueTeller dihapus dan front serta back diatur kembali ke nilai 0.

Fungsi viewQueue():
Fungsi ini digunakan untuk menampilkan seluruh elemen dalam antrian.

Jika ada elemen yang kosong di posisi tertentu dalam array, maka akan menampilkan "(kosong)" di posisi tersebut.

Output:

```
PS D:\struktur data pemograman\modul8\output> & .\'guided3.exe'
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
```

4. Unguided

1.

```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{data, 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;
    }
}

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

int main() {
    int pilihan;
    string nama;

    do {
        cout << "Menu:\n1. Tambah Antrian\n2. Hapus Antrian\n3. Lihat Antrian\n4. Keluar\nPilih: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan nama: ";
                cin >> nama;
                enqueueAntrian(nama);
                break;
            case 2:
                dequeueAntrian();
                break;
            case 3:
                viewQueue();
                break;
            case 4:
                cout << "Keluar program.\n";
                break;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (pilihan != 4);

    return 0;
}

```

Output:

```
Masukkan nama: Faishal
Menu:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan nama: Arif
Menu:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 2
Menu:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 3
Data antrian:
1. Arif
```

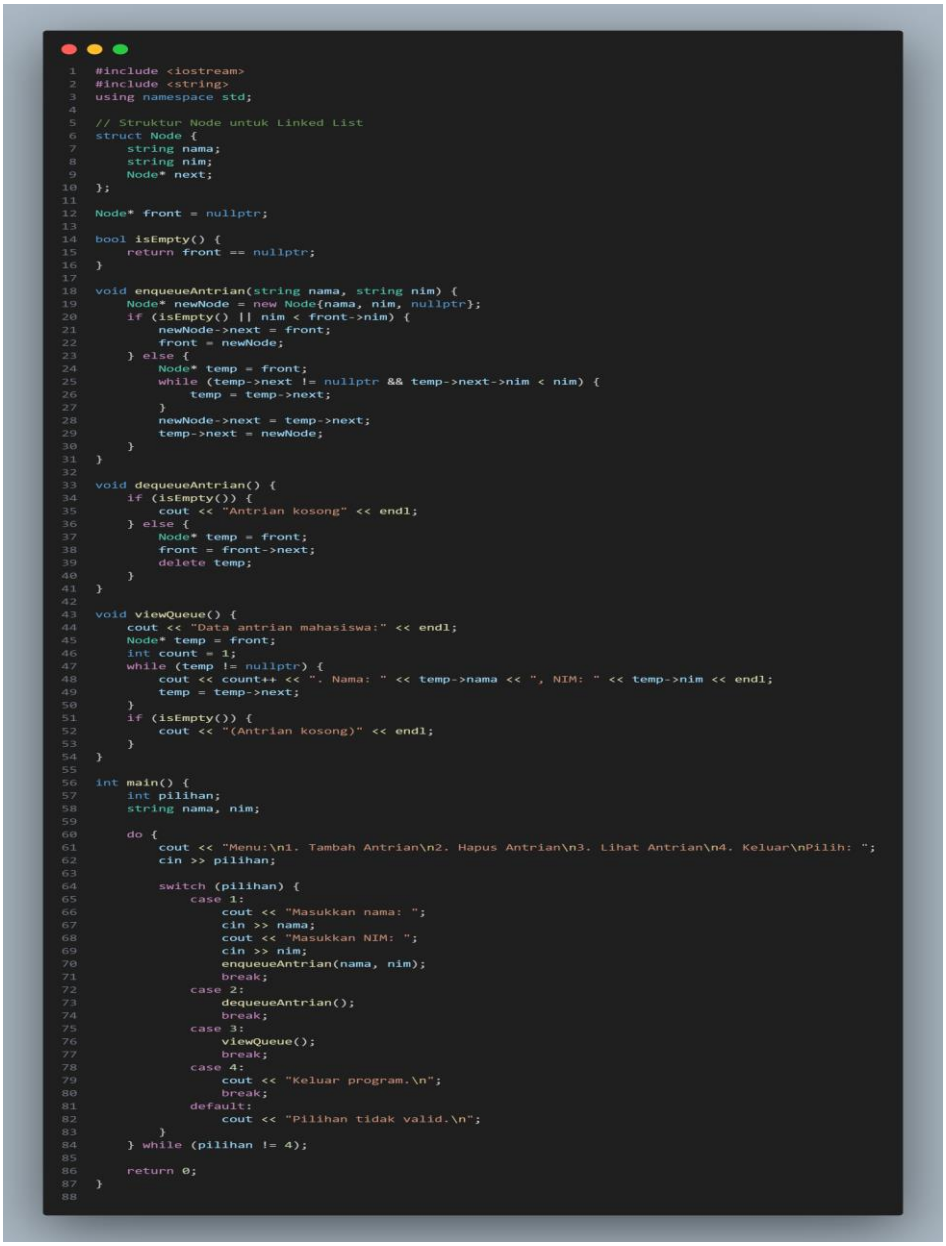
2.

