STACK

TIM AJAR

ALGORITMA DAN STRUKTUR DATA

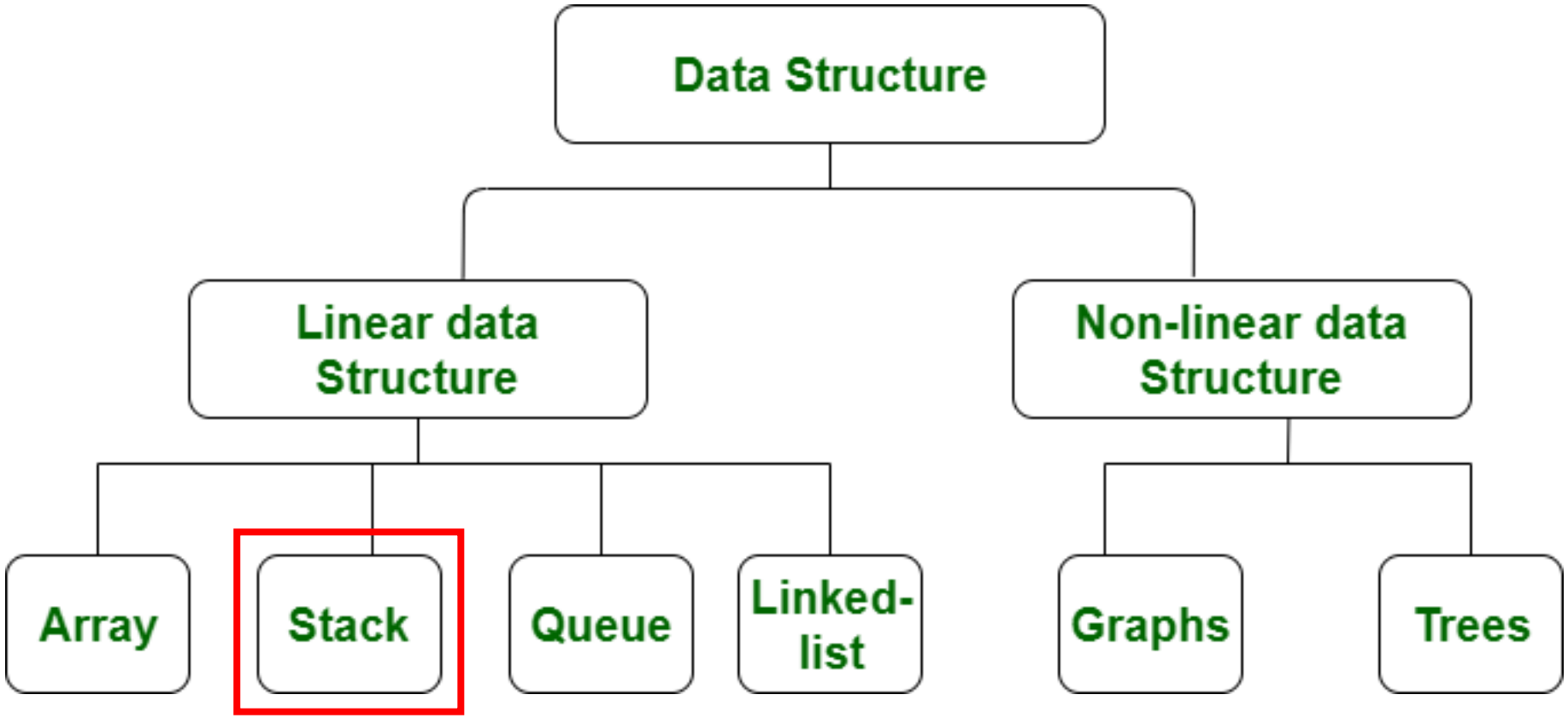
2024/2025

Capaian Pembelajaran

Setelah mempelajari materi Stack, mahasiswa diharapkan mampu

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

Jenis Struktur Data

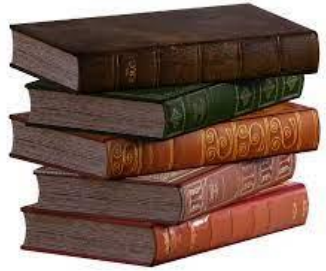


Definisi Struktur Data Linear

- 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**
- Contoh ilustrasi:



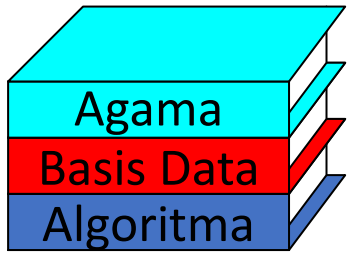

Penerapan Stack

- **Compiler**
  Compiler menggunakan stack untuk menghitung nilai ekspresi seperti 2 + 4 / 5 * (7 - 9) dengan mengubahnya menjadi bentuk prefix atau postfix
- **Reverse String (Membalik kata)**
  Menyimpan semua huruf pada stack kemudian dikeluarkan satu persatu dalam urutan terbalik menggunakan konsep LIFO
- **Convert Desimal ke Biner**
  Membagi bilangan desimal berulang kali dengan 2 dan memasukkan sisa setiap pembagian ke dalam stack hingga menjadi 0, kemudian hasilnya dikeluarkan
- **Browser History**
  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

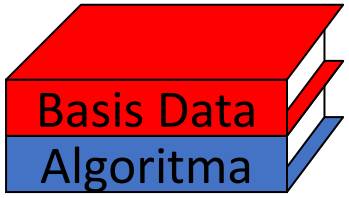
Konsep Stack

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

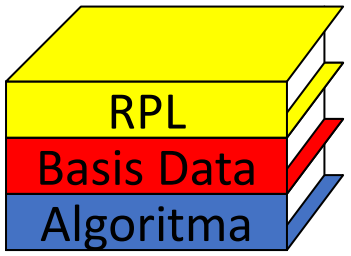
1. Keadaan awal



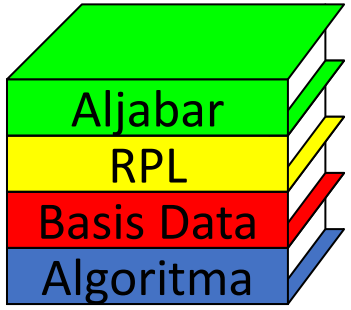
2. Setelah mengambil "Agama"



3. Setelah menambah "RPL"

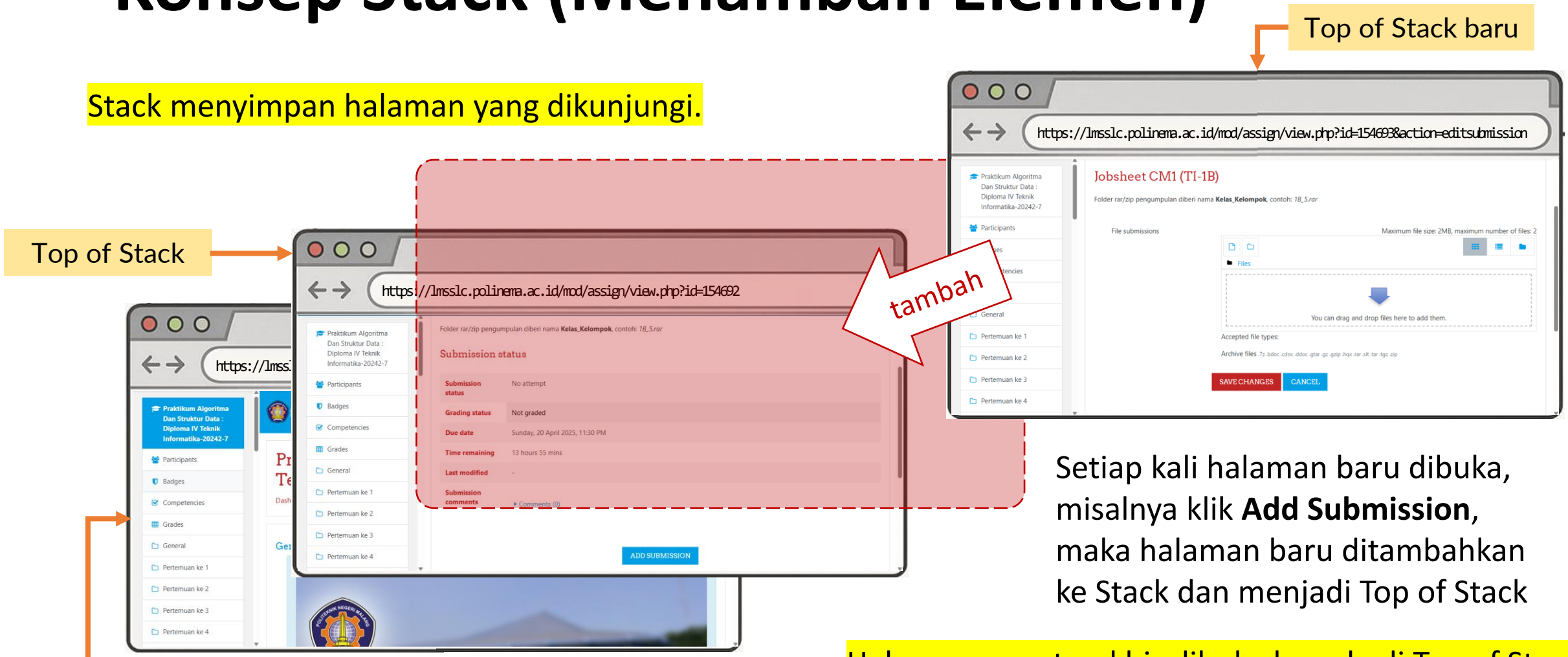


4. Setelah menambah "Aljabar"



Konsep Stack (Menambah Elemen)

