

# **LAPORAN PRAKTIKUM**

## **MODUL 8**

### **QUEUE**



**Disusun Oleh :**

Farhan Kurniawan (2311104073)

**Kelas:**

SE-07-2

**Dosen :**

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

**PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY PURWOKERTO**

**2024**

## I. TUJUAN

1. Mempelajari prinsip kerja *Queue* berdasarkan konsep FIFO (*First In, First Out*).
2. Mengembangkan pemahaman tentang perbedaan antara implementasi *Queue* menggunakan array dan *linked list*.
3. Melatih kemampuan untuk melakukan operasi *insert* (penambahan elemen) dan *delete* (penghapusan elemen) pada *Queue* berbasis *linked list*.
4. Menjelaskan cara kerja *Queue* dalam konteks struktur data *linked list*, baik *single linked list* maupun *double linked list*.
5. Mengaplikasikan teori *Queue* pada pemrograman nyata menggunakan bahasa pemrograman C.

## II. LANDASAN TEORI

Queue (dibaca: kyu) merupakan struktur data yang dapat diumpamakan seperti sebuah antrian. Misalnya, antrian pada loket pembelian tiket kereta api. Orang yang pertama kali masuk dalam antrian akan mendapatkan pelayanan terlebih dahulu, sedangkan orang yang terakhir masuk akan mendapatkan layanan paling akhir. Dengan kata lain, prinsip dasar dari Queue adalah FIFO (*First In, First Out*), yaitu proses yang pertama masuk akan diakses terlebih dahulu.

Dalam pengimplementasian struktur Queue pada bahasa pemrograman C, dapat digunakan tipe data array maupun *linked list*. Namun, dalam praktikum ini, hanya akan dibahas pengimplementasian Queue dalam bentuk *linked list*.

Implementasi Queue menggunakan *linked list* sebenarnya tidak jauh berbeda dengan operasi pada list biasa, bahkan cenderung lebih sederhana. Hal ini sesuai dengan sifat FIFO, di mana proses *delete* hanya dilakukan pada bagian *Head* (depan list) dan proses *insert* selalu dilakukan pada bagian *Tail* (belakang list), atau sebaliknya, tergantung pada persepsi yang digunakan. Dalam penerapannya, Queue dapat diimplementasikan menggunakan *single linked list* maupun *double linked list*.

## III. GUIDED

1. Guided 1

```

1  #include <iostream>
2  #define MAX 100
3
4  using namespace std;
5
6  class Queue {
7  private:
8      int front, rear;
9      int arr[MAX];
10 public:
11
12     Queue() {
13         front = -1;
14         rear = -1;
15     }
16
17     bool isFull() {
18         return rear == MAX - 1;
19     }
20
21     bool isEmpty() {
22         return front == -1 || front > rear;
23     }
24
25     void enqueue(int x) {
26         if (isFull()) {
27             cout << "Queue Overflow\n";
28             return;
29         }
30         if (front == -1) front = 0;
31         arr[++rear] = x;
32     }
33
34     void dequeue() {
35         if (isEmpty()) {
36             cout << "Queue Underflow\n";
37             return;
38         }
39         front++;
40     }
41
42     int peek() {
43         if (!isEmpty()) {
44             return arr[front];
45         }
46         cout << "Queue is empty\n";
47         return -1;
48     }
49
50     void display() {
51         if (isEmpty()) {
52             cout << "Queue is empty\n";
53             return;
54         }
55         for (int i = front; i <= rear; i++) {
56             cout << arr[i] << " ";
57         }
58         cout << "\n";
59     }
60 };
61
62 int main() {
63     Queue q;
64
65     q.enqueue(10);
66     q.enqueue(20);
67     q.enqueue(30);
68
69     cout << "Queue elements: ";
70     q.display();
71
72     cout << "Front element: " << q.peek() << "\n";
73
74     q.dequeue();
75     cout << "After dequeuing, queue elements: ";
76     q.display();
77
78     return 0;
79 }
80
81
82

```

Hasil run:

```
PS D:\Praktikum\C++\Modul8> cd "d:\Praktikum\C++\Modul8\" ;  
Queue elements: 10 20 30  
Front element: 10  
After dequeuing, queue elements: 20 30  
PS D:\Praktikum\C++\Modul8>
```

