**LAPORAN AWAL STRUKTUR DATA**

**Pertemuan ke-9**

**Linear Singly Linked List**

****

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, struktur data Linked List merupakan salah satu bentuk penyimpanan data dinamis yang sangat penting.

Tidak seperti array yang bersifat statis (ukuran tetap), Linked List mampu menambah atau menghapus elemen secara fleksibel selama program berjalan.

Linked List tersusun dari sekumpulan node yang saling terhubung melalui pointer. Setiap node berisi data dan pointer yang menunjuk ke node berikutnya.

Hubungan antar-node inilah yang membentuk suatu rantai data (list).

## 1. Pengertian Linear Singly Linked List

Linear Singly Linked List (Daftar Berantai Tunggal) adalah jenis Linked List di mana setiap node hanya memiliki satu pointer yang menunjuk ke node berikutnya,

dan node terakhir menunjuk ke NULL. Struktur ini bersifat linear satu arah, artinya penelusuran hanya dapat dilakukan dari node pertama menuju node terakhir.

Setiap node terdiri atas dua bagian utama:

* Data Field → berisi nilai atau informasi yang disimpan.
* Pointer Field (Next) → berisi alamat node berikutnya.

## 2. Karakteristik Linear Singly Linked List

* Terdiri dari rangkaian node yang saling terhubung secara searah.
* Node pertama disebut HEAD, sedangkan node terakhir menunjuk ke NULL.
* Tidak memiliki indeks seperti array; untuk mengakses data, harus dilakukan traversal dari HEAD.
* Ukurannya dinamis, dapat bertambah atau berkurang saat program berjalan.
* Operasi dasar meliputi: insert, delete, search, dan display.

## 3. Kelebihan dan Kekurangan

* Kelebihan:
* Memori digunakan secara efisien karena dialokasikan hanya saat dibutuhkan.
* Mudah menambah atau menghapus elemen di tengah list tanpa perlu menggeser elemen lain.
* Ukuran list tidak dibatasi selama memori masih tersedia.
* Kekurangan:
* Tidak dapat diakses secara langsung (harus traversal dari awal).
* Membutuhkan ruang tambahan untuk menyimpan pointer.
* Operasi pencarian relatif lebih lambat dibanding array.

## 4. Struktur dan Ilustrasi

HEAD → [Data|Next] → [Data|Next] → [Data|NULL]

## 5. Operasi Dasar pada Singly Linked List

* Inisialisasi: HEAD = NULL.
* Penyisipan (Insert): dapat dilakukan di awal, di tengah, atau di akhir.
* Penghapusan (Delete): menghapus node tertentu dengan memperbarui pointer node sebelumnya.
* Penelusuran (Traversal): menampilkan isi list dengan melintasi setiap node mulai dari HEAD hingga NULL.
* Pencarian (Search): mencari data tertentu secara berurutan.

## 6. Algoritma Dasar

ALGORITMA INISIALISASI\_LIST

HEAD ← NULL

ALGORITMA INSERT\_AWAL

Alokasi node baru

Isi data ke node baru

next(baru) ← HEAD

HEAD ← baru

ALGORITMA DELETE\_AWAL

Jika HEAD = NULL maka tampilkan "List kosong."

else

hapus ← HEAD

HEAD ← next(HEAD)

dealokasi(hapus)

## 7. Contoh Program C++

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* head = NULL;

void insertAwal(int nilai) {

Node\* baru = new Node();

baru->data = nilai;

baru->next = head;

head = baru;

}

void tampilkanList() {

Node\* temp = head;

while (temp != NULL) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL" << endl;

}

int main() {

insertAwal(10);

insertAwal(20);

insertAwal(30);

cout << "Isi Linked List: ";

tampilkanList();

return 0;

