

LAPORAN PRAKTIKUM
STRUKTUR DATA 8
"QUEUE"



Oleh:

NAMA: Ammar Dzaki Nandana

NIM: 2311104071

KELAS: SE 07 02

DOSEN: Wahyu Andi Saputra

PRODI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2023/2024

I. TUJUAN

Tujuan dari praktikum ini adalah untuk memahami dan mengimplementasikan konsep **queue** dalam bahasa pemrograman C++. Praktikum ini dirancang agar mahasiswa dapat:

Memahami Konsep Queue

1. Memahami struktur data **queue** sebagai antrian dengan prinsip *First In, First Out (FIFO)*.
2. Mempelajari perbedaan antara queue statis dan queue dinamis.

Menerapkan Operasi Dasar Queue

Mengimplementasikan operasi dasar seperti:

1. **Enqueue**: Menambahkan elemen ke antrian.
2. **Dequeue**: Menghapus elemen dari antrian.
3. **Peek/Front**: Mengakses elemen di depan antrian.
4. **isEmpty**: Memeriksa apakah antrian kosong.
5. **isFull**: Memeriksa apakah antrian penuh (untuk queue statis).

Mengembangkan Program dengan Queue

Membuat program menggunakan queue untuk menyelesaikan masalah dunia nyata, seperti simulasi antrian di kasir, pengelolaan tugas, atau sistem pemrosesan data.

Mengeksplorasi Variasi Queue

Memahami dan mengimplementasikan variasi queue, seperti:

Circular Queue: Queue yang elemen terakhirnya terhubung ke elemen pertama.

Priority Queue: Queue di mana elemen dengan prioritas lebih tinggi diproses terlebih dahulu.

Mengoptimalkan Penggunaan Memori

Menerapkan **queue dinamis** menggunakan pointer untuk mengoptimalkan alokasi memori dibandingkan queue statis.

Menggunakan STL Queue

Memanfaatkan library **Standard Template Library (STL)** queue di C++ untuk implementasi yang lebih efisien dan efektif.

Mengembangkan Logika Pemrograman

Meningkatkan kemampuan mahasiswa dalam menyelesaikan masalah algoritma menggunakan pendekatan berbasis struktur data.

II. DASAR TEORI

1. *Queue* (dibaca : kyu) merupakan struktur data yang dapat diumpamakan seperti sebuah antrian. Misalkan antrian pada loket pembelian tiket Kereta Api. Orang yang akan mendapatkan pelayanan yang pertama adalah orang pertamakali masuk dalam antrian tersebut dan yang terakhir masuk dia akan mendapatkan layanan yang terakhir pula. Jadi prinsip dasar dalam *Queue* adalah **FIFO** (*First in Fisrt out*), proses yang pertama masuk akan diakses terlebih dahulu. Dalam pengimplementasian struktur *Queue* dalam C dapat menggunakan tipe data *array* dan *linked list*.

Dalam praktikum ini hanya akan dibahas pengimplementasian *Queue* dalam bentuk *linked list*. Implementasi *Queue* dalam *linked list* sebenarnya tidak jauh berbeda dengan operasi *list* biasa, malahan lebih sederhana. Karena sesuai dengan sifat FIFO dimana proses *delete* hanya dilakukan pada bagian **Head** (depan *list*) dan proses *insert* selalu dilakukan pada bagian **Tail** (belakang *list*) atau sebaliknya, tergantung dari persepsi masing-masing. Dalam penerapannya *Queue* dapat diterapkan dalam *single linked list* dan *double linked list*.

2. Karakteristik Queue

- **FIFO (First In, First Out):** Elemen pertama yang masuk akan menjadi elemen pertama yang keluar.
- **Operasi Terbatas:**
 - Elemen hanya dapat ditambahkan di **belakang** (rear) menggunakan operasi *enqueue*.
 - Elemen hanya dapat dihapus dari **depan** (front) menggunakan operasi *dequeue*.
- Digunakan untuk situasi di mana urutan proses harus dipertahankan.