## 2. Guided 2

```

1  #include <iostream>
2  #define MAX 100
3
4  using namespace std;
5
6  class Queue {
7  private:
8      int front, rear;
9      int arr[MAX];
10 public:
11
12     Queue() {
13         front = -1;
14         rear = -1;
15     }
16
17     bool isFull() {
18         return rear == MAX - 1;
19     }
20
21     bool isEmpty() {
22         return front == -1 || front > rear;
23     }
24
25     void enqueue(int x) {
26         if (isFull()) {
27             cout << "Queue Overflow\n";
28             return;
29         }
30         if (front == -1) front = 0;
31         arr[++rear] = x;
32     }
33
34     void dequeue() {
35         if (isEmpty()) {
36             cout << "Queue Underflow\n";
37             return;
38         }
39         front++;
40     }
41
42     int peek() {
43         if (!isEmpty()) {
44             return arr[front];
45         }
46         cout << "Queue is empty\n";
47         return -1;
48     }
49
50     void display() {
51         if (isEmpty()) {
52             cout << "Queue is empty\n";
53             return;
54         }
55         for (int i = front; i <= rear; i++) {
56             cout << arr[i] << " ";
57         }
58         cout << "\n";
59     }
60 };
61
62 int main() {
63     Queue q;
64
65     q.enqueue(10);
66     q.enqueue(20);
67     q.enqueue(30);
68
69     cout << "Queue elements: ";
70     q.display();
71
72     cout << "Front element: " << q.peek() << "\n";
73
74     q.dequeue();
75     cout << "After dequeuing, queue elements: ";
76     q.display();
77
78     return 0;
79 }
80
81
82

```

Hasil Run program:

```
PS D:\Praktikum\C++\Modul8> cd "d:\Praktikum\C++\Modul8\" ;  
Queue elements: 10 20 30  
Front element: 10  
After dequeuing, queue elements: 20 30
```

### 3. Guided 3

```

1  #include<iostream>
2
3  using namespace std;
4
5  const int maksimalQueue = 5; // Maksimal antrian
6  int front = 0; // Penanda antrian
7  int back = 0; // Penanda
8  string queueTeller[5]; // Fungsi pengecekan
9
10 bool isFull() { // Pengecekan antrian penuh atau tidak
11     if (back == maksimalQueue) {
12         return true; // =1
13     } else {
14         return false;
15     }
16 }
17
18 bool isEmpty() { // Antriannya kosong atau tidak
19     if (back == 0) {
20         return true;
21     } else {
22         return false;
23     }
24 }
25
26 void enqueueAntrian(string data) { // Fungsi menambahkan antrian
27     if (isFull()) {
28         cout << "Antrian penuh" << endl;
29     } else {
30         if (isEmpty()) { // Kondisi ketika queue kosong
31             queueTeller[0] = data;
32             front++;
33             back++;
34         } else { // Antriannya ada isi
35             queueTeller[back] = data;
36             back++;
37         }
38     }
39 }
40
41 void dequeueAntrian() { // Fungsi mengurangi antrian
42     if (isEmpty()) {
43         cout << "Antrian kosong" << endl;
44     } else {
45         for (int i = 0; i < back; i++) {
46             queueTeller[i] = queueTeller[i + 1];
47         }
48         back--;
49     }
50 }
51
52 int countQueue() { // Fungsi menghitung banyak antrian
53     return back;
54 }
55
56 void clearQueue() { // Fungsi menghapus semua antrian
57     if (isEmpty()) {
58         cout << "Antrian kosong" << endl;
59     } else {
60         for (int i = 0; i < back; i++) {
61             queueTeller[i] = "";
62         }
63         back = 0;
64         front = 0;
65     }
66 }
67
68 void viewQueue() { // Fungsi melihat antrian
69     cout << "Data antrian teller:" << endl;
70     for (int i = 0; i < maksimalQueue; i++) {
71         if (queueTeller[i] != "") {
72             cout << i + 1 << ". " << queueTeller[i] << endl;
73         } else {
74             cout << i + 1 << ". (kosong)" << endl;
75         }
76     }
77 }
78
79 int main() {
80     enqueueAntrian("Andi");
81     enqueueAntrian("Maya");
82
83     viewQueue();
84     cout << "Jumlah antrian = " << countQueue() << endl;
85
86     dequeueAntrian();
87     viewQueue();
88     cout << "Jumlah antrian = " << countQueue() << endl;
89
90     clearQueue();
91     viewQueue();
92     cout << "Jumlah antrian = " << countQueue() << endl;
93
94     return 0;
95 }
96

```

Hasil run:

```
PS D:\Praktikum\C++\Modul8> cd "d:\Praktikum\C++\Modul8\" ;
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 D:\Praktikum\C++\Modul8>
```