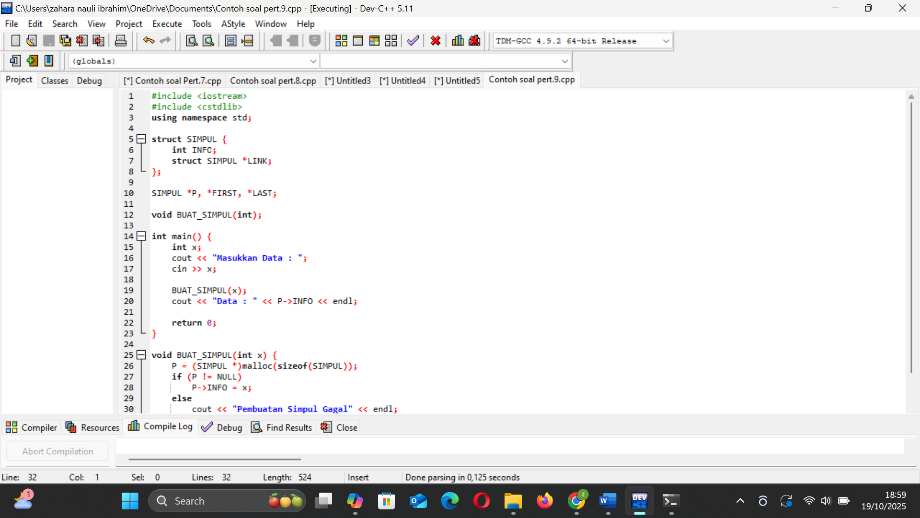
}

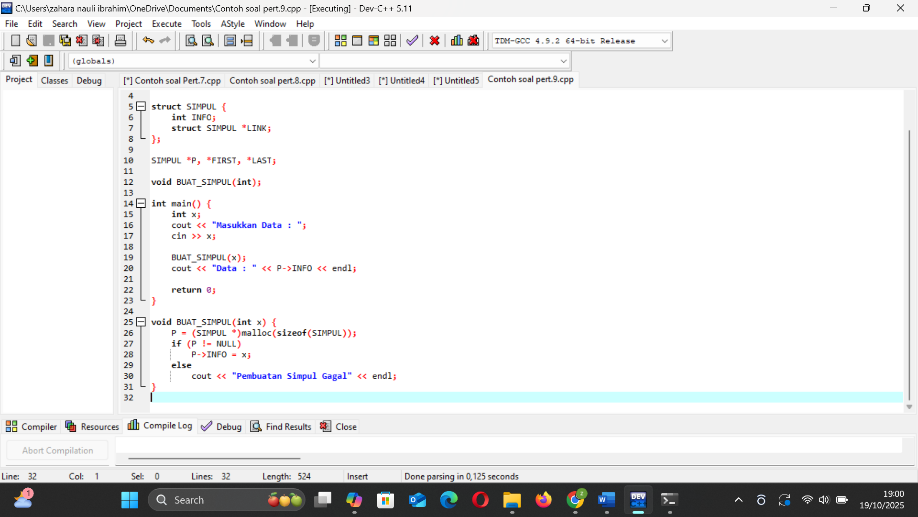
Hasil Output:

Isi Linked List: 30 -> 20 -> 10 -> NULL

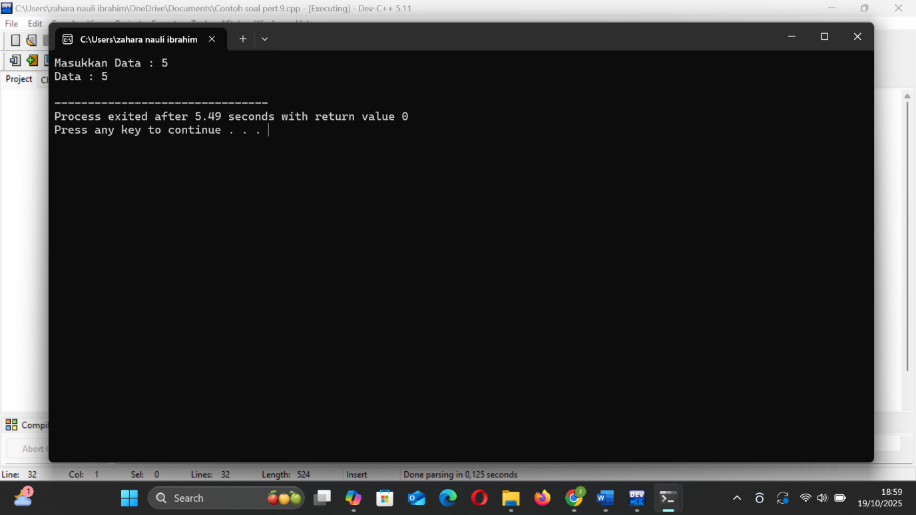
# LAPORAN PRAKTIKUM

# **1) Program Implementasi DeQueue**



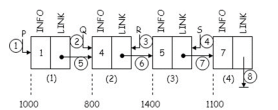


Hasil Output

****

LATIHAN

1) **Perhatikan penggalan linked List seperti gambar berikut:**

****

1. **Sebutkan nama dan isi tiap-tiap pointer**
2. **Sebutkan pointer-pointer yang bernilai sama**
3. **Sebutkan TRUE atau FALSE kondisi pada intruksi tiap pernyataan dibawah ini :**
4. **if(P->LINK==R)**
5. **if(Q->LINK==R->LINK)**
6. **if(Q->LINK->LINK==S->LINK)**
7. **if(Q==R) v. if(Q-LINK==R)**
8. **if(R->LINK->INFO==5)**
9. **if(Q->INFO==4)**