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

// Struktur Node untuk Linked List
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{nama, nim, 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;
    }
}

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

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

    do {
        cout << "Menu:\n1. Tambah Antrian\n2. Hapus Antrian\n3. Lihat Antrian\n4. Keluar\nPilih: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan nama: ";
                cin >> nama;
                cout << "Masukkan NIM: ";
                cin >> nim;
                enqueueAntrian(nama, nim);
                break;
            case 2:
                dequeueAntrian();
                break;
            case 3:
                viewQueue();
                break;
            case 4:
                cout << "Keluar program.\n";
                break;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (pilihan != 4);

    return 0;
}
```

Output:

```
Masukkan nama: Faishal
Masukkan NIM: 2311104066
Menu:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan nama: Arif
Masukkan NIM: 2311104066
Menu:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 3
Data antrian mahasiswa:
1. Nama: Faishal, NIM: 2311104066
2. Nama: Arif, NIM: 2311104066
```

3.



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

// Struktur Node untuk Linked List
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{nama, nim, nullptr};
    if (isEmpty() || nim < front->nim) {
        newNode->next = front;
        front = newNode;
    } else {
        Node* temp = front;
        while (temp->next != nullptr && temp->next->nim < nim) {
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }
}

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

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

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

    do {
        cout << "Menu:\n1. Tambah Antrian\n2. Hapus Antrian\n3. Lihat Antrian\n4. Keluar\nPilih: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan nama: ";
                cin >> nama;
                cout << "Masukkan NIM: ";
                cin >> nim;
                enqueueAntrian(nama, nim);
                break;
            case 2:
                dequeueAntrian();
                break;
            case 3:
                viewQueue();
                break;
            case 4:
                cout << "Keluar program.\n";
                break;
            default:
                cout << "Pilihan tidak valid.\n";
        }
    } while (pilihan != 4);

    return 0;
}

```

Output:

```
Pilih: 1
Masukkan nama: Arif
Masukkan NIM: 04076
Menu:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan nama: Setiawan
Masukkan NIM: 04066
Menu:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Keluar
Pilih: 3
Data antrian mahasiswa:
1. Nama: Setiawan, NIM: 04066
2. Nama: Arif, NIM: 04076
```

5. Kesimpulan

Pada praktikum ini, kita mempelajari implementasi struktur data antrian (queue) dengan berbagai metode, termasuk menggunakan array, linked list, dan teknik manipulasi elemen dalam antrian. Melalui tiga jenis implementasi yang berbeda, kita memahami konsep dasar queue, seperti operasi enqueue untuk menambahkan elemen, dequeue untuk menghapus elemen, serta fungsi tambahan seperti peek untuk melihat elemen pertama tanpa menghapusnya dan display untuk menampilkan semua elemen. Dari praktikum ini, dapat disimpulkan bahwa queue adalah struktur data yang sangat berguna dalam mengelola data secara berurutan, seperti dalam antrian pelanggan, penjadwalan tugas, dan aplikasi lainnya yang memerlukan pemrosesan berdasarkan urutan kedatangan.