# LAPORAN PRAKTIKUM

# MODUL 7
# QUEUE

**Disusun oleh:**
**Tegar Bangkit Wijaya**
**NIM: 2311102027**

**Dosen Pengampu:**

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

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

# BAB I
# TUJUAN PRAKTIKUM

A. TUJUAN PRAKTIKUM

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

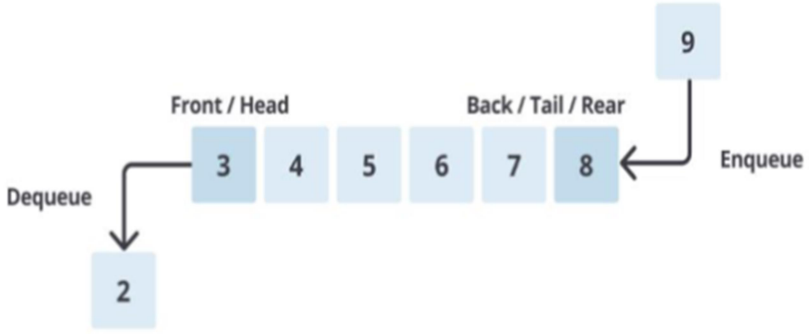
# BAB II

# DASAR TEORI

## B.DASAR TEORI

Queue adalah struktur data yang digunakan untuk menyimpan data dengan metode FIFO (First-In First-Out). Data yang pertama dimasukkan ke dalam queue akan menjadi data 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




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 diinputkan akan berada paling dengan 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 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

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.

# BAB III

## GUIDED

### 1. Guided 1

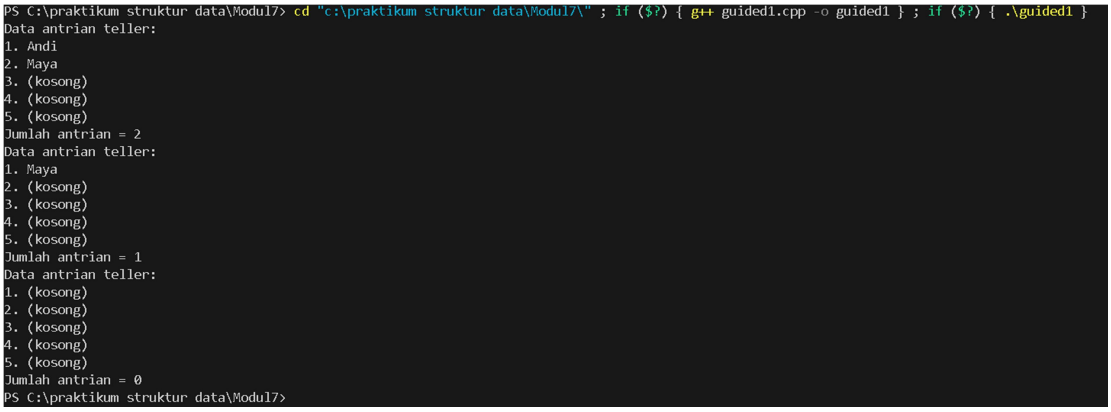
**Source code**

```
#include <iostream>

using namespace std;

const int maksimalQueue = 5;  // Maksimal antrian
int front = 0;                // Penanda antrian
int back = 0;                 // Penanda
string queueTeller[5];        // Fungsi pengecekan

bool isFull() {               // Pengecekan antrian penuh atau
tidak
  if (back == maksimalQueue) {
    return true;  // =1
  } else {
    return false;
  }
}

bool isEmpty() {  // Antriannya kosong atau tidak
  if (back == 0) {
    return true;
  } else {
    return false;
  }
}

void enqueueAntrian(string data) {  // Fungsi menambahkan antrian
  if (isFull()) {
    cout << "Antrian penuh" << endl;
  } else {
    if (isEmpty()) {  // Kondisi ketika queue 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;
}
```

**Screenshoot program**




**Deskripsi program**

Program berikut adalah implementasi dasar dari struktur data antrian (queue) menggunakan array statis dengan kapasitas tetap sebesar 5 elemen. Program ini mencakup beberapa fungsi dasar untuk mengelola antrian, termasuk menambahkan elemen ke antrian (enqueue), menghapus elemen dari antrian (dequeue), memeriksa apakah antrian penuh atau kosong, menghitung jumlah elemen dalam antrian,

mengosongkan seluruh elemen dalam antrian, dan menampilkan isi antrian.