Penjelasan

1. Sebutkan nama dan isi tiap-tiap pointer

* Di gambar ada pointer variabel P, Q, R, S dan juga field LINK di tiap node. Isi dan arti tiap pointer:
* P → menunjuk ke node di alamat 1000, yang INFO = 1.

Penjelasan sederhana: P menunjuk ke simpul pertama (isi = 1).

* Q → menunjuk ke node di alamat 800, yang INFO = 4.

Penjelasan: Q menunjuk ke simpul kedua (isi = 4).

* R → menunjuk ke node di alamat 1400, yang INFO = 5.

Penjelasan: R menunjuk ke simpul ketiga (isi = 5).

* S → menunjuk ke node di alamat 1100, yang INFO = 7.

Penjelasan: S menunjuk ke simpul keempat (isi = 7).

* Selain itu, isi field LINK pada tiap node (nilai alamat yang disimpan di field LINK):
* Node di alamat 1000 (INFO=1) → LINK = 800.
* Node di alamat 800 (INFO=4) → LINK = 1400.
* Node di alamat 1400 (INFO=5) → LINK = 1100.
* Node di alamat 1100 (INFO=7) → LINK = NULL (akhir list).

Kesimpulan 1.a:

Tiap pointer (P,Q,R,S) menunjuk ke simpul berbeda: P→1000(1), Q→800(4), R→1400(5), S→1100(7). LINK tiap node menyimpan alamat node berikutnya seperti daftar di atas.

1. Pointer-pointer yang bernilai sama

Yang dimaksud di sini adalah: apakah ada pointer variabel yang memiliki nilai sama dengan nilai di field LINK suatu node?

* Dari gambarnya terlihat hubungan ini:
* Q == nilai LINK pada node alamat 1000.

(Karena node 1000.LINK = 800 dan Q = 800.)

* R == nilai LINK pada node alamat 800.

(Karena node 800.LINK = 1400 dan R = 1400.)

* S == nilai LINK pada node alamat 1400.

(Karena node 1400.LINK = 1100 dan S = 1100.)

* NULL (nilai link node terakhir) tidak sama dengan P, Q, R, atau S di gambar.

Penjelasan sederhana: pointer Q, R, S masing-masing sama dengan isi field LINK dari simpul sebelumnya. P tidak sama dengan field LINK mana pun di gambar.

Kesimpulan 1.b:

pasangan sama-nilainya:

* Q == (node 1000).LINK
* R == (node 800).LINK
* S == (node 1400).LINK

1. Tentukan TRUE / FALSE untuk tiap instruksi
2. if (P->LINK == R)

* P->LINK = 800 (link dari node 1000)
* R = 1400
* → FALSE (800 ≠ 1400).

Penjelasan sederhana: link dari node yang ditunjuk P menunjuk ke 800, bukan 1400.

1. if (Q->LINK == R->LINK)

* Q->LINK = link dari node 800 = 1400
* R->LINK = link dari node 1400 = 1100
* → FALSE (1400 ≠ 1100).

Penjelasan: link setelah Q adalah 1400, link setelah R adalah 1100

beda.

1. if (Q->LINK->LINK == S->LINK)

* Q->LINK = 1400 → Q->LINK->LINK = link dari node 1400 = 1100
* S->LINK = link dari node 1100 = NULL
* → FALSE (1100 ≠ NULL).

Penjelasan: dua langkah dari Q membawa ke alamat 1100; S->LINK adalah NULL tidak sama.

1. if (Q == R)

* Q = 800, R = 1400
* → FALSE.

Penjelasan: Q dan R menunjuk simpul berbeda.

1. if (Q->LINK == R)

* Q->LINK = 1400, R = 1400
* → TRUE.

Penjelasan sederhana: link di simpul yang ditunjuk Q memang menunjuk ke simpul yang ditunjuk R.

1. if (R->LINK->INFO == 5)

* R->LINK = link dari node 1400 = 1100 → node 1100.INFO = 7
* → FALSE (7 ≠ 5).

