LAPORAN PRAKTIKUM

Modul 8 Queue

Disusun Oleh:

Yogi Hafidh Maulana - 2211104061

SE06-02

Asisten Praktikum :

Muhammad Faza Zulian Gesit Al Barru

Aisyah Hasna Aulia

Dosen Pengampu :

Yudha Islami Sulistya, S.Kom., M.Cs

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

A. Tujuan

- Memahami konsep dasar dari struktur data queue, serta memahami prinsip kerja FIFO (First-In First-Out) yang diterapkan pada queue.
- Mampu menerapkan operasi pada queue, seperti operasi enqueue (menambahkan elemen), dequeue (mengeluarkan elemen), dan mengecek status queue (kosong atau penuh).
- Menerapkan operasi untuk menampilkan data dalam queue dan menghitung jumlah elemen yang ada di dalamnya.
- Mengimplementasikan berbagai metode untuk memanipulasi data dalam queue, seperti peek (melihat elemen terdepan) dan clear (menghapus semua elemen).
- Mengimplementasikan queue menggunakan array atau linked list sesuai dengan kebutuhan aplikasi dan kasus penggunaan yang relevan.

B. Landasan Teori

- Pengertian Quaue

  Queue adalah salah satu struktur data yang digunakan untuk menyimpan dan mengelola data secara berurutan, yang mengikuti prinsip FIFO (First In, First Out). Dalam struktur data queue, elemen pertama yang dimasukkan adalah elemen pertama yang akan dikeluarkan. Konsep ini mirip dengan antrean dalam kehidupan sehari-hari, seperti antrean di kasir, di mana orang yang datang pertama kali adalah orang yang dilayani terlebih dahulu. Queue memiliki dua ujung, yaitu front (depan) dan rear (belakang). Elemen baru ditambahkan pada rear, sementara elemen yang paling lama berada di front akan dihapus saat melakukan operasi dequeue.

  Queue dapat diimplementasikan dengan dua cara utama, yaitu menggunakan **array** atau **linked list**. Pada implementasi dengan array, dua pointer, yaitu **front** dan **rear**, digunakan untuk melacak posisi elemen pertama dan terakhir. Sementara pada implementasi dengan linked list, elemen-elemen dalam queue dihubungkan oleh pointer, di mana front dan rear akan menunjuk pada node pertama dan terakhir dalam daftar.

- Perbedaan Queue dengan Stack

  Queue sering dibandingkan dengan stack, yang juga merupakan struktur data linear. Namun, terdapat perbedaan mendasar antara keduanya. Pada stack, operasi penambahan (push) dan penghapusan (pop) elemen dilakukan pada satu ujung yang disebut top, mengikuti prinsip LIFO (Last In, First Out). Artinya, elemen terakhir yang dimasukkan akan menjadi elemen pertama yang dihapus. Sebaliknya, pada queue, elemen ditambahkan di belakang dan dihapus dari depan, mengikuti prinsip FIFO. Sebagai contoh, jika sebuah antrean di kasir adalah queue, maka pelanggan pertama yang datang akan dilayani pertama kali, sedangkan dalam stack, pelanggan yang terakhir datang akan dilayani lebih dahulu.

C. Guided

1) Guided 1

Code:

```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;
}
```

Output:

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
PS D:\PROJECT\C++ Project\Pertemuan8>
```

Deskripsi Program:

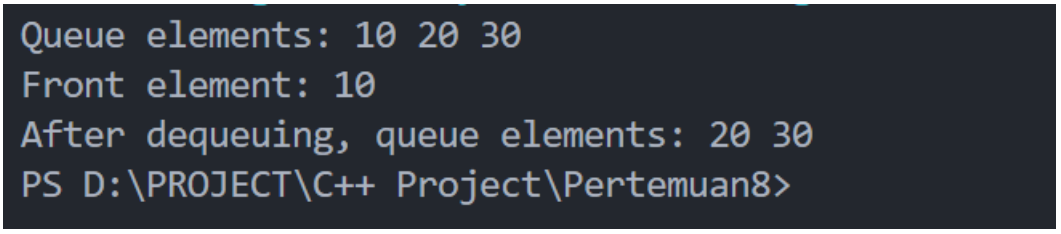
Program ini mengimplementasikan antrian (queue) menggunakan array statis dengan ukuran maksimum 100. Dalam program, ada beberapa operasi utama pada antrian: enqueue untuk menambah elemen ke dalam antrian, dequeue untuk mengeluarkan elemen dari antrian, peek untuk melihat elemen yang ada di depan antrian tanpa menghapusnya, dan display untuk menampilkan seluruh elemen dalam antrian. Program dimulai dengan membuat objek antrian, kemudian menambahkan beberapa elemen menggunakan enqueue. Setelah itu, program menampilkan elemen-elemen yang ada di antrian, memeriksa elemen paling depan menggunakan peek, dan menampilkan kondisi antrian setelah melakukan operasi pengeluaran (dequeue). Program juga menangani kondisi overflow (antrian penuh) dan underflow (antrian kosong) dengan memberikan pesan yang sesuai.

2) Guided 2

Code:

```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;
}
```

Output:



Deskripsi Program:

Program ini mengimplementasikan antrian (queue) menggunakan struktur data linked list, di mana setiap elemen antrian disimpan dalam node yang memiliki dua bagian: data dan pointer yang mengarah ke node berikutnya. Program dimulai dengan mendefinisikan kelas Node untuk setiap elemen, yang memiliki data dan pointer next. Kelas Queue memiliki dua pointer, front dan rear, yang masing-masing menunjuk ke elemen pertama dan terakhir di antrian. Operasi utama dalam program ini adalah enqueue untuk menambahkan elemen ke belakang antrian, dequeue untuk menghapus elemen dari depan antrian, peek untuk melihat elemen depan tanpa menghapusnya, dan display untuk menampilkan seluruh elemen antrian. Jika antrian kosong, operasi dequeue dan peek akan memberikan pesan yang sesuai. Program mengelola antrian dengan benar, menangani penambahan dan penghapusan elemen serta memastikan elemen terakhir dan pertama selalu diperbarui.

3) Guided 3

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 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:

```
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
PS D:\PROJECT\C++ Project\Pertemuan8>
```

Deskripsi Program:

Program ini mengelola antrian dengan kapasitas maksimum 5 elemen menggunakan array queueTeller yang menyimpan nama orang dalam antrian. Terdapat beberapa operasi utama: enqueueAntrian untuk menambahkan elemen ke antrian, dequeueAntrian untuk menghapus elemen dari depan antrian, countQueue untuk menghitung jumlah elemen yang ada di antrian, clearQueue untuk menghapus seluruh elemen, dan viewQueue untuk menampilkan kondisi antrian. Fungsi isFull dan isEmpty digunakan untuk memeriksa apakah antrian penuh atau kosong. Pada saat penambahan elemen (enqueue), program memeriksa apakah antrian sudah penuh atau kosong, dan jika kosong, elemen ditambahkan ke posisi pertama. Sedangkan pada penghapusan elemen (dequeue), seluruh elemen digeser ke kiri, dan elemen terakhir dihapus. Program juga menghitung jumlah elemen antrian dan memberikan informasi tentang status antrian setiap kali operasi dilakukan.

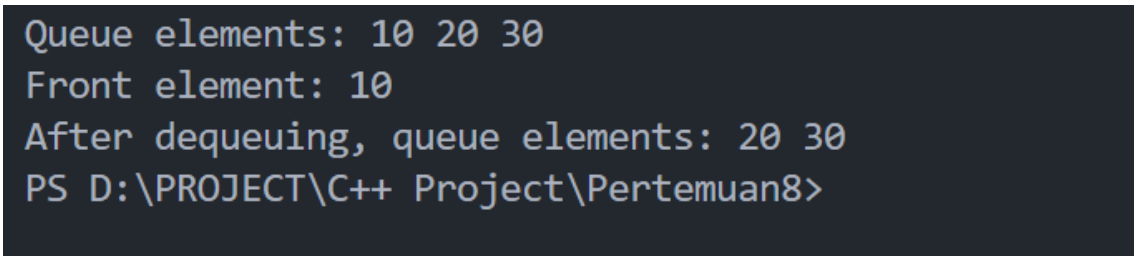
D. Unguided

1) Unguided 1

Code:

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

// Struktur Node untuk linked list
struct Node
{
    int data;
    Node *next;
};

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

public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }

    // Mengecek apakah queue kosong
    bool isEmpty()
    {
        return front == nullptr;
    }

    // Menambahkan elemen ke dalam queue
    void enqueue(int x)
    {
        Node *newNode = new Node();
        newNode->data = x;
        newNode->next = nullptr;

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

    // Mengeluarkan elemen dari queue
    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Queue Underflow\n";
            return;
        }

        Node *temp = front;
        front = front->next;
        delete temp;

        // Jika setelah dequeue queue menjadi kosong
        if (front == nullptr)
        {
            rear = nullptr;
        }
    }

    // Melihat elemen terdepan
    int peek()
    {
        if (!isEmpty())
        {
            return front->data;
        }
        cout << "Queue is empty\n";
        return -1;
    }

    // Menampilkan semua elemen dalam queue
    void display()
    {
        if (isEmpty())
        {
            cout << "Queue is empty\n";
            return;
        }

        Node *temp = front;
        while (temp != nullptr)
        {
            cout << temp->data << " ";
            temp = temp->next;
        }
        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";

    q.dequeue(); // Dequeue element

    cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}
```

