

# **LAPORAN PRAKTIKUM**

## **MODUL 7 QUEUE**



**Disusun oleh:  
Bayu Kuncoro Adi  
NIM: 2311102031**

**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. Mahasiswa mampu menjelaskan definisi dan konsep dari double queue.
2. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue.
3. Mahasiswa mampu menerapkan operasi tampil data pada queue.

## BAB II

### DASAR TEORI

#### A. DASAR TEORI

##### 1. Pengertian 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 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.



#### FIRST IN FIRST OUT (FIFO)

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 atas 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.

## 2. Jenis- jenis operasi pada Queue:

Operasi pada Queue melibatkan serangkaian fungsi yang memberikan kita kemampuan untuk berinteraksi dengan struktur data ini. Mari kita jabarkan beberapa operasi dasar yang dapat kita lakukan pada Queue:

### a. Berdasarkan Implementasinya

Linear/Simple Queue: Elemen-elemen data disusun dalam barisan linear dan penambahan serta penghapusan elemen hanya terjadi pada dua ujung barisan. Contoh Linear Queue:

Enqueue (Tambahkan Elemen):

- Elemen pertama: 1
- Elemen kedua: 2
- Elemen ketiga: 3
- Elemen keempat: 4
- Elemen kelima: 5

Dequeue (Hapus Elemen):

- Elemen pertama: 1 (dihapus)
- Elemen yang tersisa: 2, 3, 4, 5

Enqueue (Tambahkan Elemen):

- Elemen pertama: 2 (sebelumnya)
- Elemen kedua: 6
- Elemen ketiga: 7
- Elemen keempat: 8
- Elemen kelima: 9

Dequeue (Hapus Elemen):

- Elemen pertama: 2 (dihapus)
- Elemen yang tersisa: 3, 4, 5, 6, 7, 8, 9

- Circular Queue: Mirip dengan jenis linear, tetapi ujung-ujung barisan terhubung satu sama lain, menciptakan struktur antrean yang berputar.  
Contoh Circular Queue:

Enqueue (Tambahkan Elemen):

- Elemen pertama: 1
- Elemen kedua: 2
- Elemen ketiga: 3
- Elemen keempat: 4
- Elemen kelima: 5

Dequeue (Hapus Elemen):

- Elemen pertama: 1 (dihapus)
- Elemen yang tersisa: 2, 3, 4, 5

Enqueue (Tambahkan Elemen):

- Elemen pertama: 2 (sebelumnya)
- Elemen kedua: 6
- Elemen ketiga: 7
- Elemen keempat: 8
- Elemen kelima: 9

Dequeue (Hapus Elemen):

- Elemen pertama: 2 (dihapus)
- Elemen yang tersisa: 3, 4, 5, 6, 7, 8, 9

Enqueue (Tambahkan Elemen):

- Elemen pertama: 3 (sebelumnya)
- Elemen kedua: 10
- Elemen ketiga: 11
- Elemen keempat: 12
- Elemen kelima: 13

Dequeue (Hapus Elemen):

- Elemen pertama: 3 (dihapus)
- Elemen yang tersisa: 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

b. Berdasarkan Penggunaan

- Priority Queue: Setiap elemen memiliki prioritas tertentu. Elemen dengan prioritas tertinggi akan diambil terlebih dahulu. Elemen dengan prioritas rendah akan dihapus setelah elemen dengan prioritas tinggi. Contoh Priority Queue:

Enqueue (Tambahkan Elemen):

- Elemen pertama: 1 (prioritas tinggi)
- Elemen kedua: 3 (prioritas sedang)
- Elemen ketiga: 2 (prioritas rendah)
- Elemen keempat: 4 (prioritas tinggi)
- Elemen kelima: 5 (prioritas rendah)

Dequeue (Hapus Elemen):

- Elemen pertama: 1 (dihapus prioritas tinggi)
- Elemen yang tersisa: 3, 2, 4, 5

- Double-ended Queue (Deque): Elemen dapat ditambahkan atau dihapus dari kedua ujung antrean Contoh Double Ended Queue (Deque):

Enqueue (Tambahkan Elemen):

- Elemen pertama: 1
- Elemen kedua: 2
- Elemen ketiga: 3
- Elemen keempat: 4
- Elemen kelima: 5

Dequeue (Hapus Elemen):

- Elemen pertama: 1 (dihapus urutan paling depan)
- Elemen yang tersisa: 2, 3, 4, 5

Dequeue (Hapus Elemen):

- Elemen pertama: 5 (dihapus urutan paling ujung belakang)
- Elemen yang tersisa: 2, 3, 4.