Penjelasan: satu langkah dari R → node 1100 yang punya INFO = 7, bukan 5.

1. if (Q->INFO == 4)

* Q menunjuk node 800, dan node 800.INFO = 4
* → TRUE.

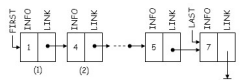
Penjelasan sederhana: Q memang menunjuk simpul yang isinya 4.

* Ringkasan TRUE / FALSE:
* i = FALSE
* ii = FALSE
* iii = FALSE
* iv = FALSE
* v = TRUE
* vi = FALSE
* vii = TRUE

Kesimpulan keseluruhan untuk Soal 1 :

* Dari gambar kita bisa membaca dua jenis nilai: alamat node (1000,800,1400,1100) dan isi INFO setiap node (1,4,5,7).
* Pointer variabel (P,Q,R,S) menunjuk ke alamat masing-masing node: P→1000(1), Q→800(4), R→1400(5), S→1100(7).
* Field LINK menyimpan alamat node berikutnya: 1000→800, 800→1400, 1400→1100, 1100→NULL.
* Dengan informasi itu kita dapat mengevaluasi setiap ekspresi pointer jadi TRUE atau FALSE sesuai perhitungan alamat/INFO di atas.

2) **Sudah ada linked list yang diilustrasikan seperti gambar dibwah ini, Simpul pertama ditunjuk oleh pointer FIRST, dan simpul terakhir ditunjuk oleh pointer LAST. Jumlah simpul tepatnya tidak diketahui, tapi dipastikan lebih dari 10 simpul. LINK dari simpul terakhir nilainya = NULL**



**Susun algoritma untuk menempatkan Pointer Q sehingga menunjuk:**

1. **Simpul no (1)**
2. **Simpul no (7)**
3. **Simpul akhir**
4. **Simpul dengan nilai INFO = 50**
5. **Simpul yang letaknya satu simpul didepan (disebelah kiri) simpul dengan nilai INFO = 50**

**Penjelasan**

1. Menempatkan pointer Q ke simpul no (1) **Algoritma:**

* Q ← FIRST

**Penjelasan sederhana:**

* Karena simpul pertama di list ditunjuk oleh pointer FIRST, maka supaya Q menunjuk simpul pertama, cukup salin alamat dari FIRST ke Q.

**Kesimpulan:**

Q sekarang menunjuk ke simpul pertama (INFO = 1).

1. Menempatkan pointer Q ke simpul no (7)

**Algoritma:**

* Q ← FIRST

while (Q->INFO ≠ 7) do

Q ← Q->LINK

end while

**Penjelasan sederhana:**

* Mulai dari simpul pertama (FIRST).
* Cek apakah INFO-nya = 7.
* Kalau belum, pindah ke simpul berikutnya lewat LINK.
* Berhenti saat menemukan simpul dengan INFO = 7.

**Kesimpulan:**

Q berhenti pada simpul yang INFO = 7.

1. Menempatkan pointer Q ke simpul akhir

**Algoritma:**

* Q ← FIRST

while (Q->LINK ≠ NULL) do

Q ← Q->LINK

end while

**Penjelasan sederhana:**

* Mulai dari simpul pertama.
* Bergerak terus ke simpul berikutnya selama masih ada LINK (tidak NULL).
* Jika LINK = NULL, berarti simpul itu adalah simpul terakhir.

**Kesimpulan:**

Q menunjuk simpul terakhir pada linked list.

1. Menempatkan pointer Q ke simpul dengan nilai INFO = 50

**Algoritma:**

* Q ← FIRST

while (Q ≠ NULL) and (Q->INFO ≠ 50) do

Q ← Q->LINK

end while

**Penjelasan sederhana:**

* Mulai dari simpul pertama.
* Bergerak ke simpul berikutnya sampai menemukan yang INFO = 50.
* Jika sampai Q = NULL, berarti simpul dengan nilai 50 tidak ditemukan.

**Kesimpulan:**

Q menunjuk simpul yang INFO = 50 jika ditemukan; jika tidak ada, Q = NULL.

1. Menempatkan pointer Q ke simpul di depan simpul dengan INFO = 50

**Algoritma:**

* Q ← FIRST

while (Q->LINK ≠ NULL) and (Q->LINK->INFO ≠ 50) do

Q ← Q->LINK

end while

**Penjelasan sederhana:**

* Mulai dari simpul pertama.
* Periksa apakah simpul berikutnya (Q->LINK) berisi INFO = 50.
* Jika belum, lanjut ke simpul berikutnya.
* Jika ya, berhenti — karena simpul saat ini (Q) adalah simpul di depan simpul dengan INFO = 50.

