LAPORAN PRAKTIKUM
MODUL 8
QUEUE

Disusun Oleh:

Satria Putra Dharma Prayudha - 21104036

SE07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. 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

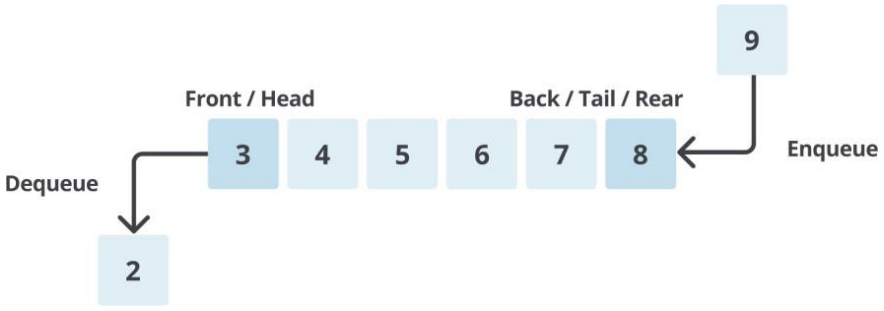
B. Landasan Teori

Landasan teori ini berdasarkan pada modul pembelajaran praktikum struktur data kali ini

2.1 Queue

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.

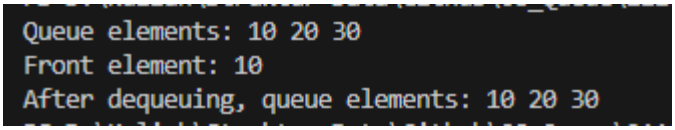
C. Guided

a. Queue Menggunakan Array

Code :

```cpp
// Queue dengan array statis

#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 :



Penjelasan :

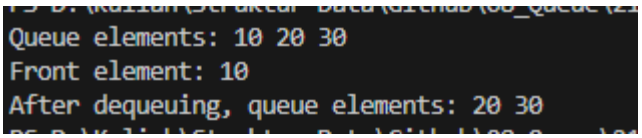
Kode ini mengimplementasikan struktur data antrian (Queue) menggunakan array statis .Kelas Queue memiliki atribut front dan rear untuk melacak posisi elemen paling depan dan belakang. Metode enqueue digunakan untuk menambahkan elemen ke antrian, sedangkan dequeue menghapus elemen dari depan. Metode peek mengembalikan elemen terdepan tanpa menghapusnya, dan display menampilkan semua elemen dalam antrian. Dalam fungsi main, beberapa elemen ditambahkan ke antrian, ditampilkan, dan elemen terdepan diakses sebelum menghapus elemen dari antrian.

b. Queue Dengan Linked Link List

Code :

```cpp
// Queue menggunakan linked list

#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 :



Penjelasan : Kode ini mengimplementasikan antrian, tetapi menggunakan struktur data linked list. Kelas Node merepresentasikan elemen dalam antrian, dengan atribut data untuk menyimpan nilai dan next untuk menunjuk ke node berikutnya. Kelas Queue memiliki pointer front dan rear untuk melacak elemen paling depan dan belakang. Metode enqueue menambahkan elemen baru ke antrian, sedangkan dequeue menghapus elemen dari depan. Metode peek mengembalikan nilai elemen terdepan, dan display menampilkan semua elemen. Fungsi main menguji fungsionalitas antrian dengan menambahkan, menampilkan, dan menghapus elemen.

c. Queue Menggunakan Dengan Kondisi Tertentu

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
            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. Maya
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 2
Data antrian teller:
1. Maya
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
```

Penjelasan : Kode ini mengimplementasikan antrian menggunakan array dengan ukuran tetap. Terdapat beberapa fungsi untuk mengelola antrian, termasuk isFull dan isEmpty untuk memeriksa status antrian. Fungsi enqueueAntrian menambahkan elemen ke antrian, sedangkan dequeueAntrian menghapus elemen dari depan dengan menggeser elemen yang tersisa. Fungsi countQueue menghitung jumlah elemen dalam antrian, dan clearQueue menghapus semua elemen. Fungsi viewQueue menampilkan semua elemen dalam antrian. Dalam fungsi main, beberapa elemen ditambahkan, ditampilkan, dan dihapus dari antrian, serta jumlah elemen diperiksa setelah setiap operasi.

D. Unguided

a. Implementasi Queue Menggunakan Linked List

Code:

