STACK

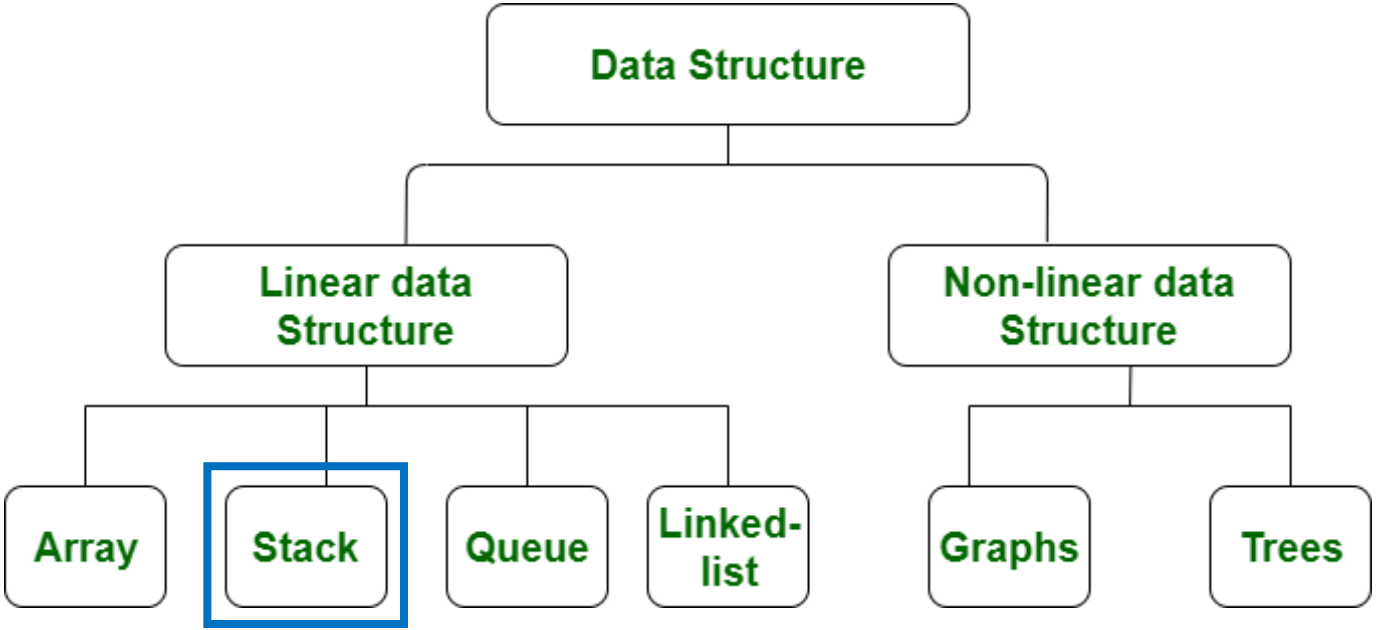
Tim Ajar Algoritma dan Struktur Data
Genap 2023/2024

Capaian Pembelajaran

Setelah mempelajari materi, mahasiswa diharapkan mampu

- Memahami konsep dasar struktur dasar Stack
- Memahami operasi-operasi pada Stack
- Memahami penerapan Stack untuk Postfix Expressions

Jenis Struktur Data

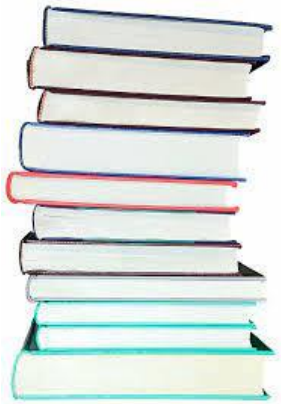


Definisi Struktur Data Linear

- Semua elemen-elemen data disusun secara berurutan atau linier. Setiap elemen melekat satu sama lain dengan elemen sebelum dan sesudahnya.
- Elemen data dapat ditelusuri (traverse) dalam sekali run
- Setiap elemen diakses atau ditempatkan di alamat memori yang berdekatan (secara berurutan)

Definisi Stack

- Stack merupakan struktur data linier yang menganut prinsip **Last In First Out (LIFO)**
- Elemen yang **terakhir masuk** ke dalam stack akan **pertama kali dikeluarkan** karena sifat stack yang membatasi operasi hanya bisa dilakukan **pada salah satu sisinya** saja (bagian atas tumpukan)
- Stack disebut juga sebagai **tumpukan**
- Ilustrasi: tumpukan buku, tumpukan piring, tumpukan koin, dll

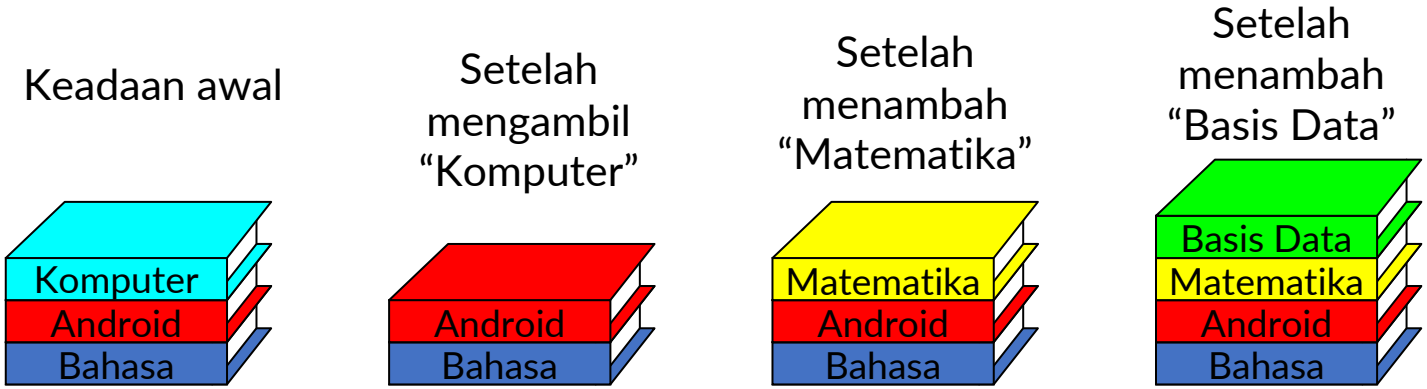


Penerapan Stack

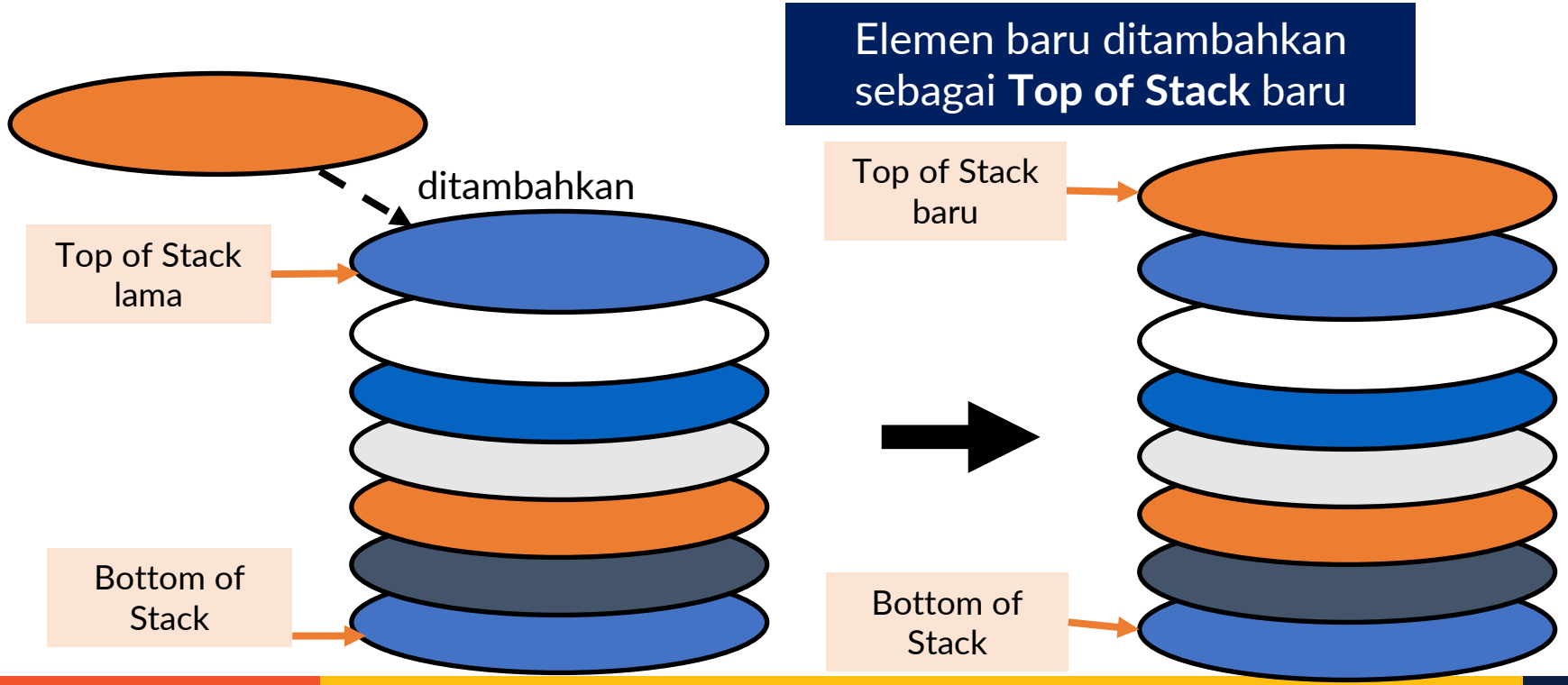
- **Membalik kata**
  Meletakkan semua huruf pada stack kemudian dikeluarkan satu persatu. Karena konsep stack adalah LIFO, maka huruf akan dikeluarkan dalam urutan terbalik
- **Compiler**
  Compiler menggunakan stack untuk menghitung nilai ekspresi seperti 2 + 4 / 5 * (7 - 9) dengan mengubahnya menjadi bentuk prefix atau postfix
- **Browser**
  Tombol Back pada browser menyimpan semua URL yang telah dikunjungi sebelumnya ke dalam stack. Setiap kali mengunjungi halaman baru, halaman itu akan ditambahkan di stack posisi atas. Saat menekan tombol Back, URL saat ini dihapus dari stack dan URL sebelumnya diakses

