

LAPORAN PRAKTIKUM
Modul 8
“QUEUE”



Disusun Oleh:

Nama : Ganes Gemi Putra

NIM : 2311104075

Kelas : SE-07-02

Dosen : WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

Memahami Konsep Queue

Mahasiswa dapat memahami definisi, prinsip kerja, dan karakteristik queue sebagai struktur data yang menerapkan konsep FIFO (First-In, First-Out).

Mengimplementasikan Operasi Queue

Mahasiswa mampu mengimplementasikan operasi dasar queue, yaitu:

- **Enqueue:** Menambahkan elemen ke dalam queue.
- **Dequeue:** Menghapus elemen dari queue.
- **Peek:** Melihat elemen di depan queue tanpa menghapusnya.

Memahami Berbagai Alternatif Implementasi Queue

Mahasiswa mampu membandingkan dan memahami tiga alternatif implementasi queue:

- Alternatif 1: Head diam, Tail bergerak.
- Alternatif 2: Head dan Tail bergerak.
- Alternatif 3: Circular Queue.

Memahami Penerapan Queue dalam Kehidupan Nyata

Mahasiswa dapat mengidentifikasi penerapan queue dalam berbagai sistem nyata, seperti manajemen antrian, penjadwalan CPU, dan buffering data.

Mengembangkan Kemampuan Pemrograman

Mahasiswa dapat mengimplementasikan queue menggunakan bahasa pemrograman (C++), baik berbasis array maupun linked list, sesuai dengan skenario yang diberikan.

Meningkatkan Pemahaman Efisiensi Algoritma

Mahasiswa dapat menganalisis efisiensi ruang dan waktu dari berbagai metode implementasi queue, termasuk solusi untuk masalah seperti "semu penuh" dalam array queue.

2. Landasan Teori

Definisi Queue

Queue adalah struktur data linear yang menggunakan prinsip FIFO (First-In, First-Out). Elemen yang pertama kali dimasukkan ke dalam queue akan menjadi elemen pertama yang dikeluarkan. Struktur data ini menyerupai antrian di dunia nyata, seperti antrean loket, di mana orang yang datang lebih dahulu akan dilayani lebih dahulu.

Konsep FIFO

FIFO merupakan aturan dasar dalam queue, di mana elemen pertama yang masuk (First In) adalah elemen pertama yang keluar (First Out). Konsep ini membedakan queue dari struktur data lain seperti stack (yang menggunakan konsep LIFO - Last In, First Out).

Komponen Utama Queue

- Head: Pointer atau indeks yang menunjuk ke elemen pertama dalam queue.
- Tail: Pointer atau indeks yang menunjuk ke elemen terakhir dalam queue.
- Array atau Linked List: Struktur penyimpanan untuk elemen queue.

Operasi Dasar Queue

- Enqueue: Menambahkan elemen baru ke akhir queue.
- Dequeue: Menghapus elemen dari awal queue.
- Peek: Mengambil elemen di awal queue tanpa menghapusnya.
- IsEmpty: Mengecek apakah queue kosong.
- IsFull: Mengecek apakah queue penuh (untuk implementasi berbasis array).

Implementasi Queue

Queue dapat diimplementasikan dengan beberapa cara:

- **Array:** Menggunakan indeks tetap untuk elemen. Kelebihan: sederhana; Kekurangan: ruang terbatas.
- **Linked List:** Menggunakan node dinamis yang saling terhubung. Kelebihan: fleksibel; Kekurangan: lebih kompleks dibanding array.
- **Circular Queue:** Variasi array queue yang memungkinkan ruang dimanfaatkan secara efisien dengan memutar indeks.

