**LAPORAN AWAL STRUKTUR DATA**

**Pertemuan ke-7**

**Circular Queue**

****

Disusun oleh:

Nama : ZAHARA NAULI IBRAHIM

NIM : 241011402500

Kelas : 03TPLP023

**Program Studi Teknik Informatika**

**Fakultas Ilmu Komputer**

**Universitas Pamulang**

Jl. Surya Kencana No. 1 Pamulang Telp (021)7412566, Fax. (021)7412566

Tangerang Selatan – Banten

# LANDASAN TEORI

Dalam dunia pemrograman, queue (antrian) merupakan salah satu struktur data linear yang sangat penting dalam pengelolaan data secara berurutan. Queue menggunakan prinsip FIFO (First In, First Out), yang berarti elemen yang pertama kali masuk akan menjadi elemen pertama yang keluar.

Konsep ini mirip dengan situasi di kehidupan nyata, seperti barisan antrian di loket tiket — orang pertama yang datang akan dilayani terlebih dahulu. Struktur data queue sering digunakan pada sistem komputer yang membutuhkan pengaturan giliran, seperti:

* Sistem antrian cetak pada printer
* Penjadwalan proses CPU (CPU scheduling)
* Komunikasi antarproses (inter-process communication)
* Buffer data (data streaming atau keyboard buffer).

## 1. Pengertian Circular Queue

Circular Queue (Antrian Melingkar) adalah pengembangan dari Linear Queue di mana posisi terakhir dari antrian terhubung kembali ke posisi pertama membentuk lingkaran.

Struktur ini dirancang untuk mengatasi keterbatasan Linear Queue yang tidak dapat memanfaatkan ruang kosong di awal array setelah beberapa elemen dihapus (dequeue).

Dengan Circular Queue, jika elemen terakhir sudah terisi dan terdapat ruang kosong di awal array, maka penambahan data baru dapat dilakukan kembali di posisi awal (indeks 0).

Konsep ini membuat penggunaan memori menjadi lebih efisien tanpa perlu menggeser data.

Circular Queue memiliki dua indikator utama:

* FRONT → menunjukkan posisi elemen pertama (data yang akan dihapus).
* REAR → menunjukkan posisi elemen terakhir (data yang terakhir dimasukkan).

## 2. Karakteristik Circular Queue

Beberapa karakteristik utama Circular Queue antara lain:

1. Struktur data berbentuk melingkar, di mana elemen terakhir terhubung kembali ke elemen pertama.
2. Operasi dilakukan dengan dua pointer utama: FRONT dan REAR.
3. Menggunakan operator modulus (%) agar indeks tidak keluar dari batas array.
4. Kondisi kosong (Empty) terjadi saat FRONT == -1.
5. Kondisi penuh (Full) terjadi saat (REAR + 1) % MAX == FRONT.
6. Operasi penambahan dan penghapusan tidak mengubah urutan elemen yang sudah ad

## 3. Kelebihan Circular Queue

* Pemanfaatan memori lebih optimal dibandingkan Linear Queue.
* Tidak perlu melakukan pergeseran elemen ketika ada penghapusan data.
* Dapat diimplementasikan menggunakan array statis maupun dinamis.
* Cocok untuk sistem yang menggunakan antrian bergilir (round robin).

## 4. Kekurangan Circular Queue

* Lebih kompleks dibanding Linear Queue karena memerlukan logika tambahan untuk pengecekan kondisi penuh dan kosong.
* Jika pointer tidak dikelola dengan benar, dapat menyebabkan kesalahan logika (overflow atau underflow).

## 5. Operasi Dasar pada Circular Queue

### 1. Inisialisasi

* Menentukan nilai awal:
* FRONT = REAR = -1;

### 2. Enqueue (Menambah Elemen)

* Menambahkan elemen baru ke antrian.
* Jika antrian kosong → FRONT = REAR = 0.
* Jika REAR sudah di posisi akhir, maka pindah ke indeks awal menggunakan operasi (REAR + 1) % MAX.

### 3. Dequeue (Menghapus Elemen)

* Menghapus elemen di bagian depan.
* Jika FRONT sama dengan REAR, maka setelah penghapusan keduanya kembali ke -1.
* Jika tidak, maka FRONT bergeser ke (FRONT + 1) % MAX.

### 4. Display (Menampilkan Elemen)

* Menampilkan seluruh isi antrian mulai dari FRONT hingga REAR, dengan memperhatikan perputaran indeks.

## 6. Kondisi Circular Queue

|  |  |  |
| --- | --- | --- |
| Kondisi | Persyaratan | Keterangan |
| Kosong (Empty) | FRONT == -1 | Tidak ada elemen dalam queue |
| Penuh (Full) | (REAR + 1) % MAX == FRONT | Semua slot array sudah terisi |
| Normal | Selain dua kondisi di atas | Queue masih dapat digunakan |

## 7. Ilustrasi Circular Queue

Misalkan terdapat Circular Queue dengan ukuran MAX = 5, berikut contoh alur operasinya:

|  |  |  |  |
| --- | --- | --- | --- |
| Operasi | FRONT | REAR | Isi Queue |
| Awal | -1 | -1 | Kosong |
| Enqueue(10) | 0 | 0 | 10 |
| Enqueue(20) | 0 | 1 | 10, 20 |
| Enqueue(30) | 0 | 2 | 10, 20, 30 |
| Dequeue() | 1 | 2 | 20, 30 |
| Enqueue(40) | 1 | 3 | 20, 30, 40 |
| Enqueue(50) | 1 | 4 | 20, 30, 40, 50 |
| Enqueue(60) | 1 | 0 | 20, 30, 40, 50, 60 |

Dapat dilihat bahwa setelah REAR mencapai indeks terakhir (4), ia kembali ke indeks 0 — inilah yang disebut melingkar (circular).