### **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

##### 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 { // Antriannya 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;
    }
}

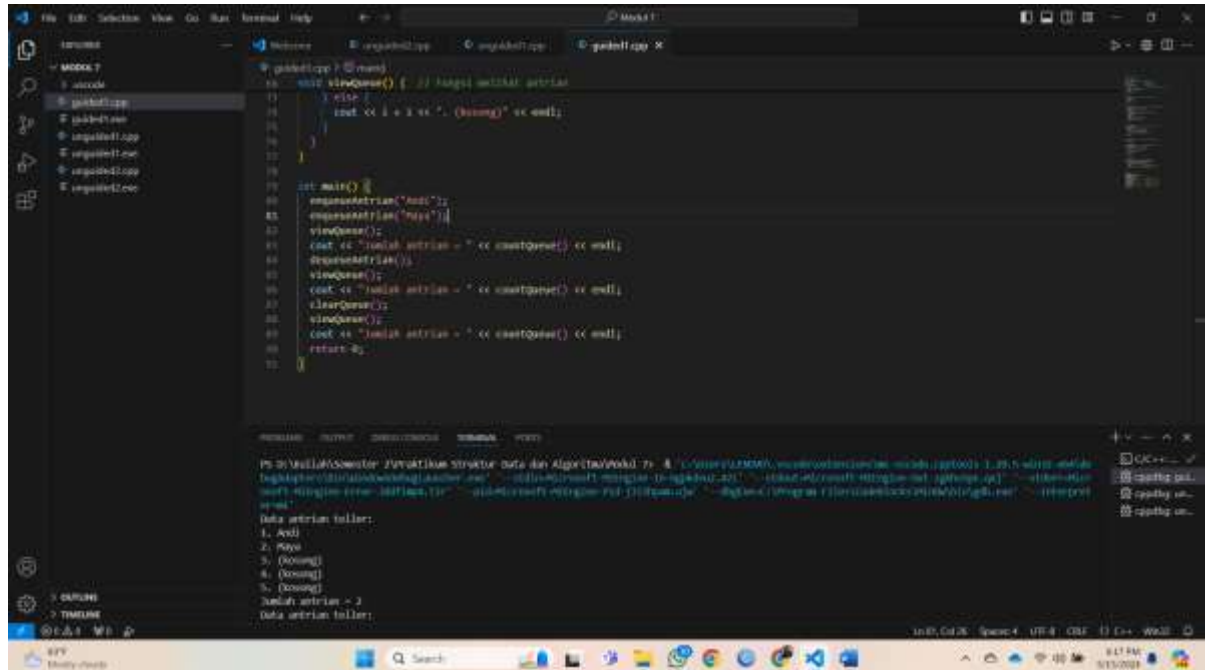
void viewQueue() { // Fungsi melihat antrian
    cout << "Data antrian teller:" << endl;
    for (int i = 0; i < maksimalQueue; i++) {
        if (queueTeller[i] != "") {
            cout << i + 1 << ". " << queueTeller[i] << endl;
        } else {
            cout << i + 1 << ". (kosong)" << endl;
        }
    }
}

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:



```
1. void viewQueue() { // fungsi untuk melihat antrian
2.     if (isEmpty())
3.         cout << "Antrian Kosong" << endl;
4.     else {
5.         for (int i = 0; i < countQueue(); i++)
6.             cout << "Antrian: " << antrian[i] << " ";
7.     }
8. }
9.
10. int main() {
11.     enqueueAntrian("Adi");
12.     enqueueAntrian("Budi");
13.     viewQueue();
14.     cout << "Jumlah antrian = " << countQueue() << endl;
15.     dequeueAntrian();
16.     viewQueue();
17.     cout << "Jumlah antrian = " << countQueue() << endl;
18.     clearQueue();
19.     viewQueue();
20.     cout << "Jumlah antrian = " << countQueue() << endl;
21.     return 0;
22. }
```

Output:

```
Data antrian taller:
1. Adi
2. Budi
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 3
Data antrian taller:
```

## Deskripsi Program:

Program ini adalah implementasi antrian menggunakan array statis di C++. Program ini menggunakan array `queueTeller` untuk menyimpan hingga lima elemen antrian dan memiliki fungsi untuk memeriksa apakah antrian penuh (`isFull`) atau kosong (`isEmpty`). Fungsi `enqueueAntrian` menambahkan elemen ke antrian, dan `dequeueAntrian` menghapus elemen dari antrian dengan menggeser elemen yang tersisa ke depan. Fungsi `countQueue` mengembalikan jumlah elemen dalam antrian, `clearQueue` mengosongkan seluruh antrian, dan `viewQueue` menampilkan isi antrian beserta status tiap posisi (kosong atau terisi). Program ini mengilustrasikan penggunaannya dengan menambahkan beberapa nama ke antrian, menampilkan antrian, menghitung elemen dalam antrian, menghapus satu elemen, dan kemudian mengosongkan antrian sepenuhnya.