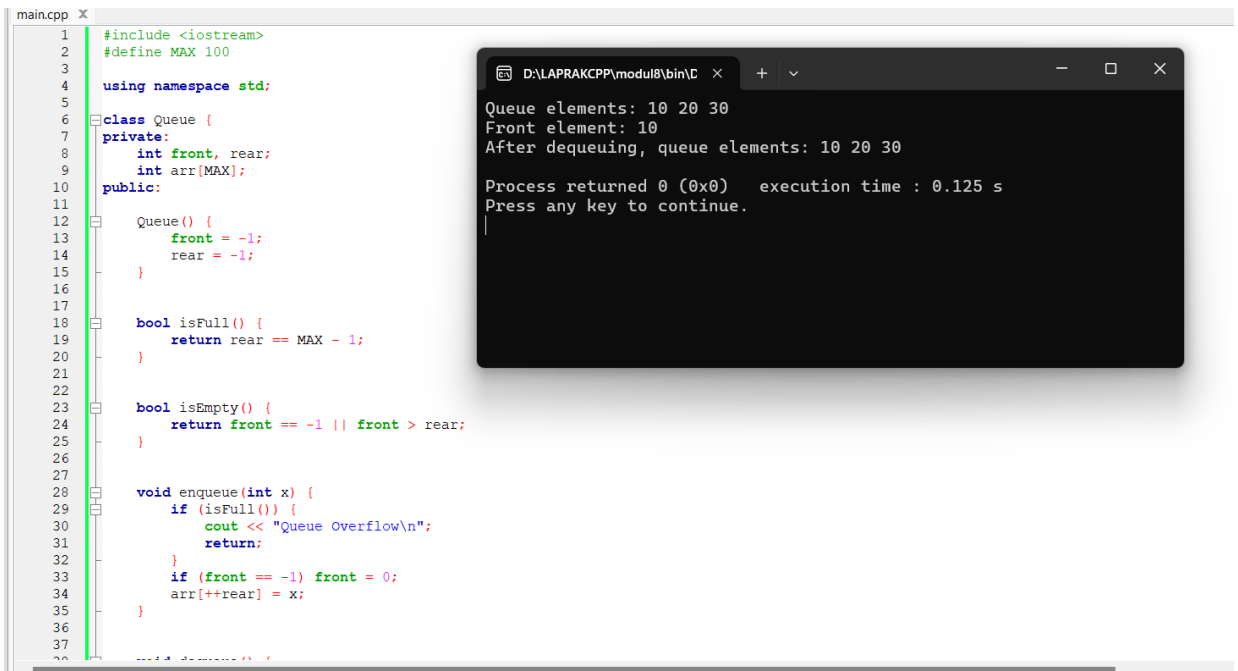
Penggunaan Queue dalam Kehidupan Nyata

- **Penjadwalan CPU:** Proses dalam sistem operasi dijadwalkan menggunakan queue.
- **Antrean Pelanggan:** Digunakan dalam sistem layanan seperti loket tiket, kasir, atau panggilan telepon.
- **Buffer Data:** Dalam jaringan komputer, queue digunakan untuk mengelola buffer paket data.

Keunggulan dan Kekurangan Queue

- **Keunggulan:** Mudah diimplementasikan dan efisien untuk mengelola data yang masuk dan keluar secara berurutan.
- **Kekurangan:** Implementasi array dapat menyebabkan pemborosan ruang jika tidak dioptimalkan.