```cpp
//Implementasi Queue Menggunakan Linked List

#include <iostream>
using namespace std;

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

// Pointer global untuk front dan rear
Node* front = nullptr;
Node* rear = nullptr;

// Fungsi untuk memeriksa apakah queue kosong
bool isEmpty() {
    return front == nullptr;
}

// Fungsi untuk menambahkan data ke dalam queue
void enqueue(int data) {
    Node* newNode = new Node{data, nullptr};
    if (isEmpty()) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << "Data " << data << " ditambahkan ke antrian.\n";
}

// Fungsi untuk menghapus data dari queue
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
    } else {
        Node* temp = front;
        front = front->next;
        cout << "Data " << temp->data << " dikeluarkan dari
antrian.\n";
        delete temp;

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

// Fungsi untuk melihat data dalam queue
void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
    } else {
        Node* temp = front;
        cout << "Data antrian: ";
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
}

int main() {
    int choice, data;

    do {
        cout << "\nMenu:\n1. Tambah Antrian (Enqueue)\n2. Hapus
Antrian (Dequeue)\n3. Lihat Antrian\n4. Keluar\nPilih: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan angka: ";
                cin >> data;
                enqueue(data);
                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:
1. Tambah Antrian (Enqueue)
2. Hapus Antrian (Dequeue)
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan angka: 10
Data 10 ditambahkan ke antrian.

Menu:
1. Tambah Antrian (Enqueue)
2. Hapus Antrian (Dequeue)
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan angka: 20
Data 20 ditambahkan ke antrian.

Menu:
1. Tambah Antrian (Enqueue)
2. Hapus Antrian (Dequeue)
3. Lihat Antrian
4. Keluar
Pilih: 3
Data antrian: 10 20
```

Penjelasan: Program ini menggunakan struktur data linked list untuk mengimplementasikan antrian (queue) dengan prinsip FIFO (First-In First-Out). Fungsi utama yang digunakan meliputi enqueue, dequeue, dan viewQueue. Fungsi enqueue menambahkan elemen baru ke bagian belakang antrian. Jika antrian kosong, elemen pertama akan menjadi front dan rear. Fungsi dequeue digunakan untuk menghapus elemen dari bagian depan antrian. Setelah elemen pertama dihapus, pointer front akan berpindah ke elemen berikutnya. Jika antrian kosong setelah penghapusan, pointer rear diatur kembali ke null. Fungsi viewQueue menampilkan seluruh elemen dalam antrian, dimulai dari elemen paling depan hingga elemen terakhir. Fungsi tambahan seperti isEmpty digunakan untuk memeriksa apakah antrian kosong sebelum operasi dilakukan.

b. Konsep Antrian (Queue) dengan atribut Nama dan NIM

Code:

```cpp
//Konsep Antrian (Queue) dengan atribut Nama dan NIM

#include <iostream>
using namespace std;

// Struktur Node untuk Linked List dengan atribut Nama dan NIM
struct Node {
    string nama;
    int nim;
    Node* next;
};

// Pointer global untuk front dan rear
Node* front = nullptr;
Node* rear = nullptr;

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

void enqueue(string nama, int nim) {
    Node* newNode = new Node{nama, nim, nullptr};
    if (isEmpty()) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << "Mahasiswa " << nama << " dengan NIM " << nim << "
ditambahkan ke antrian.\n";
}

void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
    } else {
        Node* temp = front;
        front = front->next;
        cout << "Mahasiswa " << temp->nama << " dengan NIM " << temp-
>nim << " dikeluarkan dari antrian.\n";
        delete temp;

