LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA

MODUL VII
QUEUE

Disusun Oleh :

Raka Andriy Shevchenko

2311102054

Dosen

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

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

A. Dasar Teori

Queue dalam konteks bahasa pemrograman C++ adalah struktur data linier yang menerapkan prinsip operasi FIFO (First In First Out). Dalam struktur data queue, elemen data yang masuk pertama akan keluar pertama. Queue dapat diterapkan dalam bahasa pemrograman apa pun, seperti C, C++, Java, Python, atau C#, dengan spesifikasi yang hampir sama.

Kelebihan dan Kekurangan Queue:

Kelebihan queue di antarnya:

- Data dalam jumlah besar dapat dikelola secara efisien.
- Operasi seperti penyisipan dan penghapusan dapat dilakukan dengan mudah karena mengikuti aturan masuk pertama keluar pertama.
- Queue berguna ketika layanan tertentu digunakan oleh banyak konsumen.
- Queue cepat untuk komunikasi antar-proses data.
- Queue dapat digunakan dalam implementasi struktur data lainnya.

Kekurangan struktur data queue adalah sebagai berikut:

- Operasi seperti penyisipan dan penghapusan elemen dari tengah cenderung banyak memakan waktu.
- Dalam queue konvensional, elemen baru hanya dapat dimasukkan ketika elemen yang ada dihapus dari antrian.
- Mencari elemen data pada struktur queue membutuhkan time complexity O(N).
- Ukuran maksimum antrian harus ditentukan sebelumnya.

Fungsi dan Kegunaan Queue:

- Queue banyak digunakan untuk menangani lalu lintas (traffic) situs web.
- Membantu untuk mempertahankan playlist yang ada pada aplikasi media player
- Queue digunakan dalam sistem operasi untuk menangani interupsi.
- Membantu dalam melayani permintaan pada satu sumber daya bersama, seperti printer, penjadwalan tugas CPU, dll.
- Digunakan dalam transfer data asinkronus misal pipeline, IO file, dan socket.

