LAPORAN PRAKTIKUM

MODUL VIII

“QUEUE (ANTREAN)”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-B

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

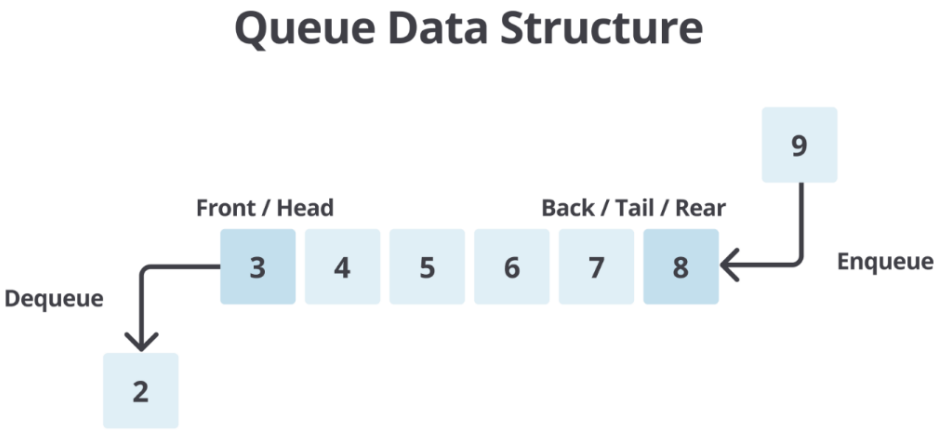
2024

1. Tujuan

- Mahasiswa mampu menjelaskan definisi dan konsep dari queue
- Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
- Mahasiswa mampu menerapkan operasi tampil data pada queue

2. Landasan Teori

Queue atau dalam bahasa Indonesia yang berarti antrean adalah struktur data yang menyusun elemen-elemen data dalam urutan linier. Prinsip dasar dari struktur data ini adalah "First In, First Out" (FIFO) yang berarti elemen data yang pertama dimasukkan ke dalam antrean akan menjadi yang pertama pula untuk dikeluarkan. Caranya bekerja adalah seperti jejeran orang yang sedang menunggu antrean di supermarket di mana orang pertama yang datang adalah yang pertama dilayani (First In, First Out). Pada struktur data ini, urutan pertama (data yang akan dikeluarkan) disebut Front atau Head. Sebaliknya, data pada urutan terakhir (data yang baru saja ditambahkan) disebut Back, Rear, atau Tail. Proses untuk menambahkan data pada antrean disebut dengan Enqueue, sedangkan proses untuk menghapus data dari antrean disebut dengan Dequeue.



Queue memiliki peran yang penting dalam berbagai aplikasi dan algoritma. Salah satu fungsi utamanya adalah mengatur dan mengelola antrean tugas atau operasi secara efisien. Dalam sistem komputasi, ia digunakan untuk menangani tugas-tugas seperti penjadwalan proses, antrean pesan, dan manajemen sumber daya.

Queue memiliki beberapa operasi dasar yang digunakan seperti:

- Create, operasi ini menginisialisasi queue dengan mendefinisikan ukuran maksimalnya (jika ada) dan mengatur pointer atau indeks awal dan akhir ke kondisi awal (misalnya front = -1 dan rear = -1 pada array-based queue).
- IsEmpty, operasi ini memeriksa kondisi queue untuk melihat apakah tidak ada elemen yang disimpan. Biasanya, queue dianggap kosong jika front == -1 (pada implementasi array) atau jika ukuran queue adalah 0 (pada implementasi berbasis linked list).

- IsFull, operasi ini memeriksa apakah queue telah mencapai kapasitas maksimalnya. Pada array-based queue, kondisi penuh seringkali ditentukan oleh rear == maxSize - 1.
- EnQueue, operasi ini menambahkan elemen baru ke bagian belakang (rear) queue. Pada array, nilai rear akan bertambah, dan elemen baru ditempatkan pada indeks tersebut. Pada linked list, node baru ditambahkan di akhir.
- DeQueue, operasi ini menghapus elemen yang berada di bagian depan (front) queue. Pada array, indeks front akan bergerak maju. Pada linked list, node pertama dihapus dari daftar.
- Clear, operasi ini mengosongkan queue dengan cara mengatur ulang pointer atau indeks ke kondisi awal. Pada linked list, setiap node dihapus dari memori.
- Tampil, operasi ini mencetak elemen queue dalam urutan dari depan ke belakang tanpa menghapus elemen tersebut.
- Push(x), operasi ini mirip dengan EnQueue, di mana elemen baru x ditambahkan ke bagian belakang queue.
- Pop(), operasi ini identik dengan DeQueue, yaitu menghapus elemen pertama (front) dari queue.

