**LAPORAN AWAL STRUKTUR DATA**

**Pertemuan ke-5**

**Double Stack**

****

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, stack (tumpukan) adalah struktur data linear yang penting untuk pengelolaan data secara berurutan. Stack menggunakan prinsip LIFO (Last In, First Out), artinya elemen terakhir yang masuk akan menjadi elemen pertama yang keluar.

Double Stack adalah pengembangan dari single stack, di mana dua stack ditempatkan dalam satu array. Konsep ini memungkinkan penggunaan ruang memori yang lebih efisien, karena kedua stack dapat bertumbuh menuju arah yang berlawanan dalam satu array tanpa mubazir.

Double Stack banyak digunakan pada sistem komputer yang membutuhkan dua struktur tumpukan dengan efisiensi memori, antara lain:

* Penjadwalan dua proses berbeda dalam satu ruang memori
* Simulasi undo-redo ganda dalam aplikasi editor
* Evaluasi ekspresi matematika yang kompleks

## 1. Pengertian Double Stack

Double Stack adalah struktur data linear di mana dua stack independen ditempatkan dalam satu array.

* Stack pertama bertumbuh dari awal array (index 0) ke kanan
* Stack kedua bertumbuh dari akhir array (index n-1) ke kiri

## 2. Ciri-ciri Double Stack

* Memiliki dua pointer utama: top1 untuk stack pertama, top2 untuk stack kedua
* Stack kosong jika top1 = -1 dan top2 = n (untuk array ukuran n)
* Stack penuh jika top1 + 1 = top2
* Setiap stack bersifat independen tetapi berbagi ruang memori yang sama

## 3. Operasi Dasar Double Stack

* Untuk Stack 1:
* push1() → menambahkan elemen di top1
* pop1() → menghapus elemen di top1
* Untuk Stack 2:
* push2() → menambahkan elemen di top2
* pop2() → menghapus elemen di top2
* Selain itu:
* isEmpty1(), isEmpty2() → mengecek apakah masing-masing stack kosong
* isFull() → mengecek apakah kedua stack sudah penuh

## 4. Implementasi Double Stack dengan Array

* Array tunggal digunakan untuk menampung dua stack
* Stack pertama bertumbuh dari kiri ke kanan, stack kedua dari kanan ke kiri
* Pointer top1 dan top2 mengatur posisi teratas masing-masing stack
* Double stack memaksimalkan penggunaan ruang array agar tidak ada ruang yang terbuang

## 5. Kelebihan Double Stack

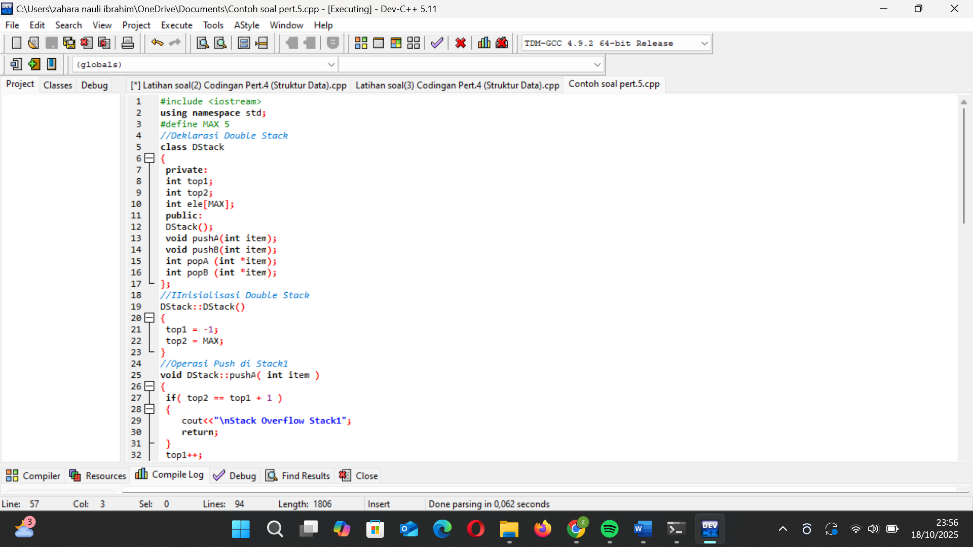
* Menghemat memori karena dua stack berbagi satu array
* Bisa menampung dua tumpukan data secara efisien
* Cocok untuk algoritma yang membutuhkan dua stack dalam satu ruang memori