3. Operasi Dasar pada Queue

1. **Enqueue:** Menambahkan elemen baru ke antrian (di belakang).
2. **Dequeue:** Menghapus elemen dari antrian (dari depan).
3. **Peek/Front:** Mengakses elemen paling depan dalam antrian.
4. **isEmpty:** Mengecek apakah antrian kosong.
5. **isFull:** Mengecek apakah antrian penuh (pada queue statis).

4. Variasi Queue

Linear Queue

- Antrian biasa dengan alokasi memori linier.
- Kekurangan: Memori di depan yang sudah di-*dequeue* tidak dapat digunakan kembali.

Circular Queue

- Mengatasi kekurangan linear queue dengan memutar indeks rear ke awal ketika mencapai batas akhir array.
- Efisiensi lebih baik dalam penggunaan memori.

Priority Queue

- Elemen dengan prioritas lebih tinggi akan dikeluarkan lebih dulu, terlepas dari urutan masuknya.

Double-Ended Queue (Deque)

- Elemen dapat ditambahkan atau dihapus di kedua ujung (depan atau belakang).

III. GUIDED

```
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     cout << "After dequeuing, queue elements: ";
75     q.display();
76
77     return 0;
78 }
```


```
1 #include <iostream>
2
3 using namespace std;
4
5 // Node untuk setiap elemen Queue
6 class Node {
7 public:
8     int data; // Data elemen
9     Node* next; // Pointer ke node berikutnya
10
11     // Konstruktor untuk Node
12     Node(int value) {
13         data = value;
14         next = nullptr;
15     }
16 };
17
18 // Kelas Queue menggunakan linked list
19 class Queue {
20 private:
21     Node* front; // Pointer ke elemen depan Queue
22     Node* rear; // Pointer ke elemen belakang Queue
23 public:
24     // Konstruktor Queue
25     Queue() {
26         front = rear = nullptr;
27     }
28
29     // Mengecek apakah Queue kosong
30     bool isEmpty() {
31         return front == nullptr;
32     }
33
34     // Menambahkan elemen ke Queue
35     void enqueue(int x) {
36         Node* newNode = new Node(x);
37         if (isEmpty()) {
38             front = rear = newNode; // Jika Queue kosong
39             return;
40         }
41         rear->next = newNode; // Tambahkan node baru ke belakang
42         rear = newNode; // Perbarui rear
43     }
44
45     // Menghapus elemen dari depan Queue
46     void dequeue() {
47         if (isEmpty()) {
48             cout << "Queue Underflow\n";
49             return;
50         }
51         Node* temp = front; // Simpan node depan untuk dihapus
52         front = front->next; // Pindahkan front ke node berikutnya
53         delete temp; // Hapus node lama
54         if (front == nullptr) // Jika Queue kosong, rear juga harus null
55             rear = nullptr;
56     }
57
58     // Mengembalikan elemen depan Queue tanpa menghapusnya
59     int peek() {
60         if (!isEmpty()) {
61             return front->data;
62         }
63         cout << "Queue is empty\n";
64         return -1; // Nilai sentinel
65     }
66
67     // Menampilkan semua elemen di Queue
68     void display() {
69         if (isEmpty()) {
70             cout << "Queue is empty\n";
71             return;
72         }
73         Node* current = front; // Mulai dari depan
74         while (current) { // Iterasi sampai akhir
75             cout << current->data << " ";
76             current = current->next;
77         }
78         cout << "\n";
79     }
80 };
81
82 // Fungsi utama untuk menguji Queue
83 int main() {
84     Queue q;
85
86     // Menambahkan elemen ke Queue
87     q.enqueue(10);
88     q.enqueue(20);
89     q.enqueue(30);
90
91     // Menampilkan elemen di Queue
92     cout << "Queue elements: ";
93     q.display();
94
95     // Menampilkan elemen depan
96     cout << "Front element: " << q.peek() << "\n";
97
98     // Menghapus elemen dari depan Queue
99     q.dequeue();
100     cout << "After dequeuing, queue elements: ";
101     q.display();
102
103     return 0;
104 }
```

```

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; front++;
32             back++;
33         } else { // Antriannya ada isi queueTeller[back] = data; back++;
34             }
35     }
36 }
37
38 void dequeueAntrian() { // Fungsi mengurangi antrian
39     if (isEmpty()) {
40         cout << "Antrian kosong" << endl;
41     } else {
42         for (int i = 0; i < back; i++) { queueTeller[i] = queueTeller[i + 1];
43             }
44         back--;
45     }
46 }
47
48 int countQueue() { // Fungsi menghitung banyak antrian
49     return back;
50 }
51
52 void clearQueue() { // Fungsi menghapus semua antrian
53     if (isEmpty()) {
54         cout << "Antrian kosong" << endl;
55     } else {
56         for (int i = 0; i < back; i++) { queueTeller[i] = "";
57             }
58         back = 0;
59         front = 0;
60     }
61 }
62
63 void viewQueue() { // Fungsi melihat antrian
64     cout << "Data antrian teller:" << endl; for (int i = 0; i < maksimalQueue; i++) {
65         if (queueTeller[i] != "") {
66             cout << i + 1 << ". " << queueTeller[i] << endl;
67         }
68     } else {
69         cout << i + 1 << ". (kosong)" << endl;
70     }
71 }
72 }
73
74 int main() {
75     enqueueAntrian("Andi");
76
77     enqueueAntrian("Maya");
78
79     viewQueue();
80     cout << "Jumlah antrian = " << countQueue() << endl;
81
82     dequeueAntrian();
83     viewQueue();
84     cout << "Jumlah antrian = " << countQueue() << endl;
85
86     clearQueue();
87     viewQueue();
88     cout << "Jumlah antrian = " << countQueue() << endl;
89
90     return 0;
91 }