## LATIHAN KELAS-UNGUIDED

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

**Program:**

```
#include <iostream>

using namespace std;

const int maksimalQueue = 5; // Maksimal antrian adalah 5

// Node untuk menyimpan data dan pointer ke node berikutnya
struct Node {
    string data;
    Node* next;
};

class Queue {
private:
    Node* front; // Node depan dari antrian
    Node* rear;  // Node belakang dari antrian

public:
    Queue() { // Konstruktor untuk menginisialisasi antrian kosong
        front = nullptr;
        rear = nullptr;
    }

    // Fungsi untuk menambahkan data ke antrian
    void enqueue_2031(const string& data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->next = nullptr;

        // Jika antrian kosong
        if (isEmpty_2031()) { // Jika antrian kosong maka front dan rear menunjuk ke newNode yang baru dibuat
            front = rear = newNode;
        } else { // Jika antrian tidak kosong maka rear menunjuk ke newNode yang baru dibuat
            rear->next = newNode;
            rear = newNode;
        }

        cout << data << " ditambahkan ke dalam antrian." << endl;
    }

    // Fungsi untuk menghapus data dari antrian
```

```

void dequeue_2031() {
    if (isEmpty_2031()) { // Jika antrian kosong maka tampilkan
pesan "Antrian kosong" dan kembalikan nilai void
        cout << "Antrian kosong." << endl;
        return;
    }

    Node* temp = front; // Simpan node front ke dalam variabel temp
untuk dihapus nantinya
    front = front->next; // Geser front ke node selanjutnya

    cout << temp->data << " dihapus dari antrian." << endl; //
Tampilkan data yang dihapus dari antrian
    delete temp; // Hapus node yang disimpan di variabel temp

    // Jika setelah penghapusan antrian menjadi kosong
    if (front == nullptr) {
        rear = nullptr;
    }
}

// Fungsi untuk menampilkan seluruh antrian
void displayQueue_2031() {
    if (isEmpty_2031()) { // Jika antrian kosong maka tampilkan
pesan "Data antrian:" dan tampilkan pesan "(kosong)"
        cout << "2031-Data antrian-2031:" << endl;
        for (int i = 0; i < maksimalQueue; i++) {
            cout << i + 1 << ". (kosong)" << endl;
        }
    } else { // Jika antrian tidak kosong maka tampilkan data
antrian yang ada
        cout << "2031-Data antrian-2031:" << endl;
        Node* current = front;
        int i = 1;
        while (current != nullptr) { // Selama current tidak
menunjuk ke nullptr maka tampilkan data antrian yang ada
            cout << i << ". " << current->data << endl;
            current = current->next;
            i++;
        }
        for (; i <= maksimalQueue; i++) { // Tampilkan pesan
"(kosong)" untuk antrian yang kosong
            cout << i << ". (kosong)" << endl;
        }
    }
}

// Fungsi untuk memeriksa apakah antrian kosong

```

```

bool isEmpty_2031() {
    return front == nullptr;
}

// Fungsi untuk mengembalikan jumlah elemen dalam antrian
int countQueue_2031() {
    int count = 0;
    Node* current = front;
    while (current != nullptr) { // Selama current tidak menunjuk
ke nullptr maka hitung jumlah elemen dalam antrian
        count++;
        current = current->next;
    }
    return count; // Kembalikan jumlah elemen dalam antrian
}

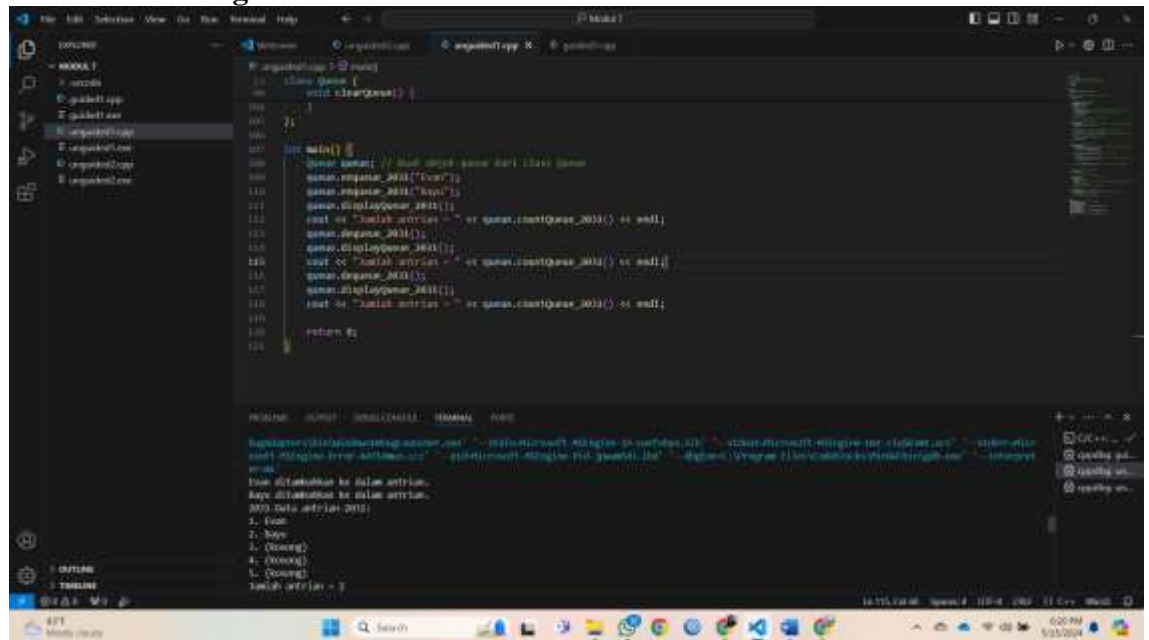
// Fungsi untuk menghapus semua elemen dalam antrian
void clearQueue() {
    while (!isEmpty_2031()) { // Selama antrian tidak kosong maka
hapus elemen dalam antrian
        dequeue_2031(); // Hapus elemen dalam antrian
    }
    cout << "Antrian telah dibersihkan." << endl;
}
};

int main() {
    Queue queue; // Buat objek queue dari class Queue
    queue.enqueue_2031("Evan");
    queue.enqueue_2031("Bayu");
    queue.displayQueue_2031();
    cout << "Jumlah antrian = " << queue.countQueue_2031() << endl;
    queue.dequeue_2031();
    queue.displayQueue_2031();
    cout << "Jumlah antrian = " << queue.countQueue_2031() << endl;
    queue.dequeue_2031();
    queue.displayQueue_2031();
    cout << "Jumlah antrian = " << queue.countQueue_2031() << endl;

    return 0;
}