Konsep Stack

- Suatu susunan koleksi data dimana data dapat ditambahkan dan dihapus. Proses ini selalu dilakukan pada bagian akhir data, yang disebut dengan **top of stack**
- Objek yang **terakhir masuk** ke dalam stack akan menjadi objek yang **pertama keluar** dari stack

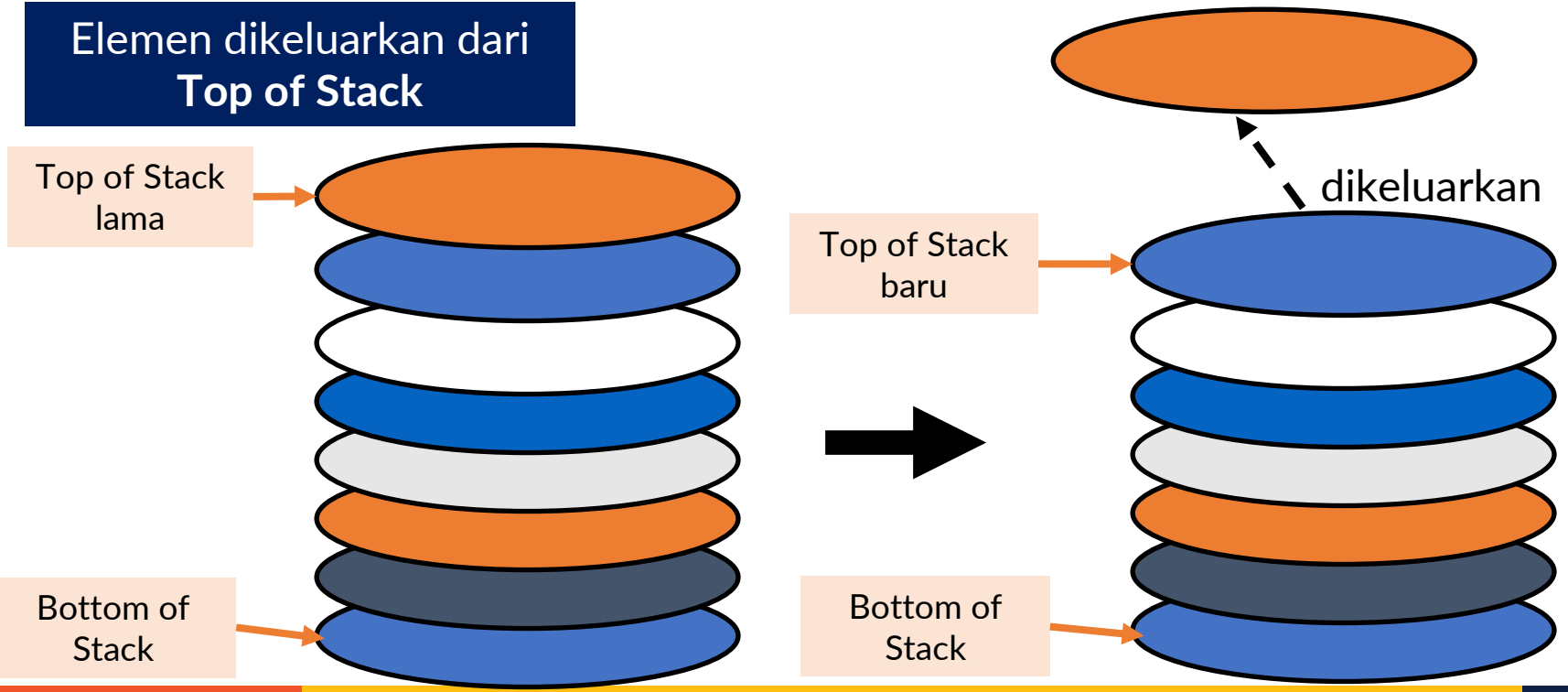


Konsep Stack (Menambah Elemen)



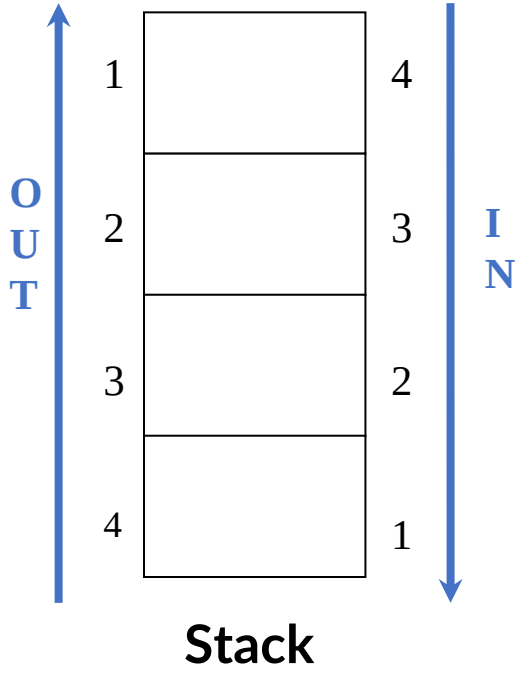
Konsep Stack (Menghapus Elemen)

Elemen dikeluarkan dari **Top of Stack**



Operasi Stack

1. **IsFull**: mengecek apakah stack sudah penuh
2. **IsEmpty**: mengecek apakah stack sudah kosong
3. **Push**: menambah elemen pada stack pada tumpukan paling atas
4. **Pop**: mengambil elemen pada stack pada tumpukan paling atas
5. **Peek**: memeriksa elemen paling atas
6. **Print**: menampilkan seluruh elemen pada stack
7. **Clear**: mengosongkan stack



Cara Kerja Stack

1. Pointer TOP digunakan untuk melacak elemen teratas dalam stack
2. Saat inisialisasi stack, tetapkan nilai TOP = -1 sehingga nanti saat mengecek apakah stack kosong digunakan perbandingan **TOP == -1**
3. Untuk memasukkan (push) elemen, **naikkan** nilai TOP dan tempatkan elemen baru di posisi indeks yang ditunjukkan oleh TOP
4. Saat mengeluarkan (pop) elemen, return elemen yang ditunjuk oleh TOP dan **kurangi** nilai TOP
5. Sebelum melakukan push, cek apakah stack sudah **penuh**
6. Sebelum melakukan pop, cek apakah stack sudah **kosong**

Deklarasi Stack

- Proses pertama yang dilakukan adalah deklarasi atau menyiapkan tempat untuk stack
- Langkah-langkah:
  1. Deklarasi class
  2. Deklarasi atribut
     a. Array data
        digunakan sebagai tempat penyimpanan data
     b. size
        digunakan untuk menentukan kapasitas penyimpanan
     c. Pointer top
        digunakan sebagai penunjuk data pada posisi akhir (atas)

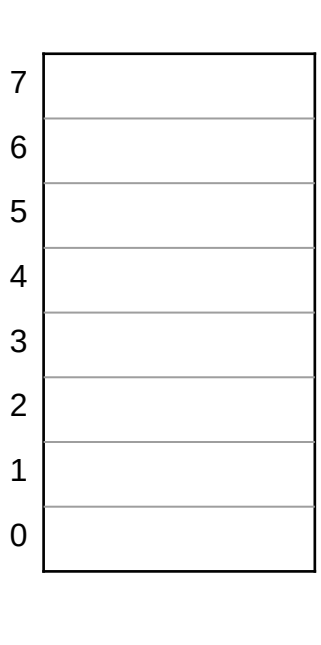
```
public class Stack {
    int data[];
    int size;
    int top;
}
```

Inisialisasi Stack

- Pada mulanya isi **top** dengan -1 karena array dimulai dari 0, yang berarti bahwa data stack dalam keadaan KOSONG
- **Top** adalah suatu variabel penanda dalam stack yang menunjukkan elemen teratas data stack sekarang
- **Top of Stack** akan selalu bergerak hingga mencapai **max** atau **size** yang menyebabkan stack PENUH

