**LAPORAN AWAL STRUKTUR DATA**

**Pertemuan ke-6**

**Linear 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 penting dalam pengelolaan data secara berurutan. Queue menggunakan prinsip FIFO (First In, First Out), artinya elemen yang pertama kali masuk akan menjadi elemen pertama yang keluar.

Linear Queue banyak digunakan pada sistem komputer yang membutuhkan pengaturan giliran atau alur proses, antara lain:

* Sistem antrian cetak pada printer
* Penjadwalan proses CPU (CPU scheduling)
* Buffer data (data streaming atau keyboard buffer)
* Antrian pelanggan atau data transaksi

## 1. Pengertian Linear Queue

Linear Queue adalah struktur data linear di mana elemen hanya bisa ditambahkan di belakang (rear) dan dihapus dari depan (front). Elemen mengikuti prinsip FIFO, sehingga elemen pertama yang masuk akan keluar lebih dahulu.

## 2. Ciri-ciri Linear Queue

* Operasi enqueue menambah elemen di belakang (rear)
* Operasi dequeue menghapus elemen dari depan (front)
* Memiliki dua pointer: front dan rear
* Queue kosong jika rear = -1 atau front > rear
* Queue penuh jika rear mencapai kapasitas maksimal

## 3. Operasi Dasar Linear Queue

* Enqueue → menambahkan elemen di belakang queue
* Dequeue → menghapus elemen dari depan queue
* isEmpty() → mengecek apakah queue kosong
* isFull() → mengecek apakah queue penuh (untuk array)

## 4. Implementasi Linear Queue dengan Array

* Menggunakan array dengan kapasitas tetap
* Pointer front menandai elemen pertama, rear menandai elemen terakhir
* Kekurangan: setelah beberapa kali dequeue, ruang di awal array tidak bisa digunakan lagi, sehingga bisa memerlukan reset atau circular queue

## 5. Implementasi Linear Queue dengan Linked List

* Node menyimpan data dan pointer ke node berikutnya
* front menunjuk ke node depan, rear menunjuk ke node belakang
* Ukuran queue menjadi dinamis, menyesuaikan jumlah elemen
* Tidak ada pemborosan ruang seperti pada array

## 6. Kelebihan Linear Queue

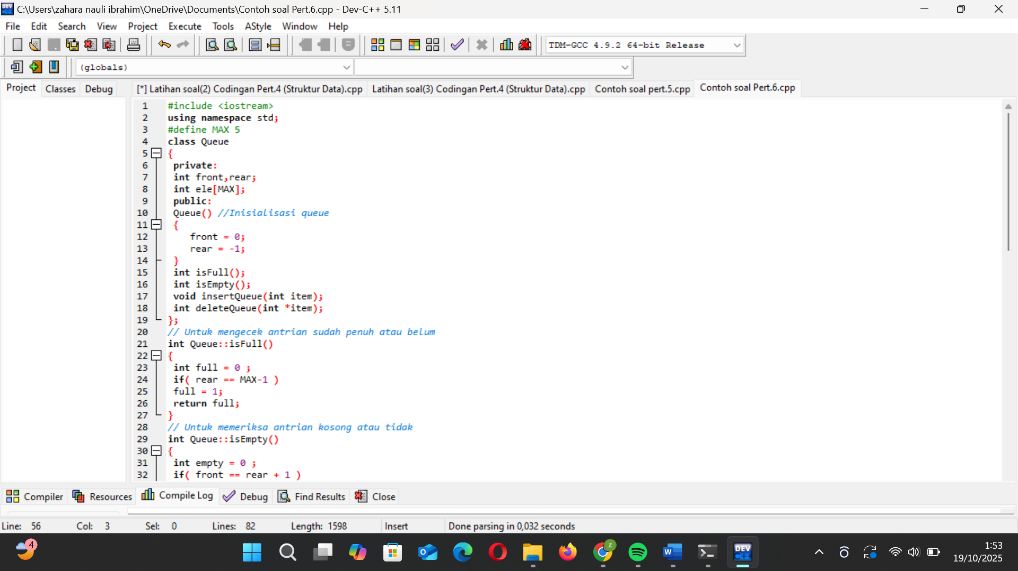
* Sederhana dan mudah diimplementasikan
* Cocok untuk antrian sederhana dan penjadwalan proses