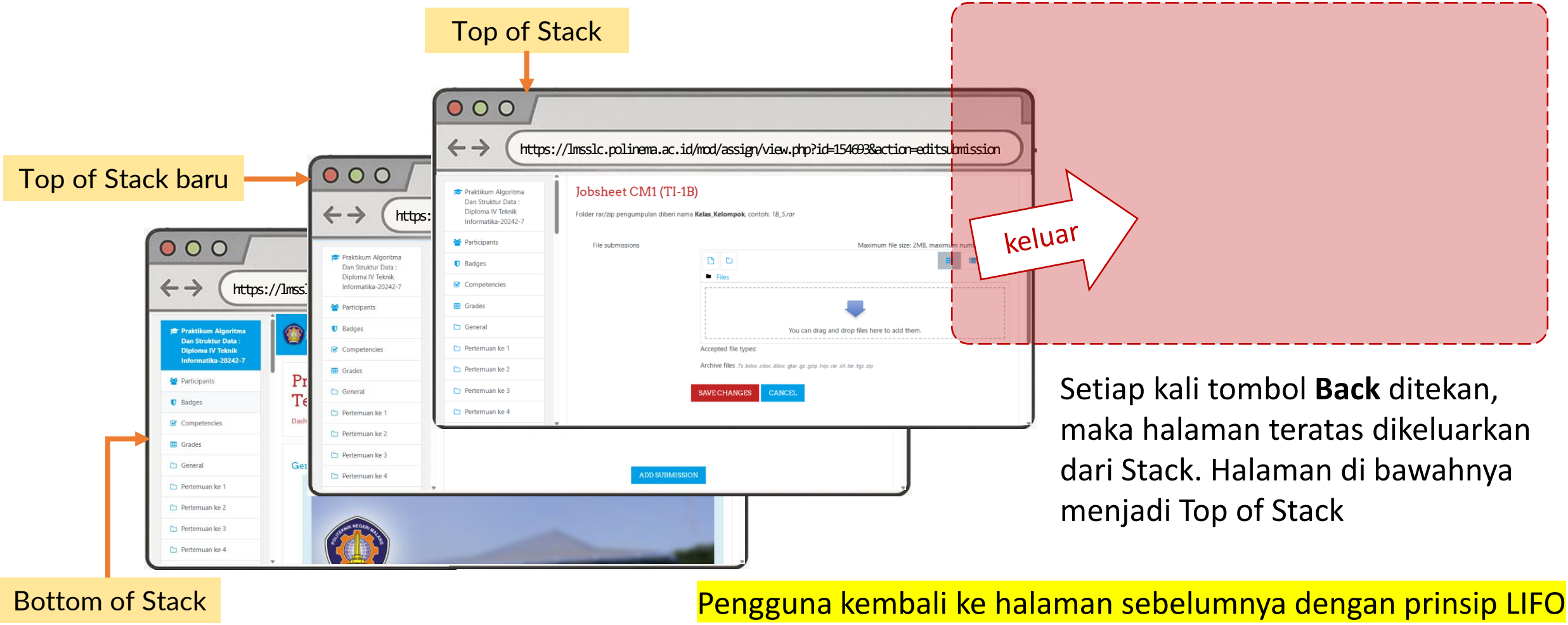
Stack menyimpan halaman yang dikunjungi.



Setiap kali halaman baru dibuka, misalnya klik **Add Submission**, maka halaman baru ditambahkan ke Stack dan menjadi Top of Stack

Halaman yang terakhir dibuka berada di Top of Stack

Konsep Stack (Menghapus Elemen)

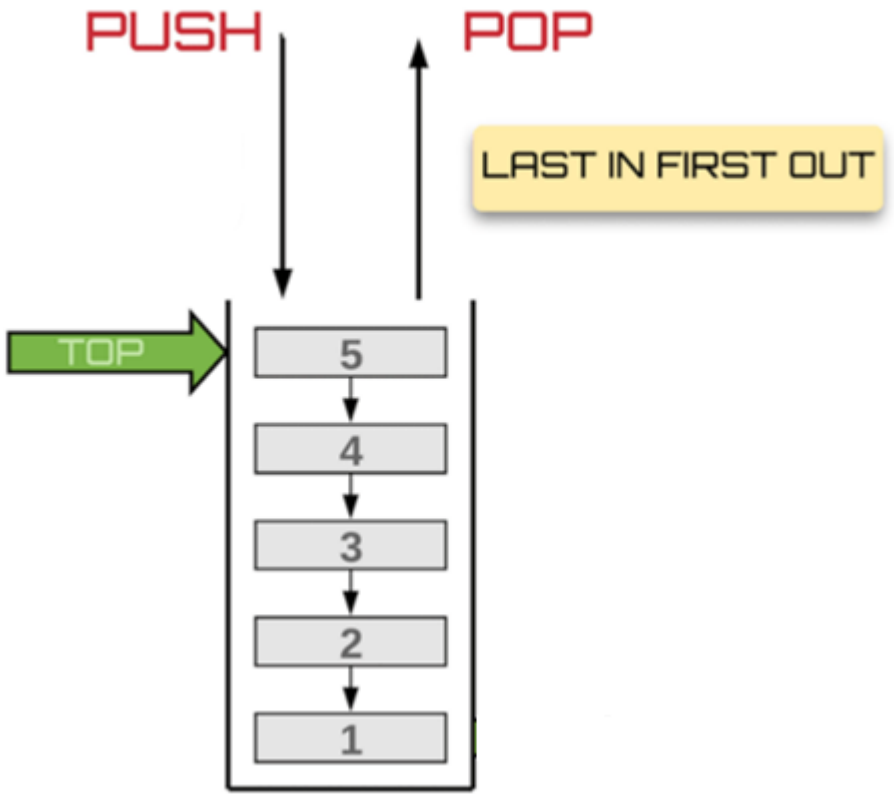


Setiap kali tombol **Back** ditekan, maka halaman teratas dikeluarkan dari Stack. Halaman di bawahnya menjadi Top of Stack

Pengguna kembali ke halaman sebelumnya dengan prinsip LIFO

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

- Deklarasi stack sebagai tempat untuk menyimpan data
- Langkah-langkah:
  1. Deklarasi class Stack
  2. Deklarasi atribut
     a. Array data
        digunakan sebagai tempat penyimpanan sejumlah data
     b. size
        digunakan untuk menentukan kapasitas penyimpanan
     c. Pointer top
        digunakan sebagai penunjuk data pada posisi akhir (atas)

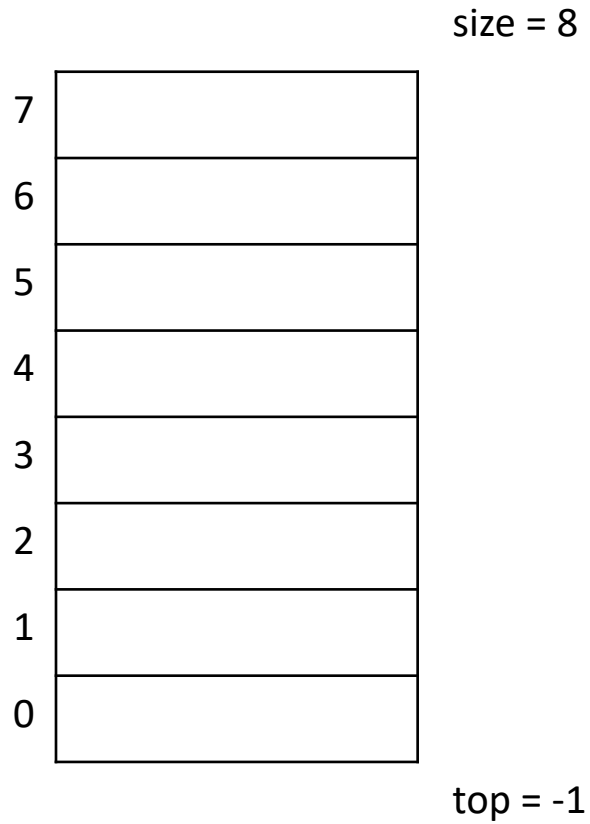
```
public class Stack {
    String data[];
    int size;
    int top;
}
```

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** akan selalu bergerak hingga mencapai **max** atau **size** yang menyebabkan stack PENUH

Inisialisasi Stack

- Ilustrasi Stack saat inisialisasi pada **konstruktor**



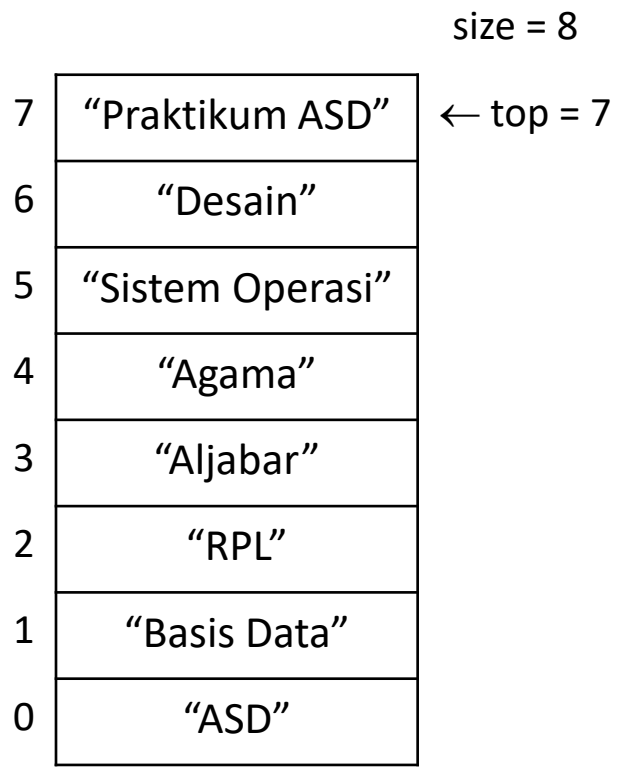

```
public Stack(int size) {
    this.size = size;
    data = new int[size];
    top = -1;
}
```

Fungsi IsFull

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

Fungsi IsFull

- Ilustrasi stack saat kondisi **Full**




```
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**, jika masih -1 maka berarti data stack masih kosong

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

}
```

