



Pertemuan 9

Stack

Tim Ajar Struktur Data 2021

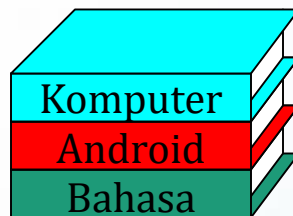
Definisi Stack

- 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

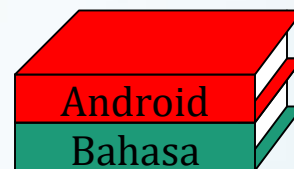
Konsep Stack

- 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

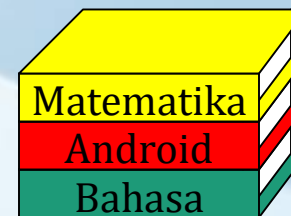
Keadaan awal



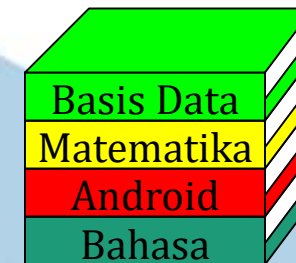
Setelah
mengambil
"Komputer"



Setelah
menambah
"Matematika"

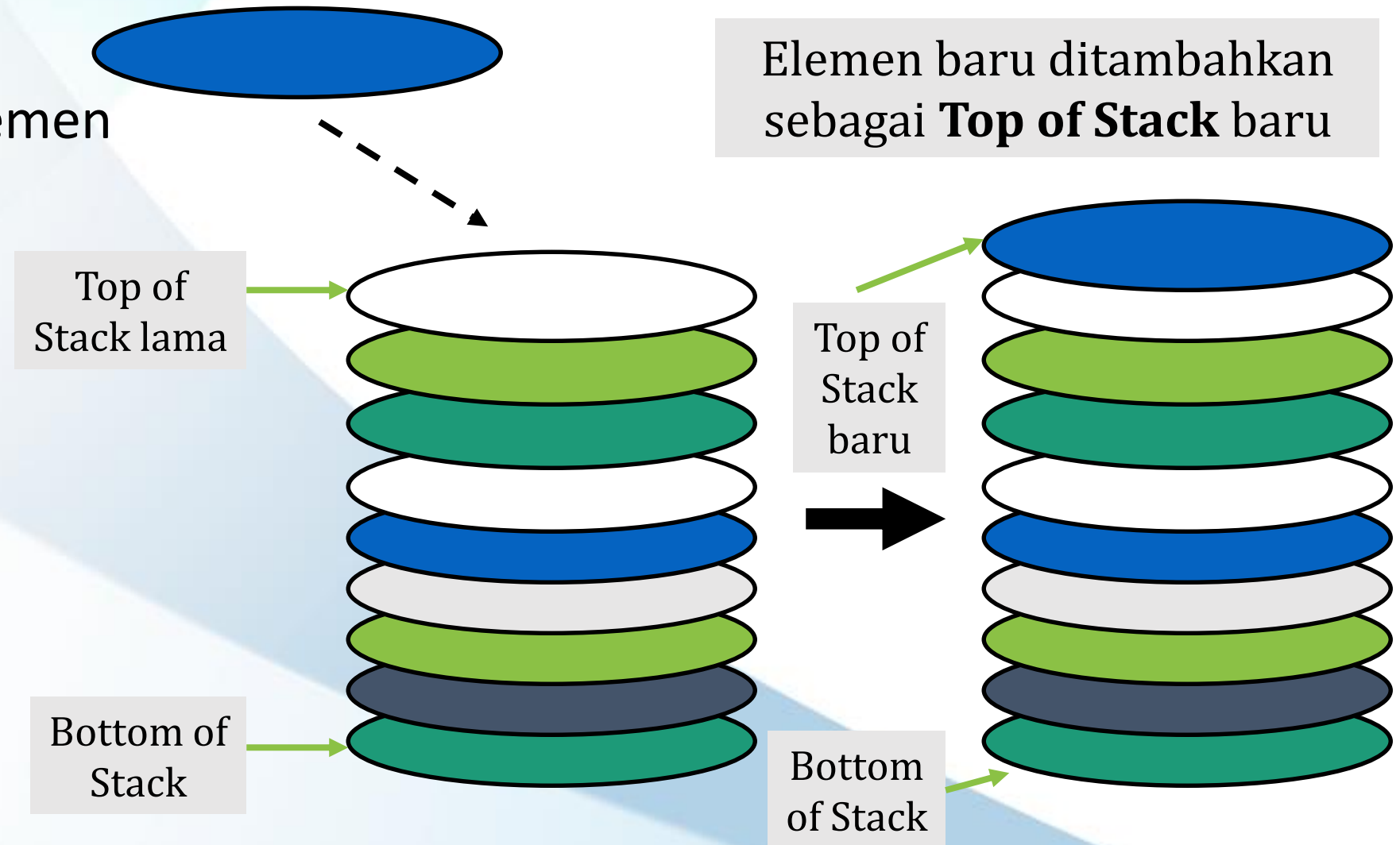


Setelah
menambah
"Basis Data"