**Kesimpulan:**

Q akan menunjuk simpul yang letaknya satu sebelum simpul dengan INFO = 50.

3) **Sudah ada Linked List seperti no 2. Diatas.**

1. **Menghitung dan mencetak jumlah simpul**
2. **Menghitung dan mencetak TotalINFO (25+12+…….+27,14)**
3. **Mencetak semua nilai INFO ke layar**
4. **Mencetak jumlah simpul yang nilai INFOnya = 50**

Algoritma

1. Menghitung dan mencetak jumlah simpul

Algoritma:

* Q ← FIRST

Count ← 0

while (Q ≠ NULL) do

Count ← Count + 1

Q ← Q->LINK

end while

print(Count)

Penjelasan sederhana:

* Mulai dari simpul pertama.
* Hitung setiap simpul yang dilewati.
* Berhenti jika sudah sampai akhir (LINK = NULL).
* Cetak hasilnya.

Kesimpulan:

Menampilkan jumlah total simpul dalam linked list.

1. Menghitung dan mencetak TotalINFO

Algoritma:

* Q ← FIRST

Total ← 0

while (Q ≠ NULL) do

Total ← Total + Q->INFO

Q ← Q->LINK

end while

print(Total)

Penjelasan sederhana:

* Mulai dari simpul pertama.
* Tambahkan nilai INFO setiap simpul ke variabel Total.
* Berhenti di simpul terakhir.
* Cetak jumlah total nilai INFO semua simpul.

Kesimpulan:

Menampilkan jumlah keseluruhan nilai INFO dalam list (contoh: 25 + 12 + … + 27 + 14).

1. Mencetak semua nilai INFO ke layar

Algoritma:

* Q ← FIRST

while (Q ≠ NULL) do

print(Q->INFO)

Q ← Q->LINK

end while

Penjelasan sederhana:

* Mulai dari simpul pertama.
* Cetak nilai INFO.
* Lanjutkan ke simpul berikutnya sampai akhir.

Kesimpulan:

Semua nilai INFO dari setiap simpul ditampilkan berurutan di layar.

1. Mencetak jumlah simpul dengan nilai INFO = 50

Algoritma:

* Q ← FIRST

Count ← 0

while (Q ≠ NULL) do

if (Q->INFO = 50) then

Count ← Count + 1

end if

Q ← Q->LINK

end while

print(Count)

Penjelasan sederhana:

* Mulai dari simpul pertama.
* Setiap kali menemukan INFO = 50, tambahkan hitungan Count.
* Lanjutkan hingga akhir list.
* Cetak hasilnya.

Kesimpulan:

Menampilkan berapa banyak simpul yang berisi nilai INFO = 50.

**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

LATIHAN 8

1) **Tulis algoritma dasar double ended queue untuk kondisi: a. Inisialisasi b. Insert sebuah record dari kanan c. Insert sebuah record dari kiri d. Delete sebuah record dari kanan e. Delete sebuah record dari kiri**

Double Ended Queue (Deque) adalah antrian yang bisa menambah (insert) atau menghapus (delete) data dari dua sisi: kiri (depan) dan kanan (belakang).

Penjelasan

Gunakan simbol:

* F → Front (depan)
* R → Rear (belakang)
* MAX → ukuran maksimum deque
* DEQUE[MAX] → array tempat data disimpan

1. Inisialisasi

* Tujuan: Mengatur kondisi awal deque agar siap digunakan.
* Algoritma: Algoritma InisialisasiDeque
* F ← -1
* R ← -1
* Selesai

Penjelasan:

* F dan R diisi -1 artinya deque kosong.
* Belum ada data, jadi posisi awal belum ditentukan.

Kesimpulan 1a:

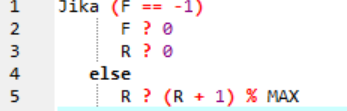
Langkah ini seperti “mengosongkan antrian” agar siap diisi nanti.

1. Insert dari Kanan (Rear)

* Algoritma: Algoritma InsertKanan(data)

1. Jika ((R + 1) % MAX == F)

* Cetak "Deque penuh, tidak bisa insert dari kanan"

1. 
2. DEQUE[R] ← data
3. Selesai

Penjelasan:

* Cek apakah deque penuh.
* Kalau masih kosong, posisi awal diisi F = R = 0.
* Kalau tidak, R maju satu posisi melingkar.
* Data baru disimpan di R.