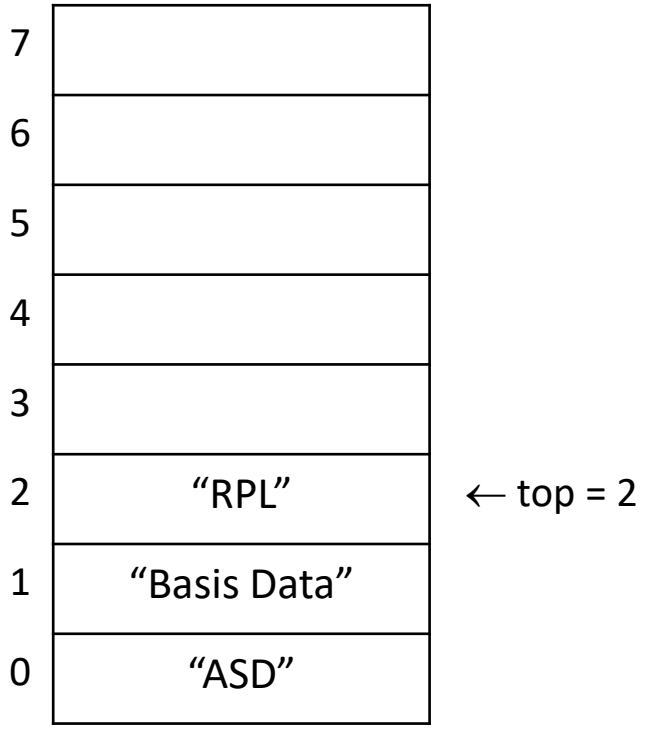
Fungsi Push

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

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

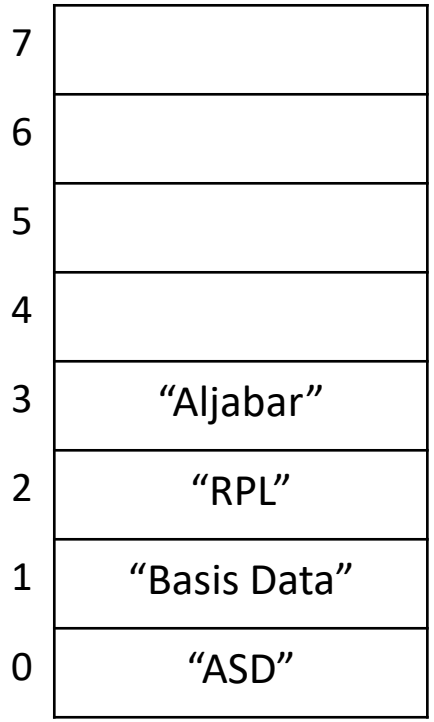
Fungsi Push

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 “Aljabar” dimasukkan ke dalam Stack

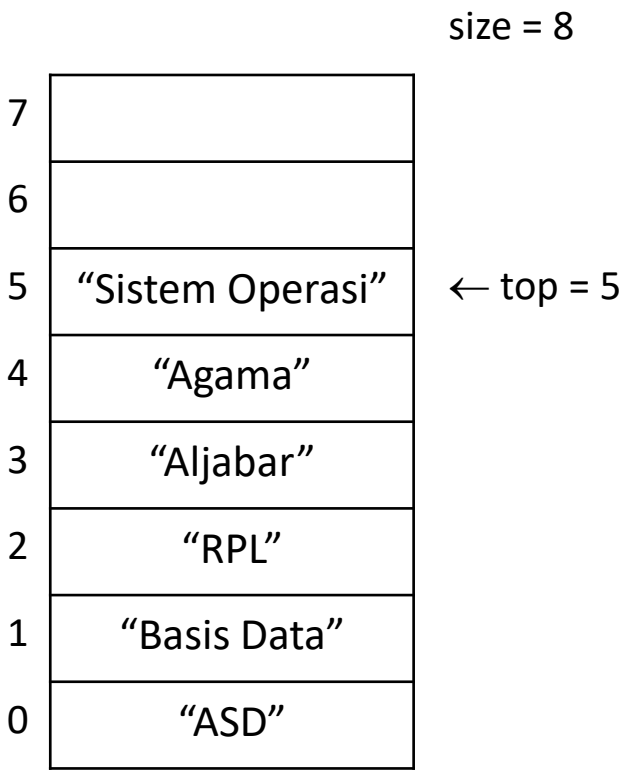
“Aljabar”

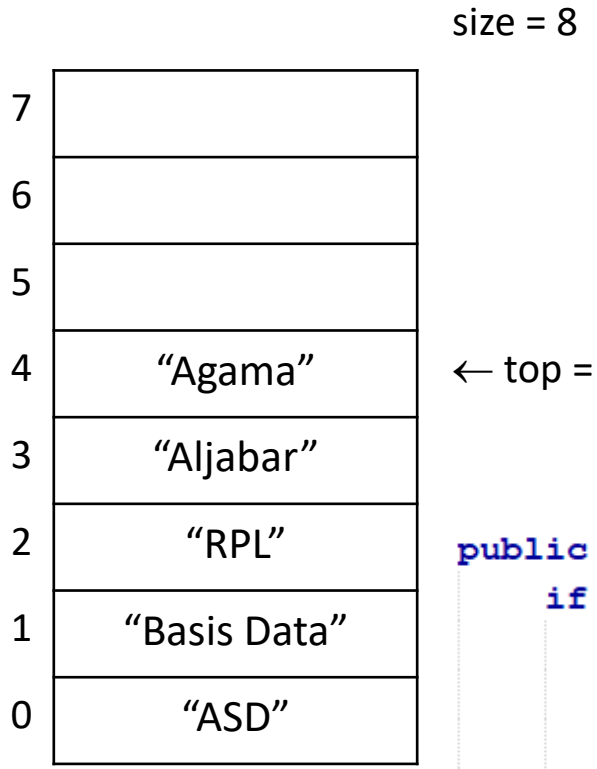
Fungsi Pop

- Untuk mengambil data stack yang terletak paling atas (data yang ditunjuk oleh **top**)
- Jika **data tidak kosong**,
  - **Tampilkan terlebih dahulu** nilai elemen teratas stack dengan mengakses indeksnya sesuai dengan top
  - Lakukan decrement nilai top, 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






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

Data “Sistem Operasi” pada posisi teratas dihapus

Fungsi Peek

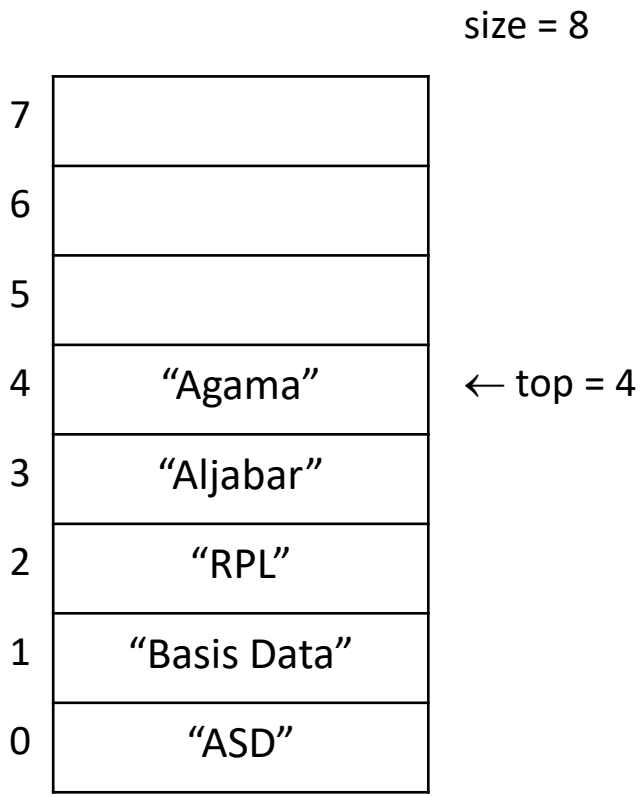
- Untuk mengakses elemen yang **ditunjuk oleh top**, yaitu elemen yang terakhir kali ditambahkan
- Operasi ini berbeda dengan pop karena tidak disertai dengan penghapusan data, namun hanya pengaksesan (pengembalian) data saja

```
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:

Agama, Aljabar, RPL, Basis Data, ASD