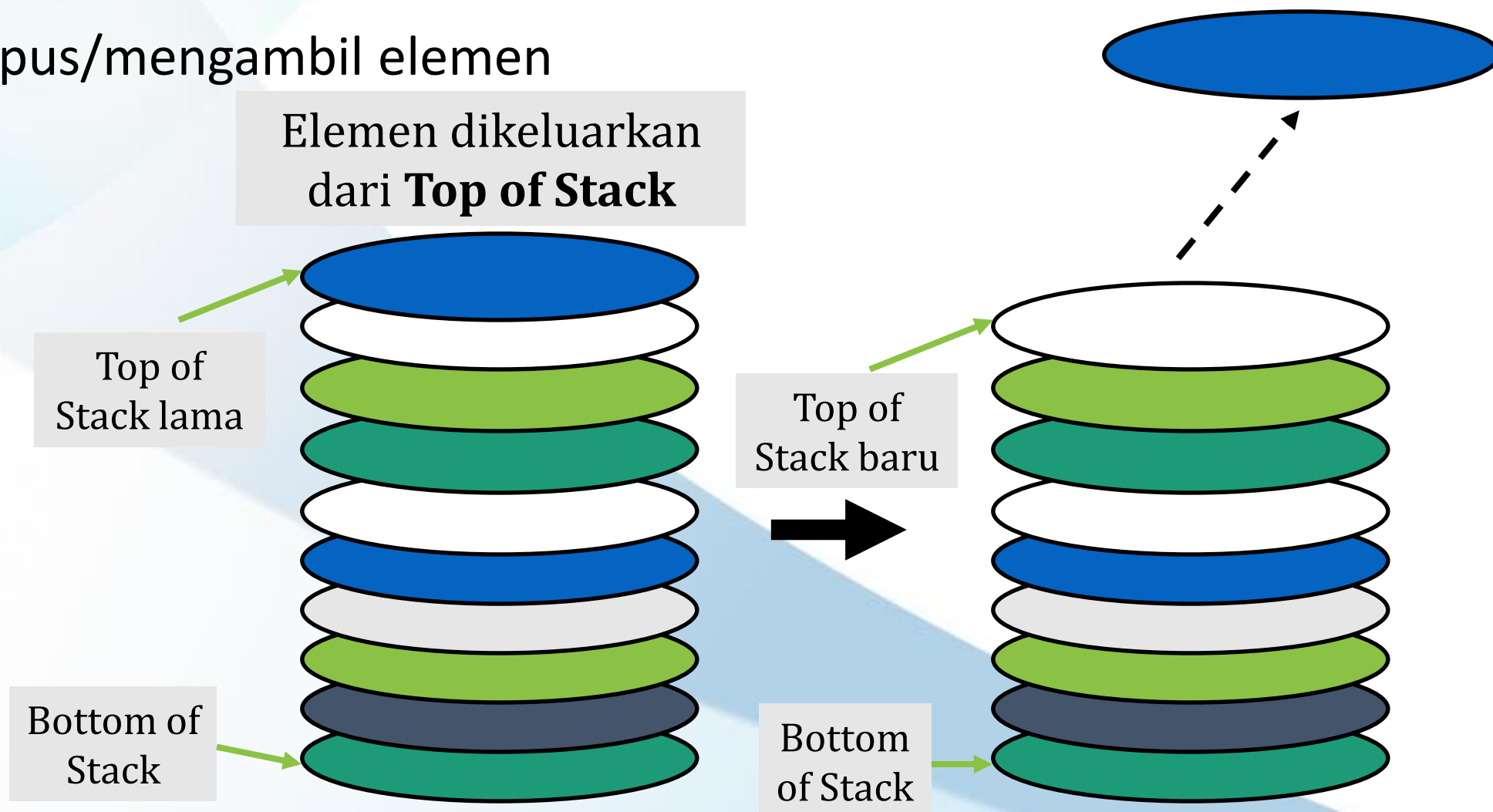
Konsep Stack

- Menambah elemen



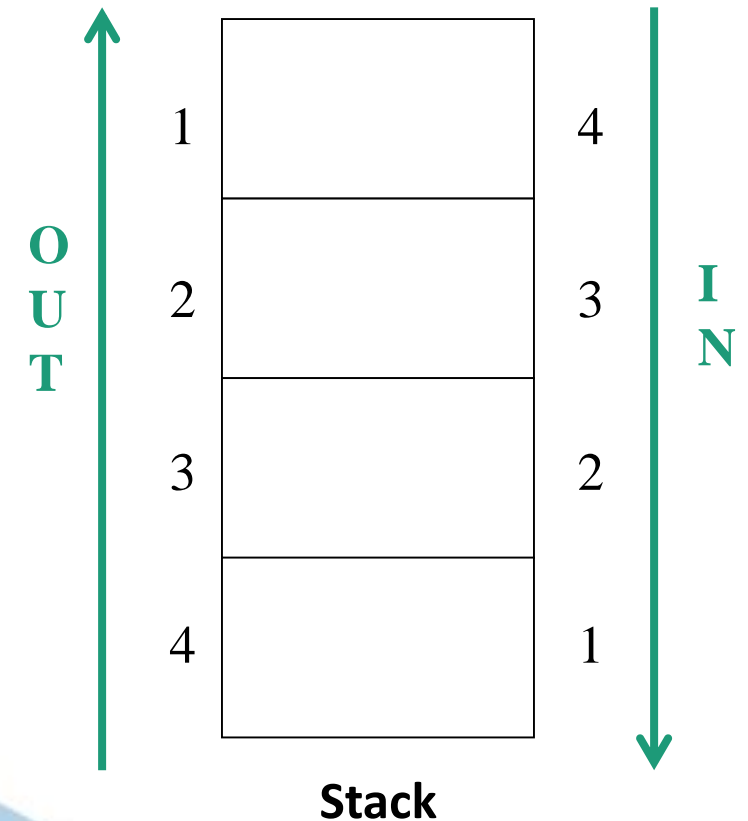
Konsep Stack

- Menghapus/mengambil elemen



Operasi Stack

- **IsFull**: mengecek apakah stack sudah penuh
- **IsEmpty**: mengecek apakah stack sudah kosong
- **Push**: menambah elemen pada stack pada tumpukan paling atas
- **Pop**: mengambil elemen pada stack pada tumpukan paling atas
- **Peek**: memeriksa elemen paling atas
- **Print**: menampilkan seluruh elemen pada stack
- **Clear**: mengosongkan stack



Deklarasi Stack

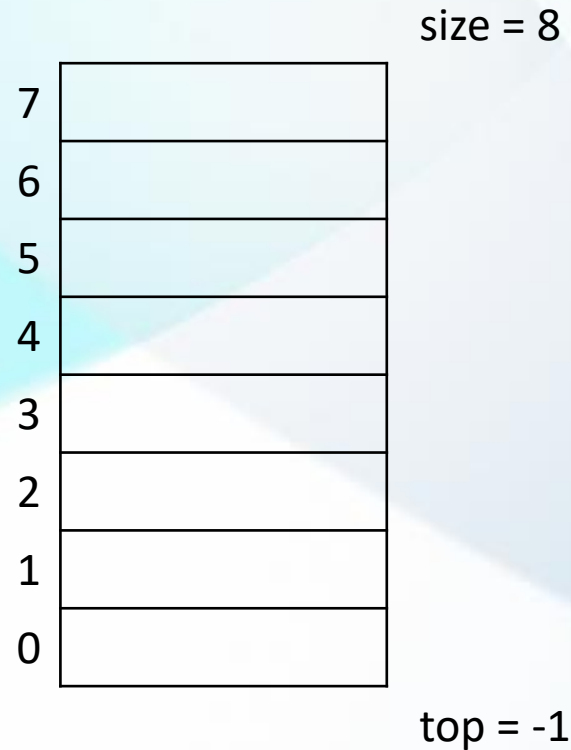
- Proses pertama yang dilakukan adalah deklarasi atau menyiapkan tempat untuk stack
- Langkah-langkah:
 - Deklarasi class
 - Deklarasi array digunakan sebagai tempat penyimpanan
 - Deklarasi size digunakan untuk menentukan kapasitas penyimpanan
 - Deklarasi pointer top digunakan sebagai penunjuk data 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