## 8. Algoritma Circular Queue

Enqueue:

IF (FRONT == 0 AND REAR == MAX - 1) OR (FRONT == REAR + 1)

THEN Queue Penuh

ELSE IF FRONT == -1

FRONT = REAR = 0

ELSE

REAR = (REAR + 1) % MAX

INSERT data ke Q[REAR]

Dequeue:

IF FRONT == -1

THEN Queue Kosong

PRINT Q[FRONT]

IF FRONT == REAR

FRONT = REAR = -1

ELSE

FRONT = (FRONT + 1) % MAX

## 9. Contoh Implementasi Program Circular Queue

#include <iostream>

using namespace std;

#define MAX 5

int Q[MAX];

int FRONT = -1, REAR = -1;

void enqueue(int data) {

if ((FRONT == 0 && REAR == MAX - 1) || (FRONT == REAR + 1)) {

cout << "Queue penuh!" << endl;

} else {

if (FRONT == -1) FRONT = 0;

REAR = (REAR + 1) % MAX;

Q[REAR] = data;

cout << data << " dimasukkan ke queue." << endl;

}

}

void dequeue() {

if (FRONT == -1) {

cout << "Queue kosong!" << endl;

} else {

cout << Q[FRONT] << " dihapus dari queue." << endl;

if (FRONT == REAR) {

FRONT = REAR = -1;

} else {

FRONT = (FRONT + 1) % MAX;

}

}

}

void display() {

if (FRONT == -1) {

cout << "Queue kosong!" << endl;

} else {

cout << "Isi Queue: ";

int i = FRONT;

while (true) {

cout << Q[i] << " ";

if (i == REAR) break;

i = (i + 1) % MAX;

}

cout << endl;

}

}

int main() {

enqueue(10);

enqueue(20);

enqueue(30);

dequeue();

enqueue(40);

enqueue(50);

enqueue(60); // Penuh

display();

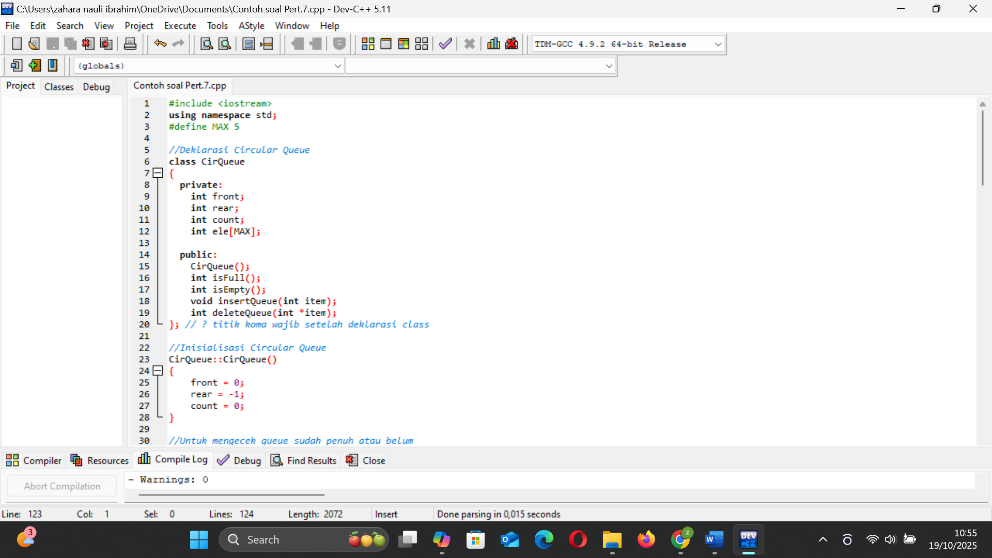
return 0;

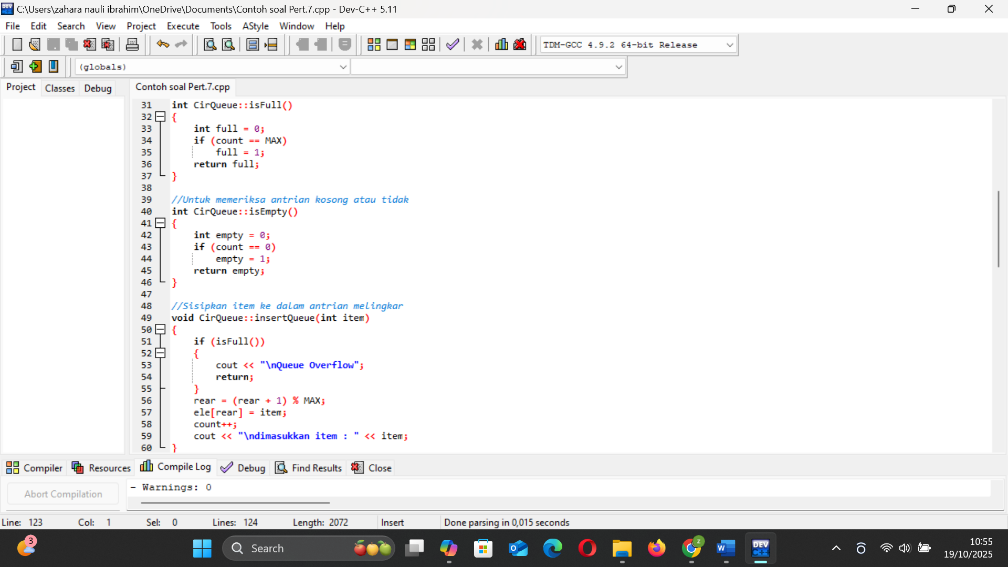
}

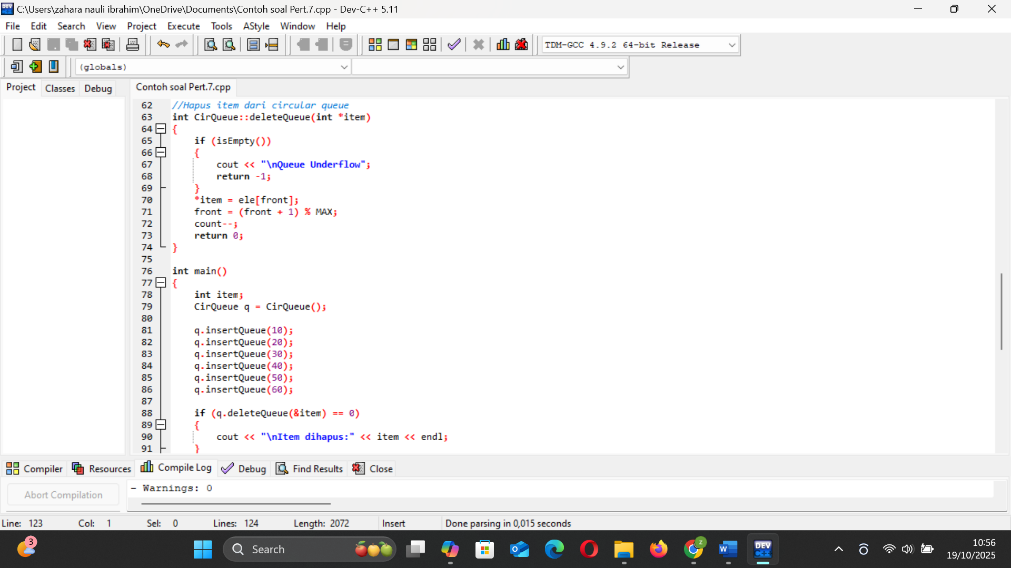
Program di atas menggambarkan implementasi Circular Queue menggunakan array statis berukuran 5 elemen.