```
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");
    }
}
```

Penerapan dalam PBO

- Untuk menyelesaikan permasalahan menggunakan penerapan Stack menggunakan program Java, maka setidaknya terdapat sejumlah class berikut:
  - Class Stack
  - Class Data, misalnya Buku, Baju, dll
  - Class Utama (main)

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

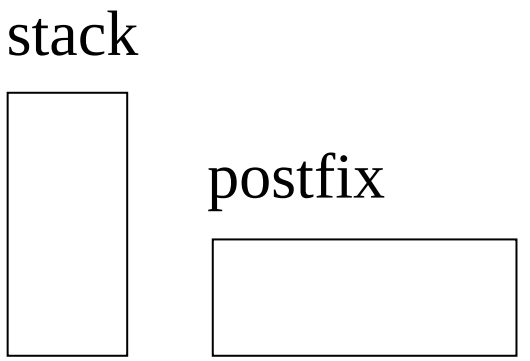
Derajat Operator Aritmatika

Urutan derajat operator aritmatika:

- Pangkat ^
- Perkalian * setara dengan pembagian / dan modulo %
- Penjumlahan + setara dengan pengurangan –
- Tanda kurung (untuk proses konversi notasi postfix)

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

- Misalkan terdapat persamaan:

  15 – (7 + 4) / 3

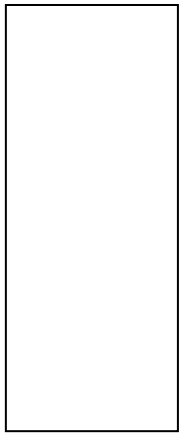
- Operasi di atas disebut notasi **infix**, notasi infix tersebut harus diubah menjadi notasi **postfix**

Studi Kasus - Penyelesaian