```

## Screenshoot Program:



```
1 // Queue using linked list
2 #include <iostream>
3 using namespace std;
4
5 struct Node {
6     int data;
7     Node* next;
8 };
9
10 Node* head = NULL;
11
12 void enqueue_2031(int data) {
13     Node* newNode = new Node;
14     newNode->data = data;
15     newNode->next = NULL;
16     if (head == NULL) {
17         head = newNode;
18     } else {
19         Node* temp = head;
20         while (temp->next != NULL) {
21             temp = temp->next;
22         }
23         temp->next = newNode;
24     }
25 }
26
27 void dequeue_2031() {
28     if (head == NULL) {
29         cout << "Queue is empty" << endl;
30     } else {
31         Node* temp = head;
32         head = temp->next;
33         delete temp;
34     }
35 }
36
37 void displayQueue_2031() {
38     if (head == NULL) {
39         cout << "Queue is empty" << endl;
40     } else {
41         Node* temp = head;
42         while (temp != NULL) {
43             cout << temp->data << " ";
44             temp = temp->next;
45         }
46         cout << endl;
47     }
48 }
49
50 int countQueue_2031() {
51     if (head == NULL) {
52         return 0;
53     } else {
54         Node* temp = head;
55         int count = 0;
56         while (temp != NULL) {
57             count++;
58             temp = temp->next;
59         }
60         return count;
61     }
62 }
63
64 void clearQueue_2031() {
65     while (head != NULL) {
66         Node* temp = head;
67         head = temp->next;
68         delete temp;
69     }
70 }
71
72 int main() {
73     enqueue_2031(10);
74     enqueue_2031(20);
75     enqueue_2031(30);
76     displayQueue_2031();
77     cout << "Count of queue is: " << countQueue_2031() << endl;
78     dequeue_2031();
79     displayQueue_2031();
80     clearQueue_2031();
81     displayQueue_2031();
82     return 0;
83 }
```

Output:

```
10 20 30
Count of queue is: 3
10 20
Queue is empty
```

## Deskripsi Program:

Program ini adalah implementasi antrian menggunakan struktur data linked list di C++. Program ini mendefinisikan kelas `Queue` dengan node sebagai elemen antriannya, di mana setiap node menyimpan data dan pointer ke node berikutnya. Fungsi `enqueue\_2031` menambahkan data ke antrian dengan menempatkan node baru di bagian belakang. Fungsi `dequeue\_2031` menghapus data dari antrian dengan menghapus node dari bagian depan. Fungsi `displayQueue\_2031` menampilkan seluruh elemen antrian, atau menunjukkan bahwa antrian kosong jika tidak ada elemen. Fungsi `isEmpty\_2031` memeriksa apakah antrian kosong, `countQueue\_2031` menghitung jumlah elemen dalam antrian, dan `clearQueue` menghapus semua elemen dalam antrian. Dalam fungsi `main`, beberapa elemen dimasukkan ke dalam antrian, ditampilkan, dihitung, dihapus, dan proses tersebut diulang untuk menunjukkan penggunaan berbagai fungsi antrian.