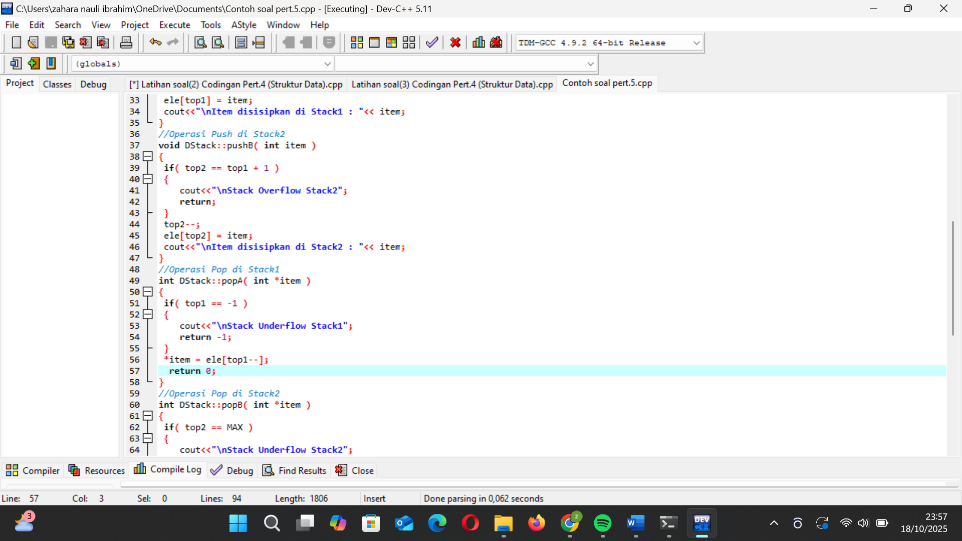
## 6. Kekurangan Double Stack

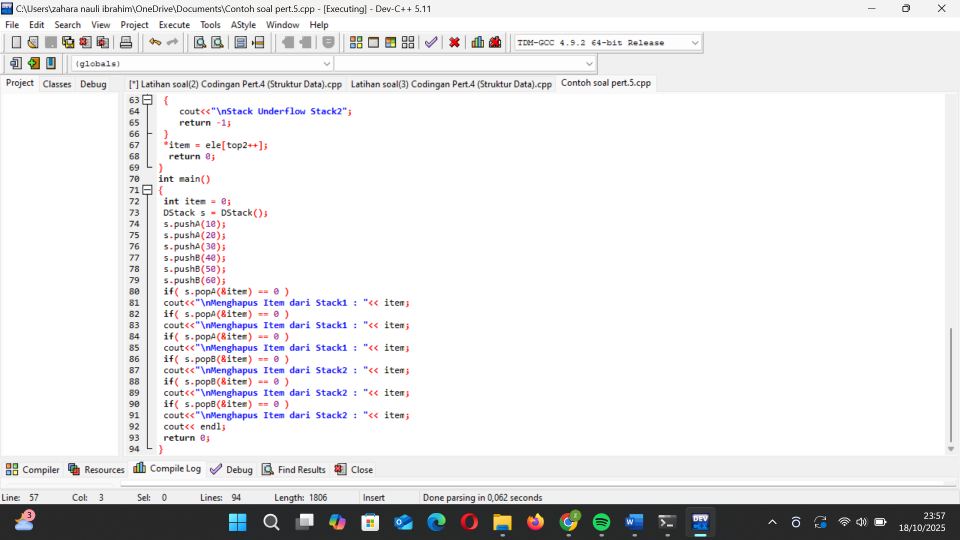
* Jika salah satu stack tumbuh terlalu besar, stack lain mungkin akan terpengaruh
* Implementasi lebih kompleks dibanding single stack
* Array statis membatasi kapasitas maksimal kedua stack

# LAPORAN PRAKTIKUM

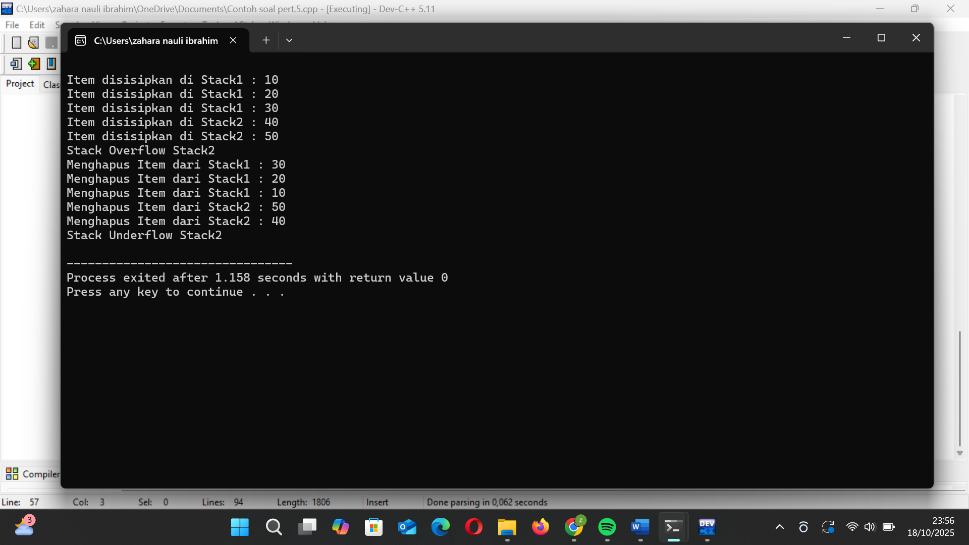
## 1) Program Implementasi Double Stack







## Hasil Output



## LATIHAN 5

### 1) Tulislah algoritma yang lengkap untuk :

### Mengisi Stack1 (PUSH1)

### Menghapus isi Stack1 (POP1)

### Mengisi Stack2 (PUSH2)

### Menghapus isi Stack2 (POP2)

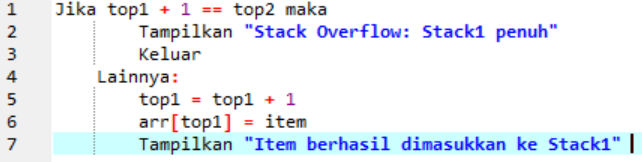
**Algoritma Lengkap untuk Operasi Double Stack**

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):



