

LAPORAN PRAKTIKUM
PERTEMUAN 8



Nama :

Razhendriya Vania Ramadhan Suganjarsarwat (2311104048)

Dosen :

WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2024

I. TUJUAN

- Memahami definisi dan konsep queue.
- Menerapkan operasi tambah, hapus, dan menampilkan data pada queue.
- Mengembangkan implementasi queue menggunakan array atau linked list.

II. TOOL

VScode

III. DASAR TEORI

Queue adalah struktur data yang menggunakan metode FIFO (First-In, First-Out). Artinya, data yang pertama masuk akan menjadi data pertama yang keluar. Queue sering digunakan dalam berbagai aplikasi seperti sistem antrean, proses multitasking, atau layanan jaringan.

Pada queue, terdapat dua operasi utama:

- **Enqueue:** Menambahkan elemen di akhir queue.
- **Dequeue:** Menghapus elemen dari awal queue.

Perbedaan antara Stack dan Queue

- **Stack** menggunakan prinsip **LIFO (Last-In, First-Out)**, di mana elemen terakhir yang masuk adalah elemen pertama yang keluar.
- **Queue** menggunakan prinsip **FIFO**, di mana elemen pertama yang masuk adalah elemen pertama yang keluar.

Operasi tambahan lainnya pada queue:

- **Peek:** Melihat elemen di awal queue tanpa menghapusnya.
- **isEmpty:** Mengecek apakah queue kosong.
- **isFull:** Mengecek apakah queue penuh.
- **Size:** Menghitung jumlah elemen di dalam queue.

IV. 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     q.enqueue(10);
65     q.enqueue(20);
66     q.enqueue(30);
67
68     cout << "Queue elements: ";
69     q.display();
70
71     cout << "Front element: " << q.peek() << "\n";
72
73     cout << "After dequeuing, queue elements: ";
74     q.display();
75
76     return 0;
77 }
```

Menggunakan array dengan ukuran tetap (MAX = 100).

Operasi enqueue menambahkan elemen di belakang, dequeue menghapus elemen dari depan.

Sederhana dan cepat, tetapi ukuran queue terbatas (statis).

```

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
24 public:
25     // Konstruktor Queue
26     Queue() {
27         front = nullptr;
28     }
29
30 // Mengcek apakah Queue kosong
31 bool isEmpty() {
32     return front == nullptr;
33 }
34
35 // Menambahkan elemen ke Queue
36 void enqueue(int x) {
37     Node* node = new Node(x);
38     if (isEmpty()) {
39         front = rear = node; // Jika Queue kosong
40         return;
41     }
42     // Tambahkan node baru ke
43     rear->next = node; // Jarak antar rear
44 }
45
46 // Menghapus elemen dari depan Queue
47 void dequeue() {
48     if (isEmpty()) {
49         cout << "Queue Underflow\n";
50         return;
51     }
52     Node* temp = front; // Simpan node depan untuk dihapus
53     front = front->next; // Pindahkan front ke node berikutnya
54     delete temp; // Hapus node lama
55     if (front == nullptr) // Jika Queue kosong, rear juga harus null
56         rear = nullptr;
57 }
58
59 // Mengembalikan elemen depan Queue tanpa menghapusnya
60 int peek() {
61     if (isEmpty()) {
62         return front->data;
63     }
64     cout << "Queue is empty\n";
65     return -1; // Nilai sentinel
66 }
67
68 // Menampilkan semua elemen di Queue
69 void display() {
70     if (isEmpty()) {
71         cout << "Queue is empty\n";
72         return;
73     }
74     Node* current = front; // Mulai dari depan
75     while (current) { // // Sampai tempat akhir
76         cout << current->data << " ";
77         current = current->next;
78     }
79     cout << "\n";
80 }
81 };
82
83 // Fungsi utama untuk menguji Queue
84 int main() {
85     Queue q;
86
87     // Menambahkan elemen ke Queue
88     q.enqueue(10);
89     q.enqueue(20);
90     q.enqueue(30);
91
92     // Menampilkan elemen di Queue
93     cout << "Queue elements: ";
94     q.display();
95
96     // Mengambil elemen depan
97     cout << "Front element: " << q.peek() << "\n";
98
99     // Menghapus elemen dari depan Queue
100    q.dequeue();
101    cout << "After dequeue, queue elements: ";
102    q.display();
103
104    return 0;
105 }

```

Menggunakan node yang terhubung (linked list), sehingga ukuran queue fleksibel.

enqueue menambahkan elemen di belakang, dequeue menghapus elemen dari depan.

Memori lebih efisien untuk data besar, tapi implementasinya lebih kompleks.