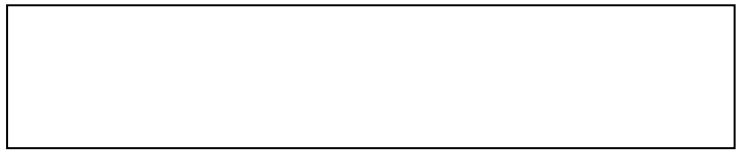
- Baca persamaan dari kiri ke kanan

15 – (7 + 4) / 3

stack

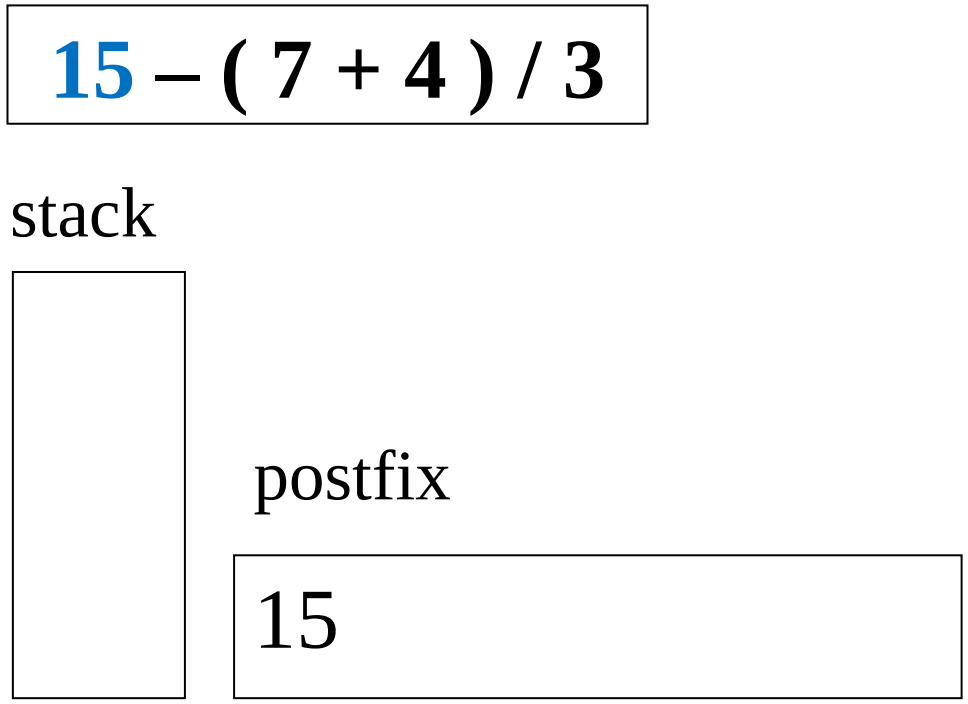


postfix



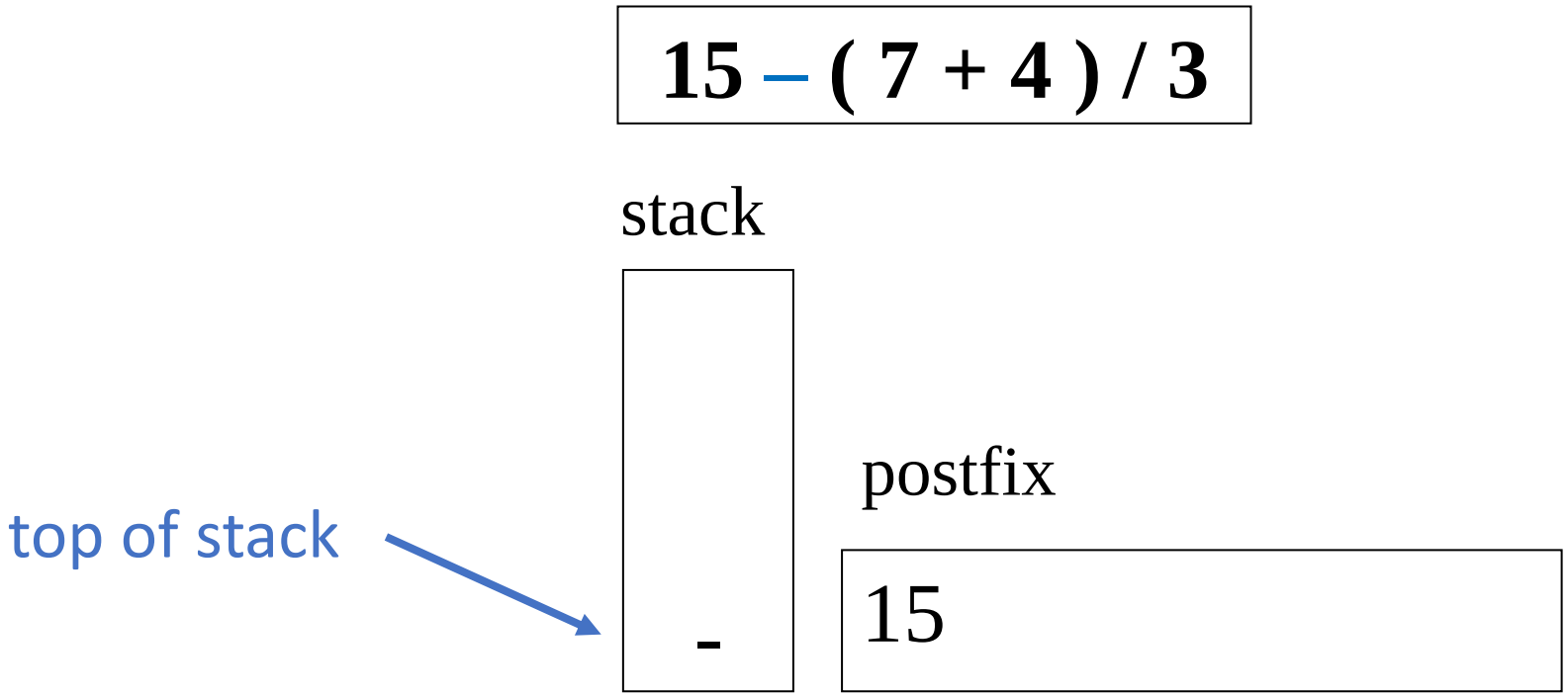
Studi Kasus - Penyelesaian

- Langkah 1: Operand 15
  Masukkan ke postfix



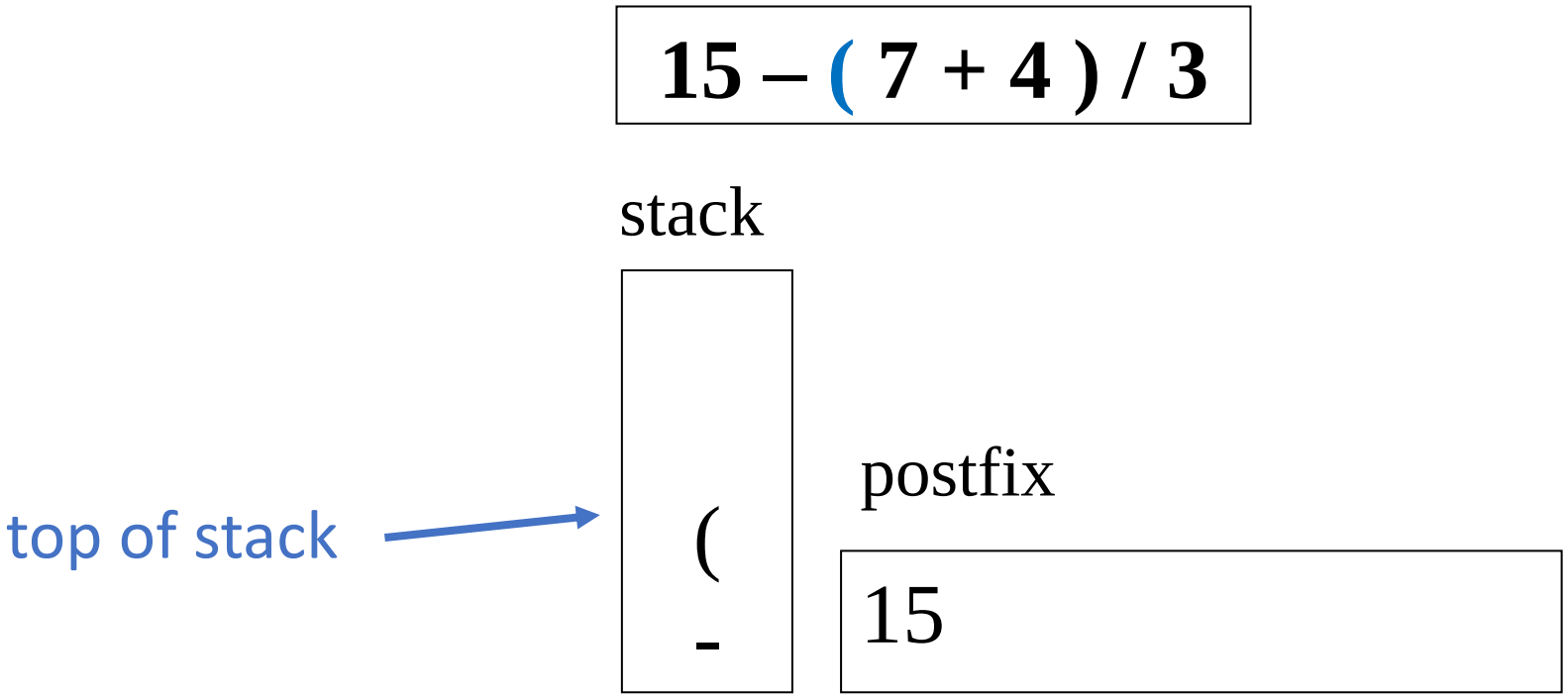
Studi Kasus - Penyelesaian

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



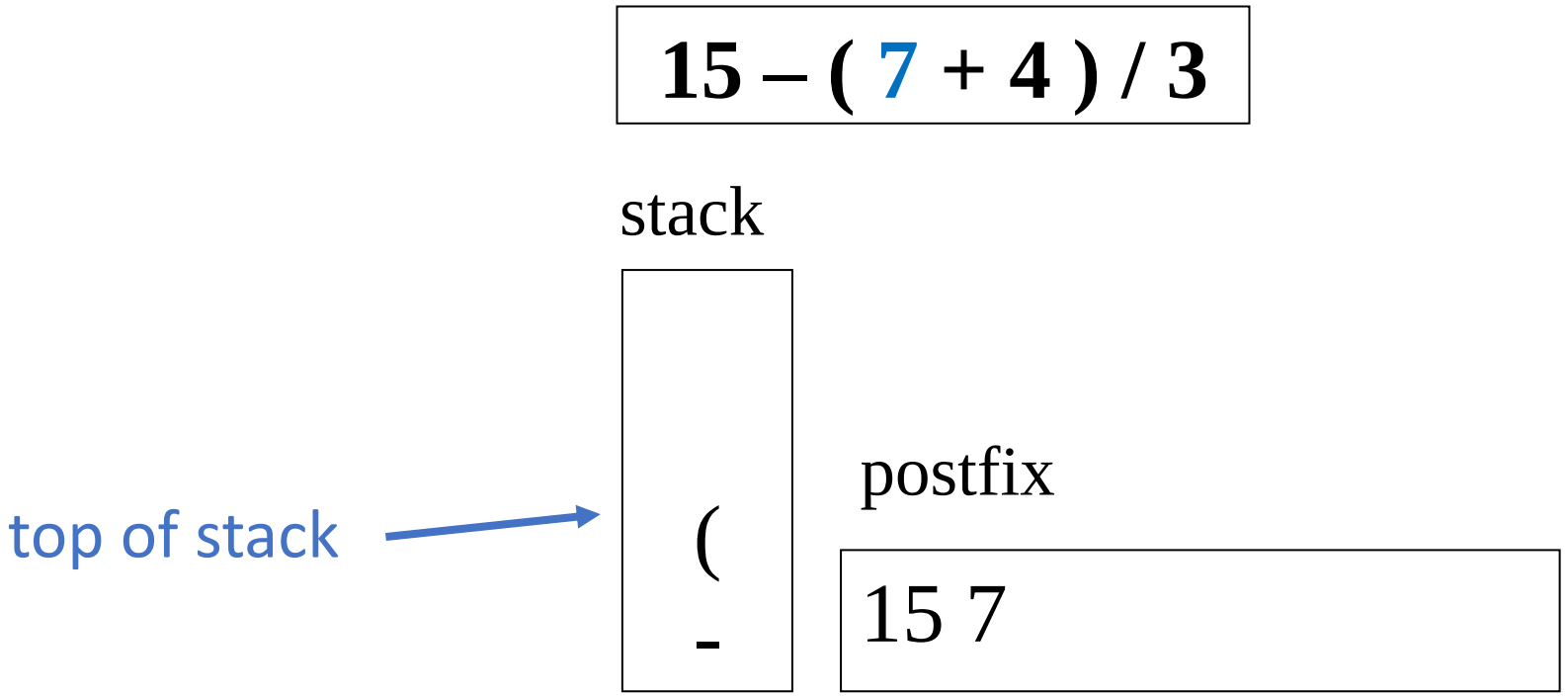
Studi Kasus - Penyelesaian

- Langkah 3: Tanda (
  Push ke stack



Studi Kasus - Penyelesaian

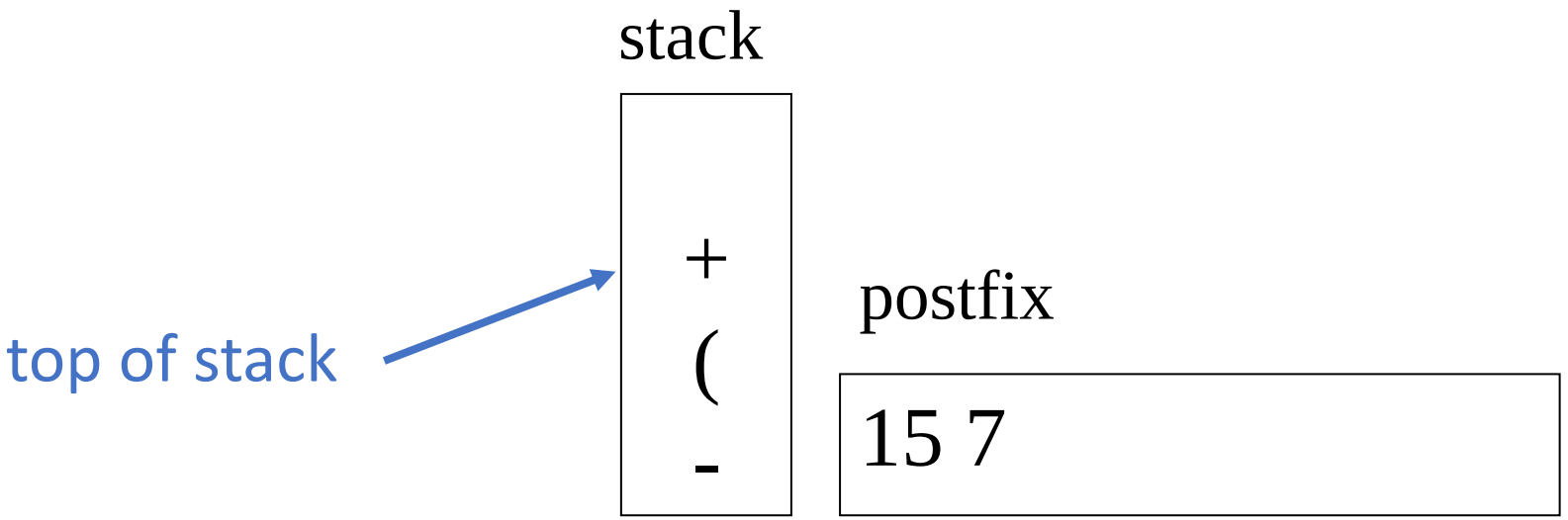
- Langkah 4: Operand 7
  Masukkan ke postfix



Studi Kasus - Penyelesaian

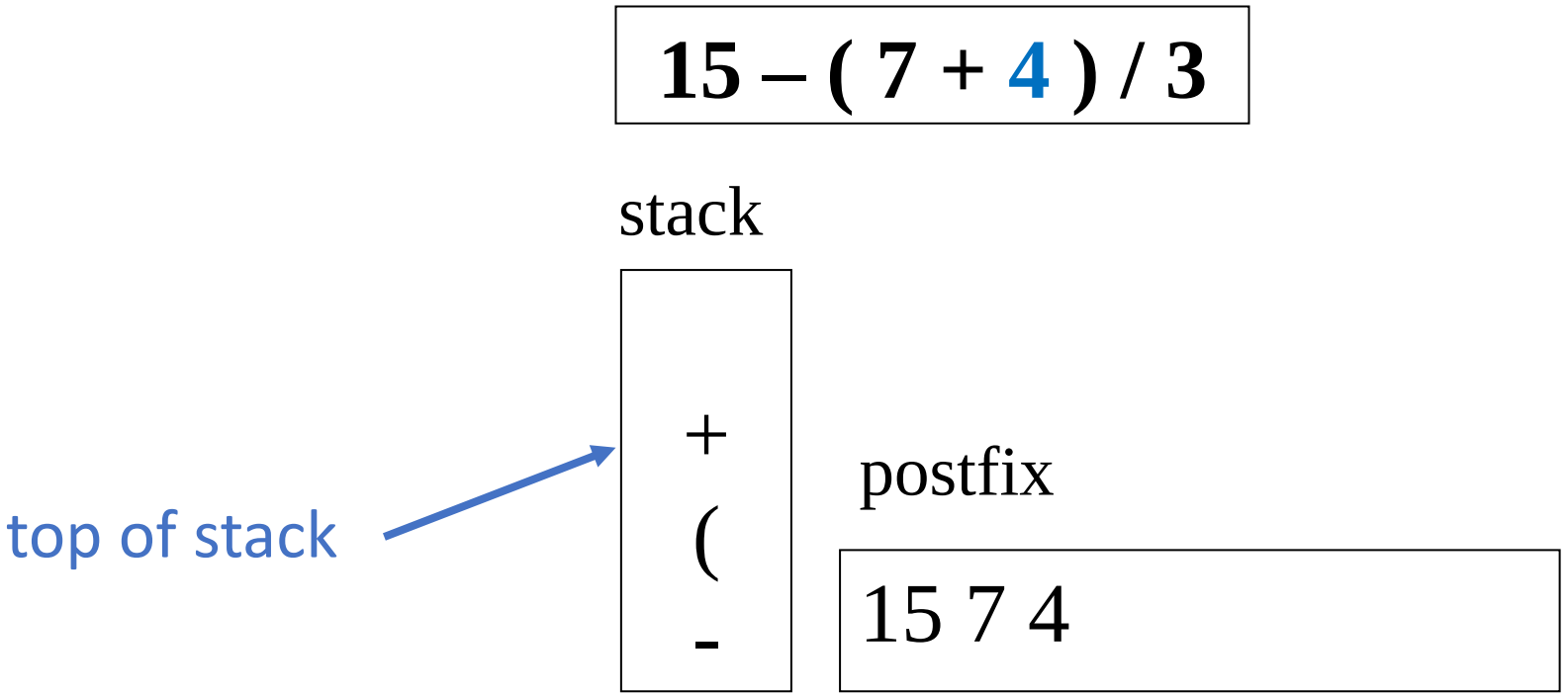
- Langkah 5: Operator +
  Push ke stack karena derajat operator Top of stack (**lebih kecil** dari derajat operator +

15 – (7 + 4) / 3



Studi Kasus - Penyelesaian

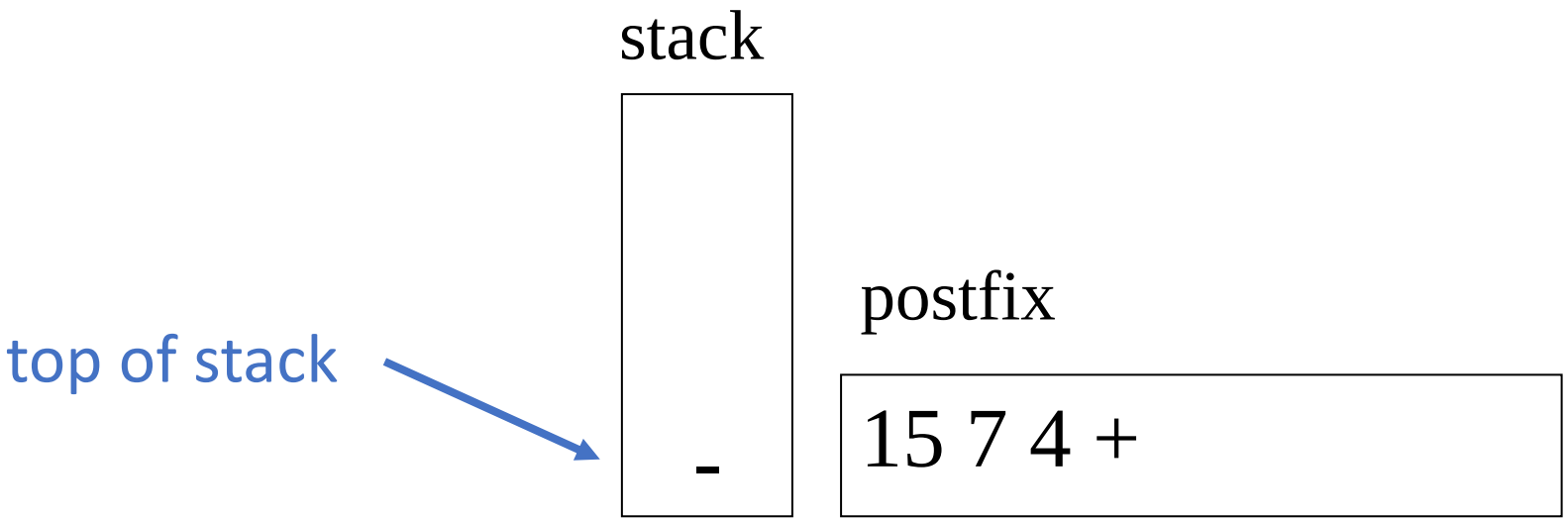
- Langkah 6: Operand 4