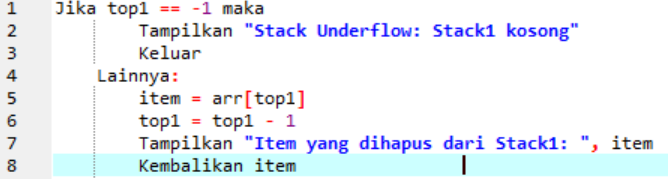
**Penjelasan**

* 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():

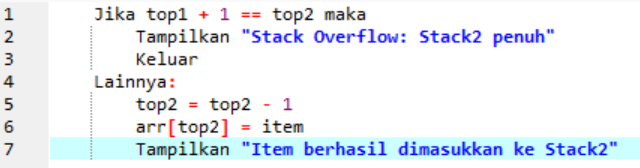


**Penjelasan**

* 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):



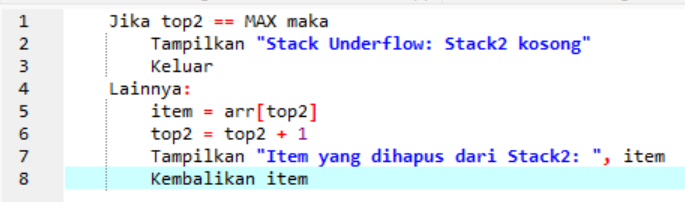
**Penjelasan**

* 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):

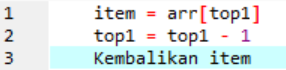


**Penjelasan**

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

1. Menghapus isi Stack1 (POP1)

Algoritma Dasar POP1():

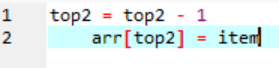


**Penjelasan**

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

1. Mengisi Stack2 (PUSH2)

Algoritma Dasar PUSH2(item):

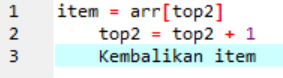


**Penjelasan**

* 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.

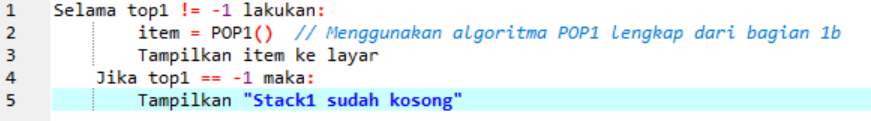
1. Baik Stack1 maupun Stack2 tak ada isinya

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():

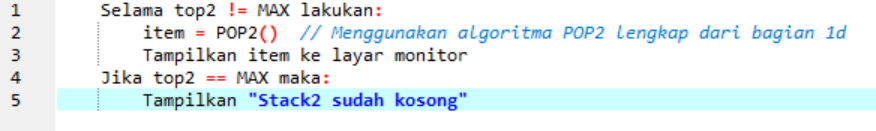


**Penjelasan**

* 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.

**LAPORAN AKHIR**

****

Disusun oleh:

Nama : ZAHARA NAULI IBRAHIM

NIM : 241011402500

Kelas : 03TPLP023

**Program Studi Teknik Informatika**

**Fakultas Ilmu Komputer**

**Universitas Pamulang**

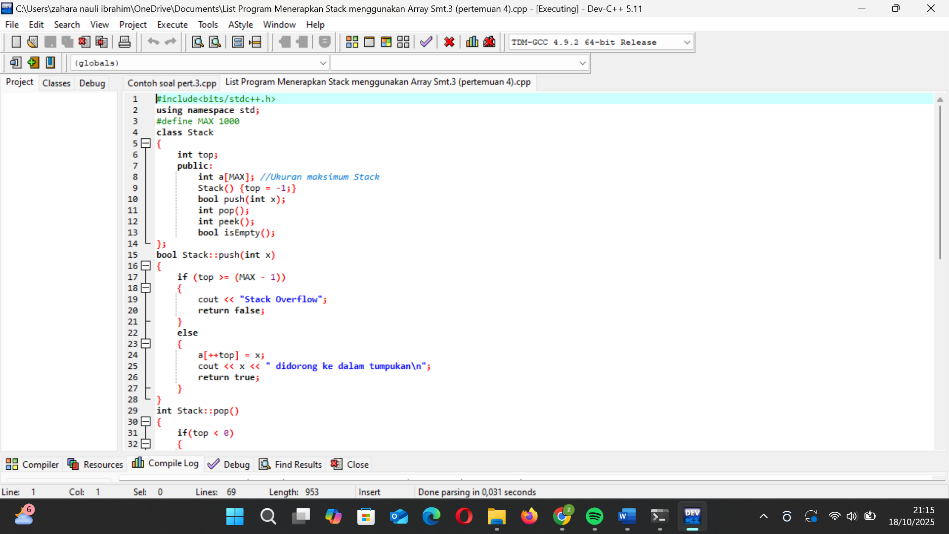
Jl. Surya Kencana No. 1 Pamulang Telp (021)7412566, Fax. (021)7412566