B. Guided

a. Guided 1

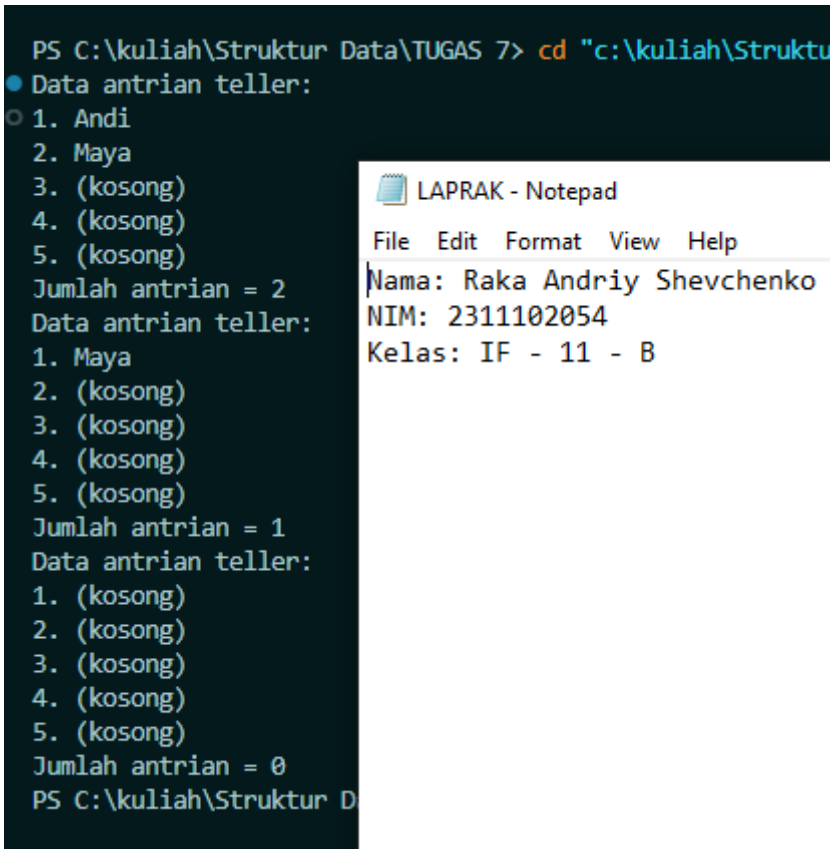
Source Code:

```
#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 koson
            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:



Deskripsi:

Program ini merupakan implementasi dari sistem antrian sederhana menggunakan array di dalam bahasa C++. Program ini memungkinkan penambahan, penghapusan, dan tampilan antrian yang disimulasikan seperti pada teller bank. Program ini menggambarkan bagaimana antrian dapat ditambahkan, dihapus, dihitung, dan ditampilkan menggunakan array statis di C++.

C. Unguided

a. Unguided 1

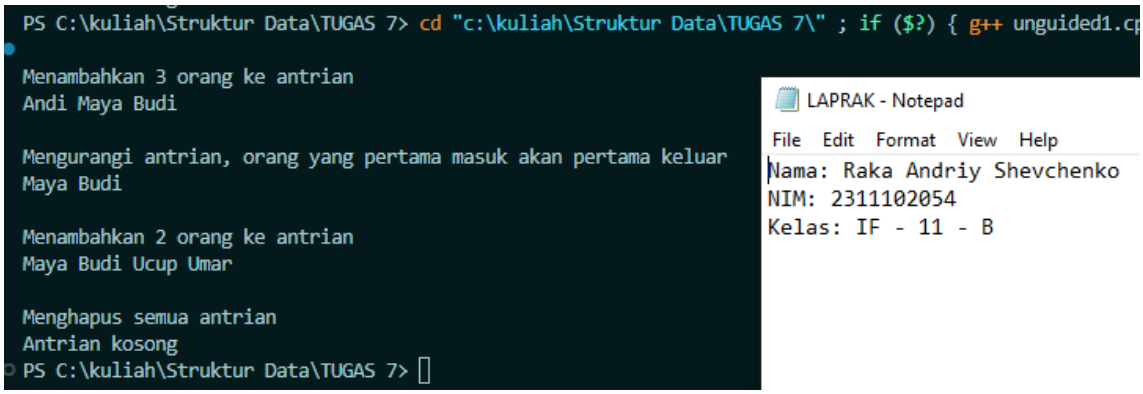
Source code:

```
#include <iostream>

using namespace std;

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

class Queue
{
private:
    Node *front;
    Node *rear;

public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }
    bool isEmpty()
    {
        return (front == nullptr);
    }
    void enqueue(string data)
    {
        Node *newNode = new Node;
        newNode->data = data;
        newNode->next = nullptr;
        if (isEmpty())
        {
            front = newNode;
            rear = newNode;
        }
        else
        {
```

```
            rear->next = newNode;
            rear = newNode;
        }
    }
    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Antrian kosong" << endl;
        }
        else
        {
            Node *temp = front;
            front = front->next;
            delete temp;
        }
    }
    void viewQueue()
    {
        if (isEmpty())
        {
            cout << "Antrian kosong" << endl;
        }
        else
        {
            Node *current = front;
            while (current != nullptr)
            {
                cout << current->data << " ";
                current = current->next;
            }
            cout << endl;
        }
    }
};
int main()
{
    Queue queue;
    cout << "\nMenambahkan 3 orang ke antrian" << endl;
    queue.enqueue("Andi");
    queue.enqueue("Maya");
    queue.enqueue("Budi");
    queue.viewQueue();
```