  Masukkan ke postfix



Studi Kasus - Penyelesaian

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

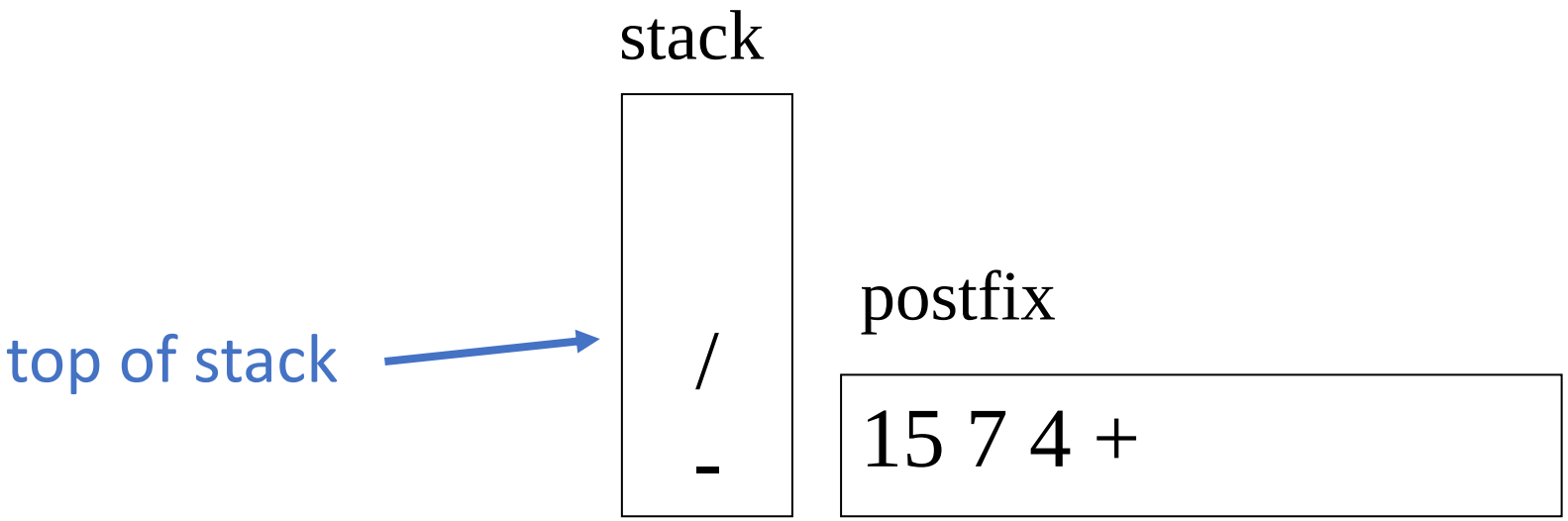
15 – (7 + 4) / 3



Studi Kasus - Penyelesaian

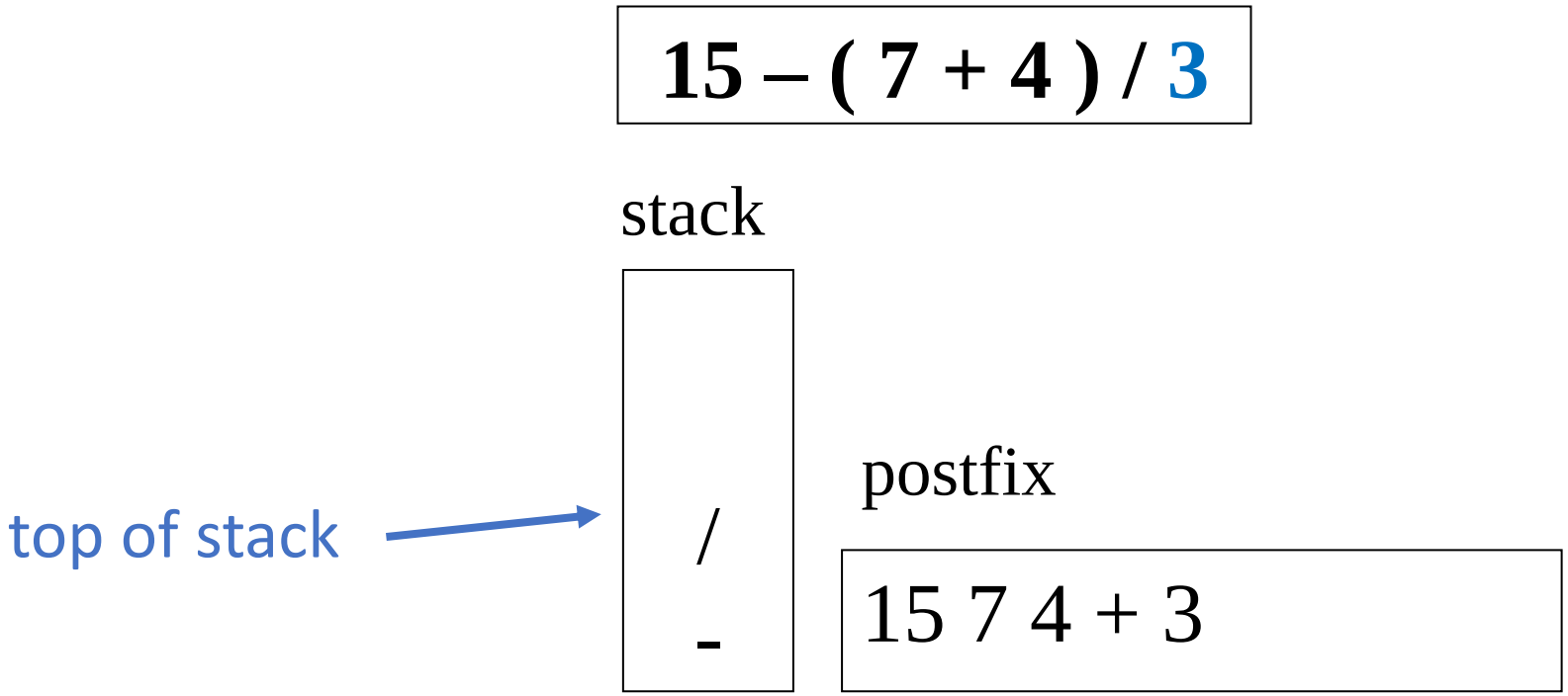
- Langkah 8: Operator /
  Push ke stack karena derajat operator Top of stack – **lebih kecil** dari derajat oprator /

15 – (7 + 4) / 3



Studi Kasus - Penyelesaian

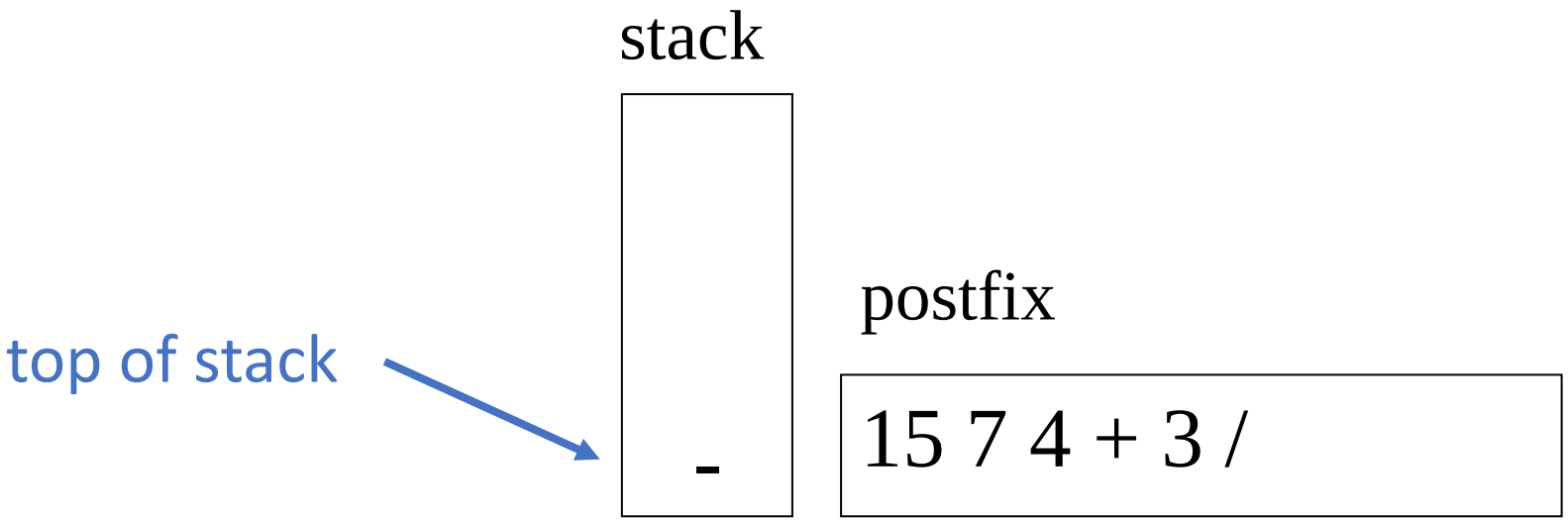
- Langkah 9: Operand 3
  Masukkan ke postfix



Studi Kasus - Penyelesaian

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

15 – (7 + 4) / 3

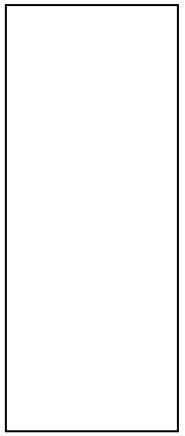


Studi Kasus - Penyelesaian

- 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 / –**

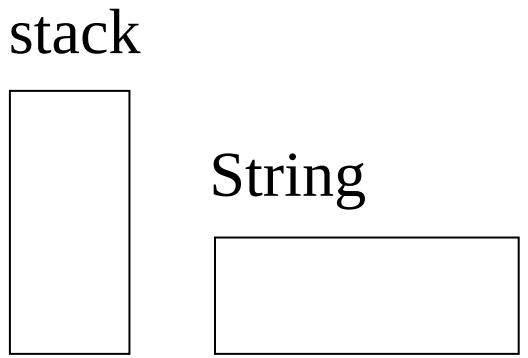
postfix

15 7 4 + 3 / -

Convert Desimal ke Biner

Algoritma Convert Desimal ke Biner

Buat dan inisialisasi stack untuk menyimpan modulo
WHILE desimal tidak sama dengan 0 **DO**