3. Guided 1 :

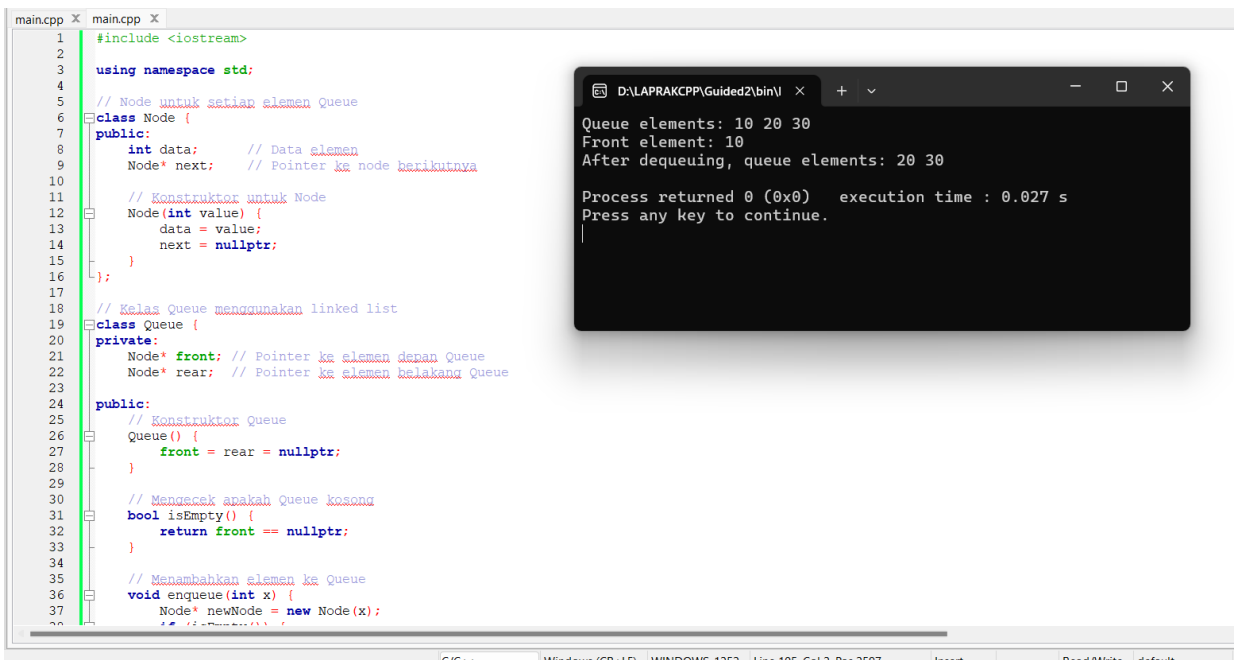


The screenshot shows a C++ IDE with a file named `main.cpp`. The code implements a queue using an array. The `Queue` class has private attributes `front` and `rear`, and a public array `arr` of size `MAX` (100). The `enqueue` method checks if the queue is full before adding an element. The `isFull` method returns `rear == MAX - 1`. The `isEmpty` method returns `front == -1 || front > rear`. The `enqueue` method increments `rear` and stores the element in `arr[rear]`. The output window shows the following text:

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30

Process returned 0 (0x0)   execution time : 0.125 s
Press any key to continue.
```

Guided 2 :

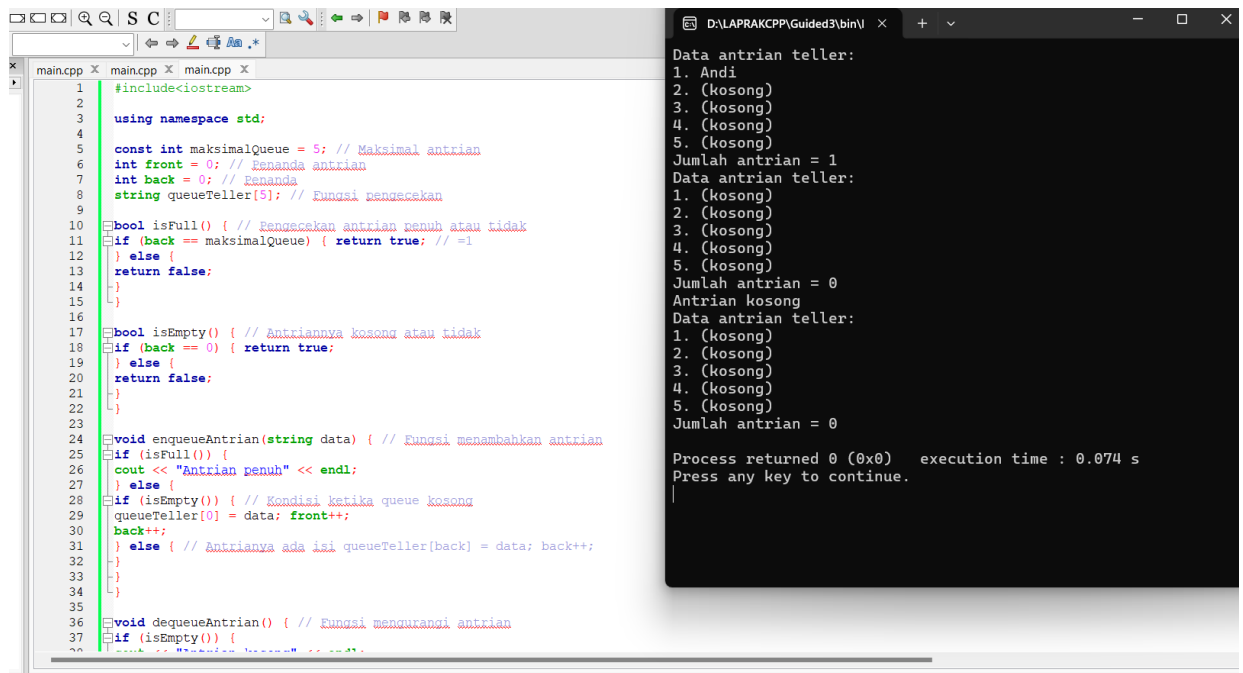


The screenshot shows a C++ IDE with a file named `main.cpp`. The code implements a queue using a linked list. The `Node` class has private attributes `data` and `next`. The `Queue` class has private attributes `front` and `rear`, and public methods `enqueue` and `isEmpty`. The `enqueue` method creates a new node and adds it to the queue. The `isEmpty` method returns `front == nullptr`. The output window shows the following text:

```
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30

Process returned 0 (0x0)   execution time : 0.027 s
Press any key to continue.
```

‘Guided 3 :



The screenshot shows a C++ IDE with a file named `main.cpp` and a terminal window. The code implements a queue using an array `queueTeller` of size 5. It includes functions `isFull()`, `isEmpty()`, `enqueueAntrian()`, and `dequeueAntrian()`. The terminal output shows the queue being filled with names and then emptied.

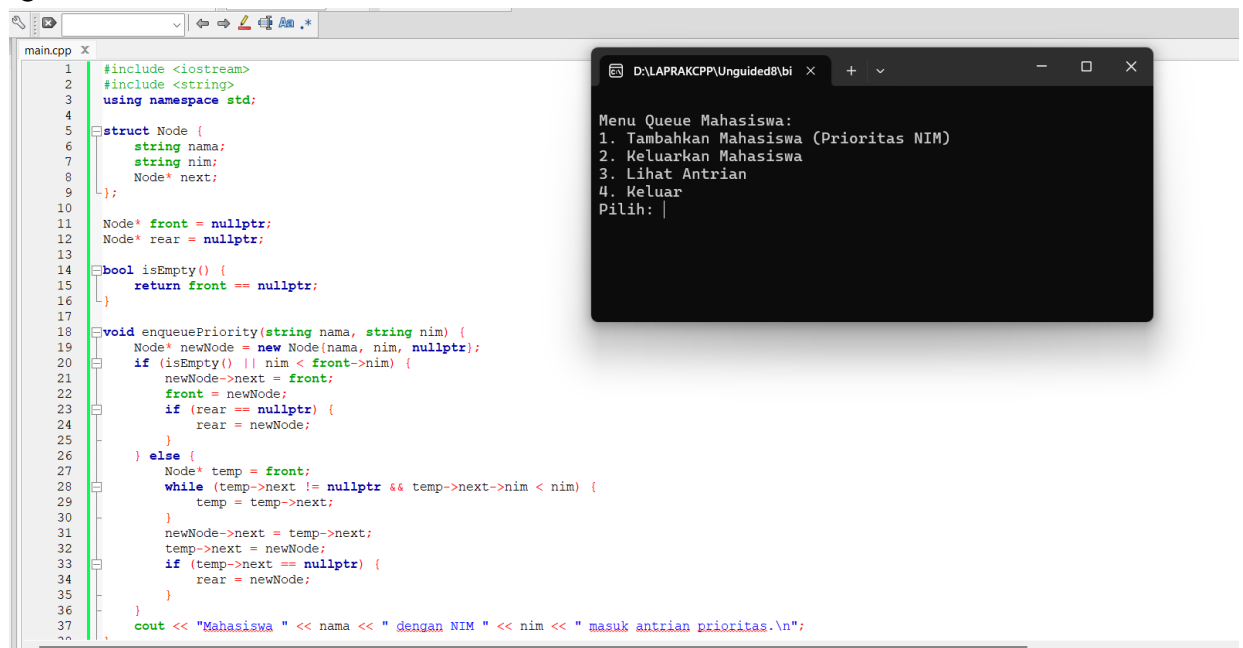
```
1 #include<iostream>
2
3 using namespace std;
4
5 const int maksimalQueue = 5; // Maksimal antrian
6 int front = 0; // Belakang antrian
7 int back = 0; // Depan antrian
8 string queueTeller[5]; // Fungsi pengacakan
9
10 bool isFull() { // Pengacakan antrian penuh atau tidak
11     if (back == maksimalQueue) { return true; // =1
12     } else {
13         return false;
14     }
15 }
16
17 bool isEmpty() { // Antriannya kosong atau tidak
18     if (back == 0) { return true;
19     } else {
20         return false;
21     }
22 }
23
24 void enqueueAntrian(string data) { // Fungsi menambahkan antrian
25     if (isFull()) {
26         cout << "Antrian penuh" << endl;
27     } else {
28         if (isEmpty()) { // Kondisi ketika queue kosong
29             queueTeller[0] = data; front++;
30             back++;
31         } else { // Antrianya ada isi queueTeller[back] = data; back++;
32             queueTeller[back] = data; back++;
33         }
34     }
35 }
36
37 void dequeueAntrian() { // Fungsi mengurangi antrian
38     if (isEmpty()) {
39         cout << "Antrian kosong" << endl;
40     }
41 }
```

Terminal Output:

```
Data antrian teller:
1. Andi
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
Antrian kosong
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0

Process returned 0 (0x0)   execution time : 0.074 s
Press any key to continue.
```

4. Unguided Queue :



The screenshot shows a C++ IDE with a file named `main.cpp` and a terminal window. The code implements an unguided queue using a linked list structure with `Node` objects. It includes functions `isEmpty()` and `enqueuePriority()`. The terminal output shows a menu for adding, removing, and viewing students in the queue.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct Node {
6     string nama;
7     string nim;
8     Node* next;
9 };
10
11 Node* front = nullptr;
12 Node* rear = nullptr;
13
14 bool isEmpty() {
15     return front == nullptr;
16 }
17
18 void enqueuePriority(string nama, string nim) {
19     Node* newNode = new Node(nama, nim, nullptr);
20     if (isEmpty() || nim < front->nim) {
21         newNode->next = front;
22         front = newNode;
23         if (rear == nullptr) {
24             rear = newNode;
25         }
26     } else {
27         Node* temp = front;
28         while (temp->next != nullptr && temp->next->nim < nim) {
29             temp = temp->next;
30         }
31         newNode->next = temp->next;
32         temp->next = newNode;
33         if (temp->next == nullptr) {
34             rear = newNode;
35         }
36     }
37     cout << "Mahasiswa " << nama << " dengan NIM " << nim << " masuk antrian prioritas.\n";
38 }
```

Terminal Output:

```
Menu Queue Mahasiswa:
1. Tambahkan Mahasiswa (Prioritas NIM)
2. Keluarkan Mahasiswa
3. Lihat Antrian
4. Keluar
Pilih: |
```