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

- Ilustrasi Stack pada saat inisialisasi

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

7	"Multimedia"	size = 8 ← top = 7
6	"Statistika"	
5	"Algoritma"	
4	"Matematika"	
3	"Basis Data"	
2	"Komputer"	
1	"Android"	
0	"Bahasa"	

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

- Ilustrasi stack pada kondisi Full

Fungsi IsEmpty

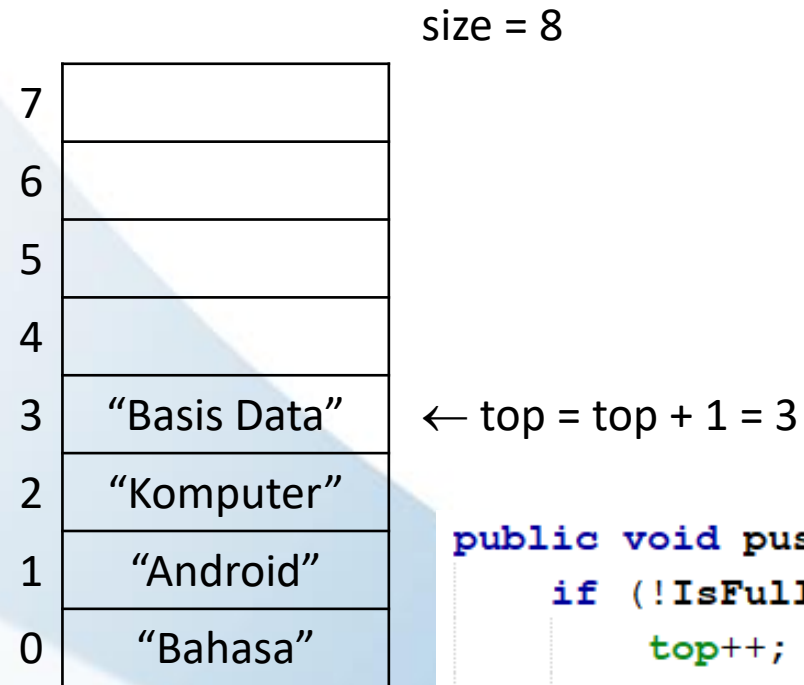
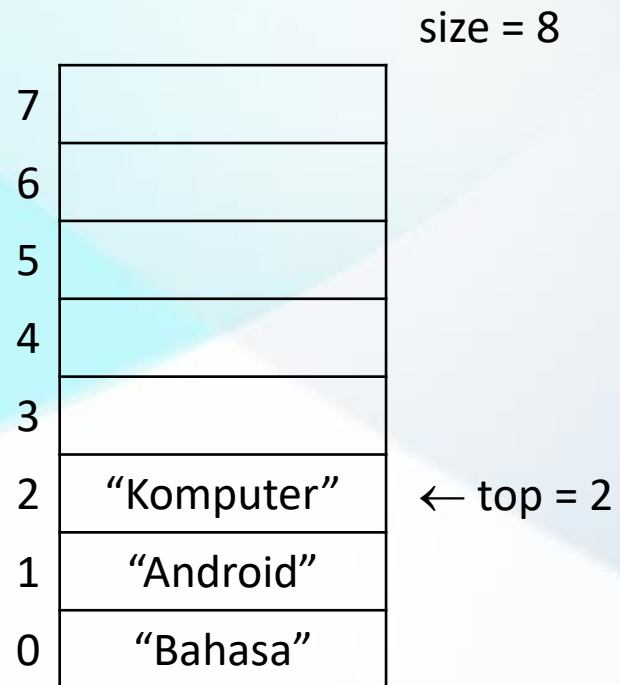
- Untuk memeriksa apakah data Stack masih **kosong**
- Dengan cara memeriksa **top of stack**, jika masih -1 maka berarti data stack masih kosong!

```
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 tidak, outputkan “Penuh”

Fungsi Push



Misalkan data baru "Basis Data"
dimasukkan ke dalam Stack

"Basis Data"

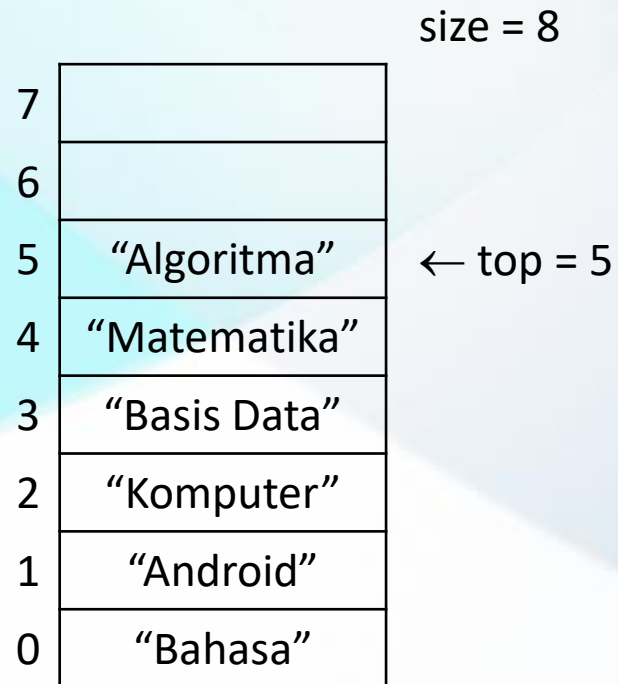
```
public void push(int dt) {  
