Stack

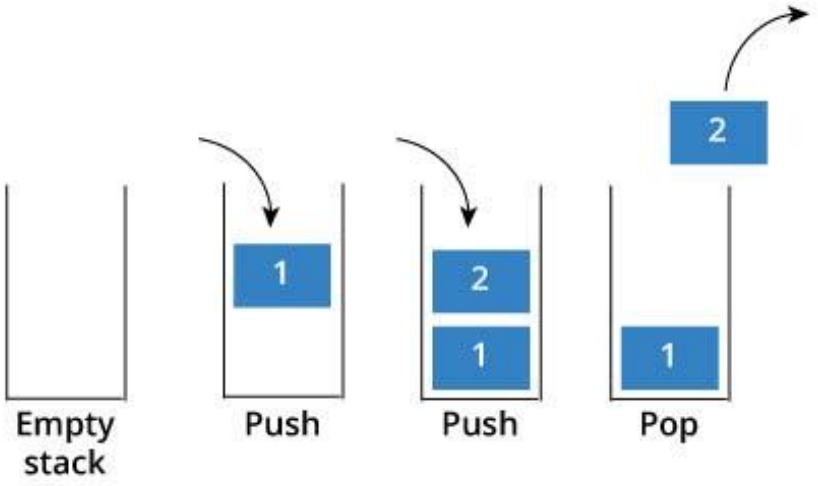
TITA KARLITA
UMI SA'ADAH
ARNA FARIZA
ENTIN MARTIANA KUSUMANINGTYAS

Materi

- Definisi Stack
- Operasi dasar stack
- Implementasi stack

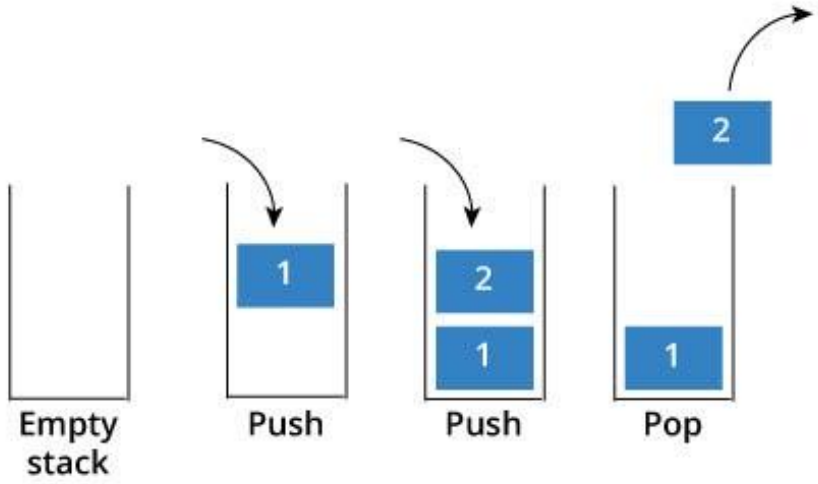
Apakah Stack ?

- Abstract Data Type (ADT)
- Diilustrasikan sebagai sebuah container yang dapat digunakan untuk menyimpan sekumpulan elemen/data/item.
- Operasi dasar stack:
  - Push : tambah data
  - Pop : ambil data