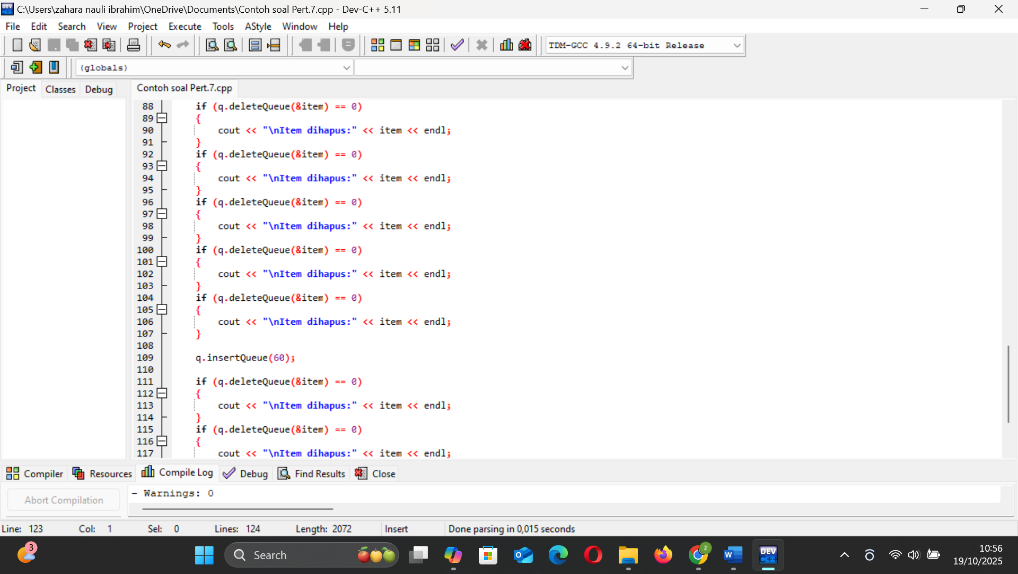
# LAPORAN PRAKTIKUM

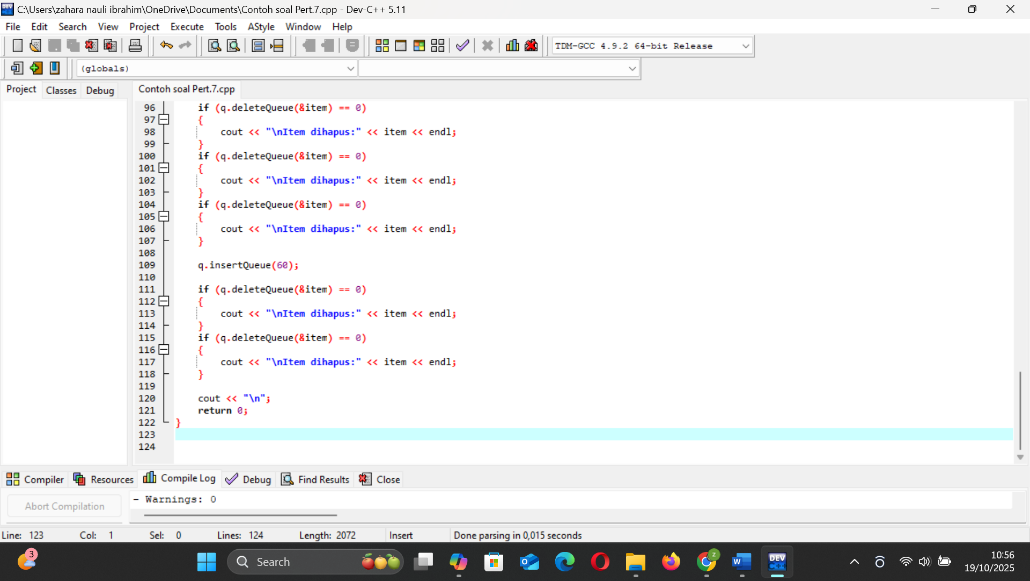
## 1.) Program untuk implementasi Circular Queue



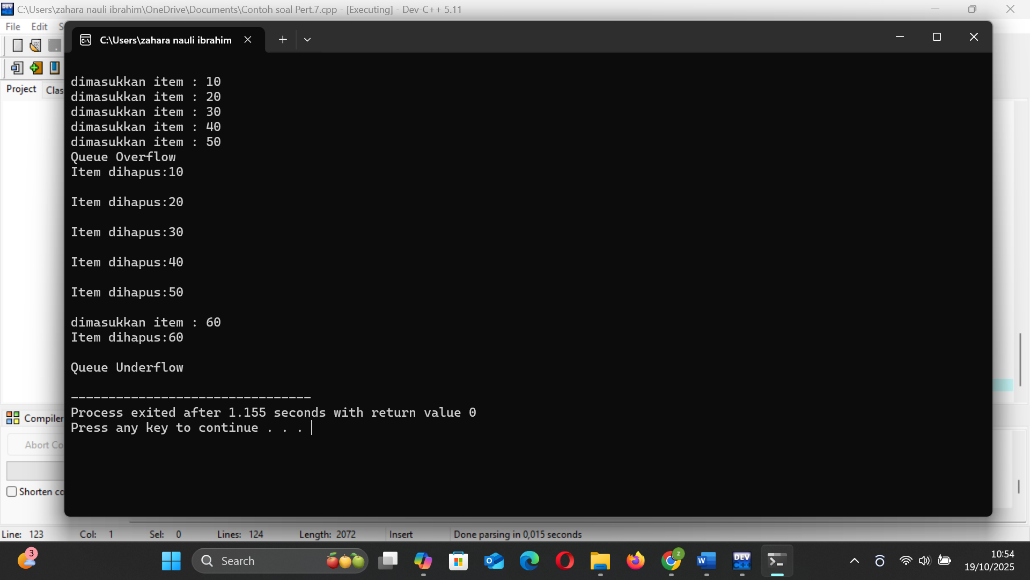








## Hasil Output



## LATIHAN 7

## 1) Sebutkan ciri circular Queue dalam kondisi:

## Kosong

## Penuh

## Bisa diisi

## Ada isinya

## Hanya berisi 10 record

## Tempat yang kosong hanya ada 10 tempat.

## Penjelasan

Misalkan:

* F = Front (penunjuk elemen depan)
* R = Rear (penunjuk elemen belakang)
* MAX = ukuran maksimum queue

|  |  |  |
| --- | --- | --- |
| **Kondisi** | **Ciri-Ciri** | **Penjelasan Sederhana** |
| **a.** Kosong | F = -1 dan R = -1 | Artinya belum ada data sama sekali. |
| **b.** Penuh | (R + 1) % MAX == F | Artinya elemen terakhir sudah diisi, dan posisi berikutnya kembali ke depan (penuh melingkar). |
| **c.** Bisa diisi | Queue tidak penuh -> (R + 1) % MAX != F | Masih ada ruang kosong untuk data baru. |
| **d.** Ada isinya | F != -1 | Berarti minimal ada satu data yang tersimpan. |
| **e.** Hanya berisi 10 record | Jumlah data = 10 | Bisa dihitung dari posisi F dan R. |
| **f.** Tempat kosong hanya ada 10 tempat | (MAX - jumlah data) = 10 | Sisa ruang kosong di queue tinggal 10 elemen. |

**Kesimpulan 1:**

Circular Queue seperti antrian yang ujungnya menyambung ke depan. Ketika penuh, Rear akan “mengejar” Front, dan ketika kosong, keduanya sama-sama -1.

## 2) Tulis Algoritma lengkap untuk:

### Algoritma INSERT Sebuah Record

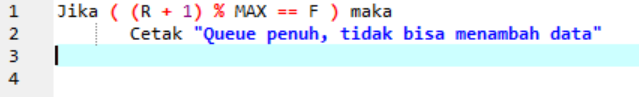
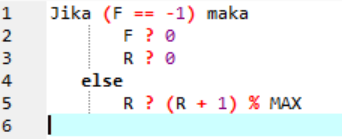
### Algoritma DELETE Sebuah Record

## Algoritma

1. Insert (Enqueue)

* Tujuan: Menambahkan data ke antrian.
* Algoritma:

Algoritma InsertCircularQueue

1. 
2. 
3. Isi Queue[R] dengan data baru
4. Selesai

**Penjelasan:**

* Langkah 1: Cek apakah penuh (tidak bisa diisi lagi).
* Langkah 2: Kalau queue kosong, set awal F = 0 dan R = 0.
* Langkah 3: Kalau tidak kosong, posisi R maju satu langkah secara melingkar.
* Langkah 4: Simpan data di posisi R.

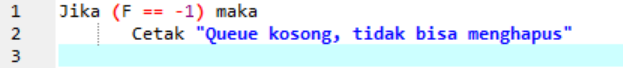
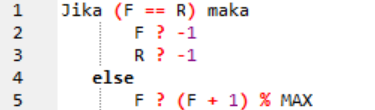
**Kesimpulan 2a:**

Insert di Circular Queue seperti memarkir mobil di antrian yang berputar — kalau penuh, kita berhenti; kalau masih ada ruang, kita isi di posisi berikutnya.

1. Delete (Dequeue

* Tujuan: Menghapus data dari antrian (dari depan).
* Algoritma:

Algoritma DeleteCircularQueue

1. 
2. Ambil data di Queue[F]
3. 
4. Selesai

**Penjelasan:**

* Langkah 1: Kalau kosong, tidak bisa dihapus.
* Langkah 2: Ambil data di depan (Front).
* Langkah 3: Kalau setelah dihapus queue jadi kosong (Front = Rear), set ulang ke -1.
* Langkah 4: Kalau tidak, geser Front ke depan secara melingkar.

**Kesimpulan 2b:**

Delete di Circular Queue seperti memanggil mobil di urutan terdepan keluar dari antrian.

## 3) Tulis Algoritma untuk mengisi antrian record per record sebanyak 10 record (Selama Belum Penuh)

## Penjelasan

**Algoritma:**

**Algoritma Isi10Record**

1. i ← 1
2. Ulangi selama (i ≤ 10) DAN (Queue belum penuh)
3. Baca data
4. Panggil InsertCircularQueue(data)
5. i ← i + 1
6. Jika Queue penuh sebelum 10 record terisi

* Cetak "Queue sudah penuh, pengisian dihentikan"

1. Selesai

**Penjelasan:**

* Langkah 1: Mulai dari record ke-1.
* Langkah 2: Selama belum penuh dan belum 10 kali, terus tambahkan data.
* Langkah 3: Kalau penuh di tengah, hentikan walau belum sampai 10.

**Kesimpulan 3:**

Algoritma ini memastikan pengisian berhenti otomatis ketika penuh — jadi tidak akan ada “kelebihan muatan”.

## 4) Tulis algoritma yang lengkap untuk mendelete isi antrian record per record sebanyak 10 record selama antrian masih ada isinya. Apabila antrian sudah kosong, walaupun belum mendelete sebanyak 10 record, maka proses delete dihentikan. (Selama Masih Ada Isi)

## Penjelasan

**Algoritma:**

**Algoritma Hapus10Record**

1. i ← 1
2. Ulangi selama (i ≤ 10) DAN (Queue belum kosong)
3. Panggil DeleteCircularQueue
4. i ← i + 1
5. Jika Queue kosong sebelum 10 record terhapus

* Cetak "Queue sudah kosong, penghapusan dihentikan"

1. Selesai

**Penjelasan:**

* Langkah 1: Mulai hitung dari record pertama yang akan dihapus.
* Langkah 2: Selama queue masih ada isinya, hapus satu per satu.
* Langkah 3: Kalau queue kosong duluan, hentikan.

**Kesimpulan 4:**

Proses delete ini seperti memanggil 10 mobil dari antrian keluar satu-satu, tapi kalau sudah habis, langsung berhenti.

## 5) Tulis untuk menghitung dan mencetak jumlah tempat(elemen) yang ada isinya bila diketahui nilai F dan R tanpa mengetahui nilai Counter.