    if (!IsFull()) {  
        top++;  
        data[top] = dt;  
    } else {  
        System.out.println("Isi stack penuh!");  
    }  
}
```



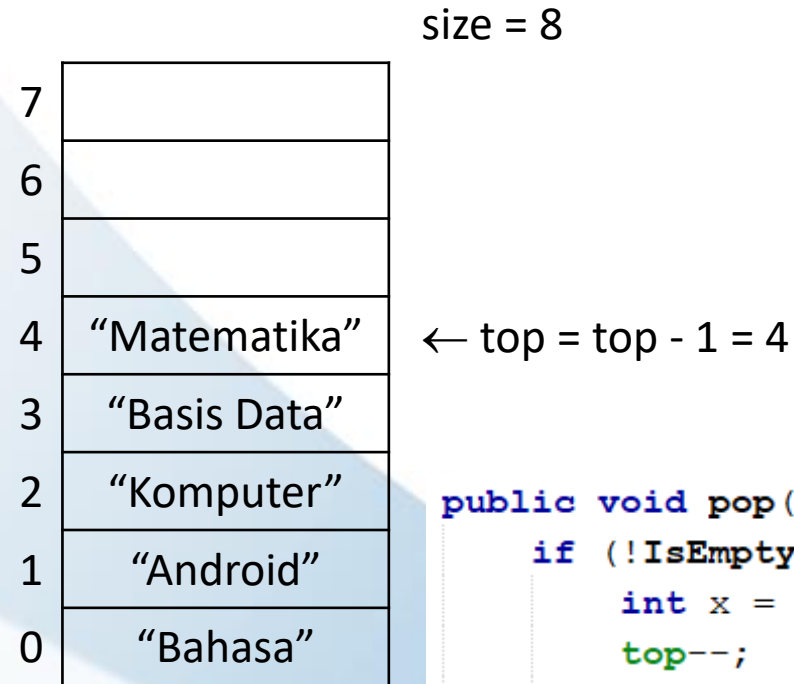
Fungsi Pop

- Untuk mengambil data stack yang terletak paling atas (data yang ditunjuk oleh **top of stack**)
- **Tampilkan terlebih dahulu** nilai elemen teratas stack dengan mengakses indeksnya sesuai dengan top of stacknya, baru dilakukan **decrement** nilai top of stacknya sehingga jumlah elemen stack berkurang

Fungsi Pop



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

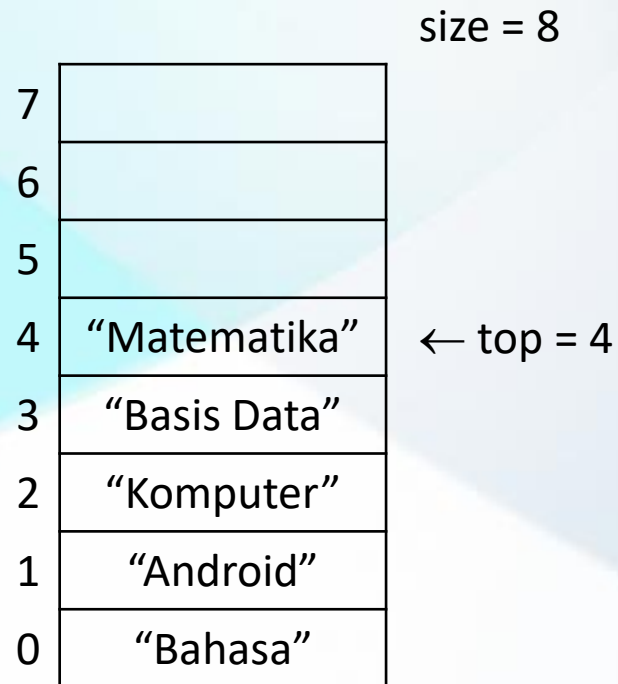
```
public void peek() {  
    System.out.println("Elemen teratas: " + data[top]);  
}
```



Fungsi Print

- Untuk menampilkan semua elemen-elemen data stack