Kesimpulan 1b:

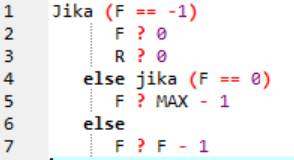
Menambah dari kanan artinya memasukkan data di “belakang” antrian.

1. Insert dari Kiri (Front)

* Algoritma: Algoritma InsertKiri(data)

1. Jika ((F == 0 dan R == MAX - 1) atau (F == R + 1))

* Cetak "Deque penuh, tidak bisa insert dari kiri"

1. 
2. DEQUE[F] ← data
3. Selesai

Penjelasan:

* Kalau penuh, tidak bisa insert.
* Kalau kosong, posisi awal diisi F = R = 0.
* Kalau F di posisi awal (0), maka mundur ke akhir (melilit).
* Data disimpan di posisi F.

Kesimpulan 1c:

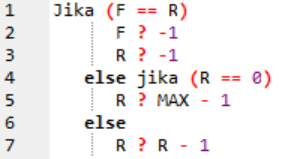
Menambah dari kiri seperti menambahkan orang di barisan paling depan.

1. Delete dari Kanan

* Algoritma: Algoritma DeleteKanan

1. Jika (F == -1)

* Cetak "Deque kosong, tidak bisa delete dari kanan"

1. Ambil data ← DEQUE[R]
2. 
3. Selesai

Penjelasan:

* Kalau kosong, tidak bisa hapus.
* Kalau hanya satu data, setelah dihapus jadi kosong.
* Kalau tidak, R mundur satu posisi melingkar.

Kesimpulan 1d:

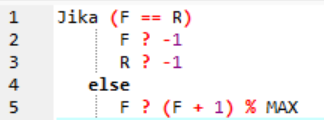
Delete kanan artinya menghapus data paling belakang.

1. Delete dari Kiri

* Algoritma: Algoritma DeleteKiri

1. Jika (F == -1)

* Cetak "Deque kosong, tidak bisa delete dari kiri"

1. Ambil data ← DEQUE[F]
2. 
3. Selesai

Penjelasan:

* Kalau kosong, tidak bisa dihapus.
* Kalau hanya satu data, setelah dihapus jadi kosong.
* Kalau lebih dari satu, F maju satu langkah ke depan.

Kesimpulan 1e:

Delete kiri artinya menghapus orang paling depan dalam antrian.

2) **Sebutkan ciri bahwa double ended queue pada: a. Kosong tidak ada isinya b. Penuh kanan, tak bisa diisi dari kanan c. Penuh kiri, tidak bisa diisi dari kiri d. Penuh total, tidak bisa diisi dari kiri maupun dari kanan e. Hanya diisi 10 pengantri.**

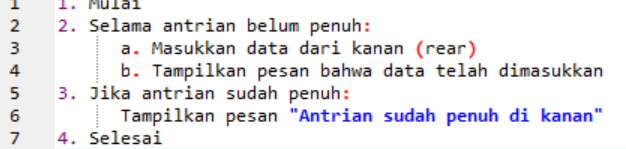
|  |  |  |
| --- | --- | --- |
| **Kondisi** | **Ciri-Ciri** | **Penjelasan Sederhana** |
| **a.** Kosong | F = -1 dan R = -1 | Tidak ada data sama sekali. |
| **b.** Penuh Kanan | (R + 1) % MAX == F | Tidak bisa lagi insert dari kanan. |
| **c.** Penuh Kiri | (F == 0 dan R == MAX - 1) atau (F == R + 1) | Tidak bisa lagi insert dari kiri. |
| **d.** Penuh Total | (R + 1) % MAX == F | Kanan dan kiri dua-duanya penuh. |
| **e.** Berisi 10 Pengantri | Jumlah elemen = 10 | Bisa dihitung dari posisi F dan R. |
|  |  |  |

Kesimpulan 2:

Kondisi deque ditentukan oleh hubungan antara Front dan Rear, apakah mereka berdekatan (penuh) atau sama-sama -1 (kosong).

3) Tulis algoritma yang lengkap untuk mengisi antrian dari kanan record per record sampai antrian penuh kanan, atau tidak bisa diisi lagi dari kanan.

