

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN ALGORITMA  
MODUL 7  
“QUEUE”**



**DISUSUN OLEH:  
TSAQIF KANZ AHMAD  
2311102075  
IF-11-B**

**DOSEN:**

**WAHYU ANDI SAPUTRA S.Pd., M.Eng.**

**PROGRAM STUDI S1 TEKNIK  
INFORMATIKA  
FAKULTAS INFORMATIKA  
INSTITUT TEKNOLOGI TELKOM PURWOKERTO  
PURWOKERTO  
2024**

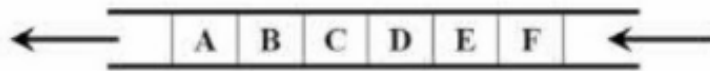
## **TUJUAN PRAKTIKUM**

1. Mampu menjelaskan definisi dan konsep dari double queue.
2. Mampu menerapkan operasi tambah, menghapus pada queue.
3. Mampu menerapkan operasi tampil data pada queue

## DASAR TEORI

### 1). PENGERTIAN QUEUE

Queue adalah suatu kumpulan data yang mana penambahan data atau elemen hanya dapat dilakukan pada sisi belakang sedangkan penghapusan atau pengeluaran elemen dilakukan pada sisi depan. Queue (Antrian) dapat diartikan sebagai suatu kumpulan data yang seolah-olah terlihat seperti ada data yang diletakkan di sebelah data yang lain. Pada pengurutannya, data masuk melalui lorong di sebelah kanan dan masuk dari terowongan sebelah kiri. Hal ini membuat Queue (Antrian) bersifat FIFO (First In First Out).



Karakteristik utama dari Queue adalah prinsip FIFO (First-In-First-Out). Elemen pertama yang dimasukkan ke dalam Queue akan menjadi elemen pertama yang diambil atau dihapus dari Queue. Elemen-elemen baru ditambahkan di ujung belakang Queue dan elemen-elemen yang sudah ada dikeluarkan dari ujung depan Queue. Dengan prinsip FIFO ini, Queue dapat membantu mengatur urutan data dan mempertahankan prioritas saat memproses elemen-elemen yang ada di dalamnya. Penggunaan Queue dalam kehidupan sehari-hari dapat ditemukan di berbagai situasi. Misalnya, saat mengantri di sebuah toko, bank, atau tempat layanan pelanggan. Orang yang pertama kali mengantri akan dilayani terlebih dahulu dan orang yang datang kemudian akan menunggu giliran mereka sesuai dengan urutan kedatangan. Selain itu, Queue juga digunakan dalam sistem pemrosesan data, seperti antrian pesan di aplikasi komunikasi atau penjadwalan tugas pada sistem operasi. Kelebihan utama pada queue data berjumlah besar yang dapat dikelola secara efisien dan efektif dengan mudah serta menangani proses dan tugas sesuai urutan kedatangan secara efisien.

### 2). Jenis-jenis Queue

#### a. Queue Linear (Implementasi Queue dengan batasan linier)

Queue Linear adalah implementasi Queue yang menggunakan struktur data linear seperti array atau linked list. Pada Queue Linear, elemen-elemen ditambahkan di ujung belakang (rear) dan dihapus dari ujung depan (front). Namun, Queue Linear memiliki batasan ukuran tetap yang dapat mengakibatkan situasi ketika Queue penuh dan tidak dapat menampung elemen baru meskipun masih ada ruang kosong pada struktur data yang digunakan.

#### b. Queue Circular (Implementasi Queue dengan memanfaatkan siklus)

Queue Circular adalah implementasi Queue yang menggunakan struktur data linear seperti array dengan pemanfaatan siklus. Pada Queue Circular, elemen-elemen tetap ditambahkan di ujung belakang (rear) dan dihapus dari ujung depan (front), namun jika Queue mencapai batas maksimum, elemen-elemen baru akan ditempatkan di awal antrian. Dengan memanfaatkan siklus, Queue Circular dapat memanfaatkan kembali ruang yang tersedia dan menghindari pemborosan memori.