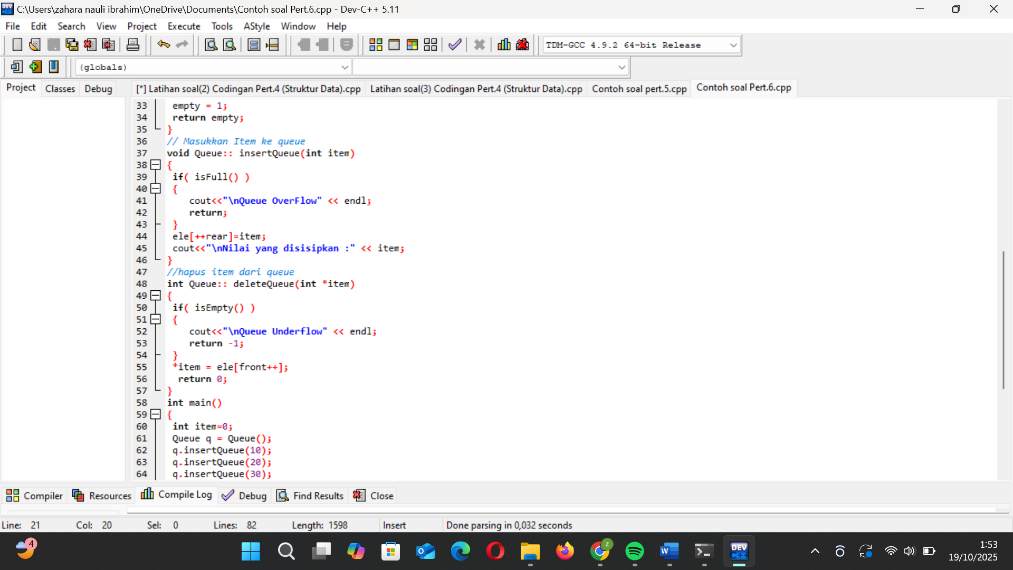
## 7. Kekurangan Linear Queue

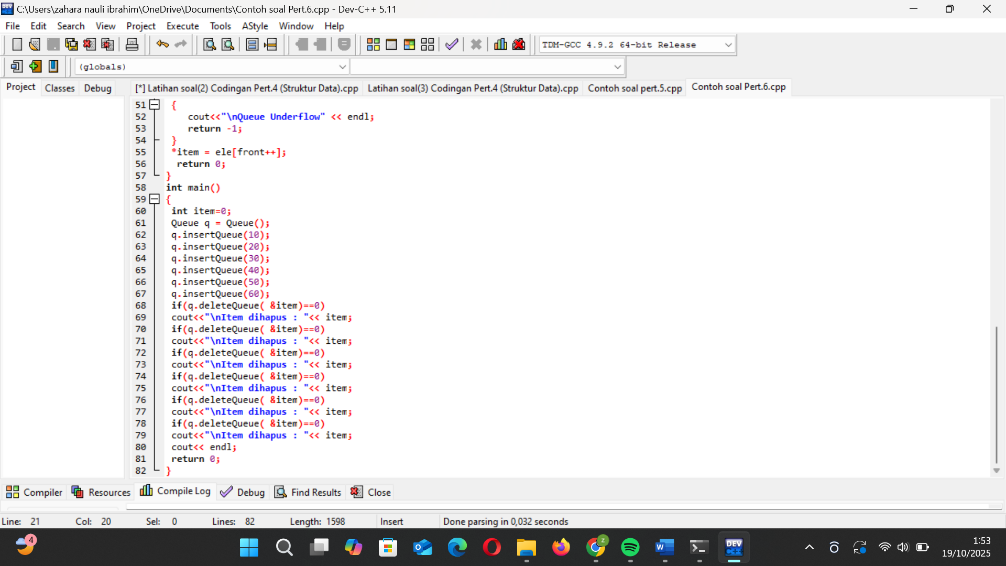
* Array statis memiliki keterbatasan kapasitas
* Setelah beberapa dequeue, ruang di awal array tidak dapat digunakan kembali
* Tidak fleksibel jika jumlah elemen berubah-ubah, kecuali menggunakan circular queue