#### IV. UNGUIDED

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



```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 // Mendefinisikan struktur node untuk data antrian
6 struct Node {
7     string data; // data yang disimpan (misalnya nama mahasiswa atau data lainnya)
8     Node* next; // Pointer ke node berikutnya
9 };
10
11 // Mendefinisikan kelas Antrian dengan linked list
12 class Queue {
13 private:
14     Node* front; // Menunjuk ke elemen pertama dalam antrian
15     Node* back; // Menunjuk ke elemen terakhir dalam antrian
16
17 public:
18     // Konstruktor untuk inisialisasi antrian
19     Queue() {
20         front = nullptr;
21         back = nullptr;
22     }
23
24     // Mengecek apakah antrian kosong
25     bool isEmpty() {
26         return front == nullptr;
27     }
28
29     // Fungsi untuk menambahkan data ke antrian (enqueue)
30     void enqueue(string data) {
31         Node* newNode = new Node; // Membuat node baru
32         newNode->data = data;
33         newNode->next = nullptr;
34
35         if (isEmpty()) {
36             front = newNode; // Jika antrian kosong, node baru menjadi front dan back
37             back = newNode;
38         } else {
39             back->next = newNode; // Menambahkan node baru di belakang
40             back = newNode; // Update back ke node baru
41         }
42     }
43
44     // Fungsi untuk mengeluarkan data dari antrian (dequeue)
45     void dequeue() {
46         if (isEmpty()) {
47             cout << "Antrian kosong" << endl;
48         } else {
49             Node* temp = front;
50             front = front->next; // Menggeser front ke node berikutnya
51             delete temp; // Menghapus node lama
52             if (front == nullptr) {
53                 back = nullptr; // Jika antrian kosong, update back menjadi null
54             }
55         }
56     }
57
58     // Fungsi untuk melihat data dalam antrian
59     void viewQueue() {
60         if (isEmpty()) {
61             cout << "Antrian kosong" << endl;
62         } else {
63             Node* current = front;
64             cout << "Data antrian:" << endl;
65             int position = 1;
66             while (current != nullptr) {
67                 cout << position << ". " << current->data << endl;
68                 current = current->next;
69                 position++;
70             }
71         }
72     }
73
74     // Fungsi untuk menghitung jumlah data dalam antrian
75     int countQueue() {
76         int count = 0;
77         Node* current = front;
78         while (current != nullptr) {
79             count++;
80             current = current->next;
81         }
82         return count;
83     }
84
85     // Fungsi untuk menghapus semua data dalam antrian
86     void clearQueue() {
87         while (!isEmpty()) {
88             dequeue(); // Menghapus data satu per satu
89         }
90         cout << "Antrian telah dikosongkan." << endl;
91     }
92 };
93
94 int main() {
95     Queue q;
96     int pilihan;
97     string data;
98
99     while (true) {
100         cout << "\nMenu:\n";
101         cout << "1. Tambah antrian\n";
102         cout << "2. Hapus antrian\n";
103         cout << "3. Lihat antrian\n";
104         cout << "4. Hapus semua antrian\n";
105         cout << "5. Hitung jumlah antrian\n";
106         cout << "6. Keluar\n";
107         cout << "Pilih menu: ";
108         cin >> pilihan;
109         cin.ignore(); // Untuk membersihkan newline setelah input pilihan menu
110
111         switch (pilihan) {
112             case 1:
113                 cout << "Masukkan data: ";
114                 getline(cin, data);
115                 q.enqueue(data);
116                 break;
117             case 2:
118                 q.dequeue();
119                 break;
120             case 3:
121                 q.viewQueue();
122                 break;
123             case 4:
124                 q.clearQueue();
125                 break;
126             case 5:
127                 cout << "Jumlah antrian: " << q.countQueue() << endl;
128                 break;
129             case 6:
130                 cout << "Terima kasih!" << endl;
131                 return 0;
132             default:
133                 cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
134         }
135     }
136 }