Fungsi `isFull` memeriksa apakah antrian telah mencapai kapasitas maksimum, sedangkan fungsi `isEmpty` memeriksa apakah antrian kosong. Fungsi `enqueueAntrian` menambahkan elemen ke antrian jika masih ada ruang, dan menampilkan pesan jika antrian penuh. Fungsi `dequeueAntrian` menghapus elemen dari antrian dan menggeser elemen-elemen yang tersisa ke depan, serta menampilkan pesan jika antrian kosong. Fungsi `countQueue` mengembalikan jumlah elemen dalam antrian. Fungsi `clearQueue` menghapus semua elemen dalam antrian dan mengatur ulang penanda depan dan belakang. Fungsi `viewQueue` menampilkan semua elemen dalam antrian, menunjukkan posisi yang kosong dengan label "(kosong)".

## LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   **Source code**

```
// Font Courier New (10)
```

   **Screenshoot program**

   **Deskripsi program**

2. **Unguided 2**

## LATIHAN KELAS - UNGUIDED

### 1. Unguided 1

**Source code**

```
#include <iostream>
#include <string>
using namespace std;
struct Node
{
    string data;
    Node *next;
};
class Queue
{
private:
    Node *front;
    Node *rear;

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

```
            front = rear = newNode;
        }
        else
        {
            rear->next = newNode;
            rear = newNode;
        }
    }
    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Queue is empty" << endl;
        }
        else
        {
            Node *temp = front;
            front = front->next;
            delete temp;
            if (front == nullptr)
            {
                rear = nullptr;
            }
        }
    }
    int count()
    {
        int cnt = 0;
        Node *current = front;
        while (current != nullptr)
        {
            cnt++;
            current = current->next;
        }
        return cnt;
    }
    void clear()
    {
        while (!isEmpty())
        {
            dequeue();
```

```
        }
    }
    void view()
    {
        cout << "Queue contents:" << endl;
        Node *current = front;
        int index = 1;
        while (current != nullptr)
        {
            cout << index << ". " << current->data << endl;
            current = current->next;
            index++;
        }
        if (index == 1)
        {
            cout << "(empty)" << endl;
        }
    }
};
int main()
{
    Queue q;
    q.enqueue("Andi");
    q.enqueue("Naya");
    q.view();
    cout << "Queue size = " << q.count() << endl;
    q.dequeue();
    q.view();
    cout << "Queue size = " << q.count() << endl;
    q.clear();
    q.view();
    cout << "Queue size = " << q.count() << endl;
    return 0;
}
```

**Screenshoot program**

```
 ($?) { .\tempCodeRunnerFile }
Queue contents:
1. Andi
2. Naya
Queue size = 2
Queue contents:
1. Naya
Queue size = 1
Queue contents:
(empty)
Queue size = 0
PS C:\praktikum struktur data\Modul7>
```

**Deskripsi program**

Program ini mendefinisikan struktur Node yang menyimpan data dalam bentuk string dan memiliki pointer ke node berikutnya. Kelas Queue menggunakan dua pointer, yaitu front dan rear, untuk menunjukkan elemen pertama dan terakhir dari antrian. Kelas ini menyediakan beberapa metode utama:

- **isFull**: Memeriksa apakah antrian penuh. Dalam konteks linked list, antrian tidak pernah penuh sehingga selalu mengembalikan false.
- **isEmpty**: Memeriksa apakah antrian kosong dengan melihat apakah front adalah nullptr.
- **enqueue**: Menambah elemen baru ke antrian. Jika antrian kosong, elemen baru menjadi elemen pertama dan terakhir. Jika tidak, elemen baru ditambahkan ke bagian belakang antrian.
- **dequeue**: Menghapus elemen dari depan antrian. Jika antrian kosong, akan menampilkan pesan bahwa antrian kosong. Jika tidak, menghapus elemen depan dan memperbarui pointer front.
- **count**: Menghitung jumlah elemen dalam antrian dengan iterasi dari depan hingga belakang.

- **clear**: Mengosongkan antrian dengan menghapus semua elemen satu per satu.
- **view**: Menampilkan semua elemen dalam antrian.

Dalam fungsi main, program menguji fungsi antrian dengan menambahkan dua elemen ("Andi" dan "Naya"), menampilkan isi antrian dan ukurannya, menghapus satu elemen, menampilkan isi antrian dan ukurannya lagi, mengosongkan antrian, serta menampilkan isi antrian dan ukurannya setelah dikosongkan. Program ini menunjukkan cara menggunakan linked list untuk mengelola antrian dinamis yang dapat menambah dan menghapus elemen tanpa batasan ukuran tetap.

2. **Unguided 2**

```
#include <iostream>
#include <string>
using namespace std;
struct Node
{
    string name;
    string studentID;
    Node *next;
};
class Queue
{
private:
    Node *front;
    Node *rear;

public:
    Queue()
    {
        front = nullptr;
        rear = nullptr;
    }
    bool isFull()
    {
        return false;
```