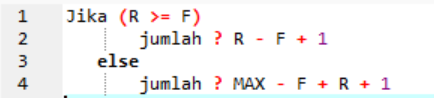
## Penjelasan

**Diketahui:** F dan R

**Ditanya:** Berapa jumlah elemen (isi Queue) sekarang?

**Algoritma:**

Algoritma HitungIsiQueue

1. 
2. 
3. Cetak "Jumlah elemen yang berisi =", jumlah
4. Selesai

**Penjelasan:**

* Jika kosong (F = -1), jumlahnya 0.
* Kalau posisi R lebih besar dari F, berarti belum melingkar.
* Kalau R lebih kecil dari F, berarti sudah melingkar, jadi hitung dari F ke akhir dan tambah dari awal ke R.
* Kesimpulan 5:
* Dengan hanya mengetahui Front dan Rear, kita masih bisa tahu berapa banyak data yang ada tanpa variabel penghitung.

**Kesimpulan Akhir :**

Circular Queue adalah bentuk antrian yang ujung belakangnya menyambung ke depan, sehingga lebih efisien dalam penggunaan ruang. Insert menambah data dari belakang, Delete menghapus dari depan. Kondisi penuh dan kosong bergantung pada posisi Front dan Rear. Proses pengisian dan penghapusan bisa diatur agar otomatis berhenti ketika sudah penuh atau kosong. Jumlah elemen bisa dihitung hanya dengan F dan R, tanpa perlu variabel tambahan.

**LAPORAN AKHIR**

****

Disusun oleh:

Nama : ZAHARA NAULI IBRAHIM

NIM : 241011402500

Kelas : 03TPLP023

**Program Studi Teknik Informatika**

**Fakultas Ilmu Komputer**

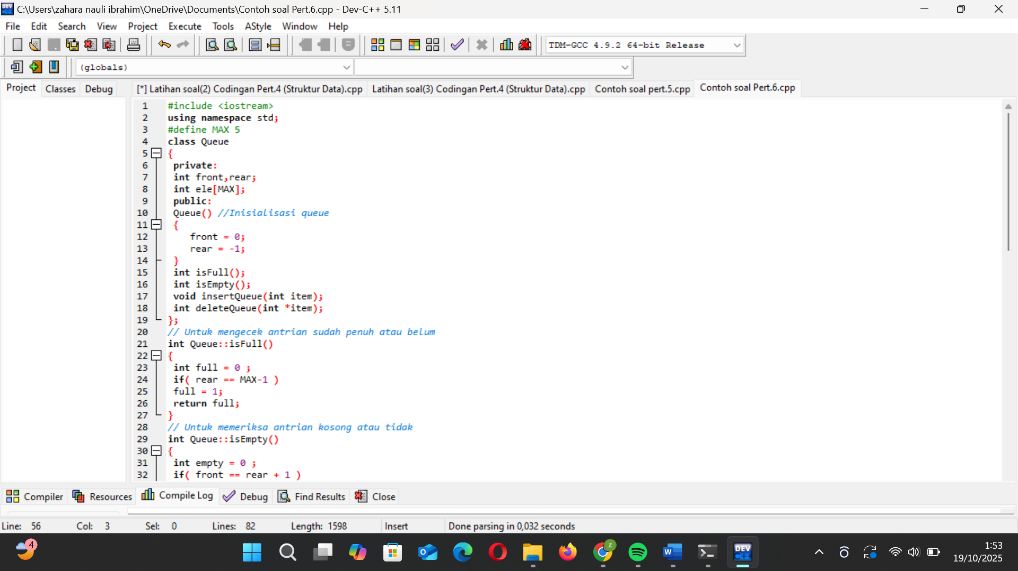
**Universitas Pamulang**

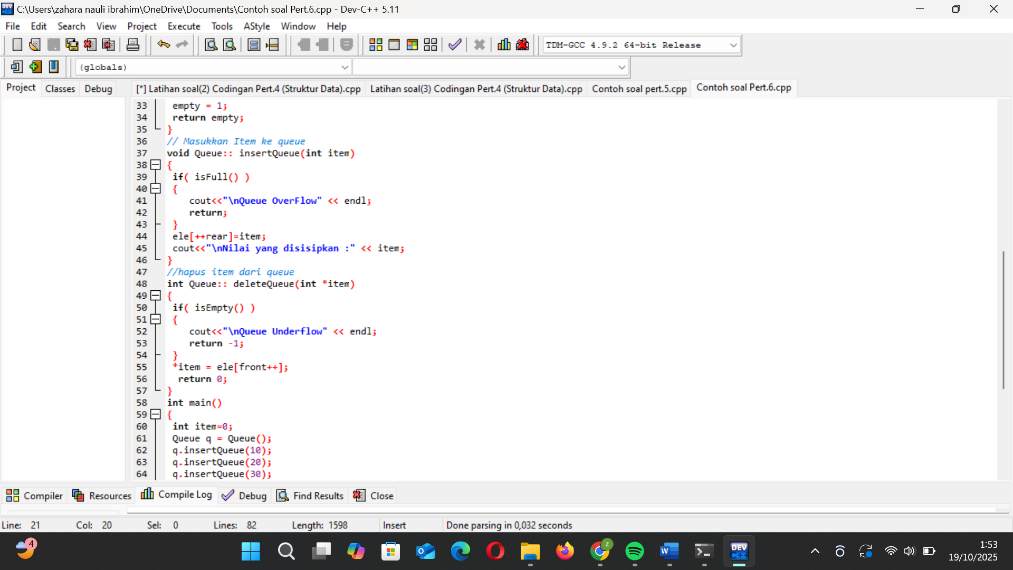
Jl. Surya Kencana No. 1 Pamulang Telp (021)7412566, Fax. (021)7412566