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

void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
    } else {
        Node* temp = front;
        while (temp != nullptr) {
            cout << "Nama: " << temp->nama << ", NIM: " << temp->nim
<< "\n";
            temp = temp->next;
        }
    }
}

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

    do {
        cout << "\nMenu:\n1. Tambah Antrian (Enqueue)\n2. Hapus
Antrian (Dequeue)\n3. Lihat Antrian\n4. Keluar\nPilih: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan Nama: ";
                cin >> nama;
                cout << "Masukkan NIM: ";
                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:

```
2. Hapus Antrian (Dequeue)
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan Nama: Satia
Masukkan NIM: 21104036
Mahasiswa Satia dengan NIM 21104036 ditambahkan ke antrian.

Menu:
1. Tambah Antrian (Enqueue)
2. Hapus Antrian (Dequeue)
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan Nama: Yudha
Masukkan NIM: 21104001
Mahasiswa Yudha dengan NIM 21104001 ditambahkan ke antrian.

Menu:
1. Tambah Antrian (Enqueue)
2. Hapus Antrian (Dequeue)
3. Lihat Antrian
4. Keluar
Pilih: 1
Masukkan Nama: Traxxis
Masukkan NIM: 2103050
Mahasiswa Traxxis dengan NIM 2103050 ditambahkan ke antrian.

Menu:
1. Tambah Antrian (Enqueue)
2. Hapus Antrian (Dequeue)
3. Lihat Antrian
4. Keluar
Pilih: 3
Nama: Satia, NIM: 21104036
Nama: Yudha, NIM: 21104001
Nama: Traxxis, NIM: 2103050
```

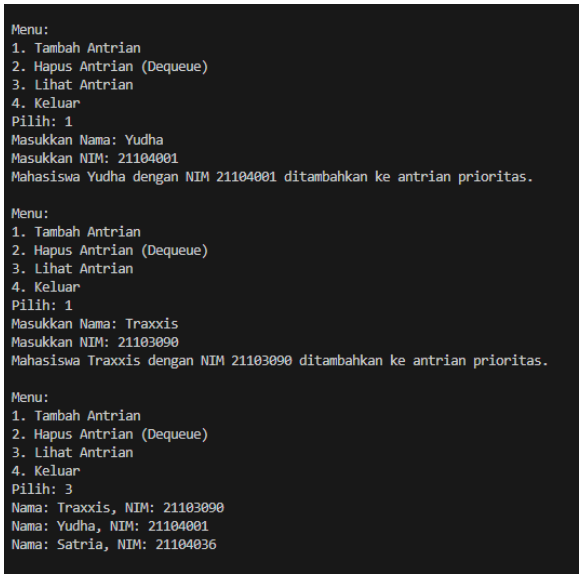
Penjelasan: Program ini mengembangkan konsep queue dengan menambahkan atribut **Nama** dan **NIM** pada setiap elemen antrian. Menggunakan linked list, setiap elemen dalam queue direpresentasikan sebagai node yang berisi data nama, NIM, dan pointer ke node berikutnya. Fungsi **enqueue** menambahkan node baru ke bagian belakang antrian, sedangkan fungsi **dequeue** menghapus node dari bagian depan. Program juga menyediakan fungsi **viewQueue** untuk menampilkan seluruh data mahasiswa dalam antrian, termasuk nama dan NIM mereka. Untuk mempermudah penggunaan, program menggunakan menu interaktif sehingga pengguna dapat memasukkan data mahasiswa, menghapus data, atau melihat daftar mahasiswa dalam antrian. Fungsi **isEmpty** memastikan bahwa operasi hanya dilakukan jika antrian tidak kosong. Dengan atribut tambahan ini, program membantu mengelola data mahasiswa dalam antrian secara terstruktur.