```

IV. UNGUIDED



```
1  #ifndef QUEUE_H
2  #define QUEUE_H
3
4  #include <iostream>
5  using namespace std;
6
7  const int MAX_SIZE = 5; // Kapasitas maksimum Queue
8
9  typedef int infotype;
10
11 struct Queue {
12     infotype info[MAX_SIZE]; // Array untuk menyimpan elemen Queue
13     int head; // Indeks elemen pertama
14     int tail; // Indeks elemen terakhir
15 };
16
17 // Fungsi dan prosedur
18 void createQueue(Queue &Q);
19 bool isEmptyQueue(const Queue &Q);
20 bool isFullQueue(const Queue &Q);
21 void enqueue(Queue &Q, infotype x);
22 infotype dequeue(Queue &Q);
23 void printInfo(const Queue &Q);
24
25 #endif
26
```



```
1  #include "queue.h"
2
3  void createQueue(Queue &Q) {
4      Q.head = 0;
5      Q.tail = -1;
6  }
7
8  bool isEmptyQueue(const Queue &Q) {
9      return Q.tail < Q.head;
10 }
11
12 bool isFullQueue(const Queue &Q) {
13     return Q.tail == MAX_SIZE - 1;
14 }
15
16 void enqueue(Queue &Q, infotype x) {
17     if (isFullQueue(Q)) {
18         cout << "Queue penuh! Tidak dapat menambah elemen." << endl;
19     } else {
20         Q.tail++;
21         Q.info[Q.tail] = x;
22     }
23 }
24
25 infotype dequeue(Queue &Q) {
26     if (isEmptyQueue(Q)) {
27         cout << "Queue kosong! Tidak dapat mengambil elemen." << endl;
28         return -1;
29     } else {
30         infotype temp = Q.info[Q.head];
31         for (int i = Q.head; i < Q.tail; i++) {
32             Q.info[i] = Q.info[i + 1];
33         }
34         Q.tail--;
35         return temp;
36     }
37 }
38
39 void printInfo(const Queue &Q) {
40     if (isEmptyQueue(Q)) {
41         cout << "Queue kosong." << endl;
42     } else {
43         cout << "Queue: ";
44         for (int i = Q.head; i <= Q.tail; i++) {
45             cout << Q.info[i] << " ";
46         }
47         cout << endl;
48     }
49 }
50
```



```
1 void enqueue(Queue &Q, infotype x) {
2     if (isFullQueue(Q)) {
3         cout << "Queue penuh! Tidak dapat menambah elemen." << endl;
4     } else {
5         if (isEmptyQueue(Q)) {
6             Q.head = 0;
7         }
8         Q.tail = (Q.tail + 1) % MAX_SIZE;
9         Q.info[Q.tail] = x;
10    }
11 }
12
13 infotype dequeue(Queue &Q) {
14     if (isEmptyQueue(Q)) {
15         cout << "Queue kosong! Tidak dapat mengambil elemen." << endl;
16         return -1;
17     } else {
18         infotype temp = Q.info[Q.head];
19         if (Q.head == Q.tail) { // Jika Queue hanya memiliki satu elemen
20             Q.head = -1;
21             Q.tail = -1;
22         } else {