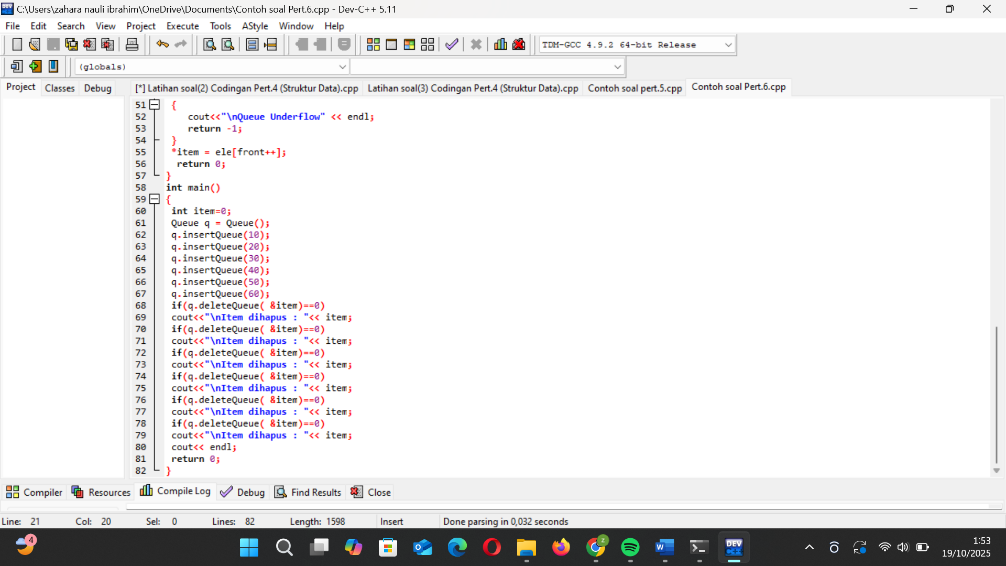
Tangerang Selatan – Banten

# TUGAS AKHIR

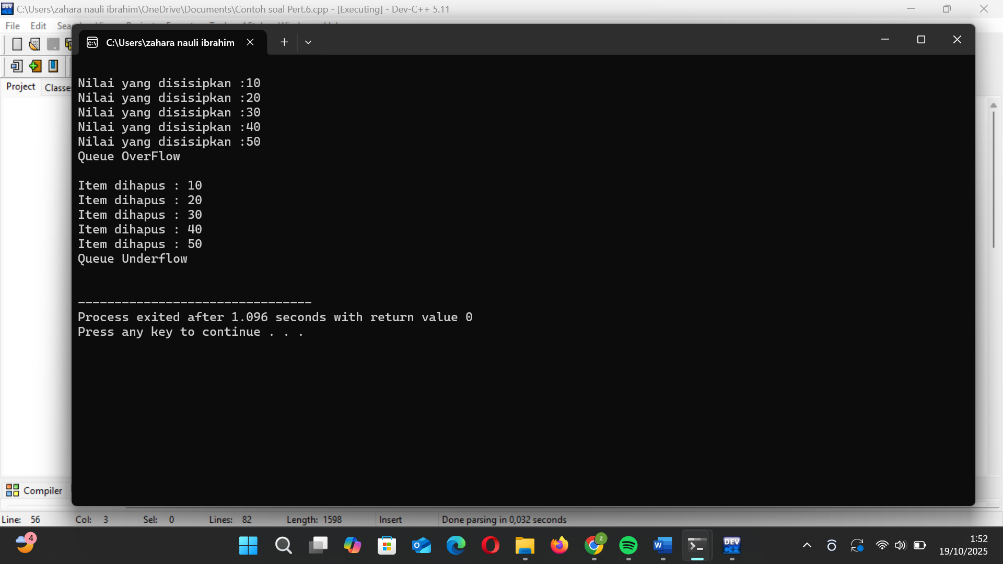
## 1) Program Implementasi Antrian Linear untuk operasi antrian







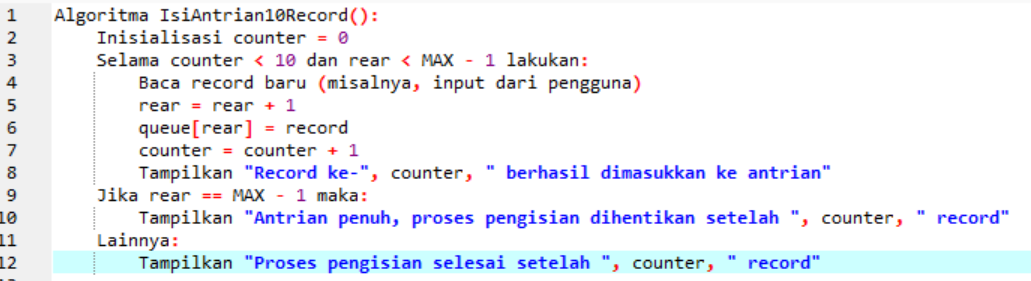
## Hasil Output



## LATIHAN 6

## 1) Tulis algoritma yang lengkap untuk mengisi antrian record per record sebanyak 10 record selama antrian belum penuh.

## Algoritma

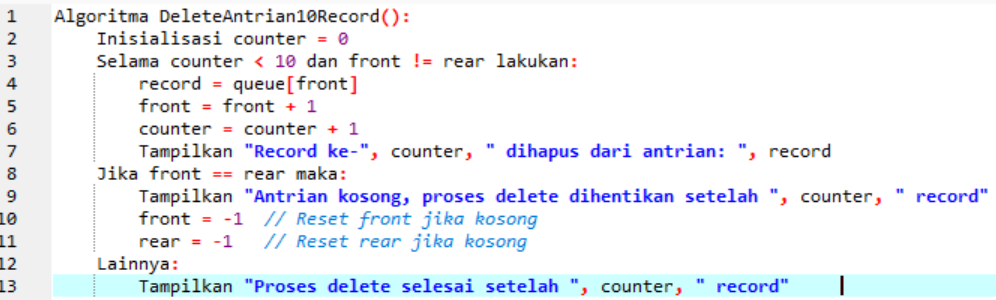


**Penjelasan**

* Inisialisasi counter = 0: Mengatur penghitung untuk melacak jumlah record yang telah dimasukkan (maksimal 10).
* Selama counter < 10 dan rear < MAX - 1 lakukan: Loop berulang selama belum mencapai 10 record dan antrian belum penuh (rear belum mencapai batas maksimal).
* Baca record baru (misalnya, input dari pengguna): Mengambil input record dari pengguna atau sumber lain.
* rear = rear + 1: Menambah pointer rear untuk menunjukkan posisi baru di array.
* queue[rear] = record: Menyimpan record di posisi rear yang baru.
* counter = counter + 1: Menambah penghitung record.
* Tampilkan "Record ke-", counter, " berhasil dimasukkan ke antrian": Memberikan konfirmasi untuk setiap record yang dimasukkan.
* Jika rear == MAX - 1 maka: Pengecekan jika antrian penuh setelah loop.
* Tampilkan "Antrian penuh, proses pengisian dihentikan setelah ", counter, " record": Pesan bahwa proses dihentikan karena penuh.
* Lainnya: Jika loop selesai tanpa penuh.
* Tampilkan "Proses pengisian selesai setelah ", counter, " record": Pesan akhir bahwa proses selesai.