# LAPORAN PRAKTIKUM

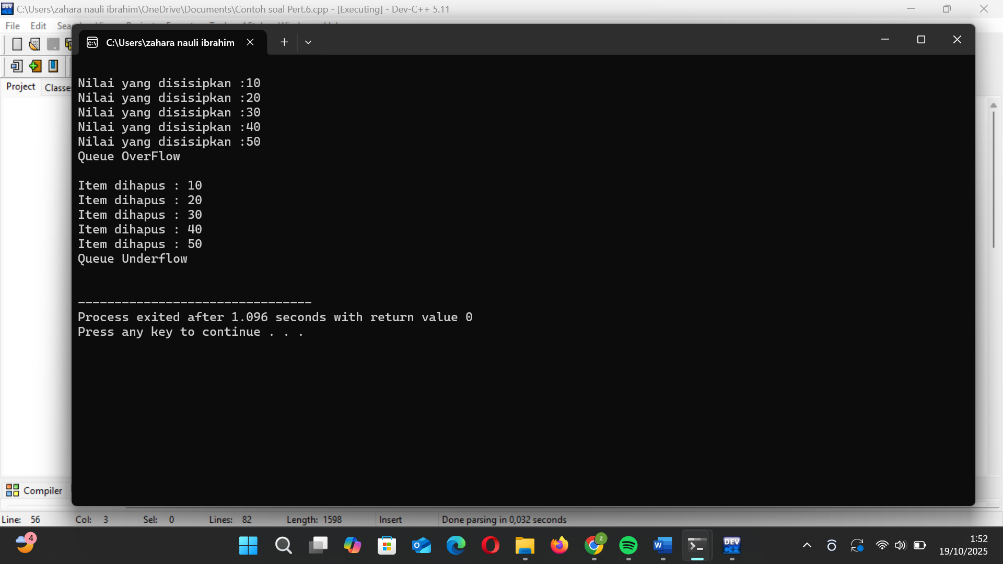
## 1) Program Implementasi Antrian Linear untuk operasi antrian