2. Dari nomor 1 buatlah konsep antri dengan atribut Nama mahasiswa dan NIM Mahasiswa

**Program:**

```
#include <iostream>
#include <string>

using namespace std;

const int maksimalQueue = 5; // Maksimal antrian adalah 5

// Node untuk menyimpan data dan pointer ke node berikutnya
struct Node {
    string nama;
    string nim;
    Node* next;
};

class Queue {
private:
    Node* front; // Node depan dari antrian
    Node* rear;  // Node belakang dari antrian

public:
    Queue() { // Konstruktor untuk menginisialisasi antrian kosong
        front = nullptr;
        rear = nullptr;
    }

    // Fungsi untuk menambahkan data ke antrian
    void enqueue_2031(const string& nama, const string& nim) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

        // Jika antrian kosong
        if (isEmpty_2031()) { // Jika antrian kosong maka front dan
            rear menunjuk ke newNode yang baru dibuat
            front = rear = newNode;
        } else { // Jika antrian tidak kosong maka rear menunjuk ke
            newNode yang baru dibuat
            rear->next = newNode;
            rear = newNode;
        }

        cout << "Mahasiswa dengan Nama: " << newNode->nama << " dan
        NIM: " << newNode->nim << " ditambahkan ke dalam antrian." << endl;
    }
}
```

```

// Fungsi untuk menghapus data dari antrian
void dequeue_2031() {
    if (isEmpty_2031()) { // Jika antrian kosong maka tampilkan
pesan "Antrian kosong" dan kembalikan nilai void
        cout << "Antrian kosong." << endl;
        return;
    }

    Node* temp = front; // Simpan node front ke dalam variabel temp
untuk dihapus nantinya
    front = front->next; // Geser front ke node selanjutnya

    cout << "Mahasiswa dengan Nama: " << temp->nama << " dan NIM: "
<< temp->nim << " dihapus dari antrian." << endl; // Tampilkan data
mahasiswa yang dihapus dari antrian
    delete temp; // Hapus node yang disimpan di variabel temp

    // Jika setelah penghapusan antrian menjadi kosong
    if (front == nullptr) {
        rear = nullptr;
    }
}

// Fungsi untuk menampilkan seluruh antrian
void displayQueue_2031() {
    if (isEmpty_2031()) { // Jika antrian kosong maka tampilkan
pesan "Data antrian:" dan tampilkan pesan "(kosong)"
        cout << "Data antrian:" << endl;
        for (int i = 0; i < maksimalQueue; i++) {
            cout << i + 1 << ". (kosong)" << endl;
        }
    } else { // Jika antrian tidak kosong maka tampilkan data
antrian yang ada
        cout << "Data antrian:" << endl;
        Node* current = front;
        int i = 1;
        while (current != nullptr) { // Selama current tidak
menunjuk ke nullptr maka tampilkan data antrian yang ada
            cout << i << ". " << "Nama: " << current->nama << ",
NIM: " << current->nim << endl;
            current = current->next;
            i++;
        }
        for (; i <= maksimalQueue; i++) { // Tampilkan pesan
"(kosong)" untuk antrian yang kosong
            cout << i << ". (kosong)" << endl;
        }
    }
}

```



```

    }
}

// Fungsi untuk memeriksa apakah antrian kosong
bool isEmpty_2031() {
    return front == nullptr;
}

// Fungsi untuk mengembalikan jumlah elemen dalam antrian
int countQueue_2031() {
    int count = 0;
    Node* current = front;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count; // Kembalikan jumlah elemen dalam antrian
}

// Fungsi untuk menghapus semua elemen dalam antrian
void clearQueue() {
    while (!isEmpty_2031()) { // Selama antrian tidak kosong maka
hapus elemen dalam antrian
        dequeue_2031(); // Hapus elemen dalam antrian
    }
    cout << "Antrian telah dibersihkan." << endl;
}

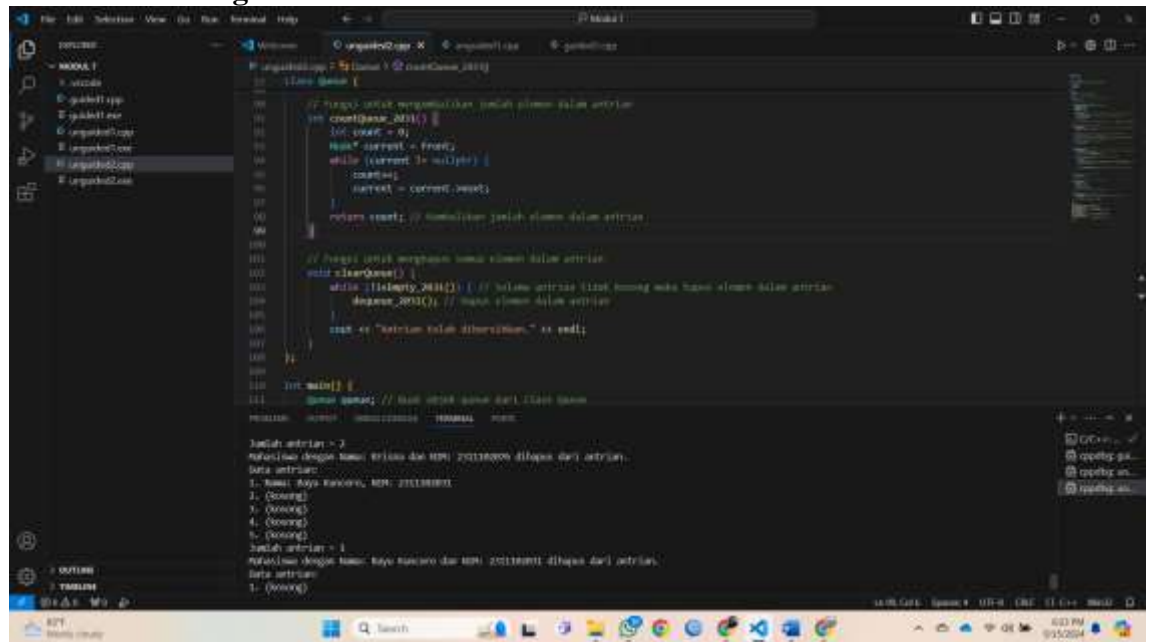
};

int main() {
    Queue queue; // Buat objek queue dari class Queue
    queue.enqueue_2031("Krisna", "2311102076");
    queue.enqueue_2031("Bayu Kuncoro", "2311102031");
    queue.displayQueue_2031();
    cout << "Jumlah antrian = " << queue.countQueue_2031() << endl;
    queue.dequeue_2031();
    queue.displayQueue_2031();
    cout << "Jumlah antrian = " << queue.countQueue_2031() << endl;
    queue.dequeue_2031();
    queue.displayQueue_2031();
    cout << "Jumlah antrian = " << queue.countQueue_2031() << endl;

    return 0;
}