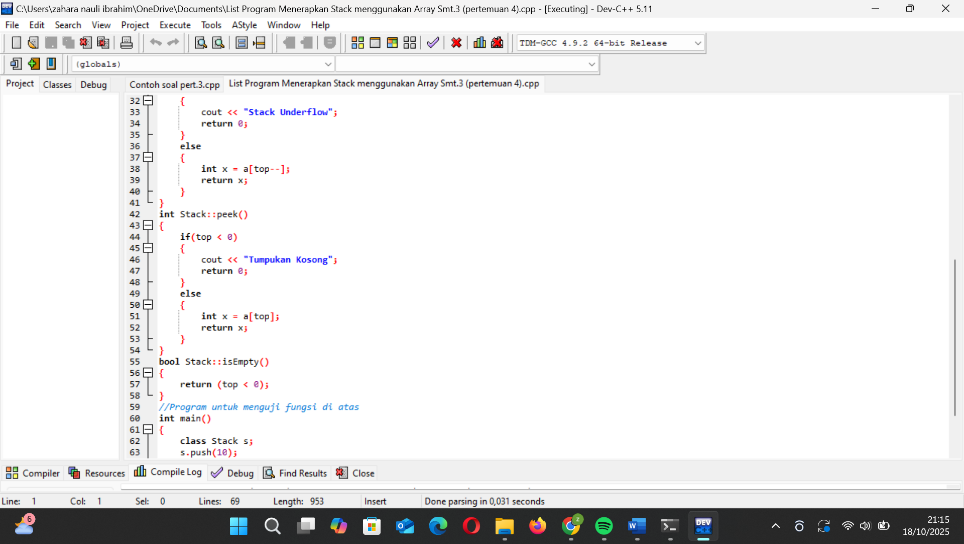
Tangerang Selatan – Banten

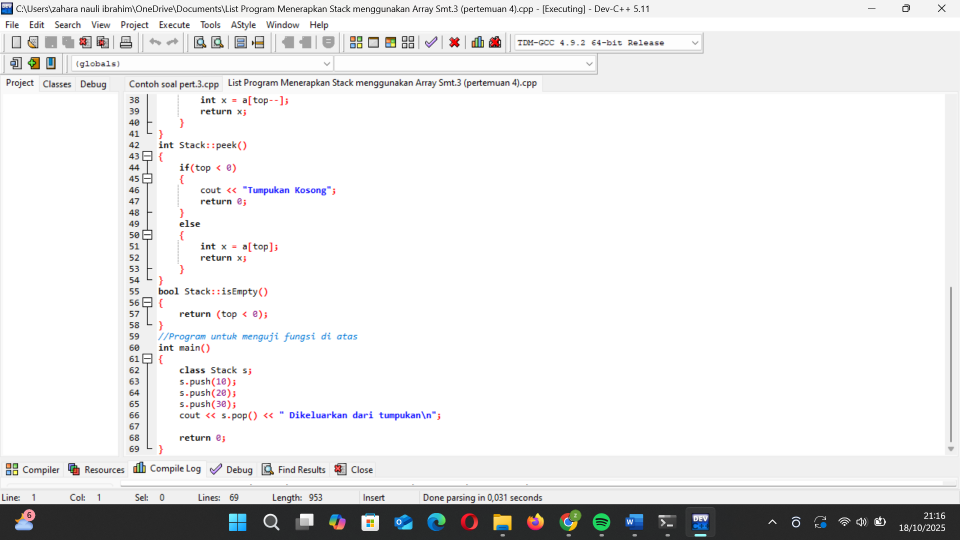
# TUGAS AKHIR

# LAPORAN PRAKTIKUM

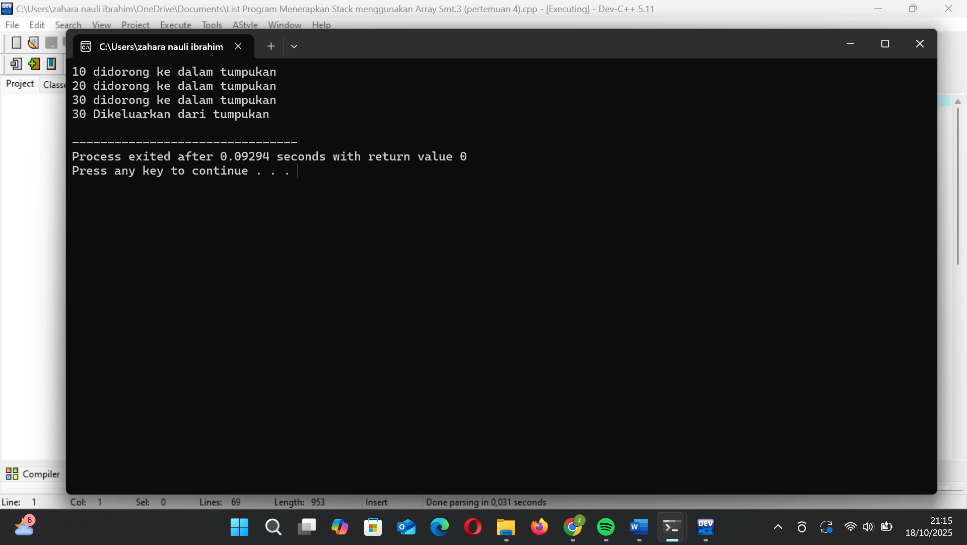
### 1) Program untuk mengimplementasikan operasi dasar pada Stack