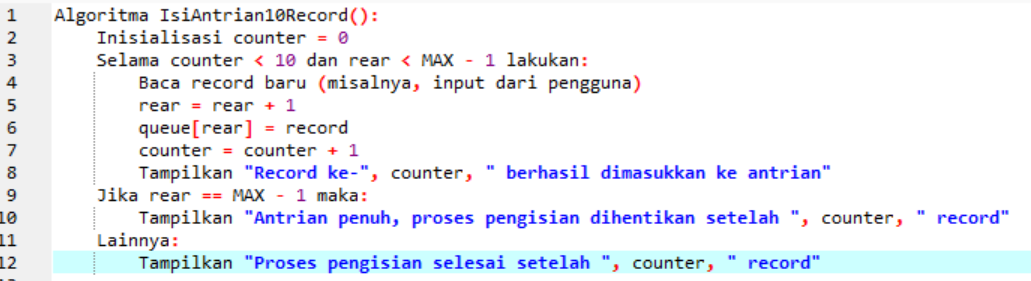
## Hasil Output



## LATIHAN 6

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

## Algoritma

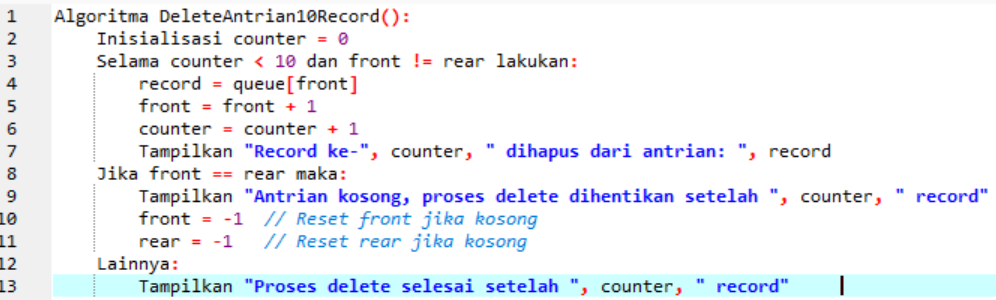


**Penjelasan**

* 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



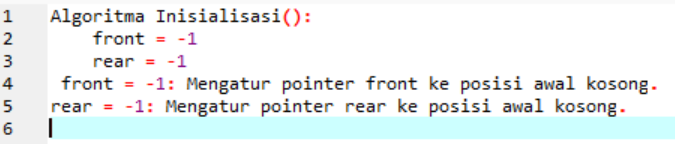
**Penjelasan**

* 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

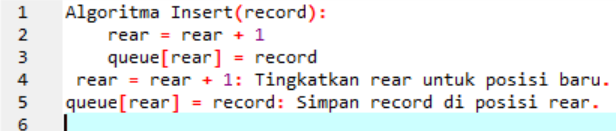
1. Inisialisasi



**Penjelasan**

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

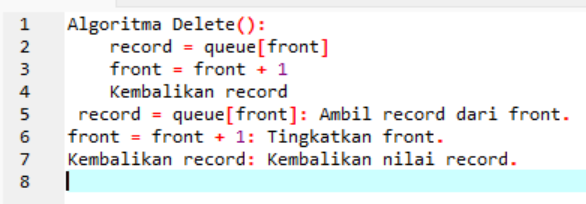
1. Insert sebuah record



**Penjelasan**

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

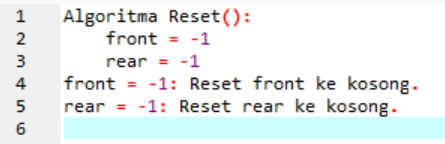
1. Delete sebuah record



**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.

**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 5

### 1) Tulislah algoritma yang lengkap untuk :

### Mengisi Stack1 (PUSH1)

### Menghapus isi Stack1 (POP1)

### Mengisi Stack2 (PUSH2)

### Menghapus isi Stack2 (POP2)

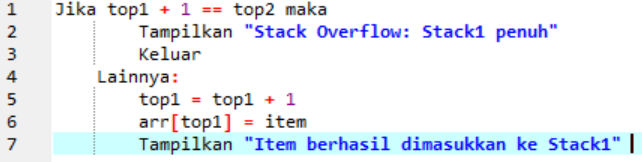
**Algoritma Lengkap untuk Operasi Double Stack**

Double stack adalah struktur data di mana dua stack (Stack1 dan Stack2) berbagi satu array dengan ukuran tetap (misalnya, MAX). Stack1 tumbuh dari indeks rendah (mulai dari -1), sedangkan Stack2 tumbuh dari indeks tinggi (mulai dari MAX). Kita menggunakan variabel top1 untuk Stack1 dan top2 untuk Stack2. Kondisi penuh terjadi jika top1 + 1 == top2. Kondisi kosong untuk Stack1 adalah top1 == -1, dan untuk Stack2 adalah top2 == MAX.

**Asumsikan array bernama arr[MAX], dengan top1 = -1 dan top2 = MAX sebagai inisialisasi awal.**

1. Mengisi Stack1 (PUSH1)

Algoritma ini menambahkan elemen ke Stack1. Periksa overflow terlebih dahulu. Algoritma PUSH1(item):