## 2) Tulis Algoritma untuk Menghapus (Delete) Record per Record sebanyak 10 record

## Algoritma



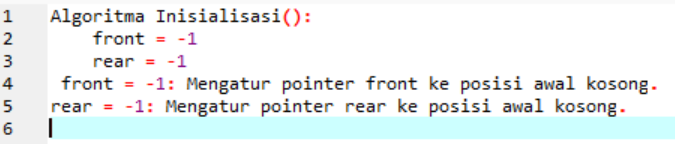
**Penjelasan**

* nisialisasi counter = 0: Mengatur penghitung untuk melacak jumlah record yang telah dihapus (maksimal 10).
* Selama counter < 10 dan front != rear lakukan: Loop berulang selama belum mencapai 10 record dan antrian tidak kosong (front belum menyamai rear).
* record = queue[front]: Mengambil record dari posisi front (elemen terdepan).
* front = front + 1: Menambah pointer front untuk menunjukkan posisi baru (elemen dihapus).
* counter = counter + 1: Menambah penghitung record.
* Tampilkan "Record ke-", counter, " dihapus dari antrian: ", record: Menampilkan record yang dihapus.
* Jika front == rear maka: Pengecekan jika antrian kosong setelah loop.
* Tampilkan "Antrian kosong, proses delete dihentikan setelah ", counter, " record": Pesan bahwa proses dihentikan karena kosong.
* front = -1 // Reset front jika kosong: Mengatur ulang front ke kondisi kosong.
* rear = -1 // Reset rear jika kosong: Mengatur ulang rear ke kondisi kosong.
* Lainnya: Jika loop selesai tanpa kosong.
* Tampilkan "Proses delete selesai setelah ", counter, " record": Pesan akhir bahwa proses selesai.

## 3) Tulis Algortima Dasar untuk Inisialisasi, Insert, Delete, Reset

## Algoritma

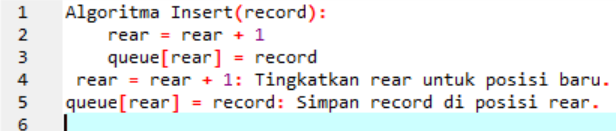
1. Inisialisasi



**Penjelasan**

* front = -1: Mengatur pointer front ke posisi awal kosong.
* rear = -1: Mengatur pointer rear ke posisi awal kosong.

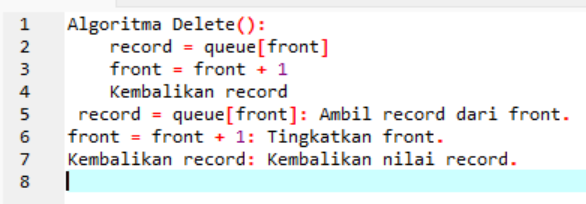
1. Insert sebuah record



**Penjelasan**

* rear = rear + 1: Tingkatkan rear untuk posisi baru.
* queue[rear] = record: Simpan record di posisi rear.

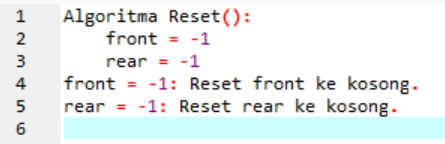
1. Delete sebuah record



**Penjelasan**

* record = queue[front]: Ambil record dari front.
* front = front + 1: Tingkatkan front.
* Kembalikan record: Kembalikan nilai record.

1. Reset



**Penjelasan**

* front = -1: Reset front ke kosong.
* rear = -1: Reset rear ke kosong.

## 4) Sebutkan ciri dari Linier Queue untuk kondisi:

## Kosong taka da isinya

## Penuh, tak bisa diisi

## Bisa diisi

## Ada isinya

## Antrian tak bisa diisi lagi, tapi belum ada isi antrian yang sudah keluar atau sudah dilayani

## Antrian perlu direset

## Penjelasan

Linear queue adalah struktur data FIFO (First In First Out) yang menggunakan array linier tanpa wrap-around. Kondisi didasarkan pada posisi front dan rear.

1. Kosong, tak ada isinya

Kondisi: front == -1 dan rear == -1 (atau front > rear jika menggunakan konvensi lain).

* Ciri: Antrian belum pernah diisi atau sudah dikosongkan sepenuhnya. Tidak ada elemen yang bisa dihapus.

1. Penuh, tak bisa diisi

Kondisi: rear == MAX - 1.

* Ciri: Array telah terisi penuh dari indeks 0 hingga MAX-1. Tidak bisa menambah elemen lagi tanpa menghapus.

1. Bisa diisi

Kondisi: rear < MAX - 1.

* Ciri: Masih ada ruang kosong di array untuk menambah elemen baru.

1. Ada isinya

Kondisi: front != -1 dan rear >= front.

* Ciri: Antrian memiliki setidaknya satu elemen yang bisa dihapus atau diproses.

1. Antrian tak bisa diisi lagi, tapi belum ada isi antrian yang sudah keluar atau sudah dilayani

Kondisi: rear == MAX - 1 dan front != -1.

* Ciri: Antrian penuh, tetapi elemen-elemen di depan (front) belum dihapus, sehingga ruang tidak bisa digunakan kembali (karakteristik linear queue tanpa circular).

1. Antrian perlu direset

Kondisi: rear == MAX - 1 dan front > 0 (atau setelah beberapa delete, front tidak direset otomatis).

* Ciri: Antrian penuh dan beberapa elemen di depan sudah dihapus, tetapi ruang kosong di depan tidak bisa digunakan. Reset diperlukan untuk mengembalikan ke kondisi awal kosong.

## 5) Jika n = 100, maka untuk: a.) Bila F = 15, dan R = 37, maka jumlah pengantri yang belum dilayani = ……. b.) Bila F = 15, dan R = 37, maka jumlah kolom yang masih bisa diisi = ……..

## Penjelasan

Jika n = 100, maka untuk:

**Asumsikan array indeks 0 hingga 99, front = F, rear = R. Jumlah elemen dalam antrian = R - F + 1 (karena linear queue, asumsikan R >= F).**

1. Bila F = 15, dan R = 37, maka jumlah pengantri yang belum dilayani = 37 - 15 + 1 = 23: Rumus jumlah elemen = R - F + 1, karena indeks dari F hingga R inklusif.
2. Bila F = 15, dan R = 37, maka jumlah kolom yang masih bisa diisi = 100 - 23 = 77: Rumus ruang kosong = n - jumlah elemen.