- Dengan cara melakukan *looping* pada semua nilai array secara **terbalik**, karena kita harus mengakses dari indeks array tertinggi 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 Hp
- Ketika operand diinputkan, maka kalkulator
 - Melakukan push ke dalam stack
- Ketika operator diinputkan, 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 inialisasi 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

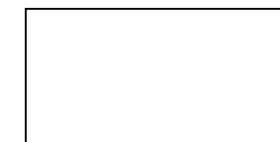
END IF

END WHILE

stack



postfix



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

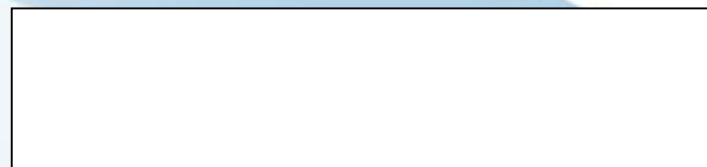
- Baca persamaan dari kiri ke kanan

$$3 + 2 * 5$$

stack



postfix



Studi Kasus 1

- Langkah 1: Operand 3
Masukkan ke postfix

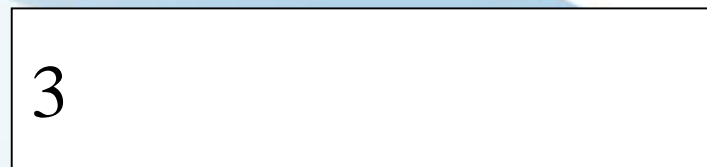
3 + 2 * 5

stack



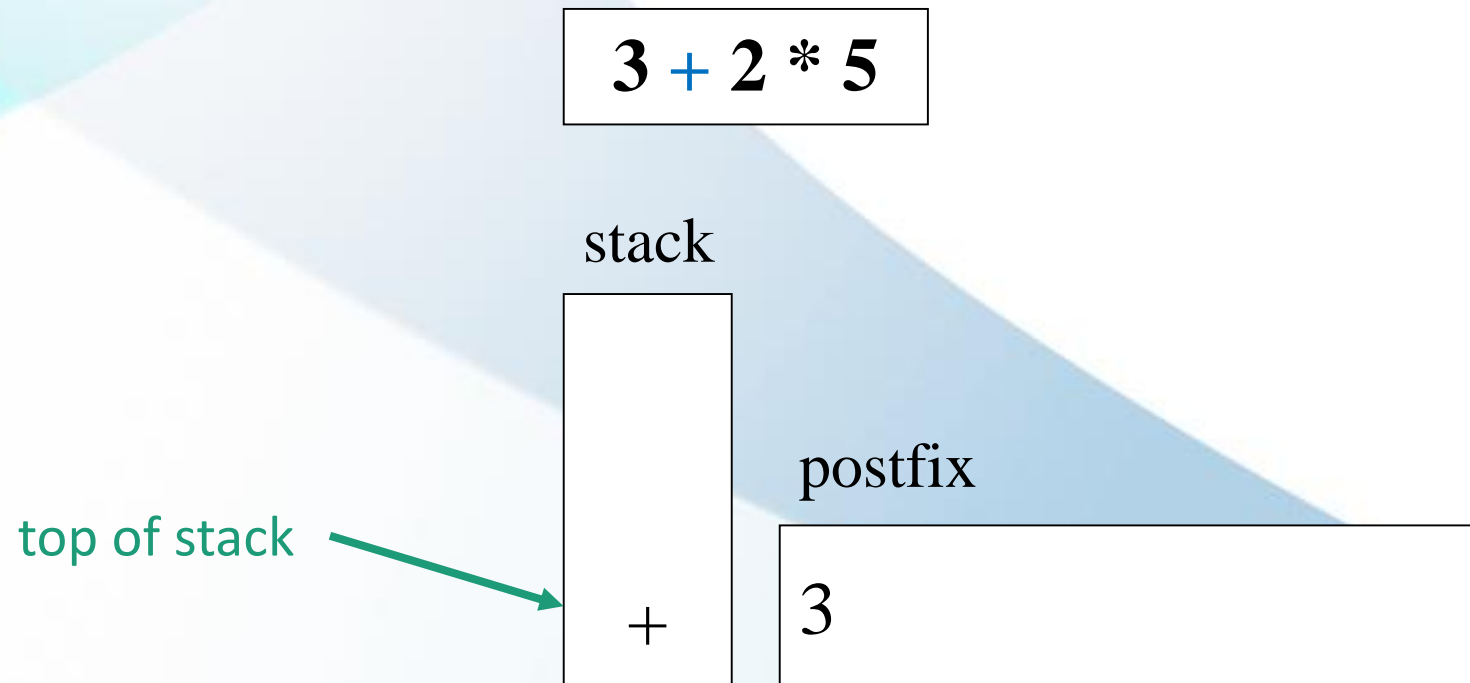
postfix

3



Studi Kasus 1

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



Studi Kasus 1

- Langkah 3: Operand 2