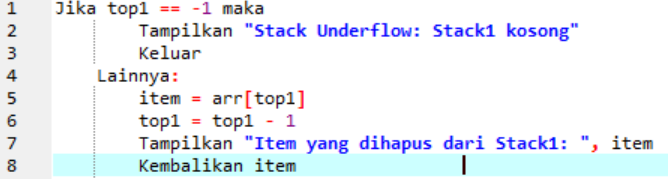
**Penjelasan**

* Jika top1 + 1 == top2 maka: Ini adalah pengecekan kondisi overflow. Jika posisi berikutnya dari top1 bertabrakan dengan top2, berarti array penuh dan tidak bisa menambah elemen ke Stack1.
* Tampilkan "Stack Overflow: Stack1 penuh": Memberikan pesan kesalahan kepada pengguna jika kondisi overflow terjadi.
* Keluar: Menghentikan algoritma untuk mencegah error.
* Lainnya: Jika tidak overflow, lanjutkan ke langkah berikut.
* top1 = top1 + 1: Menambah indeks top1 untuk menunjukkan posisi baru di array.
* arr[top1] = item: Menyimpan item di posisi top1 yang baru.
* Tampilkan "Item berhasil dimasukkan ke Stack1": Konfirmasi bahwa operasi berhasil.

1. Menghapus isi Stack1 (POP1)

Algoritma ini menghapus elemen teratas dari Stack1. Periksa underflow terlebih dahulu.

Algoritma POP1():

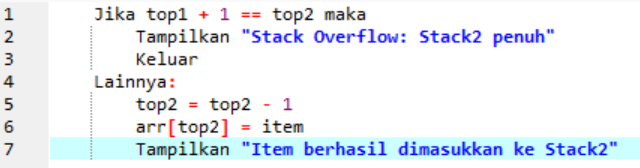


**Penjelasan**

* Jika top1 == -1 maka: Pengecekan underflow. Jika top1 masih di posisi awal (-1), berarti Stack1 kosong.
* Tampilkan "Stack Underflow: Stack1 kosong": Pesan kesalahan jika stack kosong.
* Keluar: Hentikan algoritma.
* Lainnya: Jika tidak kosong, lanjutkan.
* item = arr[top1]: Ambil elemen teratas dari Stack1.
* top1 = top1 - 1: Kurangi top1 untuk menunjukkan posisi baru (elemen dihapus).
* Tampilkan "Item yang dihapus dari Stack1: ", item: Tampilkan elemen yang dihapus.
* Kembalikan item: Mengembalikan nilai elemen untuk digunakan di luar algoritma.

1. Mengisi Stack2 (PUSH2)

Algoritma ini menambahkan elemen ke Stack2. Periksa overflow terlebih dahulu. Algoritma PUSH2(item):



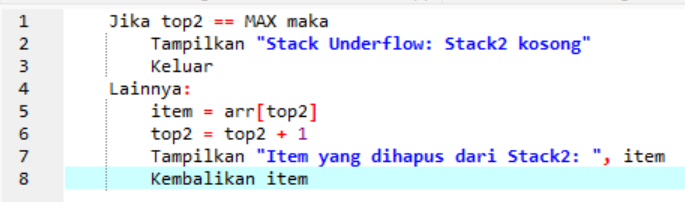
**Penjelasan**

* Jika top1 + 1 == top2 maka: Pengecekan overflow untuk Stack2 (sama seperti PUSH1, karena array bersama).
* Tampilkan "Stack Overflow: Stack2 penuh": Pesan kesalahan.
* Keluar: Hentikan.
* Lainnya: Lanjutkan jika aman.
* top2 = top2 - 1: Kurangi top2 untuk menunjukkan posisi baru (Stack2 tumbuh ke bawah).
* arr[top2] = item: Simpan item di posisi top2 baru.
* Tampilkan "Item berhasil dimasukkan ke Stack2": Konfirmasi.

1. Menghapus isi Stack2 (POP2)

Algoritma ini menghapus elemen teratas dari Stack2. Periksa underflow terlebih dahulu.

Algoritma POP2():



**Penjelasan**

* Jika top2 == MAX maka: Pengecekan underflow. Jika top2 kembali ke MAX, Stack2 kosong.
* Tampilkan "Stack Underflow: Stack2 kosong": Pesan kesalahan.
* Keluar: Hentikan.
* Lainnya: Lanjutkan.
* item = arr[top2]: Ambil elemen teratas dari Stack2.
* top2 = top2 + 1: Tambah top2 untuk menunjukkan posisi baru (elemen dihapus).
* Tampilkan "Item yang dihapus dari Stack2: ", item: Tampilkan elemen.
* Kembalikan item: Kembalikan nilai.