```
    cout << "\nMengurangi antrian, orang yang pertama masuk akan
pertama keluar " << endl;
    queue.dequeue();
    queue.viewQueue();
    cout << "\nMenambahkan 2 orang ke antrian" << endl;
    queue.enqueue("Ucup");
    queue.enqueue("Umar");
    queue.viewQueue();
    cout << "\nMenghapus semua antrian" << endl;
    while (!queue.isEmpty())
    {
        queue.dequeue();
    }
    queue.viewQueue();
    return 0;
}
```

Output:



Deskripsi:

Program ini memanfaatkan kelas Queue yang berisi beberapa metode untuk mengelola antrian, seperti menambah elemen ke antrian (enqueue), menghapus elemen dari antrian (dequeue), dan menampilkan seluruh elemen dalam antrian (viewQueue). Antrian diimplementasikan dengan dua penunjuk (front dan rear) yang mengacu pada elemen pertama dan terakhir dalam antrian.

Pada awal program, konstruktor Queue menginisialisasi penunjuk front dan rear dengan nilai nullptr yang menandakan bahwa antrian kosong. Fungsi isEmpty() digunakan untuk memeriksa apakah antrian kosong. Fungsi

enqueue(string data) menambahkan elemen baru ke antrian. Jika antrian kosong, elemen baru tersebut menjadi elemen pertama, jika tidak, elemen tersebut ditambahkan di belakang elemen terakhir. Fungsi dequeue() menghapus elemen pertama dari antrian dan mengeluarkan pesan jika antrian kosong. Fungsi viewQueue() menampilkan semua elemen dalam antrian dari depan ke belakang, dan mengeluarkan pesan jika antrian kosong.

b. Unguided 2

Source Code:

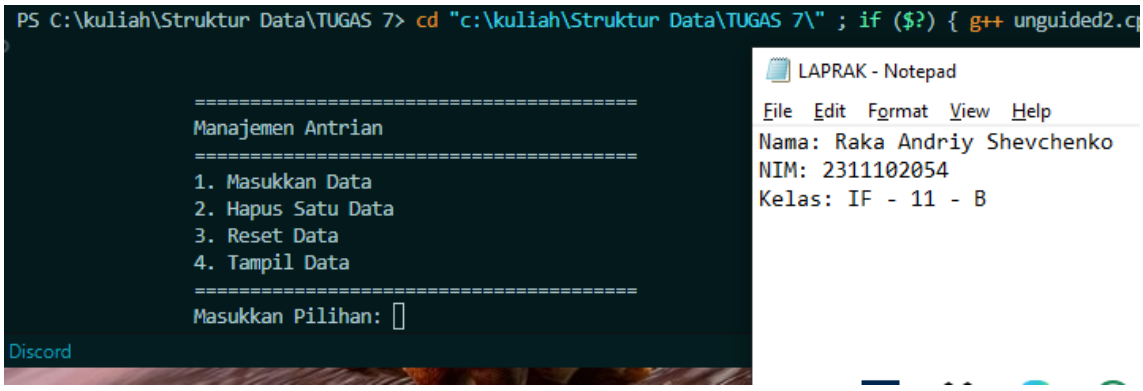
```
#include <iostream>
#include <conio.h>

using namespace std;
struct Node
{
    string nama;
    int nim;
    Node *next;
};
class Queue
{
private:
    Node *front;
    Node *rear;

public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }
    bool isEmpty()
    {
        return (front == nullptr);
    }
    void enqueue(string nama, int nim)
    {
        Node *newNode = new Node;
        newNode->nama = nama;
        newNode->nim = nim;
```

```
        newNode->next = nullptr;
        if (isEmpty())
        {
            front = newNode;
            rear = newNode;
        }
        else
        {
            rear->next = newNode;
            rear = newNode;
        }
    }
    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Antrian kosong" << endl;
        }
        else
        {
            Node *temp = front;
            front = front->next;
            delete temp;
        }
    }
    void viewQueue()
    {
        if (isEmpty())
        {
            cout << "Antrian kosong" << endl;
        }
        else
        {
            Node *current = front;
            while (current != nullptr)
            {
                cout << "\t\t" << current->nama << " NIM : " <<
current->nim << endl;
                current = current->next;
            }
            cout << endl;
        }
    }
```