Output:



Deskripsi Code:

Program ini mengimplementasikan antrian (queue) menggunakan linked list, di mana setiap elemen antrian disimpan dalam node yang memiliki dua bagian: data dan pointer next yang menunjuk ke elemen berikutnya. Kelas Queue memiliki dua pointer utama, yaitu front untuk elemen pertama dan rear untuk elemen terakhir. Ada beberapa operasi yang dilakukan: enqueue untuk menambahkan elemen ke belakang antrian, dequeue untuk menghapus elemen dari depan antrian, peek untuk melihat elemen terdepan tanpa menghapusnya, dan display untuk menampilkan semua elemen dalam antrian. Program akan memeriksa apakah antrian kosong atau tidak sebelum melakukan operasi dequeue atau peek. Saat elemen dihapus, pointer front bergerak ke node berikutnya, dan jika antrian menjadi kosong setelah penghapusan, pointer rear juga diatur menjadi nullptr. Program ini memungkinkan penambahan dan penghapusan elemen secara dinamis tanpa batasan kapasitas tetap, seperti pada array.

2) Unguided 2

Code:

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

// Struktur Node untuk linked list
struct Node
{
    string nama;
    string nim;
    Node *next;
};

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

public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }

    // Mengecek apakah queue kosong
    bool isEmpty()
    {
        return front == nullptr;
    }

    // Menambahkan elemen ke dalam queue
    void enqueue(string nama, string nim)
    {
        Node *newNode = new Node();
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

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

    // Mengeluarkan elemen dari queue
    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Queue Underflow\n";
            return;
        }

        Node *temp = front;
        front = front->next;
        delete temp;

        // Jika setelah dequeue queue menjadi kosong
        if (front == nullptr)
        {
            rear = nullptr;
        }
    }

    // Melihat elemen terdepan
    void peek()
    {
        if (!isEmpty())
        {
            cout << "Front - Nama: " << front->nama << ", NIM: " << front->nim << endl;
        }
        else
        {
            cout << "Queue is empty\n";
        }
    }

    // Menampilkan semua elemen dalam queue
    void display()
    {
        if (isEmpty())
        {
            cout << "Queue is empty\n";
            return;
        }

        Node *temp = front;
        while (temp != nullptr)
        {
            cout << "Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
            temp = temp->next;
        }
        cout << "\n";
    }
};

int main()
{
    Queue q;
    int n;
    string nama, nim;

    cout << "Masukkan jumlah mahasiswa: ";
    cin >> n;
    cin.ignore(); // Membersihkan newline character setelah cin

    for (int i = 0; i < n; i++)
    {
        cout << "Masukkan Nama Mahasiswa ke-" << (i + 1) << ": ";
        getline(cin, nama);
        cout << "Masukkan NIM Mahasiswa ke-" << (i + 1) << ": ";
        getline(cin, nim);
        q.enqueue(nama, nim);
    }

    cout << "\nQueue elements:\n";
    q.display();

    cout << "\nElemen terdepan (Front) queue: ";
    q.peek();