Unguided 2 :

```
main.cpp x main.cpp x
1 #include <iostream>
2 using namespace std;
3
4 const int MAX_SIZE = 5; // Ukuran maksimal queue
5 typedef int infotype;
6
7 struct Queue {
8     infotype info[MAX_SIZE];
9     int head, tail;
10 };
11
12 // Prototipe fungsi
13 void createQueue(Queue &Q);
14 bool isEmptyQueue(Queue Q);
15 bool isFullQueue(Queue Q);
16
17 // Alternatif 1: Head Diam, Tail Bergerak
18 void enqueueAlt1(Queue &Q, infotype x);
19 infotype dequeueAlt1(Queue &Q);
20
21 // Alternatif 2: Head dan Tail Bergerak
22 void enqueueAlt2(Queue &Q, infotype x);
23 infotype dequeueAlt2(Queue &Q);
24
25 // Alternatif 3: Circular Queue
26 void enqueueAlt3(Queue &Q, infotype x);
27 infotype dequeueAlt3(Queue &Q);
28
29 void printInfo(Queue Q);
30
31 int main() {
32     Queue Q;
33     createQueue(Q);
34
35     int pilihan, alternatif, nilai;
36     cout << "Pilih Alternatif Queue:\n";
37     cout << "1. Head Diam, Tail Bergerak\n";
38     cout << "2. Head dan Tail Bergerak\n";
39     cout << "3. Circular Queue\n";
40     cout << "4. Keluar\n";
41     cout << "Pilihan: ";
42     int input;
43     while (input != 4) {
44         input = getche();
45         if (input == '\n') continue;
46         pilihan = input - '0';
47         if (pilihan < 1 || pilihan > 4) continue;
48         cout << "\n";
49         switch (pilihan) {
50             case 1:
51                 cout << "Pilih Alternatif Queue:\n";
52                 cout << "1. Head Diam, Tail Bergerak\n";
53                 cout << "2. Head dan Tail Bergerak\n";
54                 cout << "3. Circular Queue\n";
55                 cout << "4. Keluar\n";
56                 cout << "Pilihan: ";
57                 int input2;
58                 while (input2 != 4) {
59                     input2 = getche();
60                     if (input2 == '\n') continue;
61                     pilihan2 = input2 - '0';
62                     if (pilihan2 < 1 || pilihan2 > 4) continue;
63                     cout << "\n";
64                     switch (pilihan2) {
65                         case 1:
67                             enqueueAlt1(Q, nilai);
68                             printInfo(Q);
69                             cout << "Data dihapus: Queue kosong!\n";
70                             break;
71                         case 2:
72                             enqueueAlt2(Q, nilai);
73                             printInfo(Q);
74                             break;
75                         case 3:
76                             enqueueAlt3(Q, nilai);
77                             printInfo(Q);
78                             break;
79                     }
80                 }
81                 break;
82             case 2:
83                 enqueueAlt2(Q, nilai);
84                 printInfo(Q);
85                 break;
86             case 3:
87                 enqueueAlt3(Q, nilai);
88                 printInfo(Q);
89                 break;
90             case 4:
91                 break;
92         }
93     }
94 }
```

```
Pilih Alternatif Queue:
1. Head Diam, Tail Bergerak
2. Head dan Tail Bergerak
3. Circular Queue
Pilihan: 1

Menu Queue:
1. Tambah Data (Enqueue)
2. Hapus Data (Dequeue)
3. Tampilkan Queue
4. Keluar
Pilihan: 2
Data dihapus: Queue kosong!
-1

Menu Queue:
1. Tambah Data (Enqueue)
2. Hapus Data (Dequeue)
3. Tampilkan Queue
4. Keluar
Pilihan: |
```

5. Kesimpulan

Pemahaman Konsep Queue:

Queue adalah struktur data yang bekerja dengan prinsip FIFO (First-In, First-Out). Elemen pertama yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Konsep ini sering digunakan dalam berbagai aplikasi seperti pengelolaan antrian dalam sistem nyata (misalnya, antrean loket atau printer).

Implementasi Queue:

- **Array:** Implementasi queue dengan array sederhana namun memiliki keterbatasan ruang karena jumlah elemen tetap.
- **Linked List:** Memungkinkan queue dengan ukuran dinamis, lebih fleksibel dibandingkan array.
- **Circular Queue:** Memanfaatkan seluruh ruang array secara efisien dengan mengatasi masalah "semu penuh."

Alternatif Implementasi Queue dengan Array:

- **Alternatif 1 (Head diam, Tail bergerak):** Mudah diimplementasikan, tetapi tidak efisien karena elemen di awal tidak dapat digunakan kembali.
- **Alternatif 2 (Head dan Tail bergerak):** Lebih efisien dalam memanfaatkan ruang, namun membutuhkan manajemen untuk menghindari kondisi "semu penuh."
- **Alternatif 3 (Circular Queue):** Pilihan terbaik untuk efisiensi memori karena ruang array dimanfaatkan secara melingkar.

Praktik Operasi Queue:

Praktikum ini menunjukkan cara menambahkan elemen (enqueue), menghapus elemen (dequeue), dan menampilkan isi queue. Operasi ini mendukung pemahaman langsung konsep FIFO.

Penerapan Queue dalam Kehidupan Nyata:

Struktur data queue digunakan dalam berbagai sistem, seperti penjadwalan CPU, antrean pelanggan, buffer data dalam jaringan, dan manajemen layanan di sistem tiket.