Masukkan ke postfix

$3 + 2 * 5$

stack



top of stack

postfix

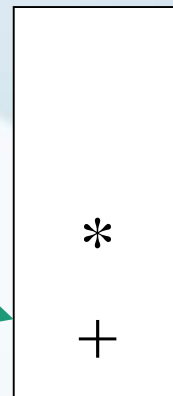
3 2

Studi Kasus 1

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

$3 + 2 * 5$

stack



top of stack

postfix

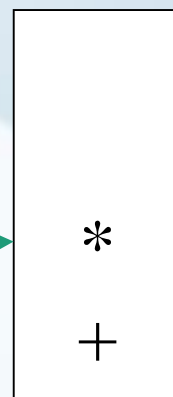
3 2

Studi Kasus 1

- Langkah 5: Operand 5
Masukkan ke postfix

$3 + 2 * 5$

stack



postfix

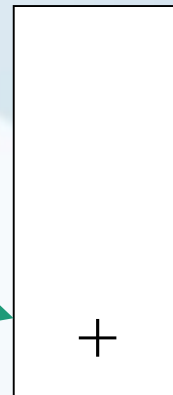
3 2 5

Studi Kasus 1

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

3 + 2 * 5

stack



top of stack

postfix

3 2 5 *

Studi Kasus 1

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

$3 + 2 * 5$

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

stack



postfix

$3\ 2\ 5\ *\ +$

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

- Baca persamaan dari kiri ke kanan

$$15 - (7 + 4) / 3$$

stack

postfix

Studi Kasus 2