3. Guided

1. Program ini adalah contoh sederhana dari implementasi queue menggunakan array. Struktur data ini sering digunakan dalam berbagai aplikasi, termasuk manajemen tugas, pemrosesan data, dan algoritma pencarian.

   Di dalam program ini terdapat Kelas Queue memiliki dua variabel anggota front dan rear untuk melacak posisi elemen depan dan belakang antrean. Kemudian digunakan fungsi untuk memeriksa status antrean. Terdapat juga operrasi antrean untuk menambahkan elemen x ke antrean. Jika antrean penuh, mencetak pesan "Queue Overflow". Jika antrean kosong (front -1), front diatur ke 0. Elemen baru ditambahkan di posisi rear yang ditingkatkan. Digunakan juga operasi dequeue untuk menghapus elemen dari antrean. Fungsi lain yang dipakai ada peek untuk mengembalikan elemen di posisi front tanpa menghapusnya, lalu ada display yang menampilkan semua elemen dalam antrean dari front ke rear. Dalam fungsi main dibuat objek Queue bernama q. Beberapa elemen (10, 20, 30) ditambahkan ke antrean menggunakan enqueue(). Elemen antrean ditampilkan dengan display(). Elemen terdepan diperiksa dengan peek(). Setelah itu, elemen dihapus dari antrean (meskipun tidak ada pemanggilan dequeue() dalam kode ini), dan antrean ditampilkan Kembali.

   Code program:

```
#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 dari program:

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
```

2. Program tersebut merupakan implementasi dari queue pada struktur data yang menggunakan linked list, yang memungkinkan ukuran antrean untuk dinamis dibandingkan dengan menggunakan array yang memiliki ukuran tetap. Setiap elemen dalam antrean diwakili oleh objek Node, yang menyimpan data dan pointer ke node berikutnya. Digunakan operasi dasar seperti enqueue yang menambahkan elemen ke belakang antrean, lalu dequeue untuk menghapus elemen dari depan antrean, peek sebagai pengakses elemen terdepan tanpa menghapusnya, dan display menampilkan semua elemen dalam antrean. Program ini juga menangani kondisi khusus seperti antrean kosong (underflow) dan mencetak pesan yang sesuai ketika operasi tidak dapat dilakukan (misalnya, mencoba mengeluarkan elemen dari antrean yang kosong).
   Code program:

```
#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 dari program:

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
```

3. Program ini merupakan implementasi dari queue yang menggunakan array untuk menyimpan elemen-elemen antrean. Dalam program ini terdapat deklarasi konstanta dan variabel global yang berfungsi untuk mendukung implementasi queue. Deklarasi kontanta maksimalQueue digunakan untuk menentukan kapasitas maksimum dari antrean. Front dan back merupakan variabel global yang berfungsi sebagai penanda untuk posisi elemen terdepan

dalam antrean dan sebagai penanda untuk posisi elemen terakhir dalam antrean. Deklarasi array queueTeller digunakan untuk menyimpan data dari antrean. Dalam hal ini, array dapat menampung hingga 5 elemen string. Setelah array tersebut, digunakan operasi dasar yang mendukung queue yaitu enqueue dan dequeue yang menambahkan dan menghapus elemen pada antrean, count untuk menghitung jumlah elemen, clear menghapus semua elemen dari antrean, lalu view yang menampilkan semua elemen dalam antrean. Program ini juga menunjukkan pentingnya manajemen memori dan pengelolaan data dalam struktur data.
Code program:

```
#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 dari program:

```
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
```

4. Unguided

1. Ubahlah penerapan konsep queue pada bagian guided dari array menjadi linked list.

   Pada program ini dibuat struktur data node dan menyesuaikan fungsi-fungsi antrian agar menggunakan linked list. Didefinisikan sebuah struktur Node yang memiliki dua atribut: data untuk menyimpan nilai

dan next untuk menunjuk ke node berikutnya. Membuat kelas queue yang memiliki pointer front dan back untuk menandai elemen depan dan belakang antrean. Digunakan fungsi enqueue dan dequeue yang akan menambahkan dan menghapus elemen pada antrean. Fungsi view, count, dan clear menampilkan elemen dalam antrian, menghitung jumlah elemen, dan menghapus semua elemen dari antrian.
Code program:

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

// Struktur untuk node dalam linked list
struct Node {
  string data;
  Node* next;
};

// Kelas Queue yang menggunakan linked list
class Queue {
private:
  Node* front; // Penanda depan antrian
  Node* back;  // Penanda belakang antrian
  int count;   // Jumlah elemen dalam antrian

public:
  Queue() {
    front = nullptr;
    back = nullptr;
    count = 0;
  }

  ~Queue() {
    clearQueue();
  }

  bool isFull() {
    // Untuk linked list, antrian tidak pernah penuh (kecuali memori habis)
    return false;
  }

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