Inisialisasi Stack

- Ilustrasi Stack saat inisialisasi pada konstruktor




```
public Stack(int size) {
    this.size = size;
    data = new int[size];
    top = -1;
}
```

Fungsi IsFull

- Untuk memeriksa apakah stack sudah penuh dengan cara memeriksa **top of stack**
- Jika top of stack sudah sama dengan size - 1, maka **full**
- Jika top of stack masih **lebih kecil** dari size - 1, maka belum full

Fungsi IsFull

- Ilustrasi stack saat kondisi Full

size = 8

7	“Multimedia”
6	“Statistika”
5	“Algoritma”
4	“Matematika”
3	“Basis Data”
2	“Komputer”
1	“Android”
0	“Bahasa”

← top = 7

```
public boolean IsFull() {
    if (top == size - 1) {
        return true;
    } else {
        return false;
    }
}
```

Fungsi IsEmpty

- Untuk memeriksa apakah data Stack masih kosong
- Dengan cara memeriksa **top of stack**, jika masih -1 maka berarti data stack masih kosong

```
public boolean IsEmpty() {
    if (top == -1) {
        return true;
    } else {
        return false;
    }

}
```

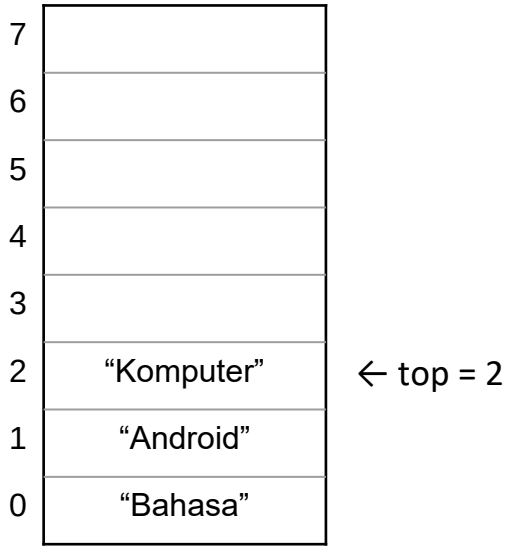
Fungsi Push

- Untuk memasukkan elemen ke data stack. Data yang diinputkan **selalu** menjadi **elemen teratas** stack (yang ditunjuk oleh **top of stack**)
- Jika **data belum penuh**,
  - Tambah satu (increment) nilai **top of stack** lebih dahulu setiap kali ada penambahan ke dalam array data stack
  - Isikan data baru ke stack berdasarkan indeks top of stack yang telah di-increment sebelumnya
- Jika sudah penuh, outputkan "Penuh"

Stack overflow: kondisi yang dihasilkan dari mencoba push elemen ke stack yang sudah penuh

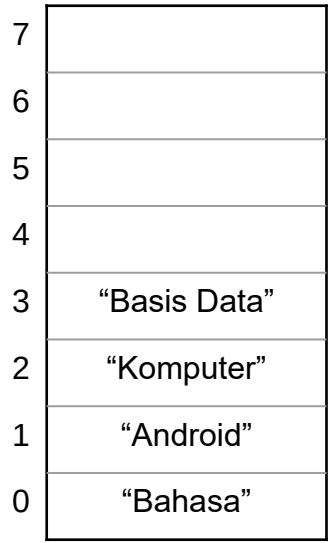
Fungsi Push

size = 8



← top = 2

size = 8



← top = top + 1 = 3

```
public void push(int dt) {
    if (!IsFull()) {
        top++;
        data[top] = dt;
    } else {
        System.out.println("Isi stack penuh!");
    }
}
```

Misalkan data baru “Basis Data” dimasukkan ke dalam Stack “Basis Data”

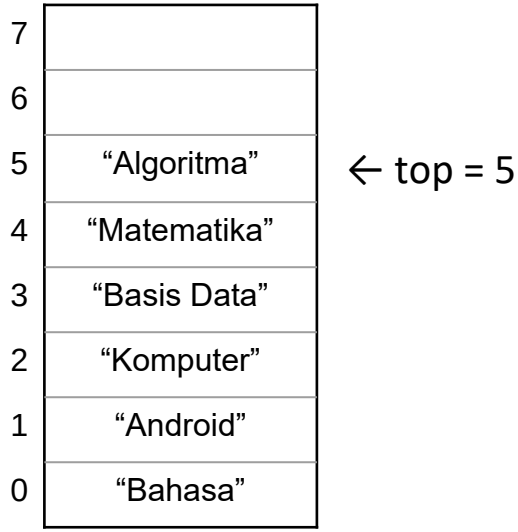
Fungsi Pop

- Untuk mengambil data stack yang terletak paling atas (data yang ditunjuk oleh **top of stack**)
- Jika **data tidak kosong**,
  - **Tampilkan terlebih dahulu** nilai elemen teratas stack dengan mengakses indeksnya sesuai dengan top of stacknya
  - Lakukan decrement nilai top of stacknya sehingga jumlah elemen stack berkurang
- Jika data kosong, outputkan “Kosong”

Stack underflow: kondisi yang dihasilkan dari mencoba pop elemen dari stack yang masih kosong

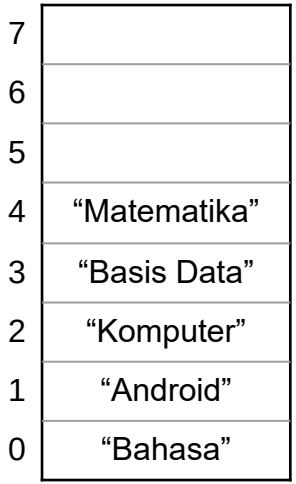
Fungsi Pop

size = 8



← top = 5

size = 8



← top = top - 1 = 4

Data “Algoritma” pada posisi teratas dihapus

```
public void pop() {
    if (!IsEmpty()) {
        int x = data[top];
        top--;
        System.out.println("Data yang keluar: " + x);
    } else {
        System.out.println("Stack masih kosong");
    }
}
```

Fungsi Peek

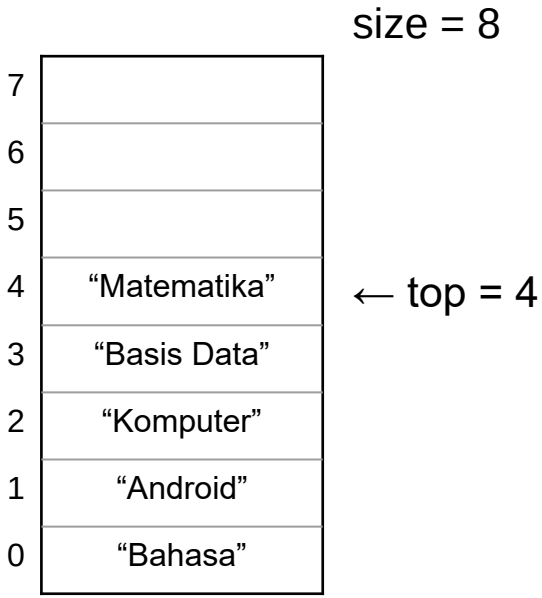
- Untuk mengakses elemen yang ditunjuk oleh top of stack, yaitu elemen yang terakhir kali ditambahkan
- Operasi ini berbeda dengan pop karena tidak disertai dengan penghapusan data, namun hanya pengaksesan (pengembalian) data saja

```
public void peek() {
    System.out.println("Elemen teratas: " + data[top]);
}
```

Fungsi Print

- Untuk menampilkan semua elemen-elemen data stack
- Dengan cara melakukan *looping* pada semua nilai array secara **terbalik**, karena pengaksesan elemen dimulai dari indeks array terbesar terlebih dahulu baru ke indeks yang lebih kecil

Fungsi Print



Pada proses print, pembacaan elemen stack dimulai dari indeks **top** sampai dengan indeks **0**
Hasilnya:
Matematika, Basis Data, Komputer, Android, Bahasa

```
public void print() {
    System.out.println("Isi stack: ");
    for (int i = top; i >= 0; i--) {
        System.out.println(data[i] + " ");
    }
    System.out.println("");
}
```

Fungsi Clear

- Untuk mengosongkan stack dengan cara mengeluarkan seluruh elemen stack

```
public void clear() {
    if (!IsEmpty()) {
        for (int i = top; i >= 0; i--) {
            top--;
        }
        System.out.println("Stack sudah dikosongkan");
    } else {
        System.out.println("Gagal! Stack masih kosong");
    }
}
```

Postfix Expressions

Expressions

Penerapan stack pada bidang aritmatika adalah penulisan ekspresi matematika, yang terdiri dari tiga jenis:

- **Notasi infix** dengan ciri-ciri:
  - Operator berada di antara operand: 3 + 4 * 2
  - Tanda kurung lebih diutamakan: (3 + 4) * 2
- **Notasi prefix**: operator dituliskan sebelum dua operand
- **Notasi postfix**: operator dituliskan setelah dua operand
- Contoh:
  - 3 + 4 * 2 → + 3 * 4 2 → 3 4 2 * +
  - (3 + 4) * 2 → * + 3 4 2 → 3 4 + 2 *