    // Dequeue beberapa elemen untuk menunjukkan perubahan queue
    cout << "\nSetelah dequeue:\n";
    q.dequeue();
    q.display();

    return 0;
}
```

Output:

```
Masukkan jumlah mahasiswa: 2
Masukkan Nama Mahasiswa ke-1: Yogi Hafidh Maulana
Masukkan NIM Mahasiswa ke-1: 2211104061
Masukkan Nama Mahasiswa ke-2: Yono Samsudin
Masukkan NIM Mahasiswa ke-2: 2211104032

Queue elements:
Nama: Yogi Hafidh Maulana, NIM: 2211104061
Nama: Yono Samsudin, NIM: 2211104032


Elemen terdepan (Front) queue: Front - Nama: Yogi Hafidh Maulana, NIM: 2211104061

Setelah dequeue:
Nama: Yono Samsudin, NIM: 2211104032
```

Deskripsi Code:

Program ini mengimplementasikan antrian (queue) menggunakan linked list untuk menyimpan data mahasiswa, yang terdiri dari nama dan NIM. Kelas Queue memiliki dua pointer, front untuk menunjuk elemen pertama dan rear untuk menunjuk elemen terakhir. Program dimulai dengan meminta pengguna untuk memasukkan jumlah mahasiswa, kemudian untuk setiap mahasiswa, program akan meminta input nama dan NIM, lalu menambahkannya ke dalam antrian menggunakan operasi enqueue. Setiap elemen mahasiswa disimpan dalam node dengan dua atribut: nama dan NIM. Setelah memasukkan semua data, program menampilkan semua elemen antrian menggunakan display dan menampilkan elemen terdepan (front) menggunakan peek. Selanjutnya, program menghapus elemen terdepan dengan operasi dequeue dan menampilkan kondisi antrian setelah elemen dihapus. Program juga menangani kondisi antrian kosong dengan menampilkan pesan yang sesuai jika operasi dilakukan pada antrian kosong.

3) Unguided 3

Code:

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

// Struktur Node untuk linked list
struct Node
{
    string nama;
    string nim;
    Node *next;
};

// Kelas Queue dengan prioritas berdasarkan NIM
class Queue
{
private:
    Node *front;
    Node *rear;

public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }

    // Mengecek apakah queue kosong
    bool isEmpty()
    {
        return front == nullptr;
    }

    // Menambahkan elemen ke dalam queue dengan prioritas berdasarkan NIM
    void enqueue(string nama, string nim)
    {
        Node *newNode = new Node();
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

        if (isEmpty())
        {
            front = rear = newNode;
        }
        else
        {
            // Menyisipkan node pada posisi yang tepat berdasarkan NIM
            Node *temp = front;
            Node *prev = nullptr;

            // Cari posisi yang tepat (NIM lebih kecil didahulukan)
            while (temp != nullptr && temp->nim < nim)
            {
                prev = temp;
                temp = temp->next;
            }

            // Menyisipkan elemen di awal atau di tengah
            if (prev == nullptr)
            {
                newNode->next = front;
                front = newNode;
            }
            else
            {
                newNode->next = temp;
                prev->next = newNode;
            }

            // Jika node baru ditambahkan di akhir
            if (newNode->next == nullptr)
            {
                rear = newNode;
            }
        }
    }

    // Mengeluarkan elemen dari queue
    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Queue Underflow\n";
            return;
        }

        Node *temp = front;
        front = front->next;
        delete temp;

        // Jika setelah dequeue queue menjadi kosong
        if (front == nullptr)
        {
            rear = nullptr;
        }
    }

    // Melihat elemen terdepan
    void peek()
    {
        if (!isEmpty())
        {
            cout << "Front - Nama: " << front->nama << ", NIM: " << front->nim << endl;
        }
        else
        {
            cout << "Queue is empty\n";
        }
    }

    // Menampilkan semua elemen dalam queue
    void display()
    {
        if (isEmpty())
        {
            cout << "Queue is empty\n";
            return;
        }

        Node *temp = front;
        while (temp != nullptr)
        {
            cout << "Nama: " << temp->nama << ", NIM: " << temp->nim << endl;
            temp = temp->next;
        }
        cout << "\n";
    }
};

int main()
{
    Queue q;
    int n;
    string nama, nim;

    cout << "Masukkan jumlah mahasiswa: ";
    cin >> n;
    cin.ignore(); // Membersihkan newline character setelah cin

    for (int i = 0; i < n; i++)
    {
        cout << "Masukkan Nama Mahasiswa ke-" << (i + 1) << ": ";
        getline(cin, nama);
        cout << "Masukkan NIM Mahasiswa ke-" << (i + 1) << ": ";
        getline(cin, nim);
        q.enqueue(nama, nim);
    }

    cout << "\nQueue elements (dengan prioritas berdasarkan NIM):\n";
    q.display();

    cout << "\nElemen terdepan (Front) queue: ";
    q.peek();

    // Dequeue beberapa elemen untuk menunjukkan perubahan queue
    cout << "\nSetelah dequeue:\n";
    q.dequeue();
    q.display();

    return 0;
}
```