- Langkah 1: Operand 15
Masukkan ke postfix

$$15 - (7 + 4) / 3$$

stack



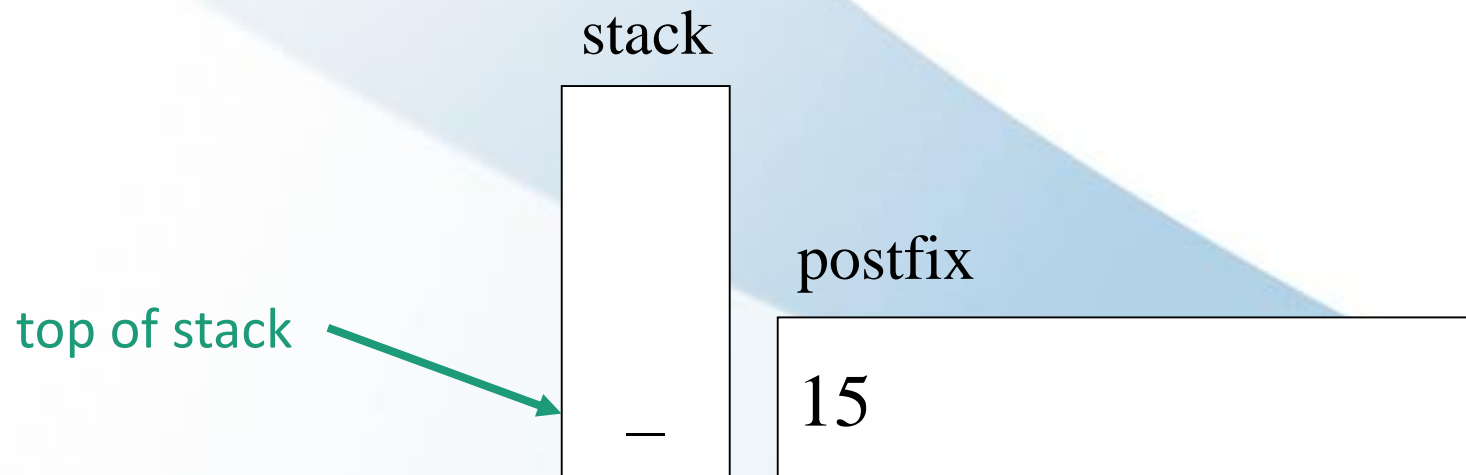
postfix

15

Studi Kasus 2

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

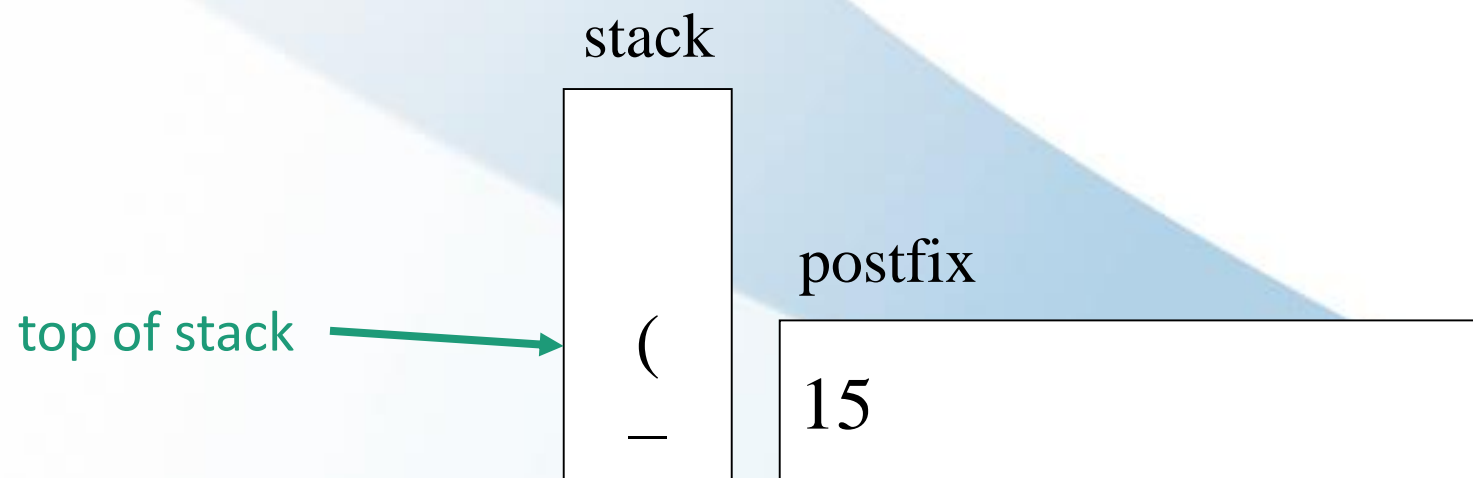
$$15 - (7 + 4) / 3$$



Studi Kasus 2

- Langkah 3: Tanda (Push ke stack

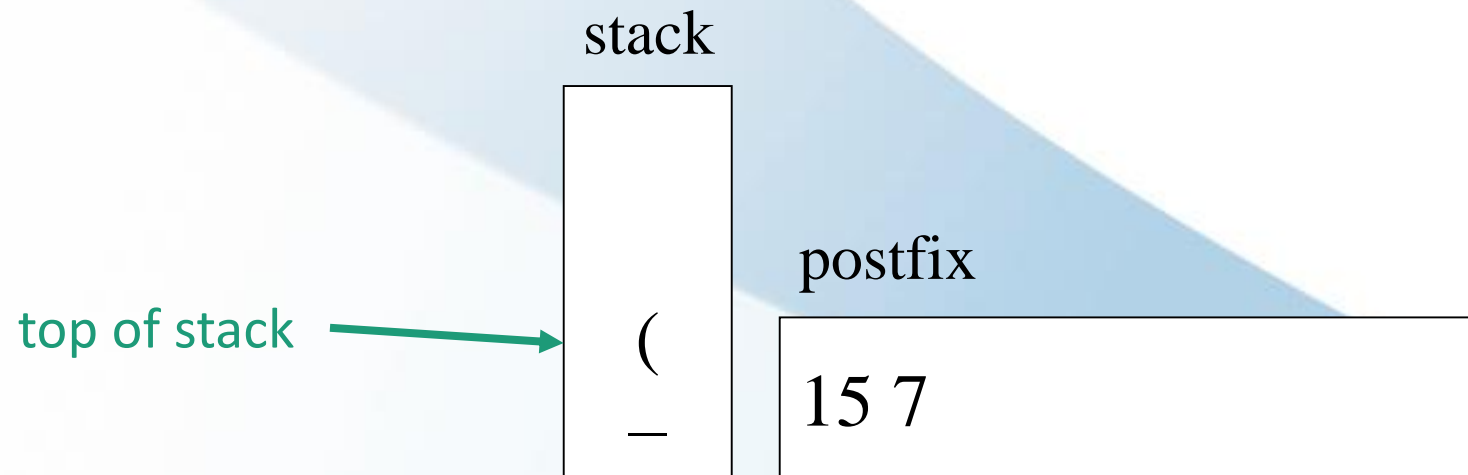
$$15 - (7 + 4) / 3$$



Studi Kasus 2

- Langkah 4: Operand 7
Masukkan ke postfix

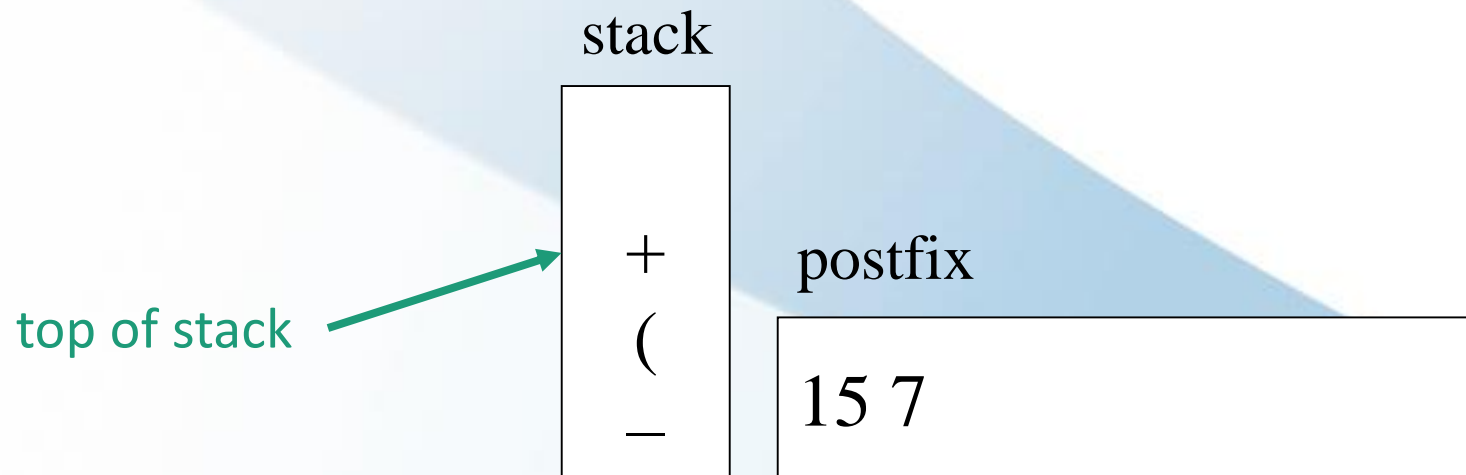
$$15 - (7 + 4) / 3$$



Studi Kasus 2

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

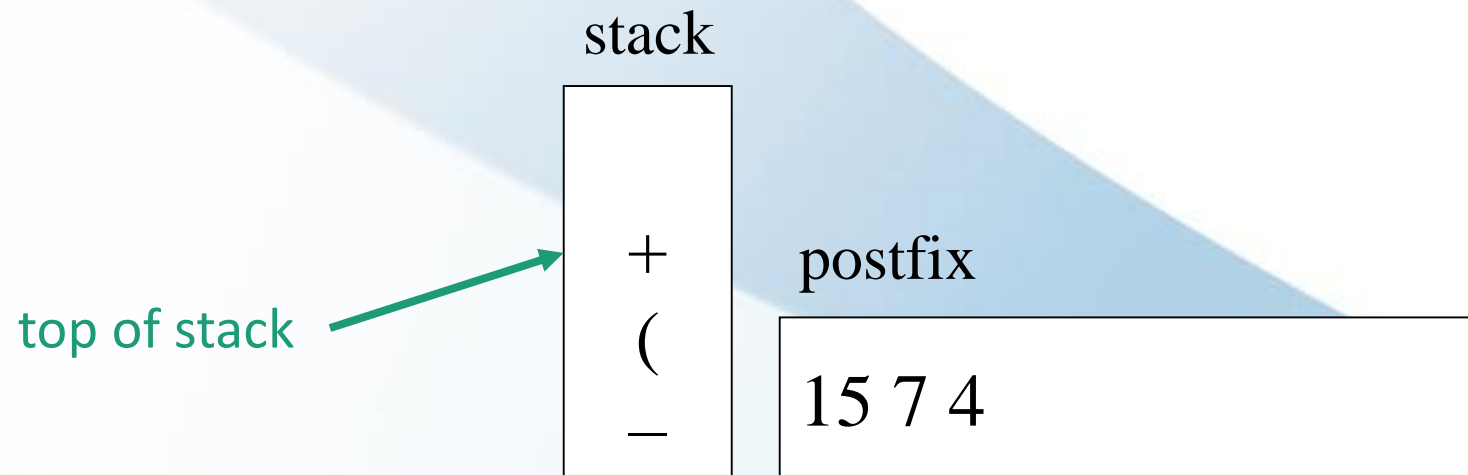
$$15 - (7 + 4) / 3$$



Studi Kasus 2

- Langkah 6: Operand 4
Masukkan ke postfix

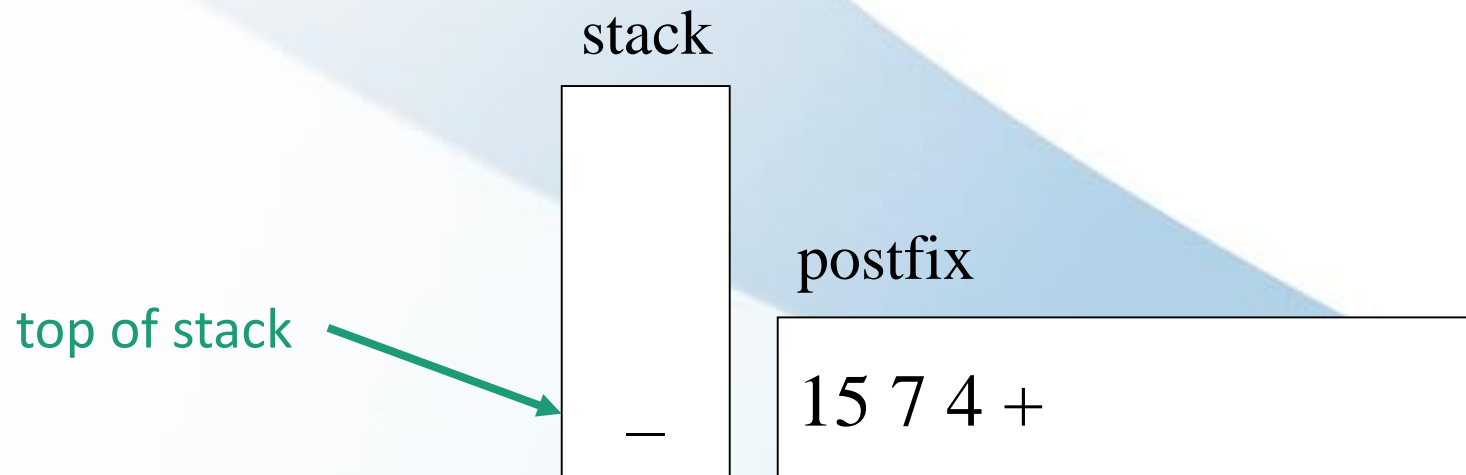
$$15 - (7 + 4) / 3$$



Studi Kasus 2

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

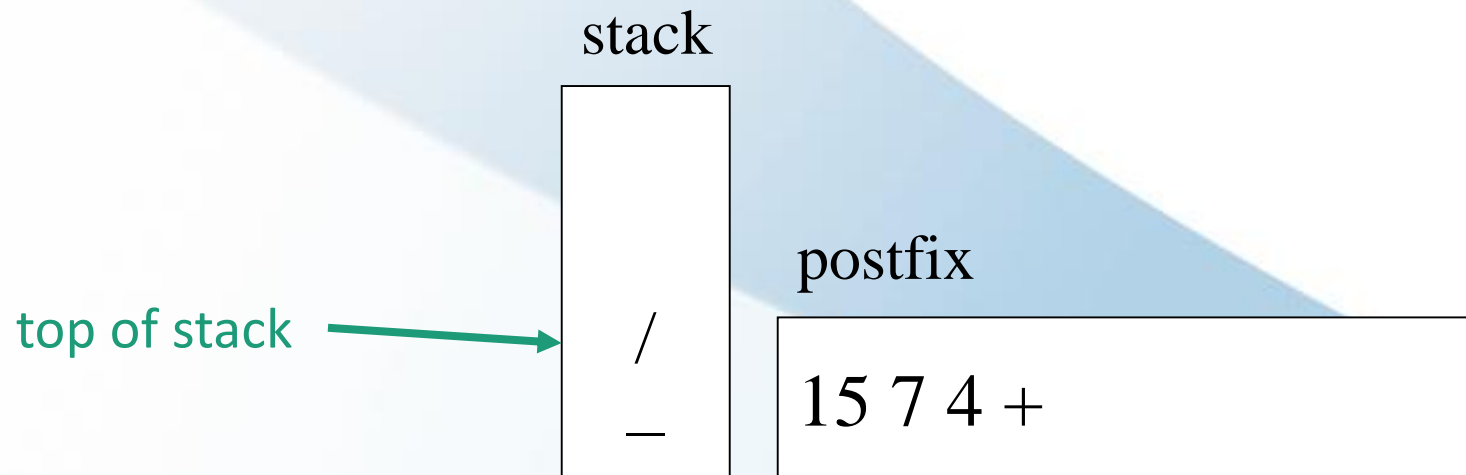