Infix Prefix Postfix

Postfix Expressions

- Biasanya, ekspresi matematika ditulis menggunakan notasi infix, namun notasi postfix adalah notasi yang digunakan oleh mesin kompilasi komputer untuk mempermudah proses pengodean. Contoh penerapannya pada kalkulator di handphone
- Ketika operand dimasukkan, maka kalkulator
  - Melakukan **push** ke dalam stack
- Ketika operator dimasukkan, maka kalkulator
  - Menerapkan operator untuk dua operand teratas pada stack
  - Melakukan pop operand dari stack
  - Melakukan push hasil operasi perhitungan ke dalam stack

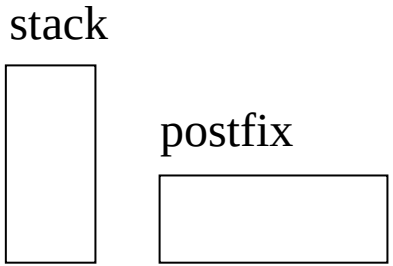
Derajat Operator Aritmatika

Urutan derajat operator aritmatika:

- Pangkat ^
- Perkalian * setara dengan pembagian / dan modulo %
- Penjumlahan + setara dengan pengurangan -

Algoritma Konversi Infix ke Postfix

Buat dan inisialisasi stack untuk menampung operator
WHILE ekspresi mempunyai token (operator dan operand) **DO**
    IF token adalah **operand**, **THEN** tambahkan ke string **postfix**
    ELSE IF token adalah tanda kurung tutup '**)**', **THEN**
        WHILE tanda kurung buka '(' belum ditemukan
            Pop operator dari stack
            Tambahkan ke string **postfix**
        END WHILE
        Hapus tanda kurung buka '(' yang ditemukan
    END IF
    IF token selanjutnya adalah tanda kurung buka '**(**', **THEN** **push** ke stack
    ELSE IF token adalah **operator**, **THEN**
        WHILE (stack is not empty) **AND** (derajat operator Top >= operator saat ini) **DO**
            Pop operator dari stack
            Tambahkan ke string **postfix**
        END WHILE
        Push operator ke stack
    END IF
END WHILE



Setelah persamaan infix terbaca, pindahkan semua isi stack (yang tersisa) ke postfix
WHILE stack is not empty
    Pop operator dari stack
    Tambahkan ke string **postfix**
END WHILE

Studi Kasus 1

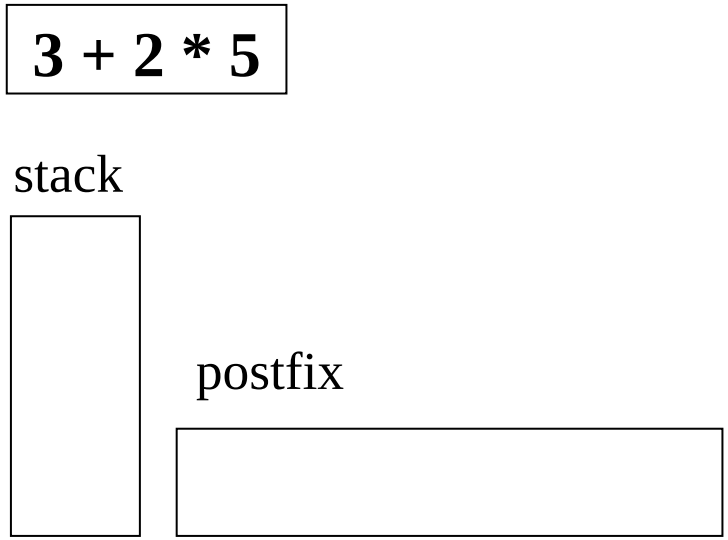
- Misalkan terdapat persamaan:

  3 + 2 * 5

- Operasi di atas disebut notasi **infix**, notasi infix tersebut harus diubah menjadi notasi **postfix**