```

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 isEmpty() { // Pengecekan antrian penuh atau tidak
11     if (back == maksimalQueue) { return true; // -1
12     } else {
13         return false;
14     }
15 }
16
17 bool isEmpty() { // Antrian 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 (isEmpty()) {
26         cout << "Antrian penuh" << endl;
27     } else {
28         if (isEmpty()) { // Kondisi ketika queue kosong
29             queueTeller[0] = data; front++;
30             back++;
31         } else { // Antrian ada isi queueTeller[back] = data; back++;
32             }
33         }
34     }
35 }
36
37 void dequeueAntrian() { // Fungsi mengurangi antrian
38     if (isEmpty()) {
39         cout << "Antrian kosong" << endl;
40     } else {
41         for (int i = 0; i < back; i++) { queueTeller[i] = queueTeller[i + 1];
42         }
43         back--;
44     }
45 }
46
47 int countQueue() { // Fungsi menghitung banyak antrian
48     return back;
49 }
50
51 void clearQueue() { // Fungsi menghapus semua antrian
52     if (isEmpty()) {
53         cout << "Antrian kosong" << endl;
54     } else {
55         for (int i = 0; i < back; i++) { queueTeller[i] = "";
56         }
57         back = 0;
58         front = 0;
59     }
60 }
61
62 void viewQueue() { // Fungsi melihat antrian
63     cout << "Data antrian teller:" << endl; for (int i = 0; i < maksimalQueue; i++) {
64         if (queueTeller[i] != "") {
65             cout << i + 1 << " : " << queueTeller[i] <<
66             endl;
67         }
68     } else {
69         cout << i + 1 << " : (kosong)" << endl;
70     }
71 }
72
73 }
74
75 int main() {
76     enqueueAntrian("Andi");
77     enqueueAntrian("Naya");
78
79     viewQueue();
80     cout << "Jumlah antrian = " << countQueue() << endl;
81     dequeueAntrian();
82     viewQueue();
83     cout << "Jumlah antrian = " << countQueue() << endl;
84     clearQueue();
85     viewQueue();
86     cout << "Jumlah antrian = " << countQueue() << endl;
87
88     return 0;
89 }

```

Array kecil dengan penanda front dan back, serta fitur tambahan seperti menghitung elemen (countQueue) dan membersihkan queue (clearQueue). Operasi dequeue memerlukan pergeseran elemen, sehingga kurang efisien dibanding metode lain.

V. UNGUIDED

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct Node {
6     string data;
7     Node* next;
8 };
9
10 Node* front = nullptr;
11 Node* rear = nullptr;
12
13 // Enqueue: Menambahkan elemen ke dalam queue
14 void enqueue(string value) {
15     Node* newNode = new Node();
16     newNode->data = value;
17     newNode->next = nullptr;
18     if (rear == nullptr) {
19         front = rear = newNode;
20     } else {
21         rear->next = newNode;
22         rear = newNode;
23     }
24 }
25
26 // Dequeue: Menghapus elemen dari depan queue
27 void dequeue() {
28     if (front == nullptr) {
29         cout << "Queue kosong.\n";
30         return;
31     }
32     Node* temp = front;
33     front = front->next;
34     if (front == nullptr) {
35         rear = nullptr;
36     }
37     delete temp;
38 }
39
40 // Menampilkan elemen queue
41 void displayQueue() {
42     Node* temp = front;
43     cout << "Isi queue:\n";
44     while (temp != nullptr) {
45         cout << temp->data << " ";
46         temp = temp->next;
47     }
48     cout << endl;
49 }
50
51 int main() {
52     enqueue("Data1");
53     enqueue("Data2");
54     displayQueue();
55     dequeue();
56     displayQueue();
57     return 0;
58 }
```

- **Fungsi Utama:** Mengimplementasikan queue untuk menyimpan data string.
- **Operasi:**
 - **Enqueue:** Menambahkan elemen di belakang queue.
 - **Dequeue:** Menghapus elemen dari depan queue.
 - **Display:** Menampilkan semua elemen di queue.
- **Ciri:** Tidak ada prioritas, elemen diproses dalam urutan FIFO (*First In, First Out*).