#### c. Priority Queue (Implementasi Queue dengan prioritas pada setiap elemen)

Priority Queue adalah implementasi Queue di mana setiap elemen memiliki prioritas tertentu. Elemen-elemen dalam Priority Queue diurutkan berdasarkan prioritas mereka, sehingga elemen dengan prioritas yang lebih tinggi akan diberikan akses lebih awal daripada elemen dengan prioritas yang lebih rendah. Prioritas dapat ditentukan

menggunakan aturan yang telah ditetapkan sebelumnya atau dengan membandingkan nilai-nilai elemen. Implementasi Priority Queue dapat menggunakan struktur data seperti heap untuk memastikan elemen-elemen tetap terurut secara tepat.

Jenis-jenis Queue ini memungkinkan kita untuk memilih implementasi yang sesuai dengan kebutuhan aplikasi. Queue Linear cocok digunakan ketika ukuran Queue tetap dan memori tidak menjadi kendala. Queue Circular bermanfaat ketika kita ingin memanfaatkan kembali ruang yang tersedia pada struktur data. Sedangkan, Priority Queue berguna ketika elemen-elemen dalam Queue memiliki prioritas yang berbeda dan kita ingin mengakses elemen dengan prioritas tertinggi terlebih dahulu.

### 3). IMPLEMENTASI QUEUE

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)**

#### A). Implementasi Queue menggunakan array

Implementasi Queue menggunakan array melibatkan penggunaan array sebagai wadah untuk menyimpan elemen-elemen dalam antrian. Dalam implementasi ini, kita menggunakan dua penunjuk, yaitu penunjuk front dan penunjuk rear, yang menunjukkan posisi elemen pertama dan terakhir dalam Queue. Ketika elemen baru ditambahkan, penunjuk rear akan maju ke posisi berikutnya, dan saat elemen dihapus, penunjuk front akan maju ke posisi berikutnya. Array memiliki keuntungan akses elemen yang cepat berdasarkan indeksnya, tetapi memiliki keterbatasan ukuran tetap yang tidak fleksibel.

#### B). Implementasi Queue menggunakan linked list

Implementasi Queue menggunakan linked list melibatkan penggunaan simpul-simpul yang saling terhubung untuk menyimpan elemen-elemen dalam antrian. Setiap simpul memiliki dua bagian, yaitu data (elemen dalam Queue) dan pointer yang menunjuk ke simpul berikutnya. Dalam implementasi ini, kita menggunakan dua penunjuk, yaitu penunjuk front dan penunjuk rear, yang menunjukkan simpul pertama dan terakhir dalam Queue. Ketika elemen baru ditambahkan, simpul baru akan ditautkan ke simpul terakhir dan penunjuk rear akan diperbarui. Ketika elemen dihapus, simpul pertama akan dilepaskan dan penunjuk front akan diperbarui. Linked list memungkinkan penambahan dan penghapusan elemen dengan fleksibilitas, tetapi memiliki overhead memori untuk menyimpan pointer tambahan.

#### C). Perbandingan antara implementasi menggunakan array dan linked list

Implementasi menggunakan array memiliki akses elemen yang cepat berdasarkan indeksnya dan tidak memerlukan alokasi memori dinamis, tetapi memiliki batasan ukuran tetap yang tidak fleksibel. Sedangkan, implementasi

menggunakan linked list memungkinkan ukuran antrian yang fleksibel, namun memiliki overhead memori untuk menyimpan pointer tambahan dan mengakses elemen membutuhkan iterasi melalui simpul-simpul. Pilihan antara array dan linked list untuk implementasi Queue bergantung pada kebutuhan aplikasi dan trade-off antara kecepatan akses dan fleksibilitas ukuran.