```
void enqueueAntrian(string data) {
  Node* newNode = new Node();
  newNode->data = data;
  newNode->next = nullptr;

  if (isEmpty()) {
    front = newNode; // Jika antrian kosong, node baru menjadi front
  } else {
    back->next = newNode; // Hubungkan node baru ke belakang antrian
  }
  back = newNode; // Update back ke node baru
  count++; // Increment jumlah antrian
}

void dequeueAntrian() {
  if (isEmpty()) {
    cout << "Antrian kosong" << endl;
    return;
  }

  Node* temp = front; // Simpan node depan untuk dihapus
  front = front->next; // Update front ke node berikutnya

  if (front == nullptr) {
    back = nullptr; // Jika antrian menjadi kosong, update back juga
  }

  delete temp; // Hapus node yang sudah tidak diperlukan
  count--; // Decrement jumlah antrian
}

int countQueue() {
  return count;
}

void clearQueue() {
  while (!isEmpty()) {
    dequeueAntrian(); // Hapus semua elemen
  }
}

void viewQueue() {
  cout << "Data antrian teller:" << endl;
  Node* current = front;
```

```
    int index = 1;

    while (current != nullptr) {
      cout << index++ << ". " << current->data << endl;
      current = current->next;
    }

    // Jika antrian kosong
    if (isEmpty()) {
      cout << "(kosong)" << endl;
    }
  }
};

int main() {
  Queue queue; // Membuat objek antrian

  queue.enqueueAntrian("Andi");
  queue.enqueueAntrian("Maya");
  queue.viewQueue();
  cout << "Jumlah antrian = " << queue.countQueue() << endl;

  queue.dequeueAntrian();
  queue.viewQueue();
  cout << "Jumlah antrian = " << queue.countQueue() << endl;

  queue.clearQueue();
  queue.viewQueue();
  cout << "Jumlah antrian = " << queue.countQueue() << endl;

  return 0;
}
```

Output dari program:

```
Data antrian teller:
1. Andi
2. Maya
Jumlah antrian = 2
Data antrian teller:
1. Maya
Jumlah antrian = 1
Data antrian teller:
(kosong)
Jumlah antrian = 0
```

2. Membuat konsep antrean dengan atribut Nama mahasiswa dan NIM Mahasiswa.

   Untuk mengubahnya kita perlu memperbarui struktur Node dan menyesuaikan fungsi-fungsi antrian. Struktur Node sekarang memiliki dua atribut yaitu nama untuk menyimpan nama mahasiswa dan nim untuk menyimpan NIM mahasiswa, serta pointer next untuk menunjuk ke node berikutnya. Disini fungsi enqueue menerima dua parameter, yaitu nama dan nim, dan menyimpannya dalam node baru. Kemudian, fungsi view menampilkan data mahasiswa dalam antrian dengan format yang jelas, menunjukkan nama dan NIM masing-masing mahasiswa. Fungsi Dequeue, Count, dan Clear tetap sama, mengelola elemen dalam antrian dengan cara yang sama seperti sebelumnya. Implementasi ini sekarang memungkinkan kita untuk mengelola antrian mahasiswa dengan menyimpan nama dan NIM mereka.

   Code program:

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

// Struktur untuk node dalam linked list
struct Node {
  string nama; // Nama mahasiswa
  string nim;  // NIM mahasiswa
  Node* next;  // Pointer ke node berikutnya
};

// Kelas Queue yang menggunakan linked list
class Queue {
private:
  Node* front; // Penanda depan antrian
  Node* back;  // Penanda belakang antrian
  int count;   // Jumlah elemen dalam antrian

public:
  Queue() {
    front = nullptr;
    back = nullptr;
    count = 0;
  }

  ~Queue() {
    clearQueue();
  }
```

```
bool isFull() {
  // Untuk linked list, antrian tidak pernah penuh (kecuali memori habis)
  return false;
}

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

void enqueueAntrian(string nama, string nim) {
  Node* newNode = new Node();
  newNode->nama = nama;
  newNode->nim = nim;
  newNode->next = nullptr;

  if (isEmpty()) {
    front = newNode; // Jika antrian kosong, node baru menjadi front
  } else {
    back->next = newNode; // Hubungkan node baru ke belakang antrian
  }
  back = newNode; // Update back ke node baru
  count++; // Increment jumlah antrian
}

void dequeueAntrian() {
  if (isEmpty()) {
    cout << "Antrian kosong" << endl;
    return;
  }

  Node* temp = front; // Simpan node depan untuk dihapus
  front = front->next; // Update front ke node berikutnya

  if (front == nullptr) {
    back = nullptr; // Jika antrian menjadi kosong, update back juga
  }

  delete temp; // Hapus node yang sudah tidak diperlukan
  count--; // Decrement jumlah antrian
}

int countQueue() {
  return count;
```

```
    }

    void clearQueue() {
        while (!isEmpty()) {
            dequeueAntrian(); // Hapus semua elemen
        }
    }

    void viewQueue() {
        cout << "Data antrian mahasiswa:" << endl;
        Node* current = front;
        int index = 1;

        while (current != nullptr) {
            cout << index++ << ". Nama: " << current->nama << ", NIM: " << current->nim << endl;
            current = current->next;
        }

        // Jika antrian kosong
        if (isEmpty()) {
            cout << "(kosong)" << endl;
        }
    }
};

int main() {
    Queue queue; // Membuat objek antrian

    queue.enqueueAntrian("Alya", "2311104076");
    queue.enqueueAntrian("Winda", "2311104020");
    queue.enqueueAntrian("Anin", "2311104017");
    queue.enqueueAntrian("Fenny", "2311104027");
    queue.enqueueAntrian("Elsa", "2311104002");
    queue.enqueueAntrian("Yani", "2311104031");
    queue.viewQueue();
    cout << "Jumlah antrian = " << queue.countQueue() << endl;

    queue.dequeueAntrian();
    queue.viewQueue();
    cout << "Jumlah antrian = " << queue.countQueue() << endl;

    queue.clearQueue();
    queue.viewQueue();
```

```
    cout << "Jumlah antrian = " << queue.countQueue() << endl;

    return 0;
}
```