### 2) Tulislah Algoritma Dasar dari:

### Mengisi Stack1 (PUSH1)

### Menghapus isi Stack1 (POP1)

### Mengisi Stack2 (PUSH2)

### Menghapus isi Stack2 (POP2)

**Algoritma dasar ini adalah versi sederhana tanpa pengecekan kondisi overflow/underflow, hanya langkah-langkah inti.**

1. Mengisi Stack1 (PUSH1)

Algoritma Dasar PUSH1(item):

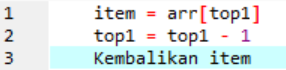


**Penjelasan**

* top1 = top1 + 1: Tingkatkan indeks top1.
* arr[top1] = item: Simpan item di posisi baru.

1. Menghapus isi Stack1 (POP1)

Algoritma Dasar POP1():

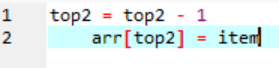


**Penjelasan**

* item = arr[top1]: Ambil elemen teratas.
* top1 = top1 - 1: Kurangi top1.
* Kembalikan item: Kembalikan nilai.

1. Mengisi Stack2 (PUSH2)

Algoritma Dasar PUSH2(item):

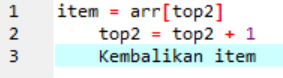


**Penjelasan**

* top2 = top2 - 1: Kurangi top2.
* arr[top2] = item: Simpan item.

1. Menghapus isi Stack2 (POP2)

Algoritma Dasar POP2():



**Penjelasan**

* item = arr[top2]: Ambil elemen teratas.
* top2 = top2 + 1: Tambah top2.
* Kembalikan item: Kembalikan nilai.

### 

### 3) Sebutkan ciri double stack dari kondisi:

### Penuh. Baik Stack1 maupun Stack2 tak bisa diisi lagi

### Baik Stack1 maupun Stack2 bisa diisi lagi

### Baik Stack1 maupun Stack2 tak ada isinya

**Ciri Double Stack dari Kondisi**

1. Penuh (Baik Stack1 maupun Stack2 tak bisa diisi lagi)

Kondisi: top1 + 1 == top2

* Penjelasan: Kedua stack telah memenuhi array bersama, sehingga tidak ada ruang lagi untuk menambahkan elemen ke salah satu stack.

1. Baik Stack1 maupun Stack2 bisa diisi lagi

Kondisi: top1 + 1 < top2

* Penjelasan: Masih ada ruang kosong di antara kedua stack, sehingga keduanya dapat menerima elemen baru.

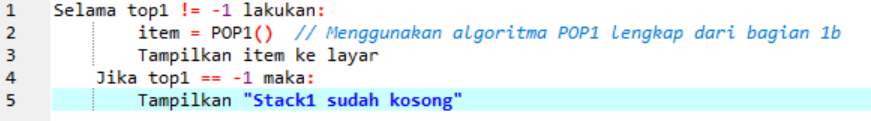
1. Baik Stack1 maupun Stack2 tak ada isinya

Kondisi: top1 == -1 dan top2 == MAX

* Penjelasan: Kedua stack kosong, dengan top1 di posisi awal bawah dan top2 di posisi awal atas.