```
};
int main()
{
    Queue queue;
    string nama;
    char opsi;
    int nim;
    do
    {
        cout <<
"\n\n\t\t======================================" << endl;
        cout << "\t\tManajemen Antrian" << endl;
        cout << "\t\t======================================"
<< endl;
        cout << "\t\t1. Masukkan Data" << endl;
        cout << "\t\t2. Hapus Satu Data" << endl;
        cout << "\t\t3. Reset Data" << endl;
        cout << "\t\t4. Tampil Data" << endl;
        cout << "\t\t======================================"
<< endl;
        int choice;
        cout << "\t\tMasukkan Pilihan: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
        {
            cin.ignore();
            cout << "\t\tMasukkan nama : ";
            getline(cin, nama);
            cout << "\t\tMasukkan NIM : ";
            cin >> nim;
            queue.enqueue(nama, nim);
            break;
        }
        case 2:
        {
            cout << "\t\t[!] 1 Data berhasil dihapus\n";
            queue.dequeue();
            break;
        }
        case 3:
        {
```

```
            cout << "\t\t[!] Data berhasil di reset\n";
            while (!queue.isEmpty())
            {
                queue.dequeue();
            }
            break;
        }
        case 4:
        {
            cout << "\t\tTampilkan Data : \n\n";
            queue.viewQueue();
            break;
        }
        }
        cout << endl << "\t\t> Kembali ke menu utama?[Y/n]\n";
        opsi = getch();
        cout << endl;
        system("cls");
        if (opsi == 'n' || opsi == 'N')
        {
            cout << endl;
            cout << "\t\tSee you :) \n";
        }
    } while (opsi == 'y' || opsi == 'Y');

    return 0;
}
```

Output:



Deskripsi:

Program ini merupakan implementasi sistem antrian (queue) menggunakan linked list dalam bahasa C++, yang dirancang untuk mengelola data mahasiswa berupa nama dan NIM (Nomor Induk Mahasiswa). Program ini memungkinkan pengguna untuk melakukan berbagai operasi seperti menambahkan data mahasiswa ke dalam antrian, menghapus data dari antrian, mereset seluruh antrian, dan menampilkan semua data dalam antrian. Antrian diimplementasikan melalui kelas Queue, yang menggunakan struktur data Node untuk merepresentasikan setiap elemen dalam antrian. Setiap Node berisi data nama, NIM, dan penunjuk ke elemen berikutnya.

Kelas Queue memiliki beberapa metode penting: isEmpty() untuk memeriksa apakah antrian kosong, enqueue(string nama, int nim) untuk menambahkan elemen baru ke antrian, dequeue() untuk menghapus elemen pertama dari antrian, dan viewQueue() untuk menampilkan semua elemen dalam antrian. Jika antrian kosong, metode yang relevan akan menampilkan pesan "Antrian kosong".

D. Kesimpulan

Dalam kesimpulan, Queue adalah struktur data linier yang menerapkan prinsip operasi FIFO (First In First Out). Queue dapat diterapkan dalam bahasa pemrograman apa pun, seperti C, C++, Java, Python, atau C#, dengan spesifikasi yang hampir sama. Queue memiliki beberapa jenis, seperti Simple Queue, Circular Queue, Priority Queue, dan Double-Ended Queue (Dequeue). Queue memiliki kelebihan seperti efisiensi dalam pengelolaan data, operasi penyisipan dan penghapusan yang mudah, dan dapat digunakan dalam implementasi struktur data lainnya. Namun, queue juga memiliki kekurangan seperti operasi penyisipan dan penghapusan elemen dari tengah yang memakan waktu, serta mencari elemen data yang membutuhkan time complexity O(N). Implementasi queue dalam C++ dapat dilakukan dengan menggunakan array atau linked list, dan contoh implementasi queue dengan menggunakan array telah diberikan dalam kode di atas.

E. Referensi

[1] Trivusi, Struktur Data Queue: Pengertian, Jenis, dan Kegunaannya. Diakses dari

Struktur Data Queue: Pengertian, Jenis, dan Kegunaannya - Trivusi

[2] Syarif Soden, Pengertian Queue Dalam C++. Diakses dari

Pengertian Queue Dalam C++ | KASKUS

[3] Muhammad Wafa, Queue dengan Array. Diakses dari

Queue dengan Array - MikirinKode