#### D). Contoh implementasi Queue pada C++

```
#include <iostream>
#include <queue>

int main() {
    // Membuat objek Queue
    std::queue<int> q;

    // Menambahkan elemen ke dalam Queue
    q.push(10);
    q.push(20);
    q.push(30);

    // Menghapus elemen dari Queue
    q.pop();

    // Mendapatkan elemen pertama dalam Queue
    int front = q.front();

    // Mendapatkan elemen terakhir dalam Queue
    int rear = q.back();

    // Mendapatkan jumlah elemen dalam Queue
    int size = q.size();

    // Memeriksa apakah Queue kosong
    bool isEmpty = q.empty();

    return 0;
}
```

#### 4). OPERASI DASAR PADA QUEUE

##### a. Enqueue (Menambahkan elemen ke dalam Queue)

Operasi Enqueue digunakan untuk menambahkan elemen baru ke dalam Queue. Elemen baru ini akan ditempatkan di posisi belakang Queue (rear). Saat melakukan Enqueue, penunjuk rear akan maju ke posisi berikutnya untuk menunjuk elemen baru tersebut.

##### b. Dequeue (Menghapus elemen dari Queue)

Operasi Dequeue digunakan untuk menghapus elemen pertama dari Queue. Elemen yang dihapus ini merupakan elemen yang berada di posisi depan Queue (front). Saat melakukan Dequeue, penunjuk front akan maju ke posisi berikutnya untuk menunjuk elemen setelahnya, dan elemen yang dihapus tidak lagi termasuk dalam Queue.

##### c. Front (Mendapatkan elemen pertama dalam Queue)

Operasi Front digunakan untuk mendapatkan elemen pertama dalam Queue tanpa menghapusnya. Dengan menggunakan operasi ini, kita dapat melihat elemen yang akan dikeluarkan selanjutnya dari Queue. Biasanya, operasi ini hanya digunakan untuk melihat elemen pertama tanpa mengubah struktur Queue.

**d. Rear (Mendapatkan elemen terakhir dalam Queue)**

Operasi Rear digunakan untuk mendapatkan elemen terakhir dalam Queue tanpa menghapusnya. Dengan menggunakan operasi ini, kita dapat melihat elemen terakhir yang telah ditambahkan ke dalam Queue. Biasanya, operasi ini hanya digunakan untuk melihat elemen terakhir tanpa mengubah struktur Queue.

**e. IsEmpty (Memeriksa apakah Queue kosong)**

Operasi IsEmpty digunakan untuk memeriksa apakah Queue kosong atau tidak. Jika Queue tidak memiliki elemen, berarti Queue kosong, dan operasi ini akan mengembalikan nilai benar (true). Jika Queue memiliki elemen, berarti Queue tidak kosong, dan operasi ini akan mengembalikan nilai salah (false).

**f. Size (Mendapatkan jumlah elemen dalam Queue)**

Operasi Size digunakan untuk mendapatkan jumlah elemen yang ada dalam Queue. Dengan menggunakan operasi ini, kita dapat mengetahui berapa banyak elemen yang saat ini ada dalam Queue tanpa mengubah struktur atau menghapus elemen.

Operasi-operasi dasar tersebut penting dalam mengelola Queue, karena mereka memungkinkan kita untuk menambahkan, menghapus, dan mengakses elemen-elemen dalam urutan yang sesuai dengan prinsip FIFO (First-In-First-Out). Pada Queue, operasi tersebut dilakukan ditempat berbeda (melalui salah satu ujung) karena perubahan data selalu mengacu pada Head, maka hanya ada 1 jenis insert maupun delete. Prosedur ini sering disebut Enqueue dan Dequeue pada kasus Queue. Untuk Enqueue, cukup tambahkan elemen setelah elemen terakhir Queue, dan untuk Dequeue, cukup "geser"kan Head menjadi elemen selanjutnya.