**Kesimpulan Akhir:**

Linear queue adalah struktur data sederhana dan efisien untuk mengelola antrian FIFO, cocok untuk aplikasi seperti sistem antrian pelanggan atau pemrosesan tugas berurutan. Operasi dasar seperti insert dan delete memerlukan pengecekan kondisi penuh/kosong untuk mencegah error, seperti pada algoritma lengkap. Algoritma pengisian dan penghapusan berulang menunjukkan cara mengelola antrian dalam batch, sedangkan ciri-ciri kondisi membantu memahami status antrian. Namun, linear queue memiliki keterbatasan karena tidak bisa menggunakan kembali ruang kosong di depan setelah delete, sehingga sering digantikan dengan circular queue untuk efisiensi memori. Jika diimplementasikan dalam kode, pastikan inisialisasi pointer benar untuk menghindari bug seperti underflow atau overflow.

# 

# Hasil dan Pembahasan

## LATIHAN 6

## Soal 1. Mengisi Antrian Record per Record Sebelum Penuh

Untuk menerapkan proses pengisian data ke dalam antrian sampai penuh secara otomatis.

### Pembahasan:

* Program mendeklarasikan array Queue[10] untuk menampung maksimal 10 record.
* Posisi awal front = 0 dan rear = -1 menandakan antrian kosong.
* Pengguna diminta memasukkan data satu per satu.
* Setiap data disimpan di posisi rear + 1, kemudian variabel jumlah bertambah.
* Ketika rear == 9, program menampilkan pesan bahwa antrian sudah penuh dan berhenti otomatis.
* Setelah pengisian selesai, seluruh data dalam antrian ditampilkan.

### Makna Algoritma:

Simulasi logika enqueue (penambahan data) dalam antrian linear sampai mencapai batas maksimal.

### Kesimpulan:

Program menggambarkan bagaimana data dimasukkan ke antrian hingga penuh, menunjukkan kondisi overflow jika kapasitas sudah habis.

## Soal 2. Operasi Dasar Queue (Inisialisasi, Insert, Delete, Reset)

Untuk memahami empat operasi utama dalam antrian linear: inisialisasi, insert, delete, dan reset.

### Pembahasan:

1. Inisialisasi:

* front = 0 dan rear = -1, artinya antrian kosong

2. Insert (Enqueue):

* Jika rear == 9, antrian penuh → tidak bisa menambah data.
* Jika belum penuh, rear bertambah satu dan data baru dimasukkan.

3. Delete (Dequeue):

* Jika front > rear, artinya antrian kosong.
* Jika ada isi, data di posisi front dihapus dan front bergeser ke depan.

4. Reset:

* front dikembalikan ke 0, rear dikembalikan ke -1, menandakan semua data dihapus

### Makna Deklarasi:

Variabel Queue[10] menampung data, sedangkan front dan rear digunakan sebagai penunjuk posisi awal dan akhir antrian.

### Kesimpulan:

Program menjelaskan seluruh siklus hidup data dalam antrian linear mulai dari kosong → isi → hapus → kosong kembali.

## Soal 3. Ciri-ciri Kondisi Linear Queue

Untuk mengetahui dan mengidentifikasi kondisi antrian berdasarkan nilai front dan rear.

### Pembahasan:

Program meminta input nilai front dan rear, kemudian menentukan status antrian berdasarkan kombinasi nilainya:

1. KOSONG: rear == -1 atau front > rear
2. PENUH: rear == 9 && front == 0
3. BISA DIISI: front == 0 && rear < 9
4. ADA ISINYA: front > 0 && rear < 9
5. TAK BISA DIISI LAGI (PERLU RESET): rear == 9 && front > 0

Fungsi Algoritma:

Menentukan kondisi logis dari antrian berdasarkan posisi pointer front dan rear.

### Kesimpulan:

Program membantu memahami kapan antrian kosong, penuh, atau perlu direset sesuai nilai posisi indeks.Hal ini penting se belum membuat sistem antrian dinamis.

## Soal 4 (Pengulangan Ciri Kondisi Queue / Penguatan Konsep)

Untuk memberikan latihan penguatan pada pengenalan kondisi queue dengan logika yang sama seperti Soal 3, agar mahasiswa benar-benar memahami perbedaan setiap kondisi antrian.

### Pembahasan:

* Variabel front dan rear tetap menjadi indikator posisi awal dan akhir data.
* Output menjelaskan kondisi queue yang valid dan menampilkan ciri-cirinya.
* Melatih pemahaman konsep overflow (penuh) dan underflow (kosong).

### Kesimpulan:

Soal ini memperkuat pemahaman visual dan logika tentang perubahan kondisi queue akibat proses insert dan delete.

## Soal 5. Menghitung Jumlah Pengantri dan Sisa Kolom Kosong

Untuk melatih kemampuan menghitung jumlah elemen aktif dalam antrian dan sisa kapasitas kosong.

### Pembahasan:

* Diketahui kapasitas antrian n = 100.
* Program meminta input front dan rear.
* Jika rear < front, berarti input tidak valid.

Dua perhitungan utama dilakukan:

1. Jumlah pengantri:

Rumus → (REAR - FRONT + 1)

Menunjukkan jumlah elemen yang masih ada di antrian.

1. Sisa kolom kosong:

Rumus → (n - REAR - 1)

Menunjukkan jumlah tempat yang masih bisa diisi.

### Fungsi Deklarasi:

front = posisi data paling depan.

rear = posisi data terakhir masuk.

n = kapasitas maksimal antrian.

### Kesimpulan:

Program memperkenalkan perhitungan matematis dalam queue dan membantu memahami hubungan antara jumlah pengantri, posisi pointer, dan kapasitas.

## Kesimpulan

1. Linear Queue bekerja dengan prinsip First In, First Out (FIFO).
2. Posisi front dan rear menentukan kondisi antrian apakah kosong, penuh, atau masih bisa diisi.
3. Operasi dasar (inisialisasi, insert, delete, reset) adalah pondasi dari sistem antrian dinamis.
4. Pemahaman konsep antrian linear membantu dalam implementasi queue pada struktur data kompleks, seperti circular queue dan priority queue.