```
27.     }
28.     bool isEmpty()
29.     {
30.         return front == nullptr;
31.     }
32.     void enqueue(string name, string studentID)
33.     {
34.         Node *newNode = new Node();
35.         newNode->name = name;
36.         newNode->studentID = studentID;
37.         newNode->next = nullptr;
38.         if (isEmpty())
39.         {
40.             front = rear = newNode;
41.         }
42.         else
43.         {
44.             rear->next = newNode;
45.             rear = newNode;
46.         }
47.     }
48.     void dequeue()
49.     {
50.         if (isEmpty())
51.         {
52.             cout << "Queue is empty" << endl;
53.         }
54.         else
55.         {
56.             Node *temp = front;
57.             front = front->next;
58.             delete temp;
59.             if (front == nullptr)
60.             {
61.                 rear = nullptr;
62.             }
63.         }
64.     }
65.     int count()
66.     {
67.         int count = 0;
```

```
68.         Node *current = front;
69.         while (current != nullptr)
70.         {
71.             count++;
72.             current = current->next;
73.         }
74.         return count;
75.     }
76.     void clear()
77.     {
78.         while (!isEmpty())
79.         {
80.             dequeue();
81.         }
82.     }
83.     void view() {
84.         cout << "Queue Student Data:" << endl;
85.         Node *current = front;
86.         int index = 1;
87.         while (current != nullptr) {
88.             cout << index << ". Name: " << current->name <<
    ", NIM: " << current->studentID << endl;
89.             current = current->next;
90.             index++;
91.         }
92.         if (index == 1) {
93.             cout << "(empty)" << endl;
94.         }
95.     }
96.};
97.
98.int main() {
99.     Queue q;
100.
101.          q.enqueue("Andi", "2023001");
102.          q.enqueue("Naya", "2023002");
103.
104.          q.view();
105.          cout << "Queue Size = " << q.count() << endl;
106.
107.          q.dequeue();
```

```
108.          q.view();
109.          cout << "Queue Size = " << q.count() << endl;
110.
111.          q.clear();
112.          q.view();
113.          cout << "Queue Size = " << q.count() << endl;
114.
115.          return 0;
116.      }
117.
```

**Screenshoot program**


**Deskripsi program**

Struktur Node menyimpan informasi berupa nama dan nomor induk mahasiswa (NIM), serta memiliki pointer ke node berikutnya. Kelas Queue menggunakan dua pointer, front dan rear, untuk menunjukkan elemen pertama dan terakhir dalam antrian.

Dalam fungsi main, program ini menguji fungsionalitas antrian dengan melakukan langkah-langkah berikut:

1. Menambah dua elemen ("Andi" dengan NIM "2023001" dan "Naya" dengan NIM "2023002").
2. Menampilkan isi antrian beserta ukurannya.
3. Menghapus satu elemen.
4. Menampilkan kembali isi antrian dan ukurannya.
5. Mengosongkan seluruh antrian.
6. Menampilkan isi antrian dan ukurannya setelah dikosongkan.

Program ini memperlihatkan bagaimana linked list digunakan untuk mengelola antrian dinamis, memungkinkan penambahan dan penghapusan elemen tanpa batasan ukuran tetap.

# BAB IV

# KESIMPULAN

Kesimpulannya, Queue (antrian) adalah salah satu jenis struktur data linier yang bekerja berdasarkan prinsip First In First Out (FIFO), di mana elemen yang pertama masuk adalah elemen yang pertama keluar. Data dalam antrian dapat berupa tipe integer, real, atau record, baik dalam bentuk sederhana maupun terstruktur. Penyisipan data dalam antrian dilakukan di satu ujung, sementara penghapusan dilakukan di ujung lainnya. Ujung untuk penyisipan disebut rear atau tail, dan ujung untuk penghapusan disebut front atau head.

Sebuah queue dalam program setidaknya harus memiliki tiga variabel: head untuk menandai bagian depan antrian, tail untuk menandai bagian belakang antrian, dan array data untuk menyimpan elemen-elemen yang dimasukkan ke dalam queue.

Beberapa operasi dasar pada queue meliputi:

- Prosedur `create` untuk membuat queue baru yang kosong.
- Fungsi `isEmpty` untuk memeriksa apakah queue kosong.
- Fungsi `isFull` untuk memeriksa apakah queue penuh.
- Prosedur `enqueue` untuk menambahkan data ke dalam queue.
- Prosedur `dequeue` untuk menghapus elemen dari posisi head pada queue.
- Fungsi `clear` untuk menghapus semua elemen dalam queue.
- Prosedur `tampil` untuk menampilkan elemen-elemen yang ada dalam queue.