```

## Screenshoot Program:



```
1 // Program untuk mengimplementasikan antrian dengan menggunakan struktur data linked list di C++
2 // Nama: Raka Kurniawan, NIM: 25118001
3 // Kelas: Informatika
4 // Tanggal: 10/10/2024
5 // Waktu: 14.00
6 // Lokasi: Kampus
7 // Dosen:
8 // Asisten:
9 // Revisi:
10 //
11 //
12 //
13 //
14 //
15 //
16 //
17 //
18 //
19 //
20 //
21 //
22 //
23 //
24 //
25 //
26 //
27 //
28 //
29 //
30 //
31 //
32 //
33 //
34 //
35 //
36 //
37 //
38 //
39 //
40 //
41 //
42 //
43 //
44 //
45 //
46 //
47 //
48 //
49 //
50 //
51 //
52 //
53 //
54 //
55 //
56 //
57 //
58 //
59 //
60 //
61 //
62 //
63 //
64 //
65 //
66 //
67 //
68 //
69 //
70 //
71 //
72 //
73 //
74 //
75 //
76 //
77 //
78 //
79 //
80 //
81 //
82 //
83 //
84 //
85 //
86 //
87 //
88 //
89 //
90 //
91 //
92 //
93 //
94 //
95 //
96 //
97 //
98 //
99 //
100 //
101 //
102 //
103 //
104 //
105 //
106 //
107 //
108 //
109 //
110 //
111 //
112 //
113 //
114 //
115 //
116 //
117 //
118 //
119 //
120 //
121 //
122 //
123 //
124 //
125 //
126 //
127 //
128 //
129 //
130 //
131 //
132 //
133 //
134 //
135 //
136 //
137 //
138 //
139 //
140 //
141 //
142 //
143 //
144 //
145 //
146 //
147 //
148 //
149 //
150 //
151 //
152 //
153 //
154 //
155 //
156 //
157 //
158 //
159 //
160 //
161 //
162 //
163 //
164 //
165 //
166 //
167 //
168 //
169 //
170 //
171 //
172 //
173 //
174 //
175 //
176 //
177 //
178 //
179 //
180 //
181 //
182 //
183 //
184 //
185 //
186 //
187 //
188 //
189 //
190 //
191 //
192 //
193 //
194 //
195 //
196 //
197 //
198 //
199 //
200 //
201 //
202 //
203 //
204 //
205 //
206 //
207 //
208 //
209 //
210 //
211 //
212 //
213 //
214 //
215 //
216 //
217 //
218 //
219 //
220 //
221 //
222 //
223 //
224 //
225 //
226 //
227 //
228 //
229 //
230 //
231 //
232 //
233 //
234 //
235 //
236 //
237 //
238 //
239 //
240 //
241 //
242 //
243 //
244 //
245 //
246 //
247 //
248 //
249 //
250 //
251 //
252 //
253 //
254 //
255 //
256 //
257 //
258 //
259 //
260 //
261 //
262 //
263 //
264 //
265 //
266 //
267 //
268 //
269 //
270 //
271 //
272 //
273 //
274 //
275 //
276 //
277 //
278 //
279 //
280 //
281 //
282 //
283 //
284 //
285 //
286 //
287 //
288 //
289 //
290 //
291 //
292 //
293 //
294 //
295 //
296 //
297 //
298 //
299 //
300 //
301 //
302 //
303 //
304 //
305 //
306 //
307 //
308 //
309 //
310 //
311 //
312 //
313 //
314 //
315 //
316 //
317 //
318 //
319 //
320 //
321 //
322 //
323 //
324 //
325 //
326 //
327 //
328 //
329 //
330 //
331 //
332 //
333 //
334 //
335 //
336 //
337 //
338 //
339 //
340 //
341 //
342 //
343 //
344 //
345 //
346 //
347 //
348 //
349 //
350 //
351 //
352 //
353 //
354 //
355 //
356 //
357 //
358 //
359 //
360 //
361 //
362 //
363 //
364 //
365 //
366 //
367 //
368 //
369 //
370 //
371 //
372 //
373 //
374 //
375 //
376 //
377 //
378 //
379 //
380 //
381 //
382 //
383 //
384 //
385 //
386 //
387 //
388 //
389 //
390 //
391 //
392 //
393 //
394 //
395 //
396 //
397 //
398 //
399 //
400 //
401 //
402 //
403 //
404 //
405 //
406 //
407 //
408 //
409 //
410 //
411 //
412 //
413 //
414 //
415 //
416 //
417 //
418 //
419 //
420 //
421 //
422 //
423 //
424 //
425 //
426 //
427 //
428 //
429 //
430 //
431 //
432 //
433 //
434 //
435 //
436 //
437 //
438 //
439 //
440 //
441 //
442 //
443 //
444 //
445 //
446 //
447 //
448 //
449 //
450 //
451 //
452 //
453 //
454 //
455 //
456 //
457 //
458 //
459 //
460 //
461 //
462 //
463 //
464 //
465 //
466 //
467 //
468 //
469 //
470 //
471 //
472 //
473 //
474 //
475 //
476 //
477 //
478 //
479 //
480 //
481 //
482 //
483 //
484 //
485 //
486 //
487 //
488 //
489 //
490 //