Output:

```
Masukkan jumlah mahasiswa: 2
Masukkan Nama Mahasiswa ke-1: Yogi Hafidh Maulana
Masukkan NIM Mahasiswa ke-1: 2211104061
Masukkan Nama Mahasiswa ke-2: Yono Suhamdi Sudirman
Masukkan NIM Mahasiswa ke-2: 2211104030

Queue elements (dengan prioritas berdasarkan NIM):
Nama: Yono Suhamdi Sudirman, NIM: 2211104030
Nama: Yogi Hafidh Maulana, NIM: 2211104061


Elemen terdepan (Front) queue: Front - Nama: Yono Suhamdi Sudirman, NIM: 2211104030

Setelah dequeue:
Nama: Yogi Hafidh Maulana, NIM: 2211104061

PS D:\PROJECT\C++ Project\Pertemuan8>
```

Deskripsi Code:

Program ini mengimplementasikan antrian (queue) dengan prioritas berdasarkan NIM mahasiswa menggunakan linked list. Kelas Queue memiliki dua pointer, front dan rear, yang menunjuk ke elemen pertama dan terakhir dalam antrian. Ketika elemen baru dimasukkan menggunakan fungsi enqueue, program akan menyisipkan elemen di posisi yang tepat berdasarkan urutan NIM yang lebih kecil, sehingga mahasiswa dengan NIM lebih kecil akan diprioritaskan lebih dahulu. Proses penyisipan dilakukan dengan mencari posisi yang tepat dalam antrian menggunakan perbandingan NIM. Fungsi dequeue digunakan untuk menghapus elemen terdepan, sementara peek menampilkan elemen terdepan tanpa menghapusnya. Fungsi display menampilkan seluruh elemen dalam antrian. Program meminta input data mahasiswa, kemudian menampilkan antrian dengan prioritas berdasarkan NIM, menampilkan elemen terdepan, dan melakukan operasi dequeue untuk menunjukkan perubahan pada antrian.

E. Kesimpulan

Queue adalah struktur data yang mengikuti prinsip FIFO (First In, First Out), di mana elemen pertama yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Konsep ini sangat berguna dalam situasi di mana data perlu diproses secara berurutan, seperti pada antrean di kasir atau antrian tugas. Operasi dasar pada queue mencakup enqueue (menambahkan elemen), dequeue (menghapus elemen), peek (melihat elemen terdepan), isEmpty (memeriksa apakah queue kosong), dan isFull (memeriksa apakah queue penuh). Queue dapat diimplementasikan menggunakan array atau linked list, masing-masing dengan kelebihan dan kekurangan tersendiri. Implementasi array lebih sederhana namun terbatas pada kapasitas tertentu, sementara implementasi linked list lebih fleksibel dalam hal ukuran tetapi memerlukan overhead memori lebih banyak karena penggunaan pointer. Secara keseluruhan, pemahaman tentang konsep dan operasi queue sangat penting bagi mahasiswa untuk dapat mengaplikasikannya dalam berbagai masalah yang melibatkan pengolahan data secara urut.

Yogi Hafidh Maulana – 2211104061©