## 4) Tulis algoritma yang lengkap untuk mengambil isi Stack1 satu persatu dan mencetaknya ke layar, sampai stack1 isinya kosong

Algoritma ini menggunakan operasi POP1 berulang hingga Stack1 kosong, sambil mencetak setiap elemen yang diambil. Algoritma CetakStack1():

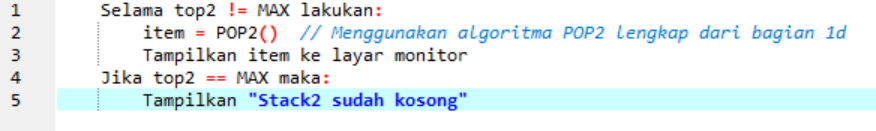


**Penjelasan**

* Selama top1 != -1 lakukan: Loop berulang selama Stack1 tidak kosong.
* item = POP1(): Panggil POP1 untuk menghapus dan ambil elemen (termasuk pengecekan underflow).
* Tampilkan item ke layar: Cetak elemen ke layar.
* Jika top1 == -1 maka: Setelah loop, konfirmasi jika kosong.
* Tampilkan "Stack1 sudah kosong": Pesan akhir.

## 5) Tulis algoritma yang lengkap untuk mengambil isi Stack2 satu persatu dan mencetaknya ke layar monitor, sampai stack2 isinya kosong.

Algoritma ini menggunakan operasi POP2 berulang hingga Stack2 kosong, sambil mencetak setiap elemen yang diambil. Algoritma CetakStack2():



**Penjelasan**

* Selama top2 != MAX lakukan: Loop selama Stack2 tidak kosong.
* item = POP2(): Panggil POP2 untuk menghapus dan ambil elemen.
* Tampilkan item ke layar monitor: Cetak elemen.
* Jika top2 == MAX maka: Konfirmasi setelah loop.
* Tampilkan "Stack2 sudah kosong": Pesan akhir.

**Kesimpulan akhir :**

Double stack adalah struktur data efisien untuk mengelola dua stack dalam satu array, menghemat memori dengan pertumbuhan dari ujung berlawanan. Operasi dasar (push dan pop) memerlukan pengecekan overflow/underflow untuk mencegah error, seperti pada algoritma lengkap. Algoritma cetak menunjukkan cara mengosongkan stack sambil menampilkan isi, yang berguna untuk debugging atau pemrosesan data. Secara keseluruhan, double stack cocok untuk aplikasi yang membutuhkan dua stack independen dengan batasan memori, tetapi memerlukan pengelolaan indeks yang hati-hati untuk menghindari tabrakan. Jika diimplementasikan dalam kode, pastikan inisialisasi top1 dan top2 benar untuk menghindari bug.

# HASIL DAN PEMBAHASAN

## LATIHAN 5

## Soal 1 Program Lengkap Operasi Double Stack (PUSH1, POP1, PUSH2, POP2)

Membuat program lengkap yang dapat melakukan seluruh operasi dasar Double Stack dalam satu menu interaktif.

### Pembahasan:

Dua stack (Stack1 dan Stack2) disimpan dalam satu array S[10].

top1 dimulai dari -1 (kiri ke kanan), dan top2 dimulai dari 10 (kanan ke kiri).

Program menyediakan menu:

1. PUSH1 → menambah data ke Stack1.
2. POP1 → menghapus data dari Stack1.
3. PUSH2 → menambah data ke Stack2.
4. POP2 → menghapus data dari Stack2.
5. Lihat isi stack → menampilkan seluruh isi Stack1 dan Stack2.
6. Keluar program.

Kondisi penuh terjadi jika top1 + 1 == top2, artinya ruang array habis.

Jika top1 == -1, maka Stack1 kosong. Jika top2 == 10, maka Stack2 kosong.

Makna Deklarasi:

S[10] adalah wadah tunggal untuk dua stack.

top1 dan top2 menjadi penanda arah penyimpanan data yang saling berlawanan.

### Kesimpulan:

Program ini menunjukkan bagaimana dua stack bisa berbagi satu ruang array tanpa tumpang tindih, dengan kontrol kondisi penuh dan kosong diatur lewat nilai top1 dan top2.

## Soal 2 Dasar Operasi Double Stack (PUSH1, POP1, PUSH2, POP2)

Memahami operasi dasar dua stack tanpa menu interaktif, secara langsung satu per satu.

### Pembahasan:

Program langsung menampilkan urutan proses:

1. PUSH1 → Memasukkan data ke Stack1.
2. POP1 → Menghapus data dari Stack1.
3. PUSH2 → Memasukkan data ke Stack2.
4. POP2 → Menghapus data dari Stack2.