## **5). PERBEDAAN STACK DAN QUEUE**

Perbedaan antara stack dan queue terdapat pada aturan penambahan dan penghapusan elemen. Pada stack, operasi penambahan dan penghapusan elemen dilakukan di satu ujung. Elemen yang terakhir kali dimasukkan akan berada paling dekat dengan ujung atau dianggap paling atas sehingga pada operasi penghapusan, elemen teratas tersebut akan dihapus paling awal, sifat demikian dikenal dengan LIFO. Pada queue, operasi tersebut dilakukan di tempat yang berbeda. Penambahan (**Enqueue**) elemen selalu dilakukan melalui salah satu ujung, menempati posisi di belakang elemen-elemen yang sudah masuk sebelumnya atau menjadi elemen paling belakang. Sedangkan penghapusan (**Dequeue**) elemen dilakukan di ujung yang berbeda, yaitu pada posisi elemen yang masuk paling awal atau paling depan. Sifat yang demikian bersifat FIFO.

## PENJELASAN GUIDED

### 1). SOURCE CODE

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

const int maksimalQueue = 5; // Batas maksimal antrian
int front = 0;                // Indeks awal antrian
int back = 0;                 // Indeks akhir antrian
string queueTeller[maksimalQueue]; // Array untuk menyimpan antrian

// Fungsi untuk memeriksa apakah antrian penuh
bool isFull()
{
    return back == maksimalQueue;
}

// Fungsi untuk memeriksa apakah antrian kosong
bool isEmpty()
{
    return back == 0;
}

// Fungsi untuk menambahkan elemen ke antrian
void enqueueAntrian(string data)
{
    if (isFull())
    {
        cout << "Antrian penuh" << endl;
    }
    else
    {
        queueTeller[back] = data;
        back++;
    }
}

// Fungsi untuk menghapus elemen dari antrian
void dequeueAntrian()
{
    if (isEmpty())
    {
        cout << "Antrian kosong" << endl;
    }
    else
    {
        for (int i = 0; i < back; i++)
        {
            queueTeller[i] = queueTeller[i + 1];
        }
        queueTeller[back - 1] = ""; // Membersihkan data terakhir
    }
}
```

```

        back--;
    }
}

// Fungsi untuk menghitung jumlah elemen dalam antrian
int countQueue()
{
    return back;
}

// Fungsi untuk mengkosongkan semua elemen dalam antrian
void clearQueue()
{
    for (int i = 0; i < back; i++)
    {
        queueTeller[i] = "";
    }
    back = 0;
    front = 0;
}

// Fungsi untuk menampilkan semua elemen dalam antrian
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;
}

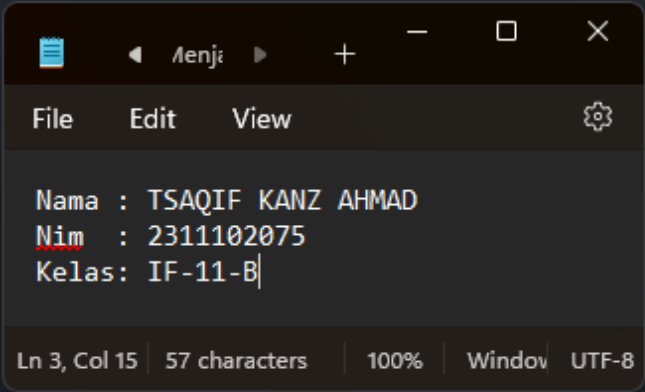
```



```
}  
    return 0;  
}
```

## SCREENSHOT OUTPUT

```
PS C:\Users\ACER> cd "C:\Users\ACER\AppData\Local\Temp\" ; if ($?) { g++ tempCodeRunnerFile.cpp  
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  
PS C:\Users\ACER\AppData\Local\Temp>
```



The screenshot shows a Notepad++ window titled 'Menjari' with a menu bar (File, Edit, View) and a status bar (Ln 3, Col 15 | 57 characters | 100% | Window | UTF-8). The text in the window is:

```
Nama : TSAQIF KANZ AHMAD  
Nim  : 2311102075  
Kelas: IF-11-B|
```

## DESKRIPSI PROGRAM

Program di atas adalah program menstimulasikan antrian teller di bank. Kode ini menggunakan array untuk menyimpan data antrian dan menyediakan fungsi untuk menambahkan, menghapus, menghitung, dan menampilkan elemen dalam antrian.

## PENJELASAN UNGUIDED

1. Ubahlah penerapan konsep queue pada bagian guided dari array menjadi linked list !
2. Dari nomor 1 buatlah konsep antri dengan atribut Nama mahasiswa dan NIM

### SOURCE CODE

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

struct Node {
    string nama;
    string nim;
    Node* next;
};

class Queue {
private:
    Node* front;
    Node* back;
    int size;

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

    bool isFull() {
        return false; // Linked list based queue is never full unless out of memory
    }

    bool isEmpty() {
        return size == 0;
    }

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

        if (isEmpty()) {
            front = newNode;
            back = newNode;
        } else {
            back->next = newNode;
            back = newNode;
        }
        size++;
    }
};
```

```

    }

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

    int countQueue() {
        return size;
    }

    void clearQueue() {
        while (!isEmpty()) {
            dequeueAntrian();
        }
    }

    void viewQueue() {
        cout << "Data antrian mahasiswa:" << endl;
        Node* current = front;
        int position = 1;
        while (current != nullptr) {
            cout << position << ". " << current->nama << " (" << current->nim << ")" << endl;
            current = current->next;
            position++;
        }
    }
};

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

    do {
        cout << "Menu Antrian Mahasiswa:" << endl;
        cout << "1. Tambah Antrian" << endl;
        cout << "2. Hapus Antrian" << endl;
        cout << "3. Lihat Antrian" << endl;
        cout << "4. Jumlah Antrian" << endl;
        cout << "5. Hapus Semua Antrian" << endl;
        cout << "0. Keluar" << endl;
        cout << "Pilih menu: ";
        cin >> pilihan;

        switch (pilihan) {

```

```

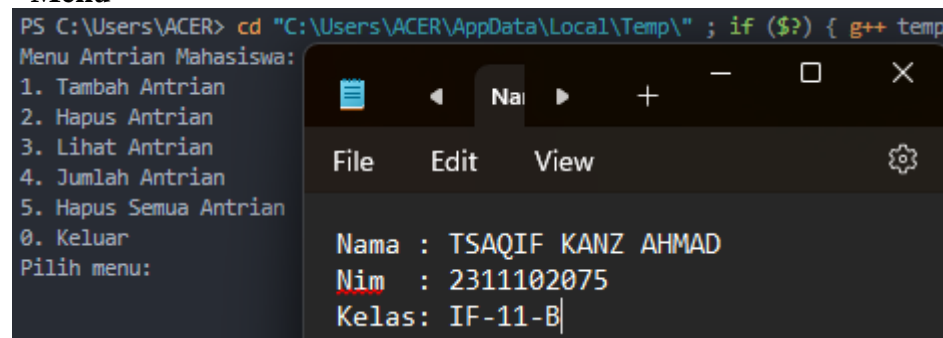
        case 1:
            cout << "Masukkan Nama: ";
            cin.ignore();
            getline(cin, nama);
            cout << "Masukkan NIM: ";
            getline(cin, nim);
            antrian.enqueueAntrian(nama, nim);
            break;
        case 2:
            antrian.dequeueAntrian();
            break;
        case 3:
            antrian.viewQueue();
            break;
        case 4:
            cout << "Jumlah antrian = " << antrian.countQueue() << endl;
            break;
        case 5:
            antrian.clearQueue();
            break;
        case 0:
            cout << "Keluar dari program." << endl;
            break;
        default:
            cout << "Pilihan tidak valid." << endl;
    }
} while (pilihan != 0);

return 0;
}