491 //
492 //
493 //
494 //
495 //
496 //
497 //
498 //
499 //
500 //
501 //
502 //
503 //
504 //
505 //
506 //
507 //
508 //
509 //
510 //
511 //
512 //
513 //
514 //
515 //
516 //
517 //
518 //
519 //
520 //
521 //
522 //
523 //
524 //
525 //
526 //
527 //
528 //
529 //
530 //
531 //
532 //
533 //
534 //
535 //
536 //
537 //
538 //
539 //
540 //
541 //
542 //
543 //
544 //
545 //
546 //
547 //
548 //
549 //
550 //
551 //
552 //
553 //
554 //
555 //
556 //
557 //
558 //
559 //
560 //
561 //
562 //
563 //
564 //
565 //
566 //
567 //
568 //
569 //
570 //
571 //
572 //
573 //
574 //
575 //
576 //
577 //
578 //
579 //
580 //
581 //
582 //
583 //
584 //
585 //
586 //
587 //
588 //
589 //
590 //
591 //
592 //
593 //
594 //
595 //
596 //
597 //
598 //
599 //
600 //
601 //
602 //
603 //
604 //
605 //
606 //
607 //
608 //
609 //
610 //
611 //
612 //
613 //
614 //
615 //
616 //
617 //
618 //
619 //
620 //
621 //
622 //
623 //
624 //
625 //
626 //
627 //
628 //
629 //
630 //
631 //
632 //
633 //
634 //
635 //
636 //
637 //
638 //
639 //
640 //
641 //
642 //
643 //
644 //
645 //
646 //
647 //
648 //
649 //
650 //
651 //
652 //
653 //
654 //
655 //
656 //
657 //
658 //
659 //
660 //
661 //
662 //
663 //
664 //
665 //
666 //
667 //
668 //
669 //
670 //
671 //
672 //
673 //
674 //
675 //
676 //
677 //
678 //
679 //
680 //
681 //
682 //
683 //
684 //
685 //
686 //
687 //
688 //
689 //
690 //
691 //
692 //
693 //
694 //
695 //
696 //
697 //
698 //
699 //
700 //
701 //
702 //
703 //
704 //
705 //
706 //
707 //
708 //
709 //
710 //
711 //
712 //
713 //
714 //
715 //
716 //
717 //
718 //
719 //
720 //
721 //
722 //
723 //
724 //
725 //
726 //
727 //
728 //
729 //
730 //
731 //
732 //
733 //
734 //
735 //
736 //
737 //
738 //
739 //
740 //
741 //
742 //
743 //
744 //
745 //
746 //
747 //
748 //
749 //
750 //
751 //
752 //
753 //
754 //
755 //
756 //
757 //
758 //
759 //
760 //
761 //
762 //
763 //
764 //
765 //
766 //
767 //
768 //
769 //
770 //
771 //
772 //
773 //
774 //
775 //
776 //
777 //
778 //
779 //
780 //
781 //
782 //
783 //
784 //
785 //
786 //
787 //
788 //
789 //
790 //
791 //
792 //
793 //
794 //
795 //
796 //
797 //
798 //
799 //
800 //
801 //
802 //
803 //
804 //
805 //
806 //
807 //
808 //
809 //
810 //
811 //
812 //
813 //
814 //
815 //
816 //
817 //
818 //
819 //
820 //
821 //
822 //
823 //
824 //
825 //
826 //
827 //
828 //
829 //
830 //
831 //
832 //
833 //
834 //
835 //
836 //
837 //
838 //
839 //
840 //
841 //
842 //
843 //
844 //
845 //
846 //
847 //
848 //
849 //
850 //
851 //
852 //
853 //
854 //
855 //
856 //
857 //
858 //
859 //
860 //
861 //
862 //
863 //
864 //
865 //
866 //
867 //
868 //
869 //
870 //
871 //
872 //
873 //
874 //
875 //
876 //
877 //
878 //
879 //
880 //
881 //
882 //
883 //
884 //
885 //
886 //
887 //
888 //
889 //
890 //
891 //
892 //
893 //
894 //
895 //
896 //
897 //
898 //
899 //
900 //
901 //
902 //
903 //
904 //
905 //
906 //
907 //
908 //
909 //
910 //
911 //
912 //
913 //
914 //
915 //
916 //
917 //
918 //
919 //
920 //
921 //
922 //
923 //
924 //
925 //
926 //
927 //
928 //
929 //
930 //
931 //
932 //
933 //
934 //
935 //
936 //
937 //
938 //
939 //
940 //
941 //
942 //
943 //
944 //
945 //
946 //
947 //
948 //
949 //
950 //
951 //
952 //
953 //
954 //
955 //
956 //
957 //
958 //
959 //
960 //
961 //
962 //
963 //
964 //
965 //
966 //
967 //
968 //
969 //
970 //
971 //
972 //
973 //
974 //
975 //
976 //
977 //
978 //
979 //
980 //
981 //
982 //
983 //
984 //
985 //
986 //
987 //
988 //
989 //
990 //
991 //
992 //
993 //
994 //
995 //
996 //
997 //
998 //
999 //
1000 //
```