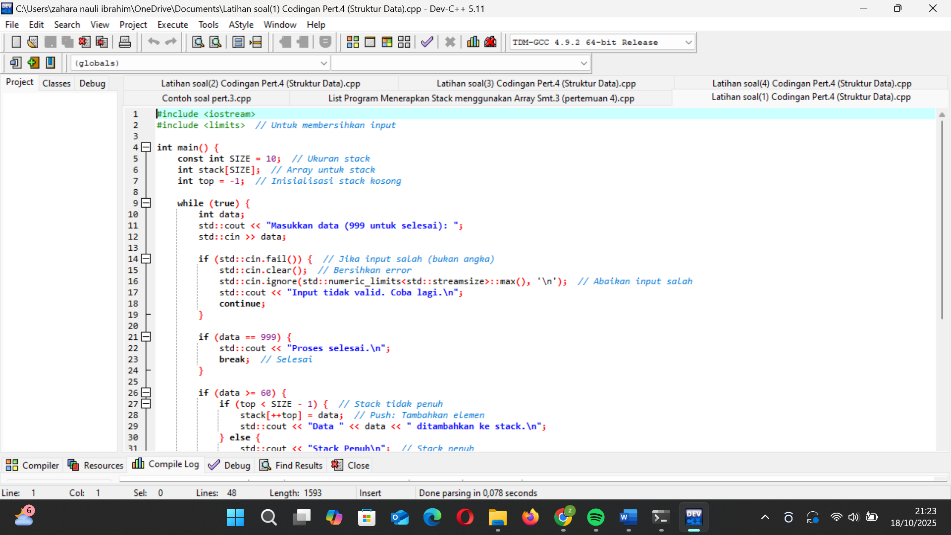


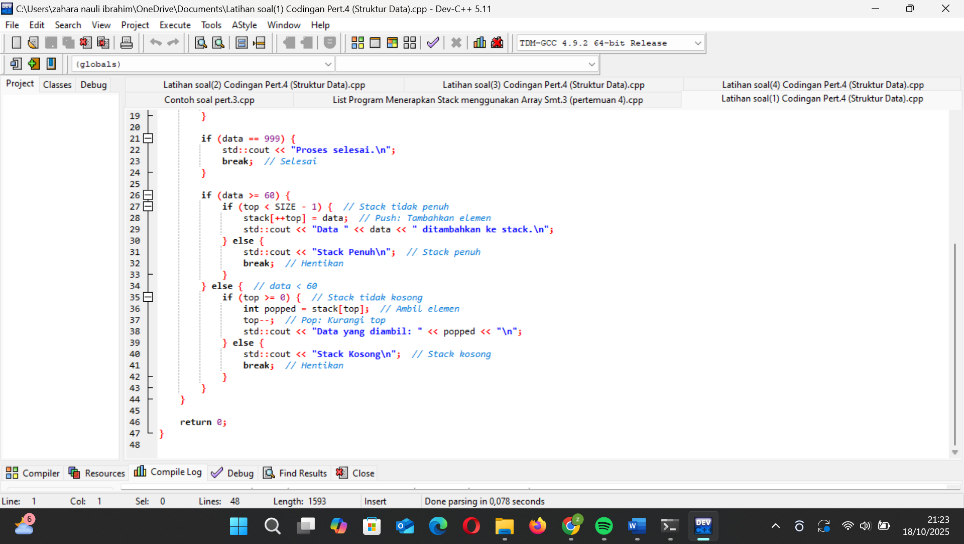
## Hasil Output



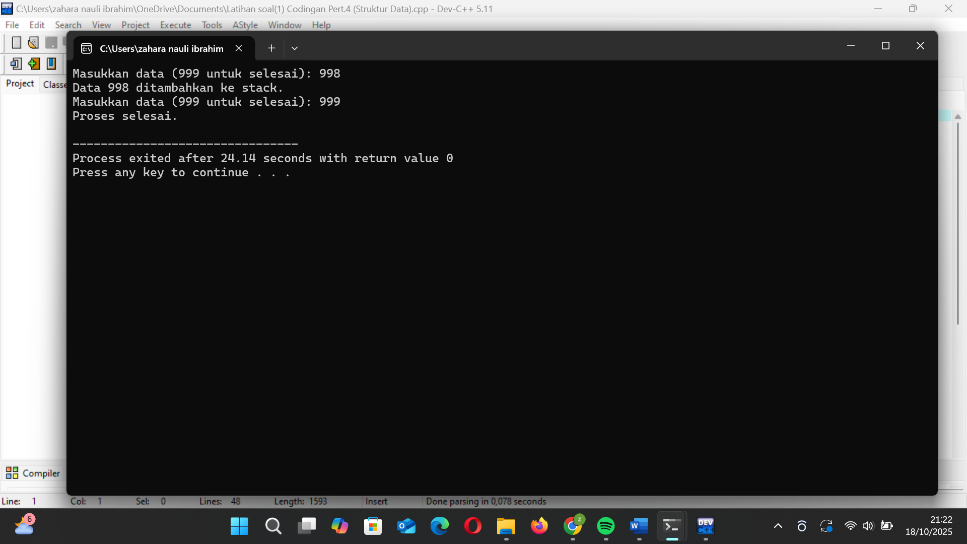
## LATIHAN 4

### 1) Program C++ stack 10 elemen: PUSH ≥ 60, POP < 60, berhenti 999.