Algoritma



Penjelasan prosesnya :

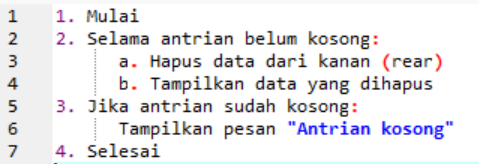
* Program akan terus menambah data dari bagian kanan (rear).
* Setiap kali menambahkan data, posisi rear akan bergeser ke kanan (atau melingkar jika circular queue).
* Proses ini terus berulang sampai antrian mencapai kapasitas maksimal (penuh).
* Saat penuh, sistem akan menghentikan proses dan memberi tahu bahwa tidak bisa menambah lagi dari kanan.

**Kesimpulan:**

Proses pengisian dari kanan akan berjalan selama masih ada ruang di antrian.  
Begitu antrian penuh, tidak ada data baru yang bisa ditambahkan lagi dari sisi kanan.

4.) Tulis algoritma lengkap untuk mendelete isi antrian dari kanan record per record sampai antrian kosong

Algoritma



**Penjelasan prosesnya:**

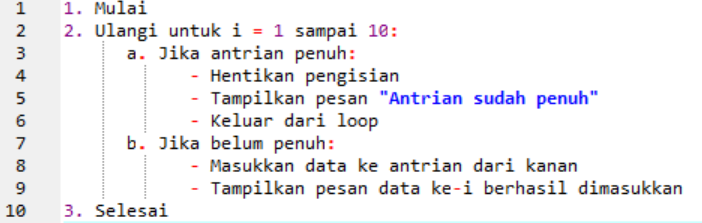
* Program akan terus menghapus data dari sisi kanan (rear) satu per satu.
* Setiap kali data dihapus, posisi rear akan bergeser ke kiri (atau ke elemen sebelumnya).
* Proses akan berhenti ketika tidak ada lagi data di dalam antrian (count == 0).
* Setelah itu akan muncul pesan bahwa antrian kosong.

**Kesimpulan:**

Proses penghapusan dari kanan dilakukan sampai semua data hilang dari antrian.  
Kalau antrian sudah kosong, tidak bisa dihapus lagi karena tidak ada data yang tersisa.

5.) Tulis algoritma yang lengkap untuk mengisi antrian dari kanan record per record sebanyak 10 record selama antrian

Algoritma



**Penjelasan prosesnya:**

* Program mencoba memasukkan 10 data satu per satu dari kanan.
* Tapi sebelum setiap pemasukan, program memeriksa dulu apakah antrian sudah penuh.
* Jika belum penuh → data dimasukkan.
* Jika sudah penuh di tengah jalan → proses langsung dihentikan, walaupun belum mencapai 10 data.

**Kesimpulan:**

Proses pengisian berhenti otomatis ketika antrian penuh, walaupun belum 10 record.  
Artinya, sistem tidak akan memaksa mengisi lebih dari kapasitas maksimal.

# HASIL DAN PEMBAHASAN

# **LATIHAN 8**

## Soal 1 Program Dasar Double Ended Queue (Deque)

Menunjukkan proses dasar pengoperasian deque: inisialisasi, insert dari kanan dan kiri, serta delete dari kanan dan kiri.

## Pembahasan

### 1. Deklarasi Variabel

* Deque[10] → array untuk menyimpan data antrian sebanyak 10.
* front dan rear → menandakan posisi depan dan belakang deque.
* data → variabel sementara untuk input data.

### 2. Inisialisasi

* front = -1 dan rear = -1 artinya deque masih kosong.

### 3. Insert dari Kanan

* Jika rear == 9, berarti deque penuh kanan.
* Jika kosong, maka front dan rear diubah ke 0.
* Jika tidak kosong, rear bertambah 1, data disimpan di posisi baru.

### 4. Insert dari Kiri

* Jika front == 0, berarti tidak bisa menambah dari kiri.
* Jika deque masih kosong, front dan rear diset ke 0.
* Jika ada isi, front dikurangi 1 agar menambah dari kiri.

### 5. Delete dari Kanan

* Jika kosong (front == -1), tidak ada data dihapus.
* Jika hanya satu data (front == rear), deque jadi kosong kembali.
* Jika lebih dari satu, rear dikurangi 1.

### 6. Delete dari Kiri

* Mirip delete kanan, tapi front bertambah 1 karena dihapus dari kiri

### Kesimpulan

* Program ini memperlihatkan cara dasar deque bekerja.
* Deque memungkinkan penambahan dan penghapusan data dari kedua sisi, membuatnya fleksibel untuk berbagai aplikasi seperti buffer data dan antrian berganda.