23             Q.head = (Q.head + 1) % MAX_SIZE;
24         }
25         return temp;
26     }
27 }
28
```




```
1  bool isFullQueue(const Queue &Q) {
2      return (Q.tail + 1) % MAX_SIZE == Q.head;
3  }
4
5  void enqueue(Queue &Q, infotype x) {
6      if (isFullQueue(Q)) {
7          cout << "Queue penuh! Tidak dapat menambah elemen." << endl;
8      } else {
9          if (isEmptyQueue(Q)) {
10             Q.head = 0;
11         }
12         Q.tail = (Q.tail + 1) % MAX_SIZE;
13         Q.info[Q.tail] = x;
14     }
15 }
16
17 infotype dequeue(Queue &Q) {
18     if (isEmptyQueue(Q)) {
19         cout << "Queue kosong! Tidak dapat mengambil elemen." << endl;
20         return -1;
21     } else {
22         infotype temp = Q.info[Q.head];
23         if (Q.head == Q.tail) { // Jika Queue hanya memiliki satu elemen
24             Q.head = -1;
25             Q.tail = -1;
26         } else {
27             Q.head = (Q.head + 1) % MAX_SIZE;
28         }
29         return temp;
30     }
31 }
32
```



```
1  #include "queue.h"
2
3  int main() {
4      Queue Q;
5      createQueue(Q);
6
7      cout << "-----" << endl;
8      cout << " H - T \t | Queue info" << endl;
9      cout << "-----" << endl;
10
11     printInfo(Q);
12     enqueue(Q, 5); printInfo(Q);
13     enqueue(Q, 2); printInfo(Q);
14     enqueue(Q, 7); printInfo(Q);
15     dequeue(Q); printInfo(Q);
16     enqueue(Q, 4); printInfo(Q);
17     dequeue(Q); printInfo(Q);
18     dequeue(Q); printInfo(Q);
19
20     return 0;
21 }
22
```

V. KESIMPULAN

Queue adalah struktur data linier yang bekerja berdasarkan prinsip **First In, First Out (FIFO)**, di mana elemen yang masuk pertama akan keluar pertama. Konsep ini menyerupai antrian dalam kehidupan sehari-hari, seperti antrean pembelian tiket atau pelanggan di kasir. Struktur data ini memiliki operasi dasar seperti **enqueue** untuk menambahkan elemen di akhir antrian, **dequeue** untuk menghapus elemen dari awal antrian, dan **peek** untuk melihat elemen terdepan tanpa menghapusnya. Terdapat berbagai variasi queue, seperti **linear queue**, **circular queue**, dan **priority queue**, yang masing-masing memiliki karakteristik dan penggunaan yang spesifik. Queue banyak digunakan dalam berbagai aplikasi nyata, seperti manajemen tugas dalam sistem operasi, simulasi antrian, dan algoritma pencarian seperti Breadth-First Search (BFS). Dengan memahami implementasi queue, baik menggunakan array maupun linked list, pengguna dapat mengoptimalkan penyimpanan dan pemrosesan data dalam berbagai konteks pemrograman.