    Hitung modulo dari desimal dengan 2
    Push modulo ke dalam stack
    Bagi desimal dengan 2 (lakukan pembagian bulat)
END WHILE
Buat sebuah string kosong untuk menyimpan bilangan biner
WHILE stack tidak kosong **DO**
    Pop modulo
    Tambahkan modulo ke dalam string biner.
END WHILE
Return string biner sebagai hasil

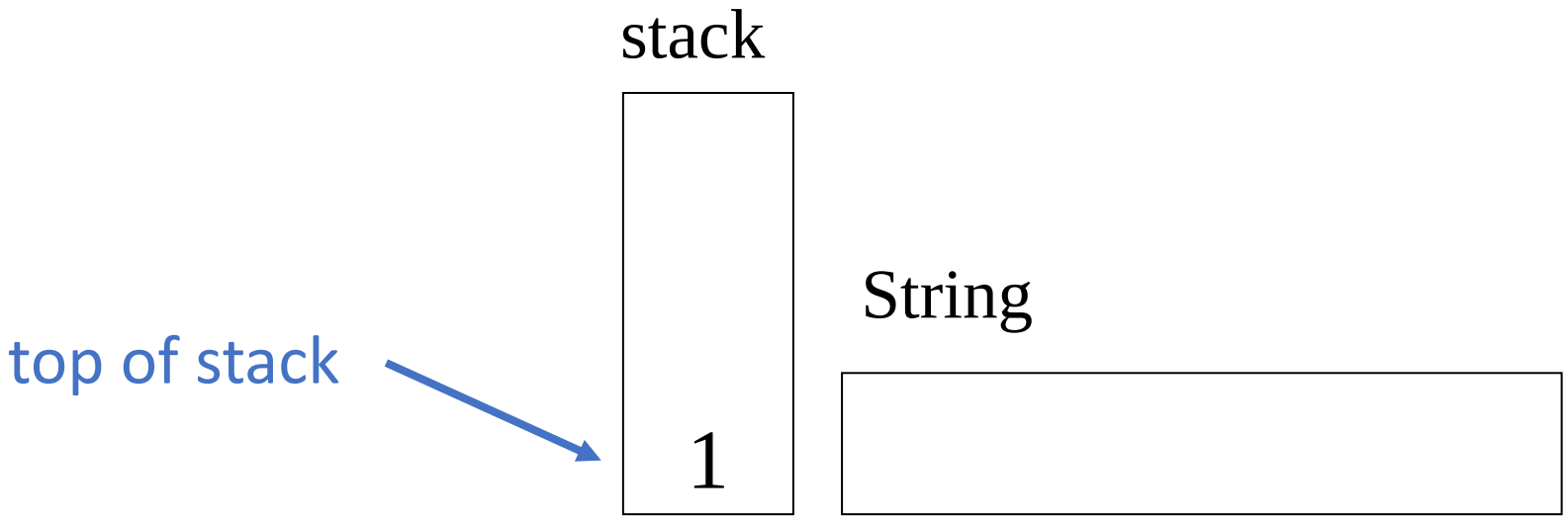


Studi Kasus

- Lakukan konversi bilangan desimal **11** ke dalam bentuk biner

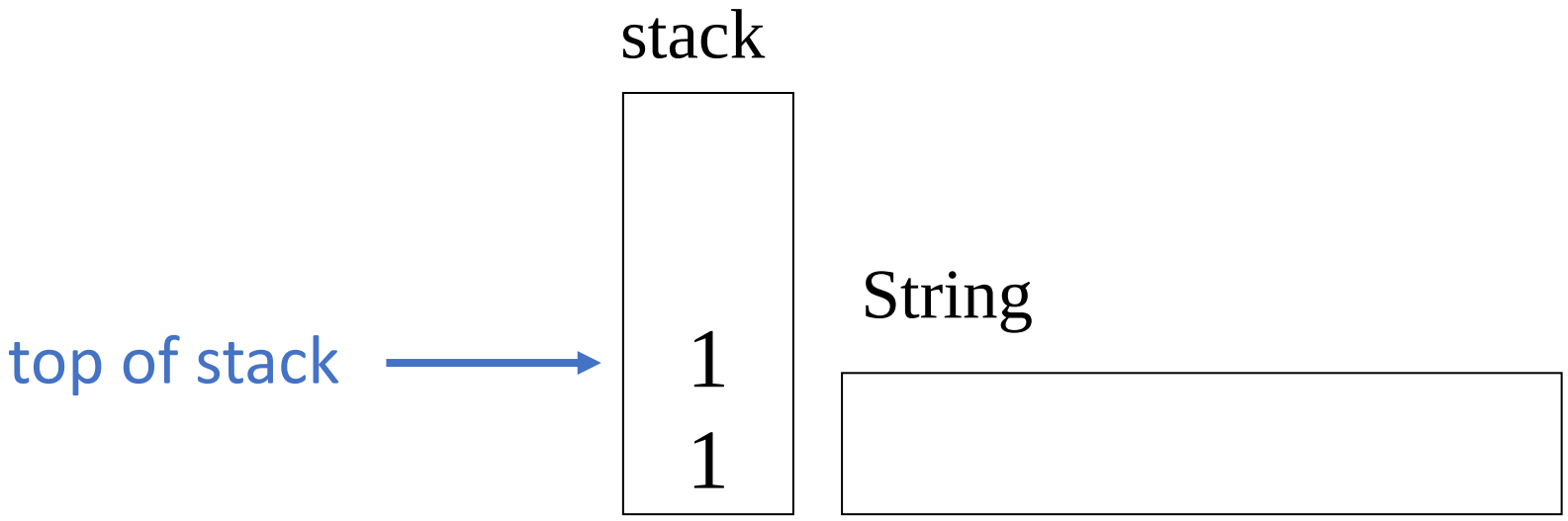
Studi Kasus - Penyelesaian

- Bilangan saat ini = **11**
- 11 % 2 = **1**, push ke stack
- 11 / 2 = **5**



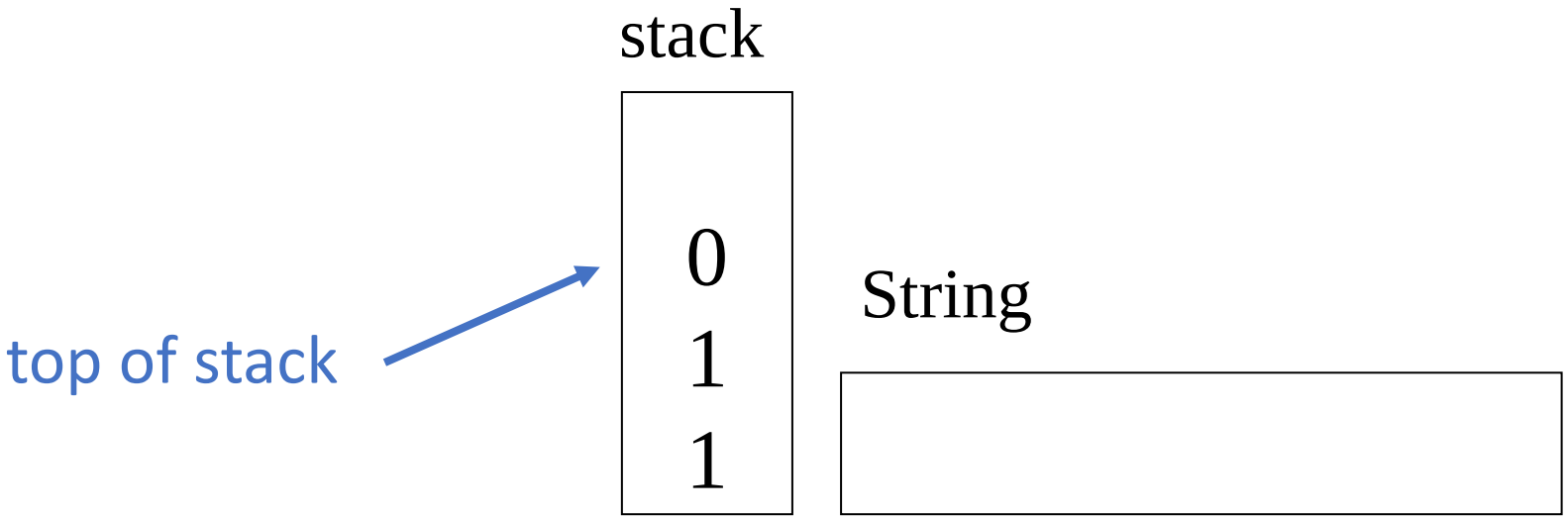
Studi Kasus - Penyelesaian

- Bilangan saat ini = **5**
- 5 % 2 = **1**, push ke stack
- 5 / 2 = **2**



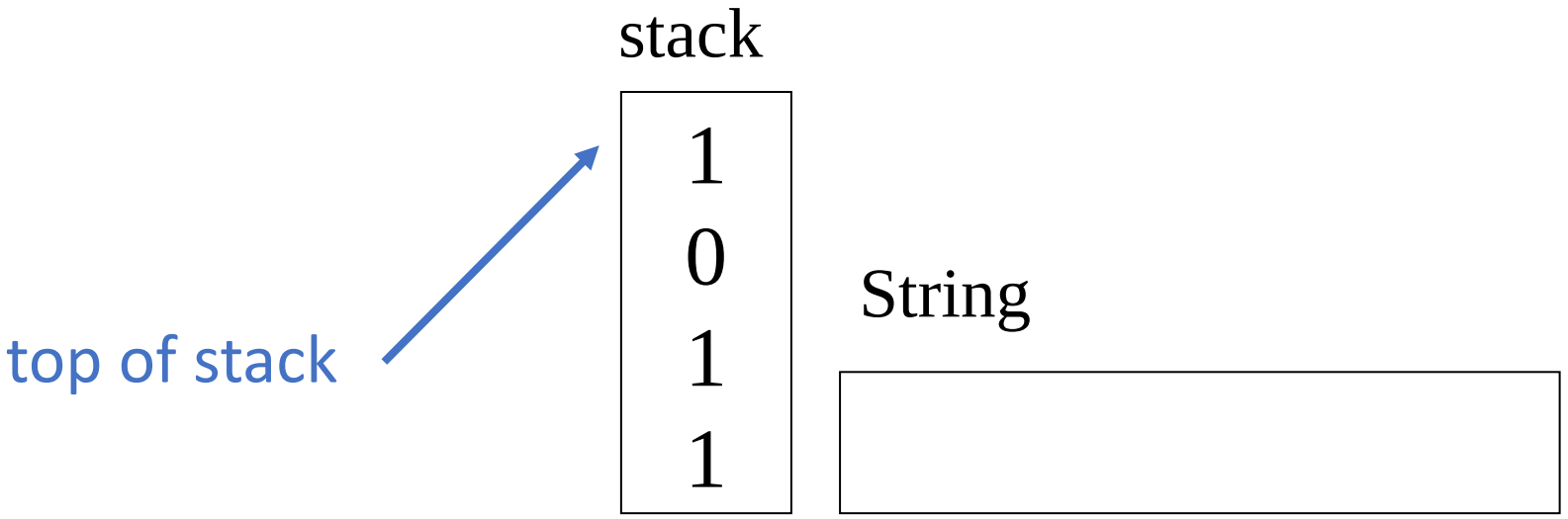
Studi Kasus - Penyelesaian

- Bilangan saat ini = **2**
- 2 % 2 = **0**, push ke stack
- 2 / 2 = **1**