```

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

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 // Mendefinisikan struktur node untuk data antrian mahasiswa
6 struct Mahasiswa {
7     string nama; // Nama mahasiswa
8     string nim; // NIM mahasiswa
9     Mahasiswa* next; // Pointer ke node berikutnya
10 };
11
12 // Mendefinisikan kelas Antrian dengan linked list
13 class Queue {
14 private:
15     Mahasiswa* front; // Menunjuk ke elemen pertama dalam antrian
16     Mahasiswa* back; // Menunjuk ke elemen terakhir dalam antrian
17 public:
18     // Konstruktor untuk inisialisasi antrian
19     Queue() {
20         front = nullptr;
21         back = nullptr;
22     }
23
24     // Menghapus antrian kosong
25     bool isEmpty() {
26         return front == nullptr;
27     }
28
29     // Fungsi untuk menambahkan data ke antrian (enqueue) dengan prioritas berdasarkan NIM
30     void enqueue(string nama, string nim) {
31         Mahasiswa* newNode = new Mahasiswa; // Membuat node baru
32         newNode->nama = nama;
33         newNode->nim = nim;
34         newNode->next = nullptr;
35
36         if (!isEmpty()) {
37             // Jika antrian kosong, node baru menjadi front dan back
38             front = newNode;
39             back = newNode;
40         } else {
41             // Jika antrian tidak kosong, kita cari posisi yang sesuai berdasarkan NIM
42             if (stoi(nim) < stoi(front->nim)) {
43                 // Jika NIM lebih kecil dari NIM di depan, insert di depan
44                 newNode->next = front;
45                 front = newNode;
46             } else {
47                 // Mencari posisi yang sesuai di antrian
48                 Mahasiswa* current = front;
49                 while (current->next != nullptr && stoi(nim) > stoi(current->next->nim)) {
50                     current = current->next;
51                 }
52                 // Menyisipkan node baru setelah node current
53                 newNode->next = current->next;
54                 current->next = newNode;
55
56                 // Jika node yang disisipkan berada di belakang, update back
57                 if (newNode->next == nullptr) {
58                     back = newNode;
59                 }
60             }
61         }
62     }
63
64     // Fungsi untuk mengeluarkan data dari antrian (dequeue)
65     void dequeue() {
66         if (!isEmpty()) {
67             cout << "Antrian kosong" << endl;
68         } else {
69             Mahasiswa* temp = front;
70             front = front->next; // Menggeser front ke node berikutnya
71             delete temp; // Menghapus node lama
72             if (front == nullptr) {
73                 back = nullptr; // Jika antrian kosong, update back menjadi null
74             }
75         }
76     }
77
78     // Fungsi untuk melihat data dalam antrian
79     void viewQueue() {
80         if (!isEmpty()) {
81             cout << "Antrian kosong" << endl;
82         } else {
83             Mahasiswa* current = front;
84             cout << "Data antrian mahasiswa:" << endl;
85             int position = 1;
86             while (current != nullptr) {
87                 cout << position << ". Nama: " << current->nama << ", NIM: " << current->nim << endl;
88                 current = current->next;
89                 position++;
90             }
91         }
92     }
93
94     // Fungsi untuk menghitung jumlah data dalam antrian
95     int countQueue() {
96         int count = 0;
97         Mahasiswa* current = front;
98         while (current != nullptr) {
99             count++;
100             current = current->next;
101         }
102         return count;
103     }
104
105     // Fungsi untuk menghapus semua data dalam antrian
106     void clearQueue() {
107         while (!isEmpty()) {
108             dequeue(); // Menghapus data satu per satu
109         }
110         cout << "Antrian telah dikosongkan." << endl;
111     }
112 };
113
114 int main() {
115     Queue q;
116     int pilihan;
117     string nama, nim;
118
119     while (true) {
120         cout << "Menu:\n";
121         cout << "1. Tambah antrian\n";
122         cout << "2. Lihat antrian\n";
123         cout << "3. Hitung jumlah antrian\n";
124         cout << "4. Hapus semua antrian\n";
125         cout << "5. Keluar\n";
126         cout << "Pilih menu: ";
127         int cin >> pilihan;
128         cin.ignore(); // Untuk membersihkan newline setelah input pilihan menu
129
130         switch (pilihan) {
131             case 1:
132                 cout << "Masukkan Nama Mahasiswa: ";
133                 getline(cin, nama);
134                 cout << "Masukkan NIM Mahasiswa: ";
135                 getline(cin, nim);
136                 q.enqueue(nama, nim);
137                 break;
138             case 2:
139                 q.dequeue();
140                 break;
141             case 3:
142                 q.viewQueue();
143                 break;
144             case 4:
145                 q.clearQueue();
146                 break;
147             case 5:
148                 cout << "Jumlah antrian: " << q.countQueue() << endl;
149                 break;
150             default:
151                 cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
152         }
153     }
154 }

```

3. Modifikasi program pada soal 1 sehingga mahasiswa dapat diprioritaskan berdasarkan NIM (NIM yang lebih kecil didahulukan pada saat output).

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 // Node untuk menyimpan data mahasiswa
6 struct Node {
7     string namaMahasiswa;
8     string nimMahasiswa;
9     Node* next;
10 };
11
12 // Pointer untuk mengelola queue
13 Node* front = nullptr; // Awal antrian
14
15 bool isEmpty() { // Mengecek apakah queue kosong
16     return (front == nullptr);
17 }
18
19 void enqueueAntrian(string nama, string nim) {
20     // Fungsi menambahkan data dengan prioritas NIM
21     Node* newNode = new Node();
22     newNode->namaMahasiswa = nama;
23     newNode->nimMahasiswa = nim;
24     newNode->next = nullptr;
25
26     if (isEmpty() || nim < front->nimMahasiswa) {
27         // Jika queue kosong atau prioritas lebih tinggi
28         newNode->next = front;
29         front = newNode;
30     } else {
31         // Sisipkan di posisi yang sesuai
32         Node* current = front;
33         while (current->next != nullptr && current->next->nimMahasiswa < nim) {
34             current = current->next;
35         }
36         newNode->next = current->next;
37         current->next = newNode;
38     }
39     cout << "Mahasiswa (" << nama << ", " << nim << ") berhasil ditambahkan ke antrian." << endl;
40 }
41
42 void dequeueAntrian() { // Fungsi menghapus data dari queue
43     if (isEmpty()) {
44         cout << "Antrian kosong, tidak ada yang dapat dihapus." << endl;
45     } else {
46         Node* temp = front;
47         front = front->next;
48         cout << "Mahasiswa (" << temp->namaMahasiswa << ", " << temp->nimMahasiswa << ") telah keluar dari antrian." << endl;
49         delete temp;
50     }
51 }
52
53 void viewQueue() { // Fungsi untuk melihat data pada queue
54     if (isEmpty()) {
55         cout << "Antrian kosong." << endl;
56     } else {
57         cout << "Data antrian berdasarkan prioritas NIM:" << endl;
58         Node* temp = front;
59         int nomor = 1;
60         while (temp != nullptr) {
61             cout << nomor++ << ". Nama: " << temp->namaMahasiswa
62                 << ", NIM: " << temp->nimMahasiswa << endl;
63             temp = temp->next;
64         }
65     }
66 }
67
68 void clearQueue() { // Fungsi untuk menghapus semua data pada queue
69     while (!isEmpty()) {
70         dequeueAntrian();
71     }
72     cout << "Semua data dalam antrian telah dihapus." << endl;
73 }
74
75 int main() {
76     int pilihan;
77     do {
78         cout << "\nMenu Antrian Mahasiswa dengan Prioritas NIM:\n";
79         cout << "1. Tambah Antrian (Enqueue)\n";
80         cout << "2. Hapus Antrian (Dequeue)\n";
81         cout << "3. Lihat Antrian\n";
82         cout << "4. Kosongkan Antrian\n";
83         cout << "5. Keluar\n";
84         cout << "Pilih menu: ";
85         cin >> pilihan;
86         cin.ignore(); // Membersihkan input buffer
87
88         switch (pilihan) {
89             case 1: {
90                 string nama, nim;
91                 cout << "Masukkan nama mahasiswa: ";
92                 getline(cin, nama);
93                 cout << "Masukkan NIM mahasiswa: ";
94                 getline(cin, nim);
95                 enqueueAntrian(nama, nim);
96                 break;
97             }
98             case 2:
99                 dequeueAntrian();
100                break;
101             case 3:
102                 viewQueue();
103                break;
104             case 4:
105                 clearQueue();
106                break;
107             case 5:
108                 cout << "Program selesai." << endl;
109                break;
110             default:
111                 cout << "Pilihan tidak valid, silakan coba lagi." << endl;
112            }
113        } while (pilihan != 5);
114
115        return 0;
116    }
117

```

## **V. KESIMPULAN**

Queue adalah salah satu struktur data yang merepresentasikan konsep antrian, di mana elemen yang pertama masuk akan menjadi elemen pertama yang diproses, sesuai dengan prinsip FIFO (First In, First Out). Konsep ini mirip dengan antrian pada loket tiket, di mana pelanggan yang datang lebih awal dilayani terlebih dahulu. Dalam pemrograman C, Queue dapat diimplementasikan menggunakan array atau linked list. Namun, implementasi dengan linked list memiliki keunggulan karena lebih fleksibel dan efisien dalam memanfaatkan memori. Operasi dasar Queue, seperti memasukkan data (enqueue) dan menghapus data (dequeue), dilakukan pada bagian belakang dan depan list, sehingga mempermudah pengelolaan data. Penggunaan linked list dalam Queue memungkinkan pengelolaan data secara dinamis, baik melalui single linked list maupun double linked list, menjadikannya pilihan yang sederhana namun sangat fungsional untuk berbagai kebutuhan praktis.