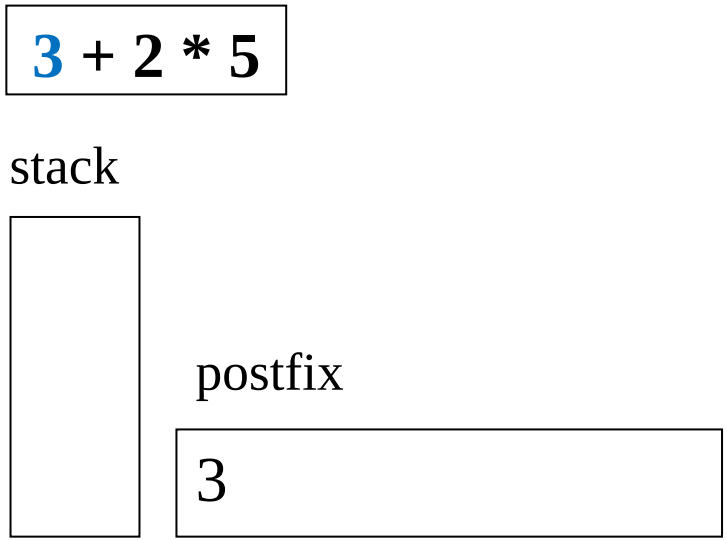
Studi Kasus 1 - Penyelesaian

- Baca persamaan dari kiri ke kanan



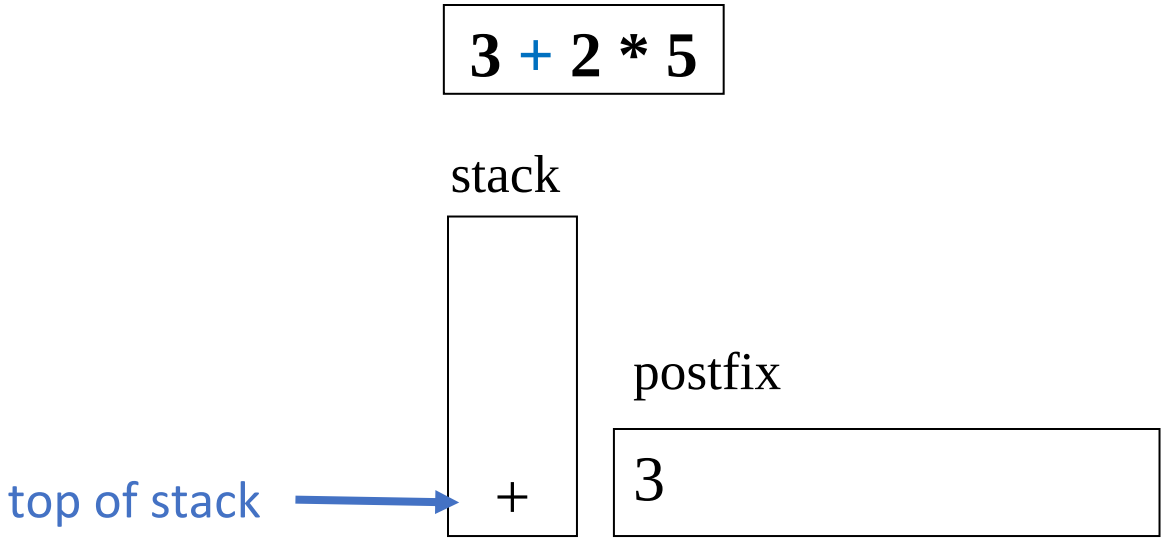
Studi Kasus 1 - Penyelesaian

- Langkah 1: Operand 3
  Masukkan ke postfix



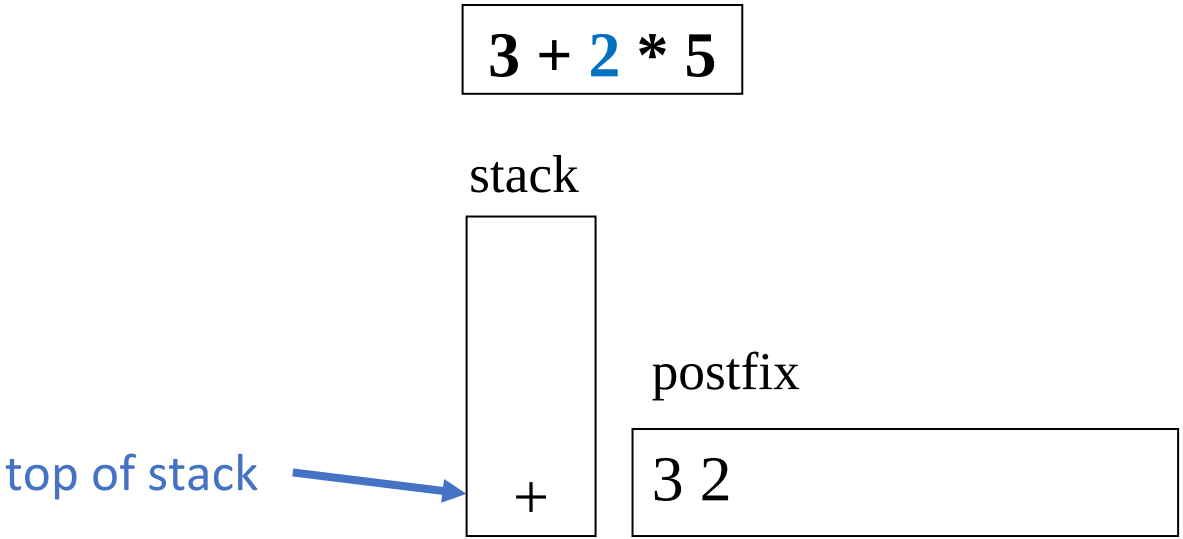
Studi Kasus 1 - Penyelesaian

- Langkah 2: Operator +
  Push ke stack karena stack masih kosong



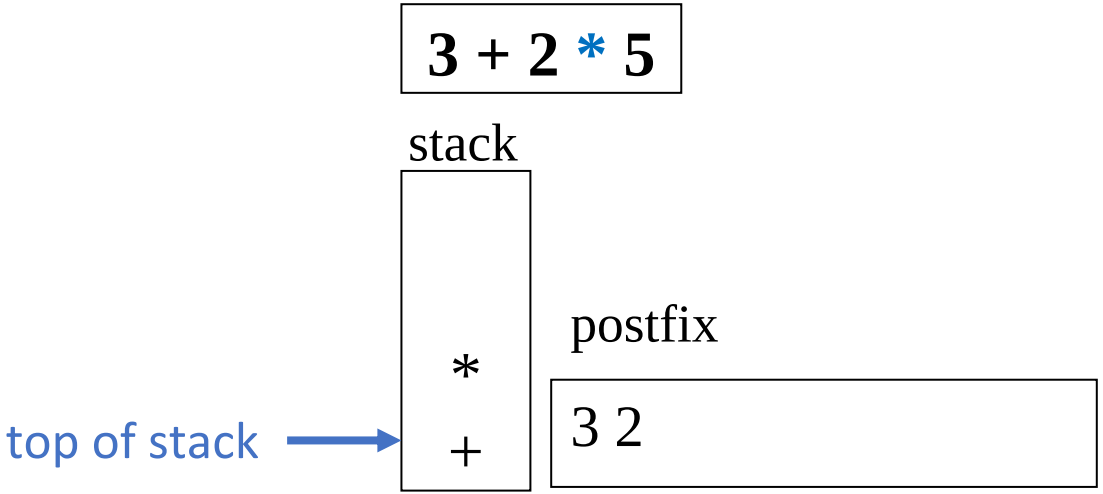
Studi Kasus 1 - Penyelesaian

- Langkah 3: Operand 2
  Masukkan ke postfix



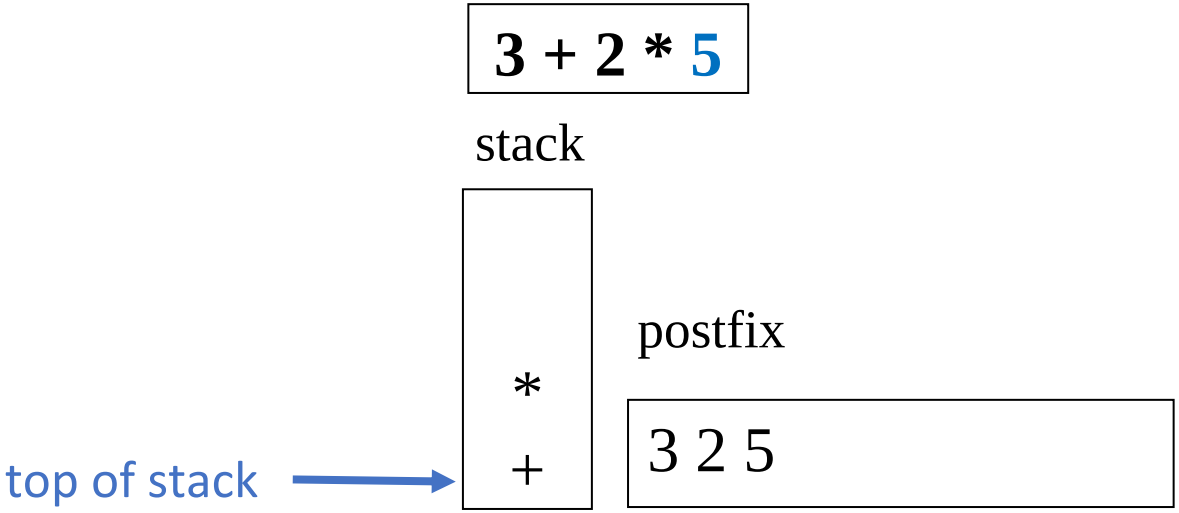
Studi Kasus 1 - Penyelesaian

- Langkah 4: Operator *
  Push ke stack karena derajat operator Top of stack + **lebih kecil** dari derajat operator *



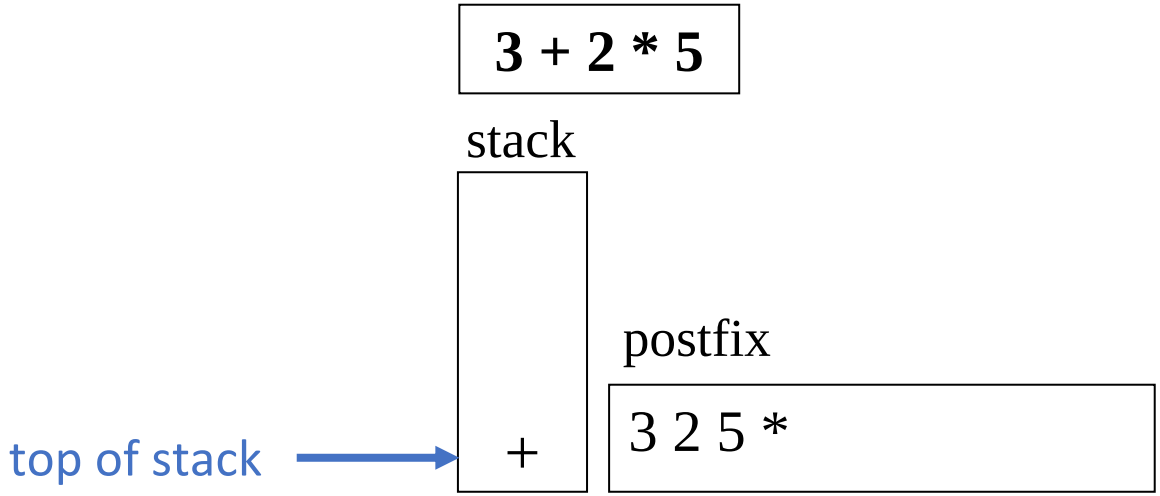
Studi Kasus 1 - Penyelesaian

- Langkah 5: Operand 5
  Masukkan ke postfix



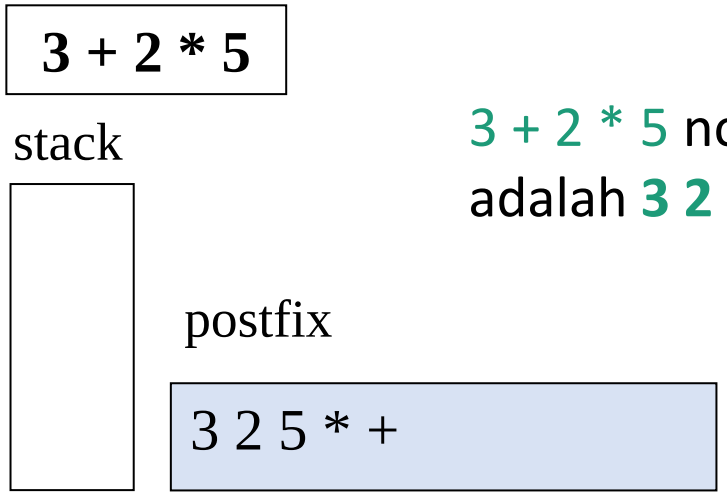
Studi Kasus 1 - Penyelesaian

- Langkah 6
  Semua persamaan sudah terbaca, pop semua isi stack dan masukkan ke postfix secara berurutan, yaitu operator * terlebih dahulu



Studi Kasus 1 - Penyelesaian

- Langkah 7
  Setelah dilakukan pop pada operator * dan dimasukkan ke postfix, selanjutnya dilakukan pop pada operator + dan dimasukkan ke postfix



3 + 2 * 5 notasi postfix-nya adalah **3 2 5 * +**

Studi Kasus 2

- Misalkan terdapat persamaan:

  15 – (7 + 4) / 3

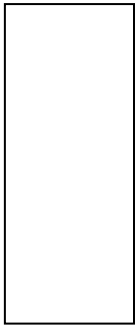
- Operasi di atas disebut notasi **infix**, notasi infix tersebut harus diubah menjadi notasi **postfix**