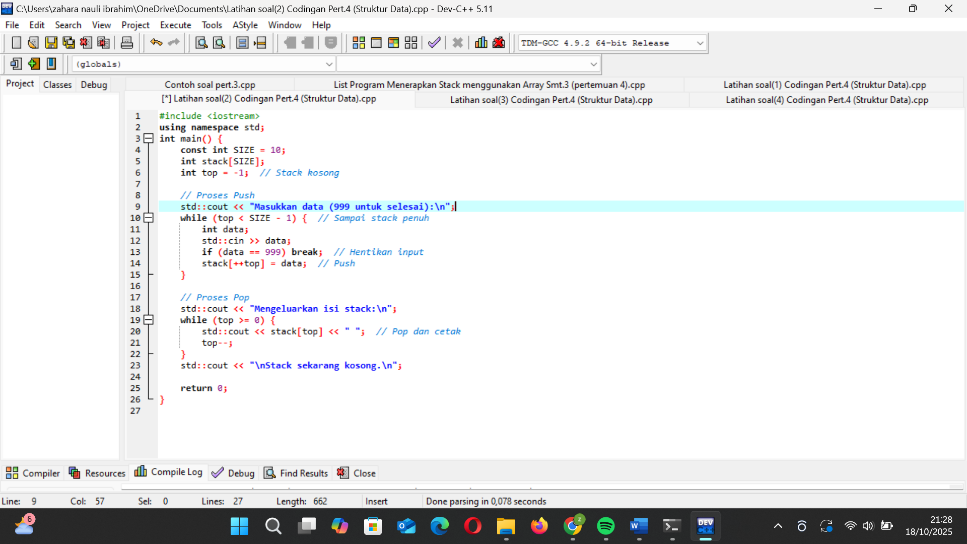




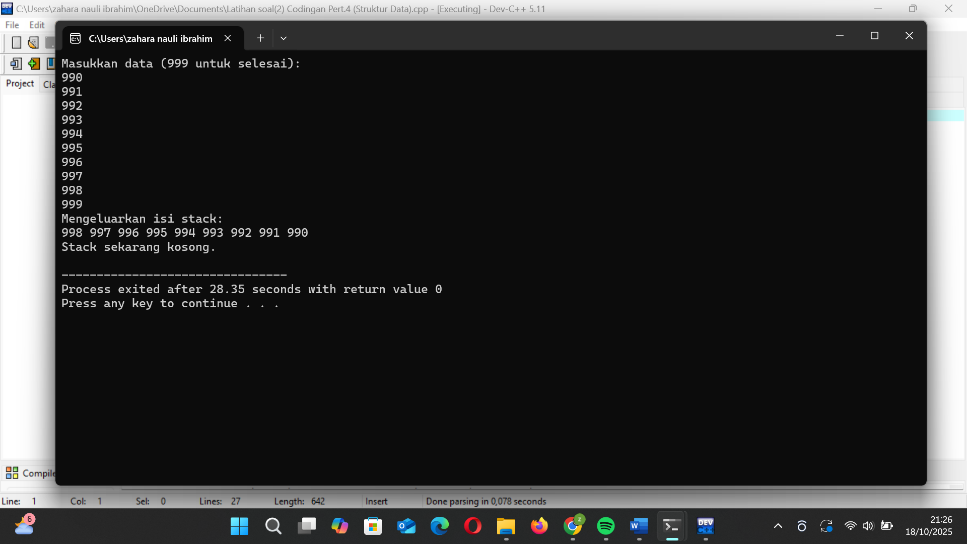
## Hasil Output



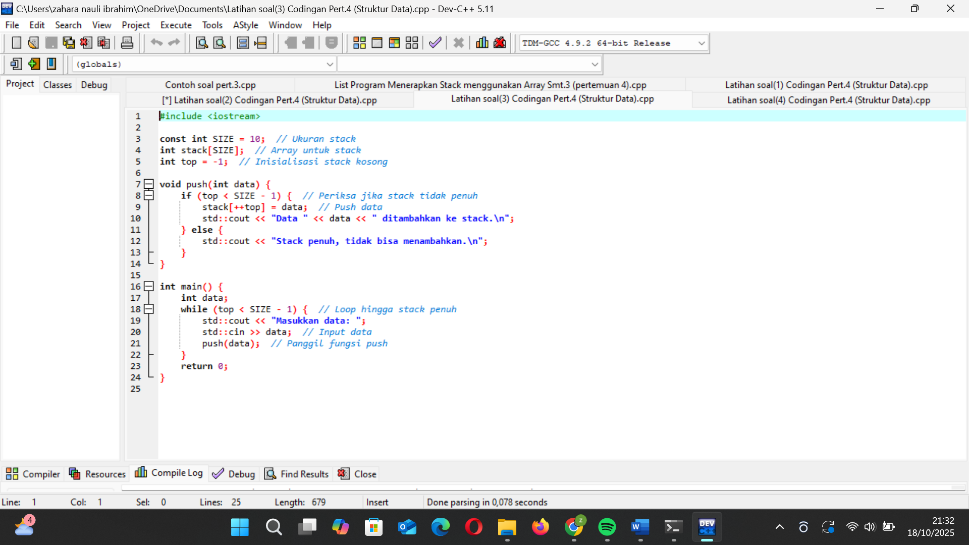
### 2.) Program PUSH hingga penuh/999, lalu POP semua isi stack.