$15 - (7 + 4) / 3$



Studi Kasus 2

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

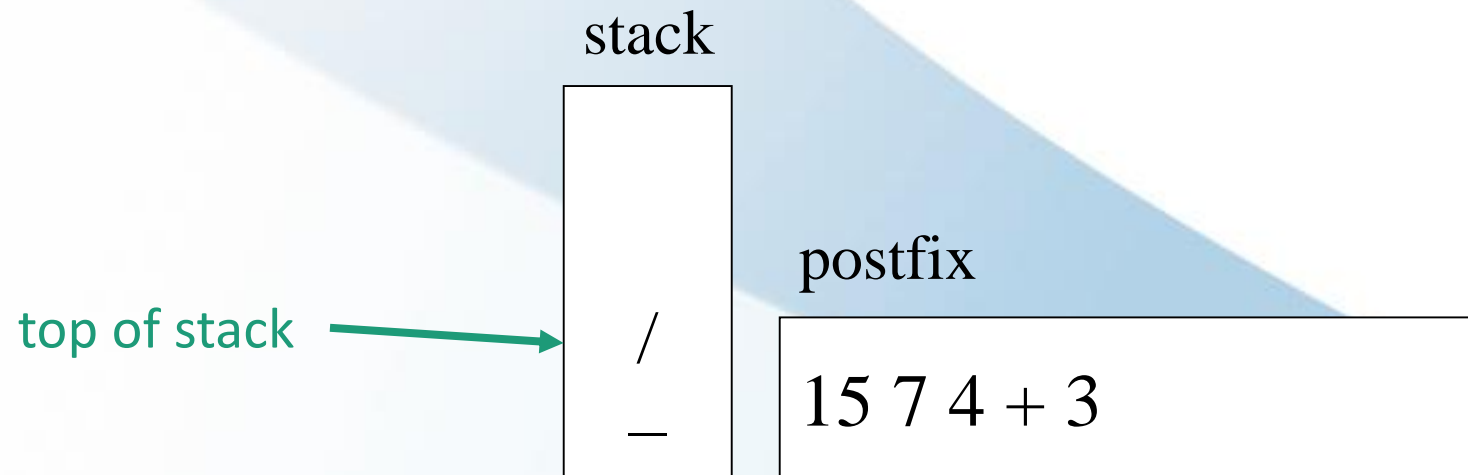
$$15 - (7 + 4) / 3$$



Studi Kasus 2

- Langkah 9: Operand 3
Masukkan ke postfix

$$15 - (7 + 4) / 3$$

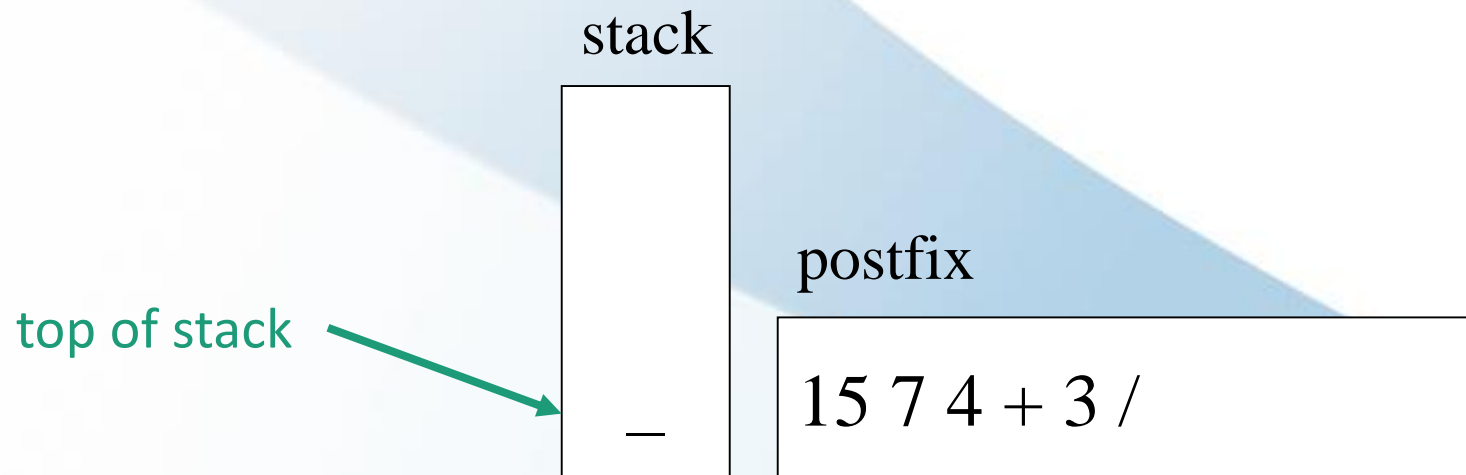


Studi Kasus 2

- Langkah 10

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

$$15 - (7 + 4) / 3$$



Studi Kasus 2

- 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



postfix

15 7 4 + 3 / –

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

Latihan

1. Tuliskan langkah-langkah pengerjaan dari beberapa rangkaian operasi stack berikut:
 - Push(8)
 - Push(3)
 - Pop()
 - Push(5)
 - Push(2)
 - Pop()
 - Pop()
 - Push(1)
2. Lakukan konversi notasi infix berikut menjadi notasi postfix!
 - a. $x + y^z - w$
 - b. $32 / 2 * 4 + 12$
 - c. $4 * (7 - 2 + 4) / 6$