c. Membuat program queue dengan prioritas berdasarkan NIM

Code:

```
// Membuat program queue dengan prioritas berdasarkan NIM

#include <iostream>
using namespace std;

// Struktur Node untuk Linked List dengan atribut Nama dan NIM
struct Node {
    string nama;
    int nim;
    Node* next;
};

// Pointer global untuk front dan rear
Node* front = nullptr;
Node* rear = nullptr;

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

// Fungsi untuk menambahkan data ke dalam queue dengan prioritas
berdasarkan NIM
void enqueuePrioritas(string nama, int nim) {
    Node* newNode = new Node{nama, nim, nullptr};

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

void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
    } else {
        Node* temp = front;
        front = front->next;
        cout << "Mahasiswa " << temp->nama << " dengan NIM " << temp-
>nim << " dikeluarkan dari antrian.\n";
        delete temp;

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

void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
    } else {
        Node* temp = front;
        while (temp != nullptr) {
            cout << "Nama: " << temp->nama << ", NIM: " << temp->nim
<< "\n";
            temp = temp->next;
        }
    }
}

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

    do {
        cout << "\nMenu:\n1. Tambah Antrian \n2. Hapus Antrian
(Dequeue)\n3. Lihat Antrian\n4. Keluar\nPilih: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Masukkan Nama: ";
                cin >> nama;
                cout << "Masukkan NIM: ";
                cin >> nim;
                enqueuePrioritas(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:



Penjelasan: Program ini memodifikasi antrian sehingga mendukung pengurutan berdasarkan atribut **NIM**. Elemen dengan NIM terkecil akan memiliki prioritas lebih tinggi dan ditempatkan lebih awal di antrian. Fungsi **enqueuePrioritas** berfungsi untuk menambahkan elemen baru ke antrian pada posisi yang sesuai berdasarkan urutan NIM. Jika antrian kosong atau elemen baru memiliki NIM lebih kecil dari elemen di bagian depan, elemen tersebut ditempatkan sebagai elemen pertama. Jika tidak, program akan mencari posisi yang tepat dalam antrian untuk menyisipkan elemen baru. Fungsi **dequeue** menghapus elemen dari bagian depan antrian, sedangkan **viewQueue** menampilkan seluruh elemen dalam antrian, termasuk nama dan NIM mereka, dalam urutan prioritas. Program ini juga menggunakan fungsi **isEmpty** untuk memeriksa apakah antrian kosong sebelum melakukan operasi. Dengan menggunakan menu interaktif, pengguna dapat menambah, menghapus, dan melihat elemen dalam antrian, yang diurutkan berdasarkan prioritas NIM.

E. Kesimpulan

Dalam praktikum ini, telah dipelajari konsep dasar dan implementasi queue sebagai salah satu struktur data penting dalam pemrograman. Queue merupakan struktur data yang mengikuti prinsip FIFO (First-In First-Out), di mana elemen pertama yang dimasukkan menjadi elemen pertama yang dikeluarkan. Selain memahami perbedaan antara queue dan stack—di mana stack menggunakan prinsip LIFO (Last-In First-Out)—pada materi ini juga telah mempelajari berbagai operasi queue, seperti enqueue untuk menambahkan elemen, dequeue untuk menghapus elemen, dan peek untuk melihat elemen di depan tanpa menghapusnya. Implementasi queue dilakukan menggunakan array maupun linked list, yang masing-masing memiliki kelebihan dan kekurangan dalam penggunaannya.

Pada praktikum ini juga dibuat sebuah program queue dengan fitur prioritas, di mana elemen dengan NIM terkecil mendapatkan prioritas lebih tinggi dalam antrian. Praktikum ini memberikan wawasan mengenai penerapan queue dalam berbagai situasi, seperti sistem antrean di kasir atau pengelolaan proses dalam sistem operasi. Dengan demikian, pada praktikum ini diperoleh pemahaman mendalam dan keterampilan praktis untuk memanfaatkan struktur data secara efisien dan terorganisir dalam menyelesaikan berbagai permasalahan komputasi.