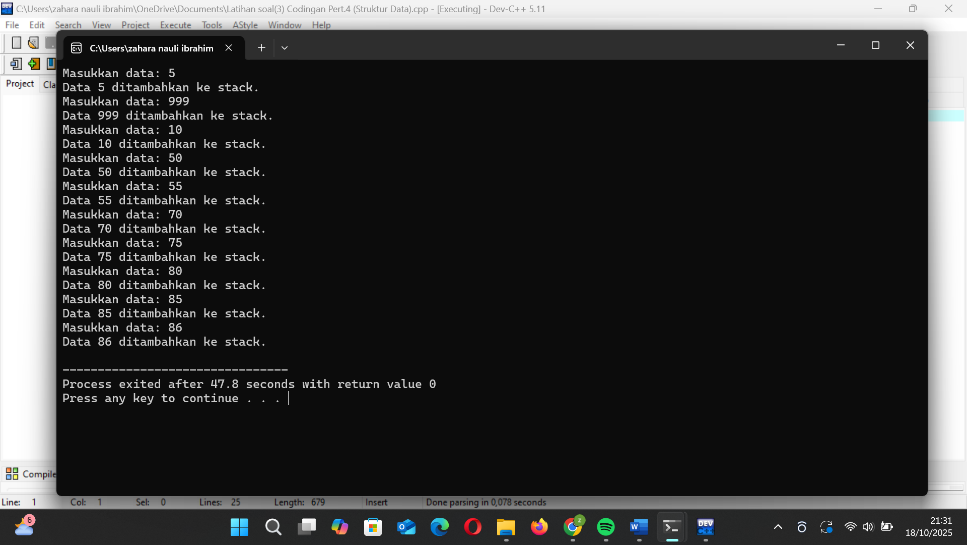
## Hasil Output



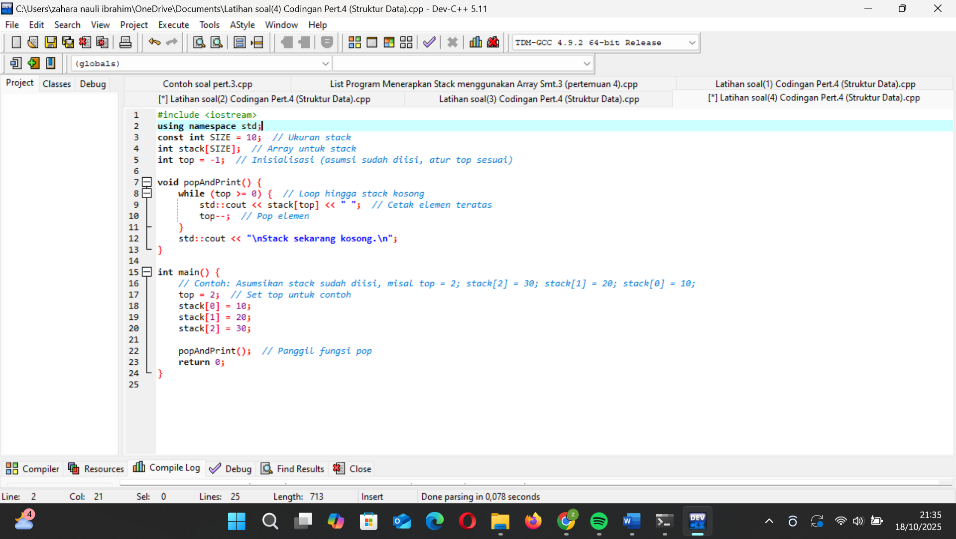
### 3) Potongan program PUSH hingga stack penuh.



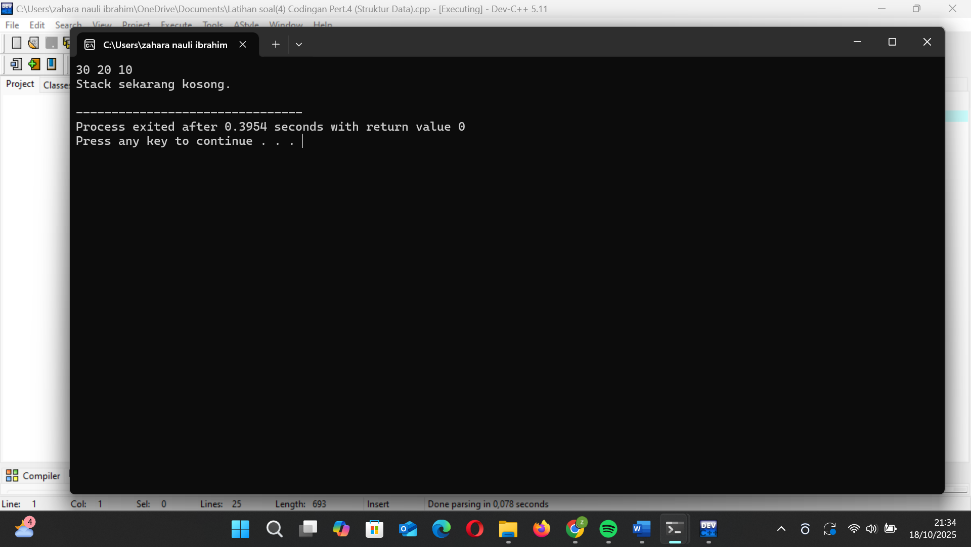
## Hasil Output



### 4.) Potongan program POP hingga stack kosong.



## Hasil Output



### 5.) Ciri stack: kosong, penuh, bisa diisi, ada isinya.

**Penjelasan Teori:**

Single stack adalah struktur data LIFO yang menggunakan array. Berikut ciri-cirinya berdasarkan kondisi, dengan penjelasan sederhana:

1. Kosong:

Ciri: top = -1 (atau nilai awal, menandakan tidak ada elemen).