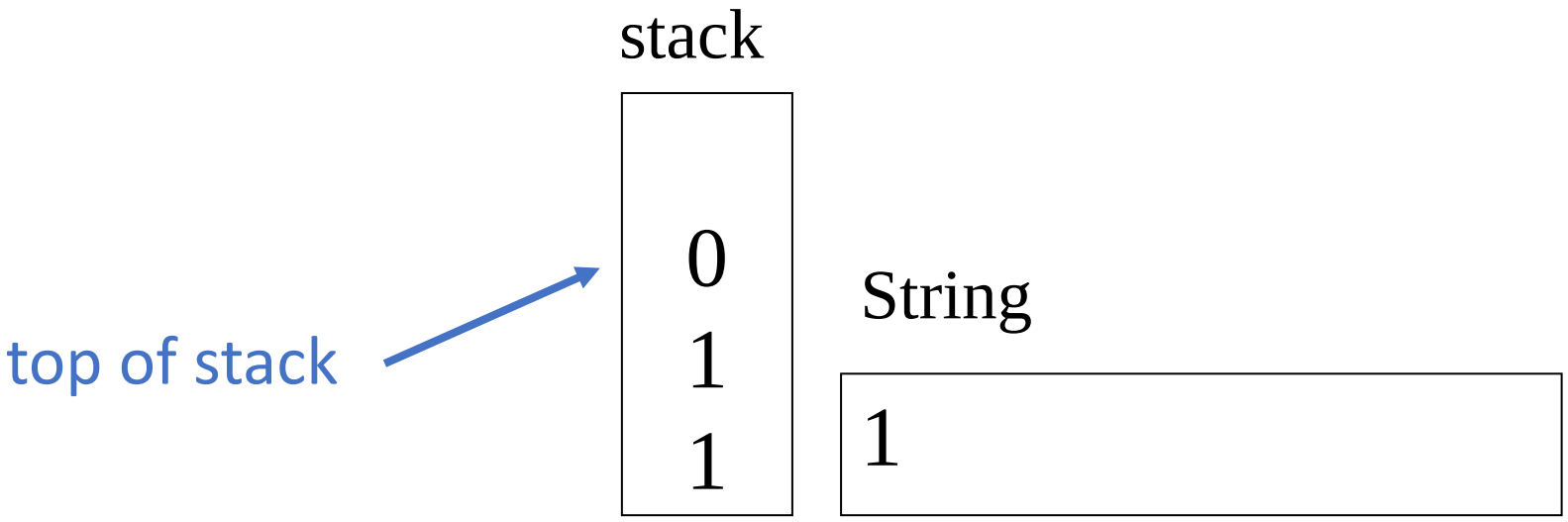
Studi Kasus - Penyelesaian

- Bilangan saat ini = **1**
- 1 % 2 = **1**, push ke stack
- 1 / 2 = **0**, karena menghasilkan 0 maka proses konversi selesai



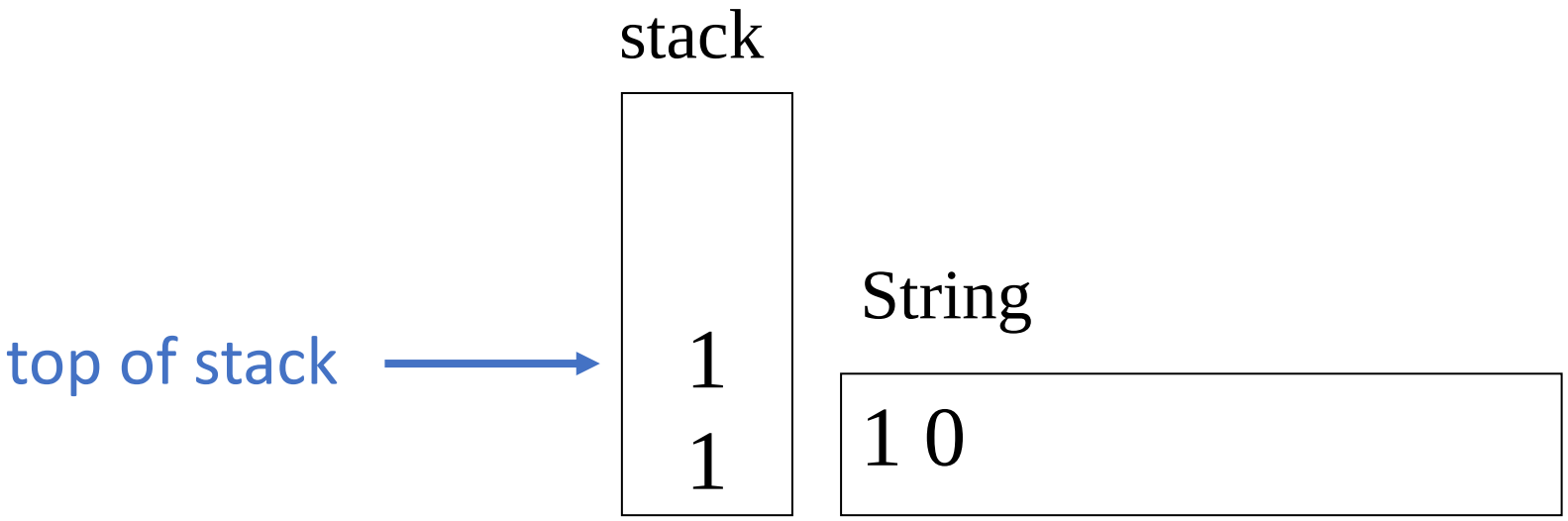
Studi Kasus - Penyelesaian

- Pop stack dan masukkan ke dalam String



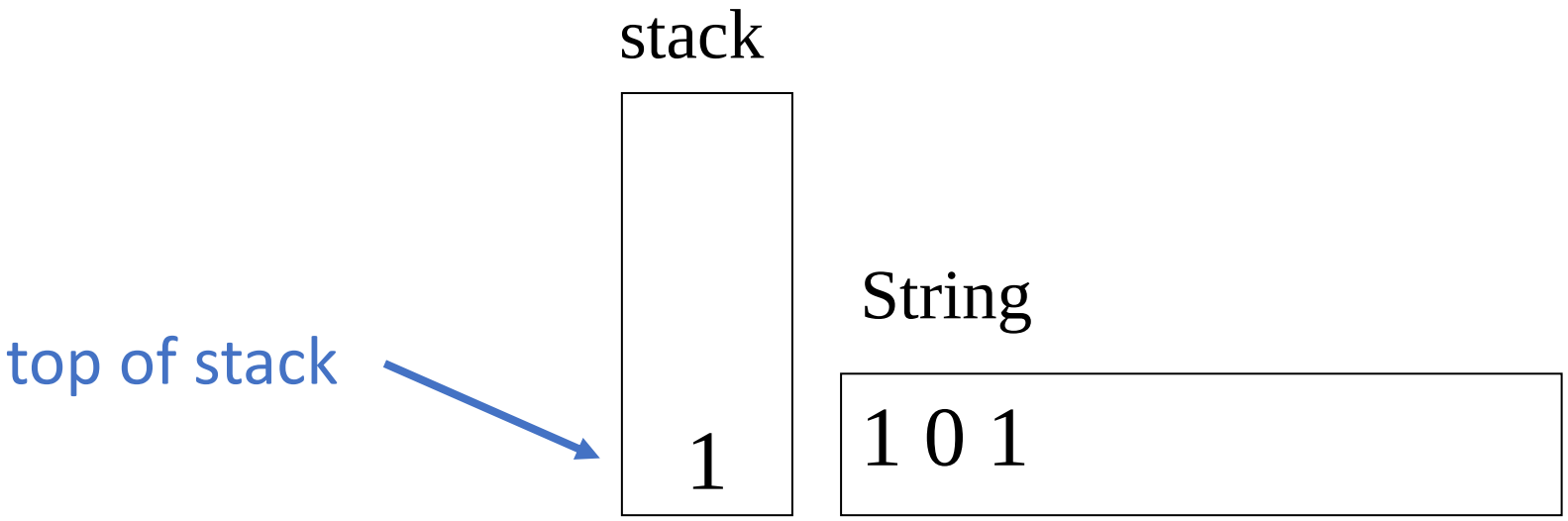
Studi Kasus - Penyelesaian

- Pop stack dan masukkan ke dalam String



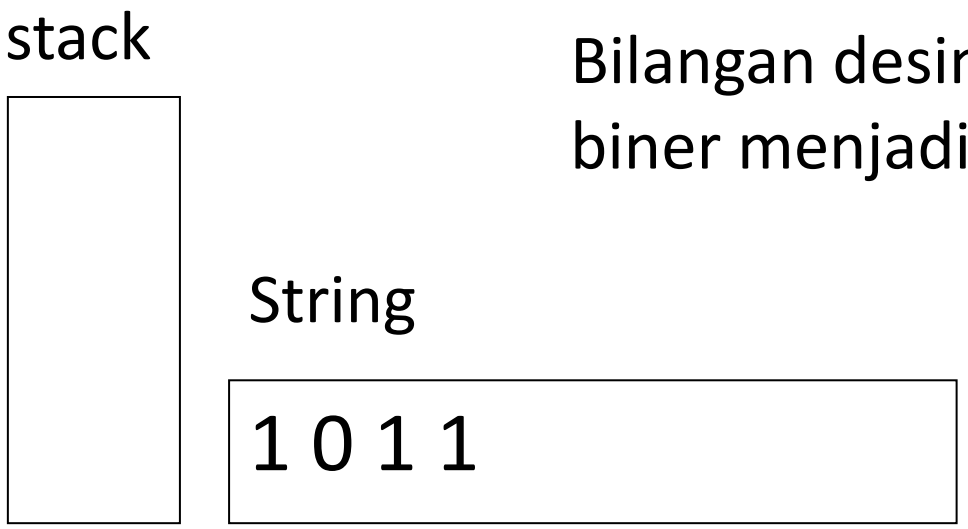
Studi Kasus - Penyelesaian

- Pop stack dan masukkan ke dalam String



Studi Kasus - Penyelesaian

- Pop stack dan masukkan ke dalam String



Bilangan desimal **11** dikonversi ke biner menjadi **1011**

Latihan

1. Lakukan konversi notasi infix berikut menjadi notasi postfix menggunakan konsep Stack!
   a. 6 % 3 + 5 ^ 2
   b. 2 + 4 * (9 - 5) / 3
   c. 12 - 3 ^ (4 % 2)
2. Lakukan konversi bilangan desimal berikut ke dalam biner menggunakan konsep Stack!
   a. 14
   b. 27