## Deskripsi Program:

Program ini adalah implementasi antrian menggunakan struktur data linked list di C++, yang menyimpan informasi mahasiswa berupa nama dan NIM. Program mendefinisikan kelas `Queue` dengan node sebagai elemen antriannya, di mana setiap node menyimpan data `nama` dan `nim` serta pointer ke node berikutnya. Fungsi `enqueue\_2031` menambahkan data mahasiswa ke antrian dengan menempatkan node baru di bagian belakang. Fungsi `dequeue\_2031` menghapus data mahasiswa dari antrian dengan menghapus node dari bagian depan dan menampilkan informasi mahasiswa yang dihapus. Fungsi `displayQueue\_2031` menampilkan seluruh elemen antrian beserta nama dan NIM mahasiswa atau menunjukkan bahwa antrian kosong jika tidak ada elemen. Fungsi `isEmpty\_2031` memeriksa apakah antrian kosong, `countQueue\_2031` menghitung jumlah elemen dalam antrian, dan `clearQueue` menghapus semua elemen dalam antrian. Dalam fungsi `main`, program mendemonstrasikan penggunaan berbagai fungsi antrian dengan menambahkan beberapa mahasiswa ke dalam antrian, menampilkan antrian, menghitung jumlah elemen, menghapus elemen, dan menampilkan antrian setelah setiap operasi.

## **BAB IV**

### **KESIMPULAN**

Queue merupakan struktur data yang mengimplementasikan prinsip FIFO (First-In First-Out). Elemen yang pertama dimasukkan akan menjadi yang pertama keluar. Implementasi queue dapat dilakukan menggunakan array atau linked list, di mana setiap elemen memiliki dua penunjuk: front yang menunjuk ke elemen pertama dan rear yang menunjuk ke elemen terakhir. Perbedaan utama antara queue dan stack adalah pada cara penambahan dan penghapusan elemen; stack menggunakan LIFO (Last-In First-Out) sementara queue menggunakan FIFO.

Ada beberapa jenis queue, seperti linear queue, circular queue, priority queue, dan double-ended queue. Masing-masing jenis memiliki cara kerja dan penggunaan yang berbeda. Operasi dasar pada queue meliputi enqueue (menambahkan elemen), dequeue (menghapus elemen), peek (mengambil elemen tanpa menghapusnya), isEmpty (memeriksa apakah queue kosong), isFull (memeriksa apakah queue penuh), dan size (menghitung jumlah elemen dalam queue). Contoh implementasi queue dengan array menunjukkan bagaimana elemen ditambahkan dan dihapus dengan menggeser posisi elemen.

Implementasi queue menggunakan linked list lebih dinamis karena tidak dibatasi oleh ukuran tetap seperti pada array. Pada implementasi ini, setiap node menyimpan data dan pointer ke node berikutnya. Fungsi enqueue menambahkan elemen baru di bagian belakang, sedangkan dequeue menghapus elemen dari bagian depan. Fungsi tambahan seperti displayQueue menampilkan seluruh elemen dalam queue, isEmpty memeriksa apakah queue kosong, countQueue menghitung jumlah elemen, dan clearQueue menghapus semua elemen dalam queue. Implementasi ini lebih fleksibel dan efisien dalam memori dibandingkan dengan array.

Contoh penerapan konsep queue pada mahasiswa dengan atribut Nama dan NIM menunjukkan penggunaan praktis dari queue dalam situasi nyata. Dengan menggunakan linked list, setiap node menyimpan nama dan NIM mahasiswa. Fungsi enqueue menambahkan mahasiswa ke dalam antrian, dan dequeue menghapus mahasiswa dari antrian sambil menampilkan informasi yang dihapus. Fungsi displayQueue menampilkan seluruh mahasiswa dalam antrian, membantu dalam pengelolaan data mahasiswa secara efektif. Implementasi ini menunjukkan bagaimana struktur data queue dapat diadaptasi untuk berbagai kebutuhan dalam pemrograman dan pengelolaan data.

## DAFTAR PUSTAKA

Karumanchi, N. (2016). *Data Structures and algorithms made easy: Concepts, problems, Interview Questions*. CareerMonk Publications

Modul 5 Queue diakses dari <https://elektro.um.ac.id/wp-content/uploads/2016/04/ASD-Modul-5-Queue.pdf>

Modul 7 Queue Praktikum Struktur Data dan Algoritma Pemrograman

Struktur Data diakses dari <https://github.com/yunusfebriansyah/struktur-data/tree/main/Circular%20Single%20Linked%20List>