## Soal 2 Program Ciri-Ciri Kondisi Double Ended Queue

Mengetahui kondisi deque apakah kosong, penuh kanan, penuh kiri, atau penuh total, berdasarkan nilai FRONT dan REAR.

## Pembahasan

### 1. Deklarasi Variabel

* front dan rear digunakan untuk menentukan posisi antrian.
* n = 10 → kapasitas maksimal deque.

### 2. Kondisi Utama

* front == -1 && rear == -1 → deque kosong.
* rear == n - 1 → penuh kanan.
* front == 0 → penuh kiri.
* front == 0 && rear == n - 1 → penuh total.

### 3. Analisis Tambahan

* Jika jumlahIsi = rear - front + 1, program dapat menilai berapa banyak data yang sedang terisi.
* Jika jumlahIsi == 0, berarti deque kosong.
* Jika jumlahIsi < 10, berarti deque masih bisa diisi sebagian.

### Kesimpulan

* Soal ini membantu mengenali status logis deque berdasarkan posisi indeks.
* Penting untuk mencegah overflow dan underflow, serta mengetahui kapan deque masih bisa digunakan.

## Soal 3 Program Mengisi Antrian dari Kanan Record per Record

Melatih proses insert data dari sisi kanan deque sampai kondisi penuh kanan tercapai.

## Pembahasan

### 1. Deklarasi

* Deque[10] sebagai wadah data.
* front dan rear diinisialisasi -1 menandakan deque kosong.
* n = 10 kapasitas maksimum.

### 2. Proses Pengisian

* Pengisian dilakukan berulang selama rear < n - 1.
* Jika deque kosong, front = 0, rear = 0.
* Setiap data baru, rear bertambah 1.
* Bila rear sudah 9, berarti deque penuh kanan.

### 3. Tampilan Akhir

* Program menampilkan seluruh isi deque sesuai urutan index.

### Kesimpulan

* Deque dapat diisi bertahap dari kanan sampai penuh.
* Pendekatan ini digunakan untuk simulasi antrian data masuk seperti buffer jaringan atau proses batch input.

## Soal 4 Program Menghapus Antrian Record per Record dari Kanan

Menunjukkan cara menghapus data (delete) dari kanan deque sampai antrian kosong.

## Pembahasan

### 1. Deklarasi

* Deque[10] untuk menyimpan data.
* front = 0, rear = n - 1 → posisi awal antrian.

### 2. Proses

* Program menghapus satu per satu dari kanan (rear--).
* Setiap kali data dihapus, jumlahHapus bertambah.
* Jika rear < front, berarti deque kosong dan proses berhenti.

### 3. Kondisi Henti

* Dihentikan jika semua data sudah dihapus.

### Kesimpulan

* Program ini menunjukkan proses pengosongan deque dengan cara menghapus dari kanan.
* Teknik ini penting dalam sistem buffer atau pengelolaan data berlapis, di mana data terakhir keluar terlebih dahulu (LIFO di sisi kanan).

## Soal 5 Program Mengisi Antrian Record Sebelum Penuh Kanan

Mendemonstrasikan pengisian deque dari sisi kanan hingga penuh kanan tercapai secara otomatis.

## Pembahasan

### 1. Deklarasi

* Deque[10] untuk menampung data.
* front dan rear diinisialisasi -1.
* jumlah → menghitung banyak data yang sudah dimasukkan.

### 2. Proses

* Program meminta input record satu per satu.
* Jika deque kosong, front dan rear diatur ke 0.
* Jika belum penuh, rear++ untuk menambah data.
* Pengisian berhenti jika rear == n - 1 (indeks terakhir).

### 3. Hasil

* Program menampilkan seluruh isi deque dengan nomor urut data.

### Kesimpulan

* Deque diisi otomatis dari kanan hingga batas maksimum.
* Metode ini bermanfaat untuk simulasi input batch yang berhenti otomatis saat kapasitas penuh.

# 

# KESIMPULAN

1. Double Ended Queue (Deque) adalah struktur data yang memungkinkan penambahan dan penghapusan data dari dua sisi.
2. Setiap kondisi deque ditentukan oleh nilai FRONT dan REAR.
3. Operasi deque lebih fleksibel dibanding queue biasa karena dapat bekerja seperti stack maupun queue.
4. Pemahaman posisi penuh kanan, penuh kiri, dan kosong penting untuk mencegah kesalahan logika dalam pengisian atau penghapusan data.