Output dari program:

```
Data antrian mahasiswa:
1. Nama: Alya, NIM: 2311104076
2. Nama: Winda, NIM: 2311104020
3. Nama: Anin, NIM: 2311104017
4. Nama: Fenny, NIM: 2311104027
5. Nama: Elsa, NIM: 2311104002
6. Nama: Yani, NIM: 2311104031
Jumlah antrian = 6
Data antrian mahasiswa:
1. Nama: Winda, NIM: 2311104020
2. Nama: Anin, NIM: 2311104017
3. Nama: Fenny, NIM: 2311104027
4. Nama: Elsa, NIM: 2311104002
5. Nama: Yani, NIM: 2311104031
Jumlah antrian = 5
Data antrian mahasiswa:
(kosong)
Jumlah antrian = 0
```

3. Modifikasi program pertama sehingga dapat diprioritaskan berdasarkan NIM. Pada program ini perlu melakukan beberapa perubahan pada fungsi enqueue. Kita akan menambahkan logika untuk menyisipkan node baru ke dalam antrian dengan mempertimbangkan urutan NIM. Logika ini diterapkan dalam fungsi **enqueue**, di mana node baru disisipkan ke dalam antrian dengan mempertimbangkan urutan NIM. Ini memastikan bahwa mahasiswa dengan NIM yang lebih kecil akan selalu berada di depan antrian. Untuk fungsi dasar lainnya masih sama seperti yang sebelumnya. Secara keseluruhan, implementasi ini memberikan solusi yang efisien dan terorganisir untuk mengelola antrian mahasiswa dengan prioritas berdasarkan NIM, memanfaatkan keunggulan dari struktur data linked list.
   Code program:

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

// Struktur untuk node dalam linked list
struct Node {
    string nama; // Nama mahasiswa
```

```
    string nim;  // NIM mahasiswa
    Node* next;  // Pointer ke node berikutnya
};

// Kelas Queue yang menggunakan linked list
class Queue {
private:
    Node* front; // Penanda depan antrian
    Node* back;  // Penanda belakang antrian
    int count;   // Jumlah elemen dalam antrian

public:
    Queue() {
        front = nullptr;
        back = nullptr;
        count = 0;
    }

    ~Queue() {
        clearQueue();
    }

    bool isFull() {
        // Untuk linked list, antrian tidak pernah penuh (kecuali memori habis)
        return false;
    }

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

    void enqueueAntrian(string nama, string nim) {
        Node* newNode = new Node();
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