Studi Kasus 2 - Penyelesaian

- Baca persamaan dari kiri ke kanan

15 – (7 + 4) / 3

stack



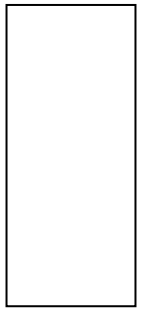
postfix

Studi Kasus 2 - Penyelesaian

- Langkah 1: Operand 15
  Masukkan ke postfix

15 – (7 + 4) / 3

stack



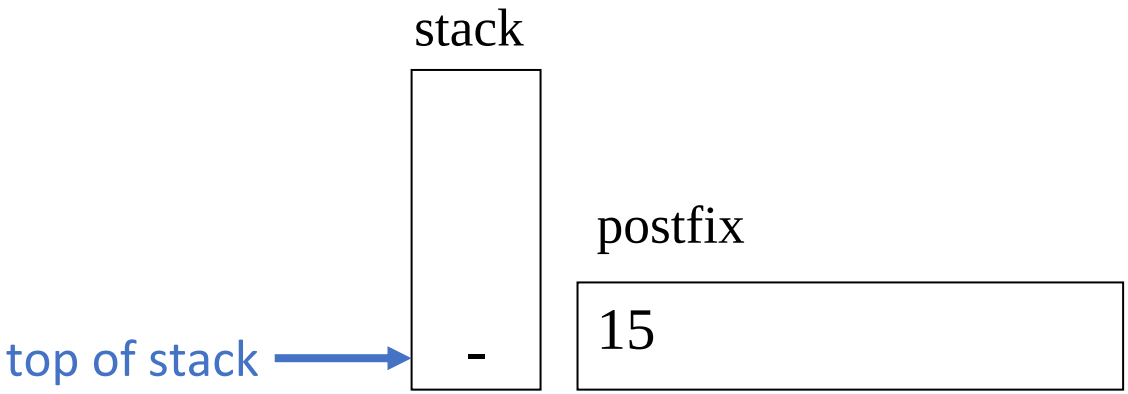
postfix

15

Studi Kasus 2 - Penyelesaian

- Langkah 2: Operator –
  Push ke stack karena stack masih kosong

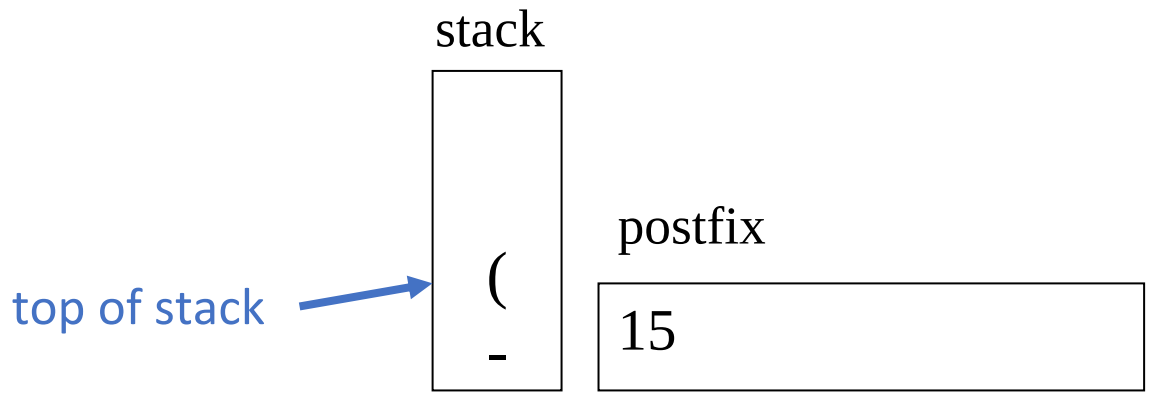
15 – (7 + 4) / 3



Studi Kasus 2 - Penyelesaian

- Langkah 3: Tanda (
  Push ke stack

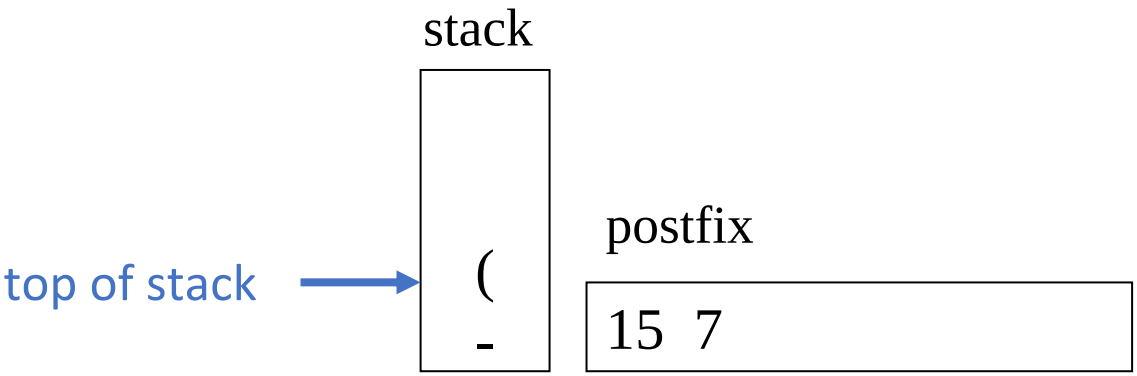
15 – (7 + 4) / 3



Studi Kasus 2 - Penyelesaian

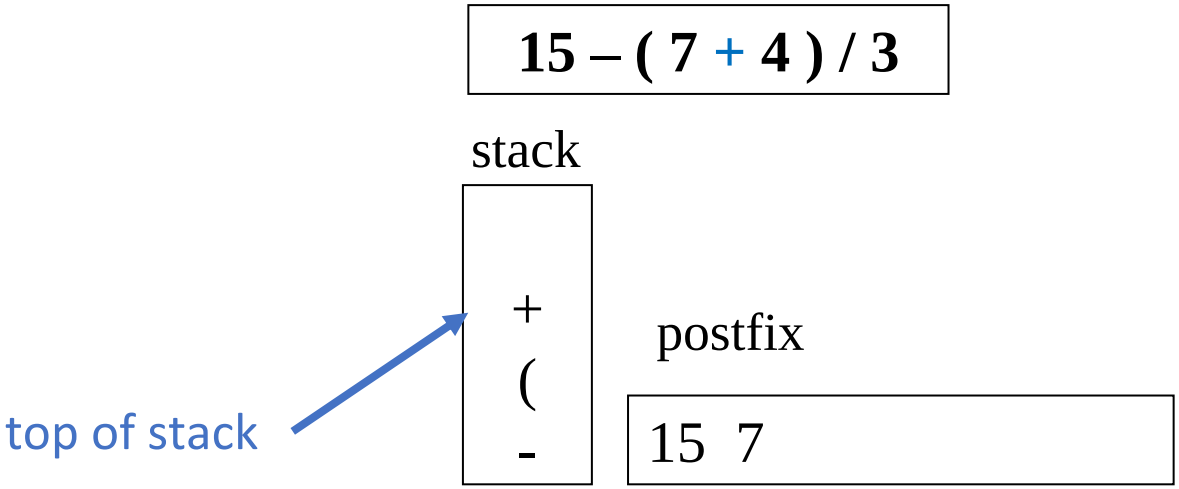
- Langkah 4: Operand 7
  Masukkan ke postfix

15 – (7 + 4) / 3



Studi Kasus 2 - Penyelesaian

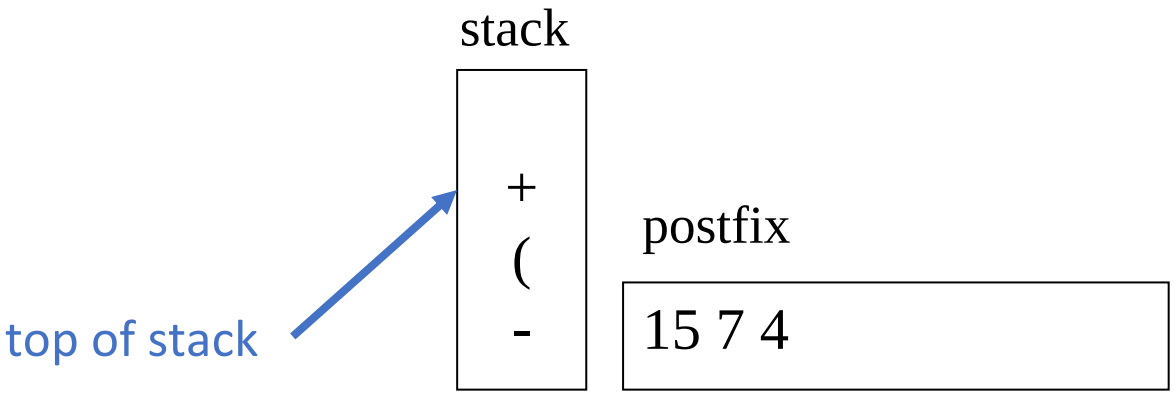
- Langkah 5: Operator +
  Push ke stack karena derajat operator Top of stack (**lebih kecil** dari derajat operator +