```

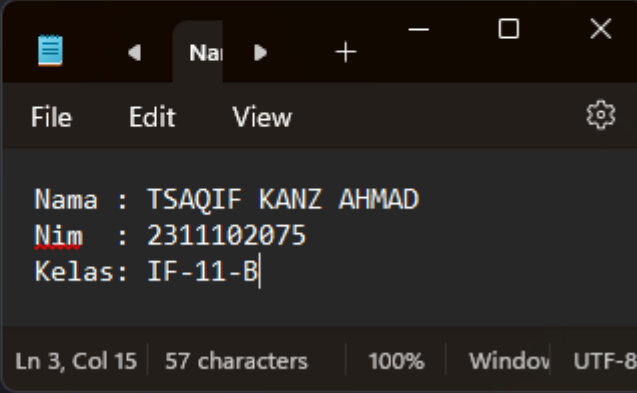
## SCREENSHOT OUTPUT

### - Menu



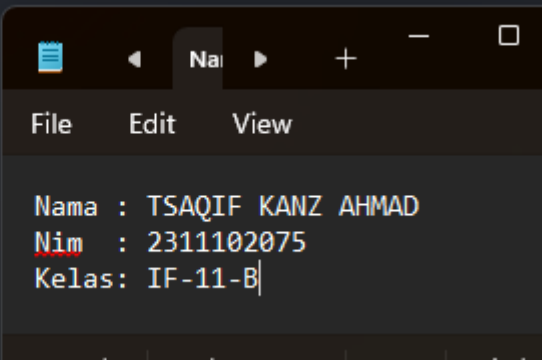
### -Menambahkan antrian

```
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 1
Masukkan Nama: Tsaqif
Masukkan NIM: 2311102075
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 1
Masukkan Nama: Naufal
Masukkan NIM: 2311102078
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 1
Masukkan Nama: Rizky
Masukkan NIM: 2311102061
```



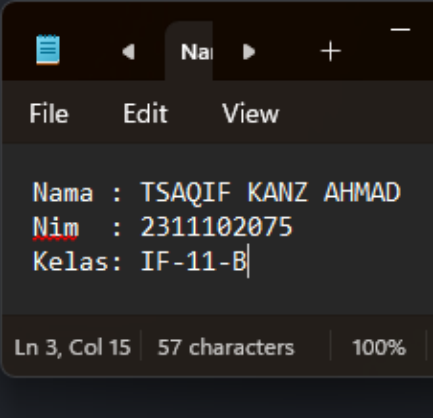
### - Melihat antrian

```
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 3
Data antrian mahasiswa:
1. Tsaqif (2311102075)
2. Naufal (2311102078)
3. Rizky (2311102061)
```



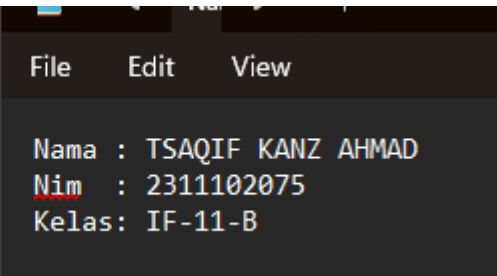
### - Menghapus antrian

```
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 2
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 3
Data antrian mahasiswa:
1. Naufal (2311102078)
2. Rizky (2311102061)
```



### - Jumlah antrian

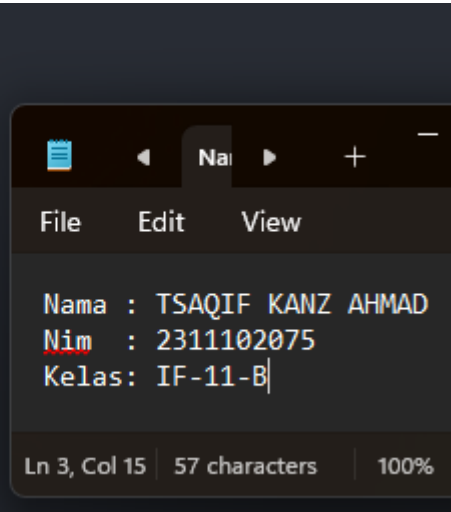
```
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 4
Jumlah antrian = 2
```



The screenshot shows a terminal window with a dark background. On the left, a menu titled 'Menu Antrian Mahasiswa:' lists five options: '1. Tambah Antrian', '2. Hapus Antrian', '3. Lihat Antrian', '4. Jumlah Antrian', and '5. Hapus Semua Antrian'. Below the menu, it says '0. Keluar' and 'Pilih menu: 4'. The output of the program is 'Jumlah antrian = 2'. On the right, a window titled 'Nama : TSAQIF KANZ AHMAD' displays user information: 'Nama : TSAQIF KANZ AHMAD', 'Nim : 2311102075', and 'Kelas: IF-11-B'. The window has a menu bar with 'File', 'Edit', and 'View'.