* Penjelasan: Stack tidak memiliki elemen, jadi operasi pop tidak mungkin. Operasi push masih bisa dilakukan jika stack tidak penuh.

1. Penuh:

Ciri: top = SIZE - 1 (misalnya, untuk SIZE=10, top=9).

* Penjelasan: Stack mencapai kapasitas maksimal, jadi operasi push tidak mungkin. Operasi pop masih bisa dilakukan jika ada elemen.

1. Bisa diisi:

Ciri: top < SIZE - 1 (ada ruang kosong di array).

* Penjelasan: Stack memiliki kapasitas untuk menambahkan elemen, yang merupakan kondisi normal sebelum mencapai batas. Ini memungkinkan operasi push.

1. Ada isinya:

Ciri: top >= 0 (setidaknya satu elemen ada).

* Penjelasan: Stack memiliki elemen, jadi operasi pop mungkin. Ini menandakan stack tidak kosong.

# HASIL DAN PEMBAHASAN

## LATIHAN 4

## Soal 1 PUSH/POP Sesuai Nilai Data

Menerapkan proses PUSH dan POP otomatis berdasarkan nilai data yang diinputkan.

## Pembahasan:

* Program menggunakan stack[10] dengan top = -1 sebagai posisi awal.
* Jika input data ≥ 60, maka data dimasukkan (PUSH) ke stack.
* Jika input data < 60, maka data dikeluarkan (POP) dari stack.
* Input 999 digunakan sebagai tanda berhenti.
* Program juga menampilkan isi stack setelah proses selesai.

Makna Algoritma:

Menunjukkan mekanisme otomatis tumpukan berdasarkan kondisi nilai, serta memperlihatkan perubahan top saat stack bertambah atau berkurang.

### Kesimpulan:

Stack dapat digunakan untuk pengambilan keputusan dinamis — data masuk atau keluar tergantung kondisi tertentu.

## Soal 2 PUSH/POP Manual (Masukkan dan Keluarkan Data)

Melatih proses dasar memasukkan (push) dan mengeluarkan (pop) data secara manual.

### Pembahasan:

* Pengguna menginput data hingga stack penuh (10 elemen) atau memasukkan 999 untuk berhenti.
* Setiap data baru menambah top satu posisi ke atas.
* Setelah selesai, program menampilkan proses POP, yaitu mengeluarkan data satu per satu mulai dari elemen paling atas (top) sampai stack kosong.
* Kondisi top = -1 menunjukkan stack kosong.

Fungsi Algoritma:

Simulasi lengkap proses pengisian dan pengosongan stack dari awal hingga akhir.

### Kesimpulan:

Soal ini memperlihatkan prinsip dasar LIFO (Last In, First Out), di mana data terakhir yang dimasukkan akan keluar terlebih dahulu.

## Soal 3 PUSH hingga Terjadi Overflow

Menunjukkan kondisi overflow (penuh) pada stack ketika data dimasukkan melebihi kapasitas maksimum.

### Pembahasan:

* Stack berukuran tetap (10 elemen).
* Program terus meminta input sampai posisi top == 9.
* Saat mencapai batas, sistem menampilkan pesan “Stack sudah penuh”.
* Setelah itu, isi stack ditampilkan dari atas ke bawah.

Makna Deklarasi dan Logika:

* Variabel top berfungsi sebagai penanda posisi teratas dari stack.
* Setiap proses push menambah nilai top.
* Overflow terjadi saat top mencapai indeks maksimum array.

### Kesimpulan:

Stack memiliki keterbatasan kapasitas; bila terus diisi tanpa pengosongan, maka akan terjadi overflow.

## Soal 4 Mengeluarkan (POP) Isi Stack

Memahami proses pengosongan stack setelah stack terisi penuh.

### Pembahasan:

* Program pertama-tama mengisi stack hingga penuh.
* Setelah penuh, proses dilanjutkan dengan mengeluarkan semua data menggunakan POP.
* Setiap data yang dikeluarkan membuat top berkurang satu.
* Saat top = -1, stack dinyatakan kosong.

Makna Algoritma:

Menunjukkan arah aliran data — semua data dihapus dari puncak ke dasar stack.

* Stack dapat dikosongkan sepenuhnya melalui proses pop berulang, dan kondisi akhir ditandai dengan top = -1.

## Soal 5 Menentukan Ciri-Ciri Kondisi Stack

Menentukan kondisi stack (kosong, penuh, atau masih bisa diisi) berdasarkan nilai variabel top.

### Pembahasan:

* Input nilai top menentukan status stack:
* top = -1 → Stack kosong
* top = 9 → Stack penuh
* 0 ≤ top < 9 → Stack bisa diisi (ada isinya tapi belum penuh)
* Program menampilkan ciri-ciri setiap kondisi agar mudah dipahami secara logis.

Fungsi Deklarasi:

Nilai top adalah indikator utama dalam mendeteksi status stack tanpa perlu membaca seluruh isi data.

# KESIMPULAN

1. Stack bekerja dengan prinsip LIFO (Last In, First Out).
2. Operasi utama yang wajib dipahami adalah PUSH (memasukkan) dan POP (mengeluarkan)
3. Kondisi overflow terjadi saat stack penuh, sedangkan underflow terjadi saat stack kosong
4. Nilai variabel top menentukan status stack saat runtime.
5. Konsep stack banyak digunakan dalam sistem rekursi, undo-redo, compiler parsing, dan manajemen memori