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct Mahasiswa {
6     string nama;
7     int nim;
8     Mahasiswa* next;
9 };
10
11 Mahasiswa* front = nullptr;
12 Mahasiswa* rear = nullptr;
13
14 // Enqueue
15 void enqueue(string nama, int nim) {
16     Mahasiswa* baru = new Mahasiswa();
17     baru->nama = nama;
18     baru->nim = nim;
19     baru->next = nullptr;
20     if (rear == nullptr) {
21         front = rear = baru;
22     } else {
23         rear->next = baru;
24         rear = baru;
25     }
26 }
27
28 // Dequeue
29 void dequeue() {
30     if (front == nullptr) {
31         cout << "Queue kosong.\n";
32         return;
33     }
34     Mahasiswa* temp = front;
35     front = front->next;
36     if (front == nullptr) {
37         rear = nullptr;
38     }
39     delete temp;
40 }
41
42 // Menampilkan Queue
43 void displayQueue() {
44     Mahasiswa* temp = front;
45     cout << "Isi queue mahasiswa:\n";
46     while (temp != nullptr) {
47         cout << temp->nama << " (" << temp->nim << ")\n";
48         temp = temp->next;
49     }
50 }
51
52 int main() {
53     enqueue("Andi", 12345);
54     enqueue("Budi", 54321);
55     displayQueue();
56     dequeue();
57     displayQueue();
58     return 0;
59 }

```

- **Fungsi Utama:** Queue untuk menyimpan informasi mahasiswa (nama dan NIM).
- **Operasi:**
 - **Enqueue:** Menambahkan mahasiswa ke belakang queue.
 - **Dequeue:** Menghapus mahasiswa dari depan queue.
 - **Display:** Menampilkan semua mahasiswa dalam queue.
- **Ciri:** Seperti kode pertama, elemen diproses dalam urutan FIFO.

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct Mahasiswa {
6     string nama;
7     int nim;
8     Mahasiswa* next;
9 };
10
11 Mahasiswa* front = nullptr;
12
13 // Enqueue dengan prioritas
14 void enqueue(string nama, int nim) {
15     Mahasiswa* baru = new Mahasiswa();
16     baru->nama = nama;
17     baru->nim = nim;
18     baru->next = nullptr;
19
20     if (front == nullptr || nim < front->nim) {
21         baru->next = front;
22         front = baru;
23     } else {
24         Mahasiswa* temp = front;
25         while (temp->next != nullptr && temp->next->nim < nim) {
26             temp = temp->next;
27         }
28         baru->next = temp->next;
29         temp->next = baru;
30     }
31 }
32
33 // Dequeue
34 void dequeue() {
35     if (front == nullptr) {
36         cout << "Queue kosong.\n";
37         return;
38     }
39     Mahasiswa* temp = front;
40     front = front->next;
41     delete temp;
42 }
43
44 // Menampilkan Queue
45 void displayQueue() {
46     Mahasiswa* temp = front;
47     cout << "List Queue mahasiswa:\n";
48     while (temp != nullptr) {
49         cout << temp->nama << " (" << temp->nim << ") \n";
50         temp = temp->next;
51     }
52 }
53
54 int main() {
55     enqueue("Andi", 12345);
56     enqueue("Budi", 54321);
57     enqueue("Cudi", 11111);
58     displayQueue();
59     dequeue();
60     displayQueue();
61     return 0;
62 }

```

Fungsi Utama: Queue dengan prioritas berdasarkan nilai NIM (semakin kecil NIM, semakin tinggi prioritas).

Operasi:

- **Enqueue:** Menambahkan mahasiswa ke posisi yang sesuai dengan prioritas.
- **Dequeue:** Menghapus elemen dengan prioritas tertinggi (NIM terkecil).
- **Display:** Menampilkan semua mahasiswa dalam queue.

Ciri: Tidak menganut FIFO, melainkan berdasarkan urutan prioritas (NIM).

VI. KESIMPULAN

Dalam laporan ini, mahasiswa berhasil memahami konsep dasar queue dengan implementasi FIFO, baik menggunakan array maupun linked list. Selain itu, modifikasi prioritas berdasarkan atribut NIM memberikan gambaran nyata tentang penerapan konsep queue dalam kehidupan nyata.