### - Hapus semua antrian

```
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 5
Menu Antrian Mahasiswa:
1. Tambah Antrian
2. Hapus Antrian
3. Lihat Antrian
4. Jumlah Antrian
5. Hapus Semua Antrian
0. Keluar
Pilih menu: 3
Data antrian mahasiswa:
```



The screenshot shows a terminal window with a dark background. On the left, a menu titled 'Menu Antrian Mahasiswa:' lists five options: '1. Tambah Antrian', '2. Hapus Antrian', '3. Lihat Antrian', '4. Jumlah Antrian', and '5. Hapus Semua Antrian'. Below the menu, it says '0. Keluar' and 'Pilih menu: 5'. The output of the program is 'Menu Antrian Mahasiswa: 1. Tambah Antrian 2. Hapus Antrian 3. Lihat Antrian 4. Jumlah Antrian 5. Hapus Semua Antrian 0. Keluar'. Below this, it says 'Pilih menu: 3' and 'Data antrian mahasiswa:'. On the right, a window titled 'Nama : TSAQIF KANZ AHMAD' displays user information: 'Nama : TSAQIF KANZ AHMAD', 'Nim : 2311102075', and 'Kelas: IF-11-B'. The window has a menu bar with 'File', 'Edit', and 'View'. At the bottom, it shows 'Ln 3, Col 15 | 57 characters | 100%'.

## DESKRIPSI PROGRAM

*Program diatas adalah program yang menggunakan linked list untuk mengimplementasikan antrian mahasiswa. Program ini menyediakan menu interaktif untuk melakukan berbagai operasi pada antrian, seperti menambah, menghapus, melihat, menghitung, dan menghapus semua mahasiswa dalam antrian..*

## KESIMPULAN

**Queue** adalah struktur data abstrak yang sangat penting dalam pemrograman dan pengembangan aplikasi dengan mengikuti prinsip First In, First Out (FIFO). Artinya, elemen data yang pertama kali dimasukkan ke dalam antrian akan menjadi yang pertama kali dikeluarkan. Queue sering digunakan untuk memodelkan situasi di mana elemen data harus diproses secara berurutan supaya adil dan teratur. Dalam kehidupan sehari-hari queue dapat ditemukan dalam berbagai aplikasi dan skenario, mulai dari penjadwalan tugas pada sistem operasi, pengelolaan antrian pesanan hingga pemodelan sistem yang kompleks. Penerapan dan pemahaman yang baik tentang queue memungkinkan untuk mengoptimalkan kinerja sistem, mengatasi masalah antrian dengan efisien, serta memprioritaskan tugas dan layanan berdasarkan tingkat kepentingan atau prioritas. Dengan begitu pengguna dapat membangun aplikasi yang lebih efisien, dapat diandalkan serta dapat mengelola antrian tugas dengan baik.

## DAFTAR PUSTAKA

- [1] Asisten Praktikum, "Modul 7 QUEUE" : [MODUL 7 - QUEUE \(1\).pdf](#)
- [2] Medium - Queue: Pengenalan, Implementasi, Operasi Dasar, dan Aplikasi :  
<https://medium.com/@furatamarizuki/queue-pengenalan-implementasi-operasi-dasar-dan-aplikasi-c5eed7e871a3>
- [3] Academia - LAPORAN PRAKTIKUM 5 ALGORITMA STRUKTUR DATA-QUEUE :  
[https://www.academia.edu/32592023/LAPORAN\\_PRAKTIKUM\\_5\\_ALGORITMA\\_STRUKTUR\\_DATA\\_QUEUE](https://www.academia.edu/32592023/LAPORAN_PRAKTIKUM_5_ALGORITMA_STRUKTUR_DATA_QUEUE)