Studi Kasus 2 - Penyelesaian

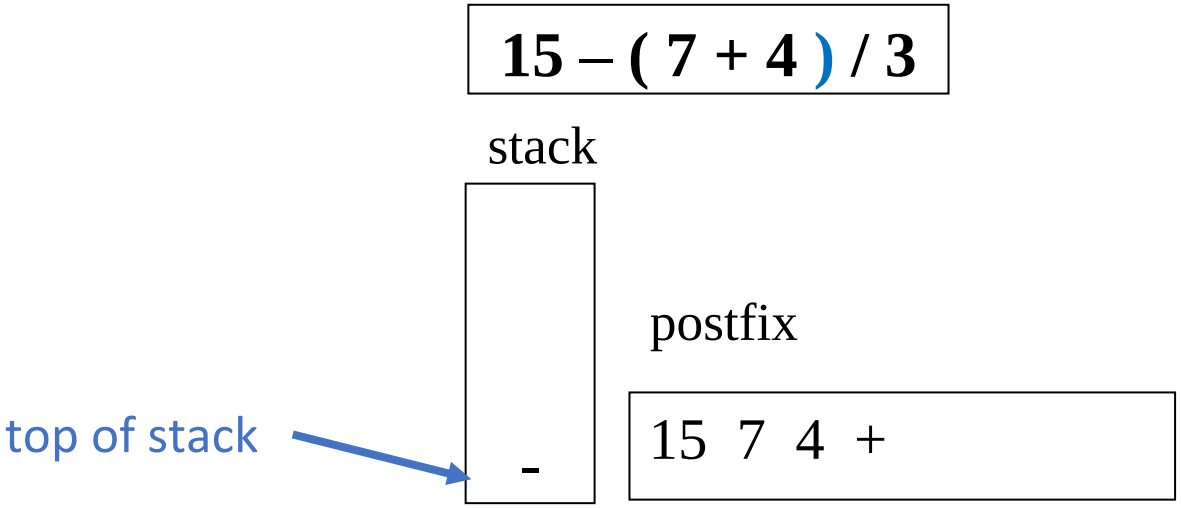
- Langkah 6: Operand 4
  Masukkan ke postfix

15 – (7 + 4) / 3



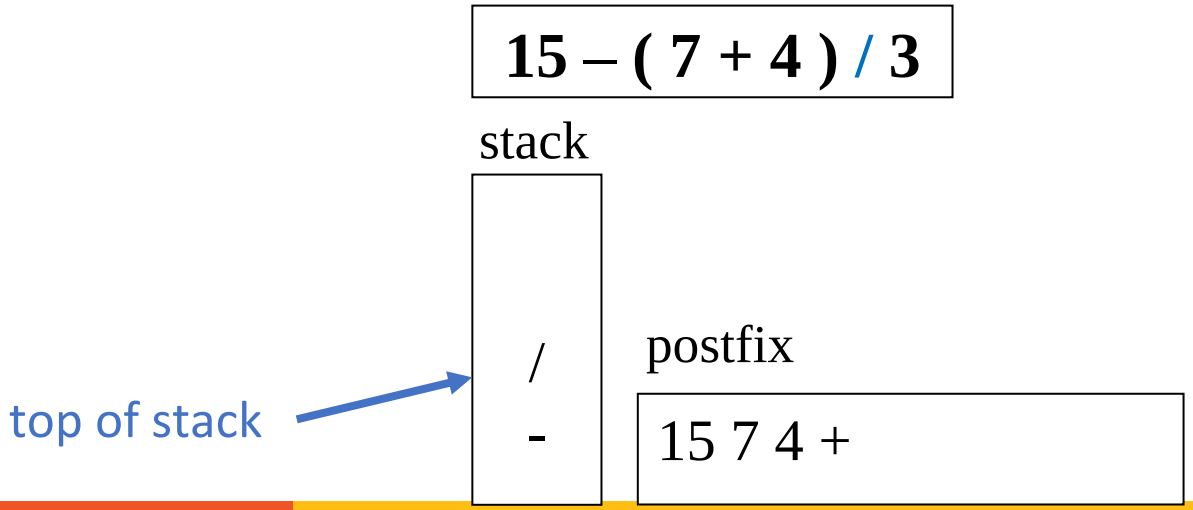
Studi Kasus 2 - Penyelesaian

- Langkah 7: Tanda)
  Pop isi stack yaitu operator +, kemudian masukkan ke postfix.
  Tanda (hanya di-pop, tidak perlu dimasukkan ke postfix



Studi Kasus 2 - Penyelesaian

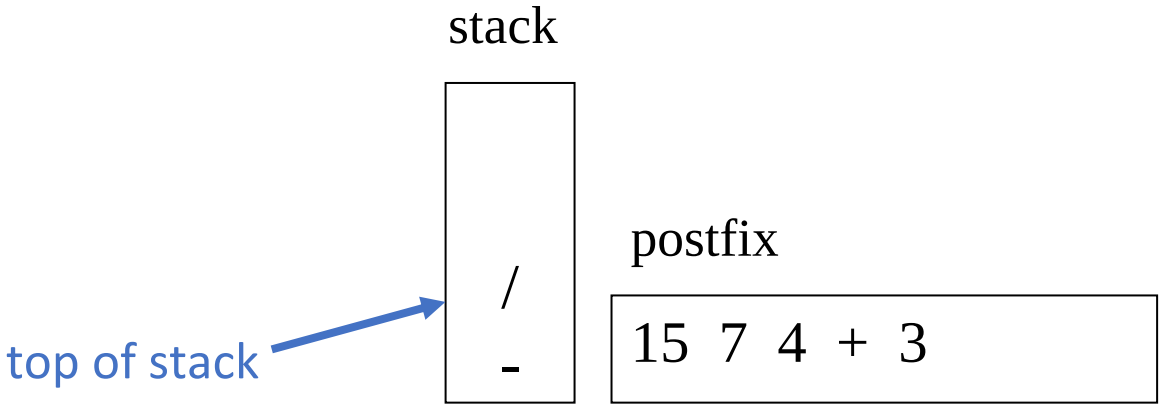
- Langkah 8: Operator /
  Push ke stack karena derajat operator Top of stack – **lebih kecil** dari derajat oprator /



Studi Kasus 2 - Penyelesaian

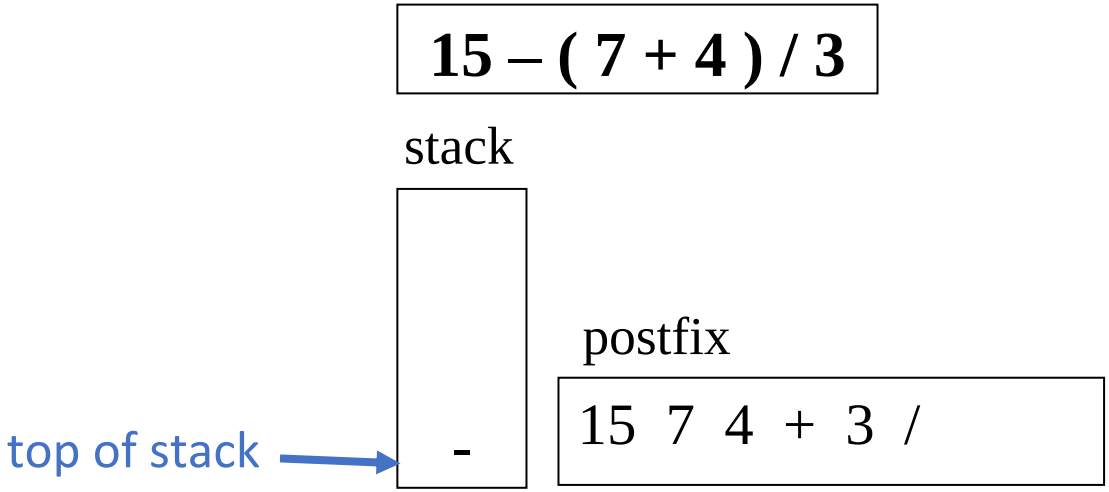
- Langkah 9: Operand 3
  Masukkan ke postfix

15 – (7 + 4) / 3



Studi Kasus 2 - Penyelesaian

- Langkah 10
  Semua ekspresi sudah terbaca, pop semua isi stack dan masukkan ke postfix secara berurutan, yaitu operator / terlebih dahulu

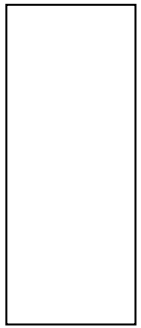


Studi Kasus 2 - Penyelesaian

- Langkah 11
  Setelah dilakukan pop pada operator / dan dimasukkan ke postfix, selanjutnya dilakukan pop pada operator – dan dimasukkan ke postfix

15 – (7 + 4) / 3

stack



15 – (7 + 4) / 3 notasi postfix-nya adalah **15 7 4 + 3 / –**

postfix

15 7 4 + 3 / -

Menghitung Ekspresi Matematika yang disusun dalam Notasi Postfix

Studi Kasus 2 - Penyelesaian

- Ekspresi matematika yang tersusun dalam bentuk notasi postfix dapat dihitung hasil akhirnya
- Contoh:

15 5 +

Hasil: 20

30 6 / 5 +

Hasil: 10

Menghitung Ekspresi Matematika

Diasumsikan **P** adalah ekspresi matematika yang ditulis dalam notasi postfix dan variabel **value** sebagai penampung hasil akhir.

1. Membaca ekspresi P dari kiri ke kanan, ulangi langkah 2 dan 3 untuk setiap elemen P
2. Jika bertemu dengan operand, push stack.
3. Jika bertemu dengan operator (opt), maka:
   - pop operand teratas dari stack, simpan dalam var1
   - Pop operand teratas dari stanck, simpan dalam var2
   - lakukan operasi (var2 opt var1), simpan hasil di variabel hitung,
   - push variable hitung ke dalam stack.
4. Pop isi stack dan simpan di variabel value.