        // Jika antrian kosong, tambahkan node baru
        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            // Jika NIM dari node baru lebih kecil dari front, masukkan di depan
            if (nim < front->nim) {
```

```
        newNode->next = front;
        front = newNode;
      } else {
        // Cari posisi untuk menyisipkan node baru
        Node* current = front;
        while (current->next != nullptr && current->next->nim < nim) {
          current = current->next; // Cari node yang NIM-nya lebih besar
        }
        // Sisipkan node baru
        newNode->next = current->next;
        current->next = newNode;

        // Jika node baru ditambahkan di belakang, update back
        if (newNode->next == nullptr) {
          back = newNode;
        }
      }
    }
    count++; // Increment jumlah antrian
}

void dequeueAntrian() {
    if (isEmpty()) {
      cout << "Antrian kosong" << endl;
      return;
    }

    Node* temp = front; // Simpan node depan untuk dihapus
    front = front->next; // Update front ke node berikutnya

    if (front == nullptr) {
      back = nullptr; // Jika antrian menjadi kosong, update back juga
    }

    delete temp; // Hapus node yang sudah tidak diperlukan
    count--; // Decrement jumlah antrian
}

int countQueue() {
    return count;
}

void clearQueue() {
    while (!isEmpty()) {
```

```
        dequeueAntrian(); // Hapus semua elemen
    }
  }

  void viewQueue() {
    cout << "Data antrian mahasiswa:" << endl;
    Node* current = front;
    int index = 1;

    while (current != nullptr) {
      cout << index++ << ". Nama: " << current->nama << ", NIM: " << current-
>nim << endl;
      current = current->next;
    }

    // Jika antrian kosong
    if (isEmpty()) {
      cout << "(kosong)" << endl;
    }
  }
};

int main() {
  Queue queue; // Membuat objek antrian

  queue.enqueueAntrian("Alya", "2311104076");
  queue.enqueueAntrian("Winda", "2311104020");
  queue.enqueueAntrian("Anin", "2311104017");
  queue.enqueueAntrian("Fenny", "2311104027");
  queue.enqueueAntrian("Elsa", "2311104002");
  queue.enqueueAntrian("Yani", "2311104031");

  queue.viewQueue();
  cout << "Jumlah antrian = " << queue.countQueue() << endl;

  queue.dequeueAntrian();
  queue.viewQueue();
  cout << "Jumlah antrian = " << queue.countQueue() << endl;

  queue.clearQueue();
  queue.viewQueue();
  cout << "Jumlah antrian = " << queue.countQueue() << endl;

  return 0;
```

```
}
```

Output dari program:

```
Data antrian mahasiswa:
1. Nama: Elsa, NIM: 2311104002
2. Nama: Anin, NIM: 2311104017
3. Nama: Winda, NIM: 2311104020
4. Nama: Fenny, NIM: 2311104027
5. Nama: Yani, NIM: 2311104031
6. Nama: Alya, NIM: 2311104076
Jumlah antrian = 6
Data antrian mahasiswa:
1. Nama: Anin, NIM: 2311104017
2. Nama: Winda, NIM: 2311104020
3. Nama: Fenny, NIM: 2311104027
4. Nama: Yani, NIM: 2311104031
5. Nama: Alya, NIM: 2311104076
Jumlah antrian = 5
Data antrian mahasiswa:
(kosong)
Jumlah antrian = 0
```

5. Kesimpulan

Dari praktikum ini dapat disimpulkan bahwa queue adalah struktur data linear yang bekerja berdasarkan prinsip First In, First Out (FIFO), di mana elemen yang pertama kali masuk akan keluar lebih dahulu. Memperkenalkan operasi-operasi dasar pada queue seperti enqueue, dequeue, count, view, clear, dsb. Pada tugas dari laporan ini juga membuat implementasi berbasis array, queue memiliki ukuran tetap sehingga membutuhkan manajemen memori yang ketat untuk menghindari kondisi *overflow* atau *underflow*. Kemudian ada, Implementasi berbasis linked list yang memungkinkan ukuran queue bersifat dinamis, memanfaatkan memori lebih efisien, dan memudahkan penambahan atau penghapusan elemen. Lalu ada modifikasi program pada guided dimana queue dikembangkan untuk menyimpan atribut kompleks, seperti nama mahasiswa dan NIM, yang menunjukkan fleksibilitas struktur data ini. Modifikasi lebih lanjut dilakukan untuk menerapkan prioritas berdasarkan NIM, di mana mahasiswa dengan NIM lebih kecil akan memiliki prioritas lebih tinggi dalam antrean.