Kondisi penuh dicek dengan top1 + 1 == top2, sedangkan kosong dicek dengan top1 == -1 (Stack1) dan top2 == 10 (Stack2).Nilai top1 bertambah setiap kali push ke Stack1, dan top2 berkurang setiap kali push ke Stack2.

Makna Algoritma:

Menjelaskan alur logika kerja dua stack yang bergerak dari arah berlawanan di dalam satu array tunggal.

### Kesimpulan:

Program ini menjelaskan langkah paling dasar bagaimana Double Stack bekerja: satu stack mengisi dari kiri ke kanan, dan stack lainnya dari kanan ke kiri, sampai ruang bertemu di tengah.

## Soal 3 Menentukan Ciri-Ciri Kondisi Double Stack

Menentukan status (kosong, penuh, bisa diisi) dari double stack berdasarkan posisi top1 dan top2.

### Pembahasan:

* Pengguna memasukkan nilai top1 dan top2.
* Program menentukan:
* Penuh → jika top1 + 1 == top2.
* Kosong → jika top1 == -1 dan top2 == 10.
* Bisa diisi → jika masih ada jarak antara top1 dan top2.
* Output menjelaskan kondisi logis dari dua stack yang saling berlawanan arah.

Makna Deklarasi:

top1 dan top2 menjadi indikator logika penuh atau kosong tanpa perlu mengecek seluruh isi array.

### Kesimpulan:

Dengan hanya melihat posisi kedua variabel penanda (top1 dan top2), kita bisa menentukan kondisi stack ganda secara cepat dan efisien.

## Soal 4 Mengambil dan Menampilkan POP Stack1 hingga Kosong

Memahami proses POP berulang hingga Stack1 benar-benar kosong.

### Pembahasan:

* Pengguna menentukan banyak data (n) yang ingin dimasukkan ke Stack1.
* Program mengisi data satu per satu (top1 bertambah tiap input).
* Setelah semua data masuk, program menampilkan proses POP, yaitu mengeluarkan data dari top1 sampai -1.
* Setelah semua data dihapus, muncul pesan “Stack1 sekarang kosong.”

Makna Algoritma:

Menunjukkan mekanisme pengosongan bertahap dari stack bagian kiri (Stack1) hingga mencapai kondisi awal.

### Kesimpulan:

Stack1 mengikuti prinsip LIFO (Last In, First Out) data terakhir yang dimasukkan keluar pertama kali.

## Soal 5 Mengambil dan Menampilkan POP Stack2 hingga Kosong

Mempelajari operasi pengosongan Stack2 yang bergerak dari arah kanan ke kiri.

### Pembahasan:

* Program meminta jumlah data (n) untuk diinput ke Stack2.
* top2 dimulai dari nilai 10, dan setiap kali data masuk, nilainya berkurang (top2--).
* Setelah pengisian selesai, program menampilkan proses pengambilan data (POP) dari top2 hingga kembali ke nilai 10.
* Saat top2 = 10, artinya Stack2 sudah kosong.

Makna Deklarasi:

Stack2 bergerak dari arah kanan ke kiri (berlawanan dengan Stack1).

Variabel top2 menunjukkan posisi data paling atas dari arah kanan.

### Kesimpulan:

Program ini memperjelas konsep arah kerja Double Stack, di mana Stack2 menggunakan ruang dari kanan ke kiri dan mengikuti prinsip LIFO yang sama.

# 

# KESIMPULAN

1. Double Stack memungkinkan dua stack berbagi satu ruang array secara efisien tanpa bentrok.
2. Stack1 tumbuh dari kiri ke kanan (top1 naik), sedangkan Stack2 tumbuh dari kanan ke kiri (top2 turun).
3. Kondisi penuh ditandai saat top1 + 1 == top2.
4. Kondisi kosong terjadi saat top1 == -1 dan top2 == 10.
5. Kedua stack tetap bekerja dengan prinsip LIFO (Last In, First Out) meskipun arah penyimpanan berbeda.