Studi Kasus

- Misalkan terdapat persamaan matematika dalam bentuk notasi postfix

  P = 5 2 6 + * 12 4 / -

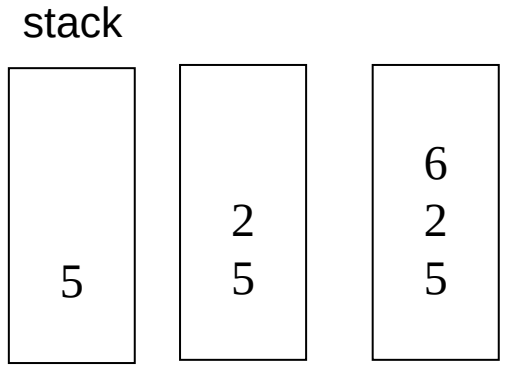
- Persamaan matematika tersebut dapat dihitung hasilnya tanpa perlu mengubah menjadi notasi infix

Studi Kasus

- Membaca ekspresi P dari kiri ke kanan:

P	5 2 6 + * 12 4 / -

- 5 adalah operand, push ke stack.
- 2 adalah operand, push ke stack.
- 6 adalah operand, push ke stack.



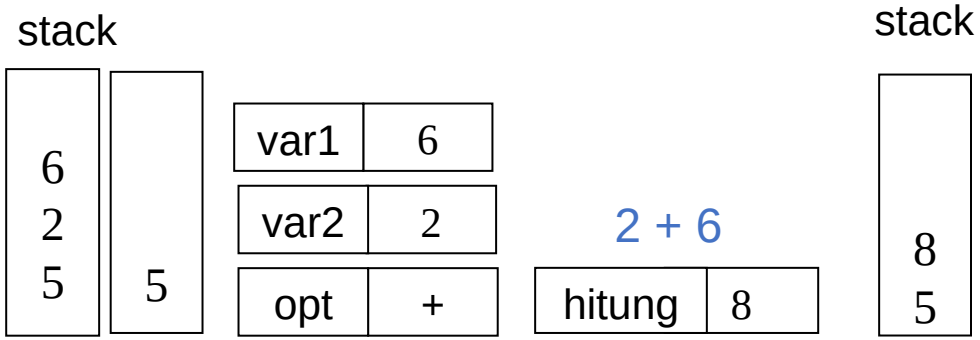
var1	
var2	

Opt	
hitung	

Studi Kasus

P	5 2 6 + * 12 4 / -

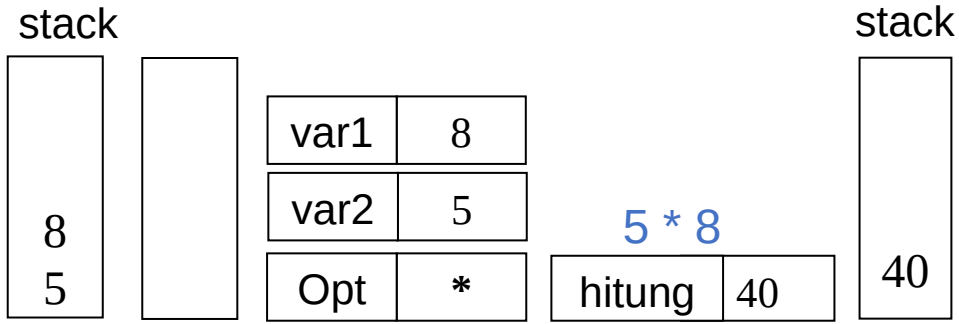
- + adalah operator, pop dua operand teratas dari stack (6 dan 2), simpan 6 ke var1 dan simpan 2 ke var2. Lakukan operasi **var2 opt var1** (2 + 6 = 8), simpan hasilnya dalam variabel hitung, push variable hitung ke stack.



Studi Kasus

P	5 2 6 + * 12 4 / -

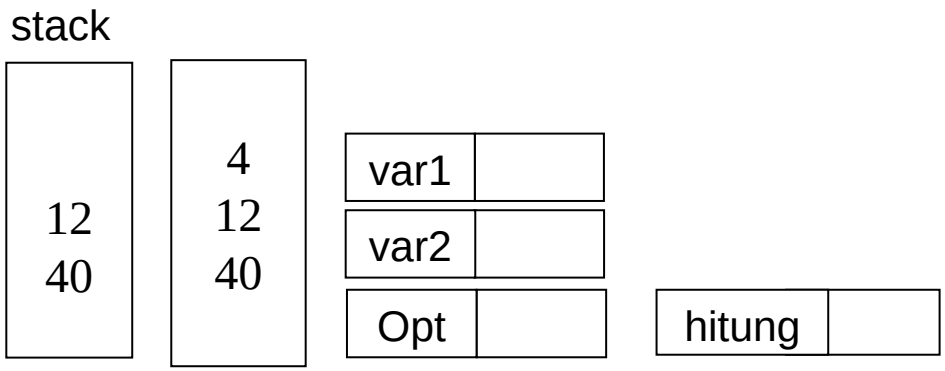
- * adalah operator, pop dua operand teratas dari stack (8 dan 5), simpan 8 ke var1 dan simpan 5 ke var2. Lakukan operasi **var2 opt var1** (5 * 8 = 40), simpan hasilnya dalam variabel hitung, push variabel hitung ke stack.



Studi Kasus

P	5 2 6 + * 12 4 / -

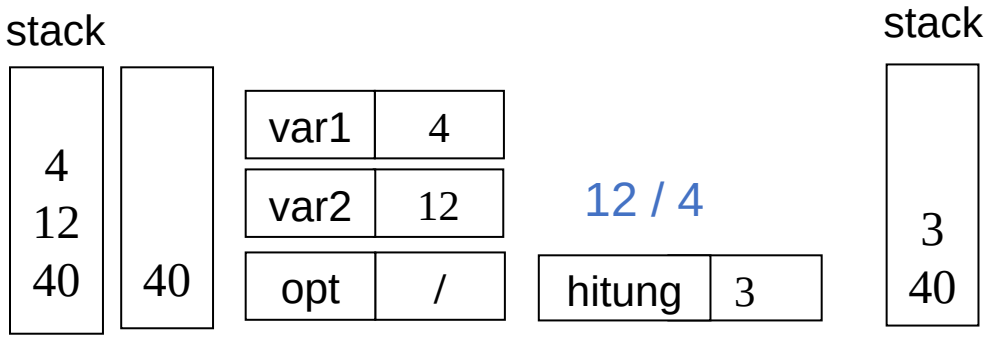
- 12 adalah operand, push ke stack.
- 4 adalah operand, push ke stack.



Studi Kasus

P	5 2 6 + * 12 4 / -

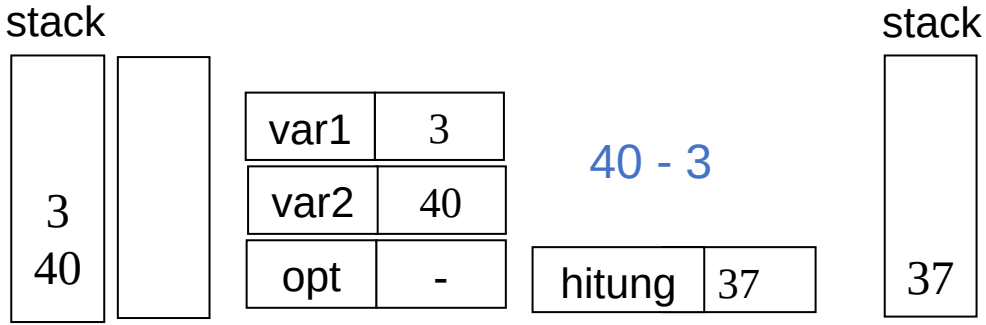
- / adalah operator, pop dua operand teratas dari stack (4 dan 12), simpan 4 ke var1, simpan 12 ke var2. Lakukan operasi var2 opt var1 (12 / 4 = 3), simpan hasilnya dalam variabel hitung, push variabel hitung ke stack.



Studi Kasus

P	5 2 6 + * 12 4 / -

- - adalah operator, pop dua operand teratas dari stack (3 dan 40), simpan 3 ke var1 dan simpan 40 ke var2. Lakukan operasi **var2 opt var1** (40 - 3 = 37), dan simpan hasilnya dalam variabel hitung, push variabel hitung ke stack.



- Pop isi stack dan simpan di variabel value sebagai hasil akhir.
- Persamaan P = 5 2 6 + *12 4 / -, hasil akhirnya adalah value = 37

Latihan

1. Tuliskan langkah-langkah pengerjaan dari beberapa rangkaian operasi stack berikut:
   1) Push(10)
   2) Push(6)
   3) Pop()
   4) Push(8)
   5) Push(2)
   6) Pop()
   7) Pop()
   8) Push(4)

   Jika kondisi kondisi awal stack kosong, berapa nilai top saat ini (setelah langkah 8)?
2. Lakukan konversi notasi infix berikut menjadi notasi postfix!
   a. x + y / z – w
   b. 28 / 2 % 7 + 12
   c. 4 * (7 – 4 + 1) ^ 3

Latihan

3. Hitung hasil ekspresi matematika berikut
   a. 15 2 * 2 / 6 –
   b. 27 12 5 % 3 * –