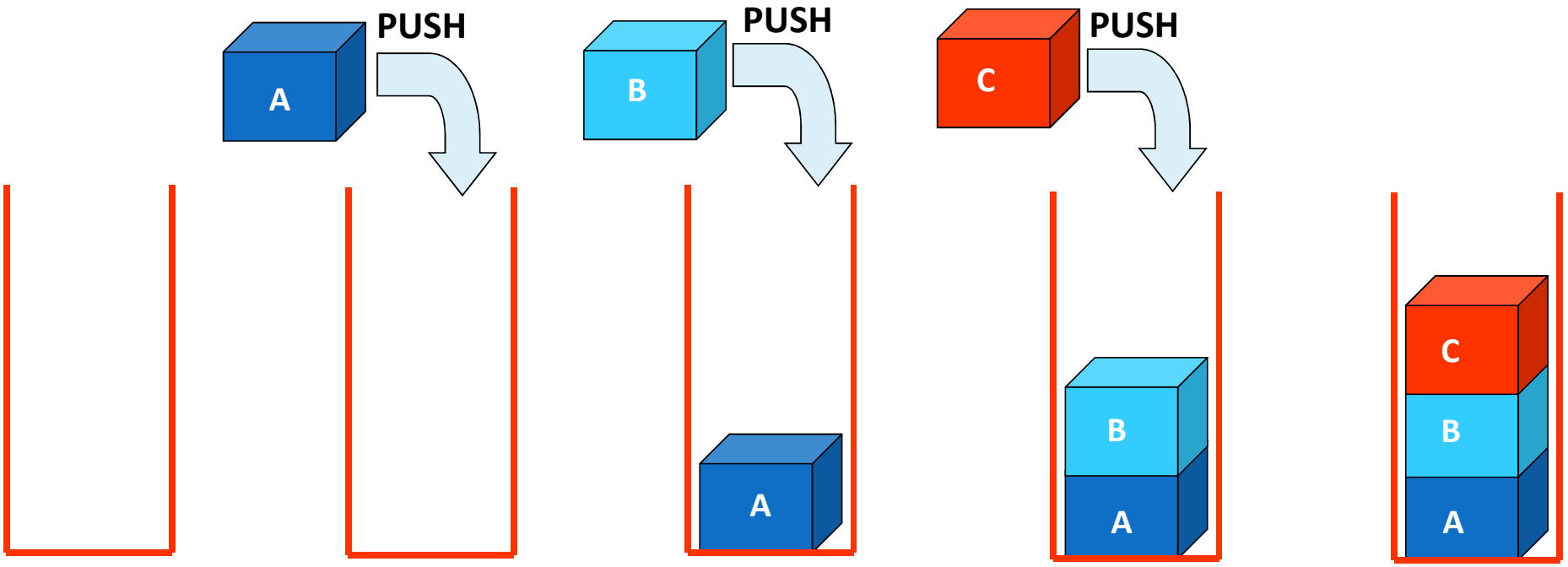


Apakah Stack ?

- Menerapkan konsep Last In First Out (LIFO)
- Operasi Push dan Pop hanya melalui satu sisi
- Data yang disimpan terakhir akan diambil lebih dahulu

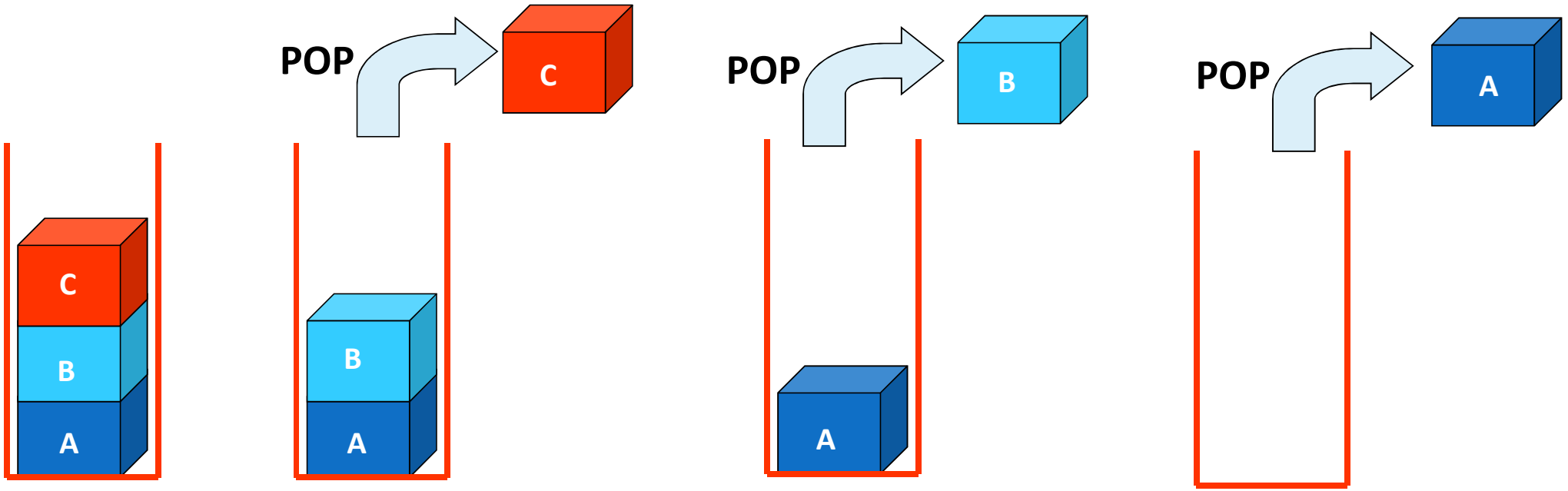


Ilustrasi operasi push



Operasi Push meletakkan elemen/item ke stack

Ilustrasi operasi POP



Operasi Pop mengambil elemen/item dari stack

Implementasi stack

- Implementasi stack menggunakan:
  - Array
  - Linked list
- Implementasi stack dengan array, kemungkinan stack dalam kondisi penuh
- Implementasi stack dengan linked list, stack tidak pernah penuh

Operasi pada Stack

- Push : menyimpan item pada stack
- Pop : mengambil item dari stack
- Inisialisasi : inisialisasi awal stack
- Penuh : stack dalam kondisi penuh
- Kosong : stack dalam kondisi kosong

Representasi Stack dengan Array

Mengguakan tipe data struktur yang terdiri dari dua field:

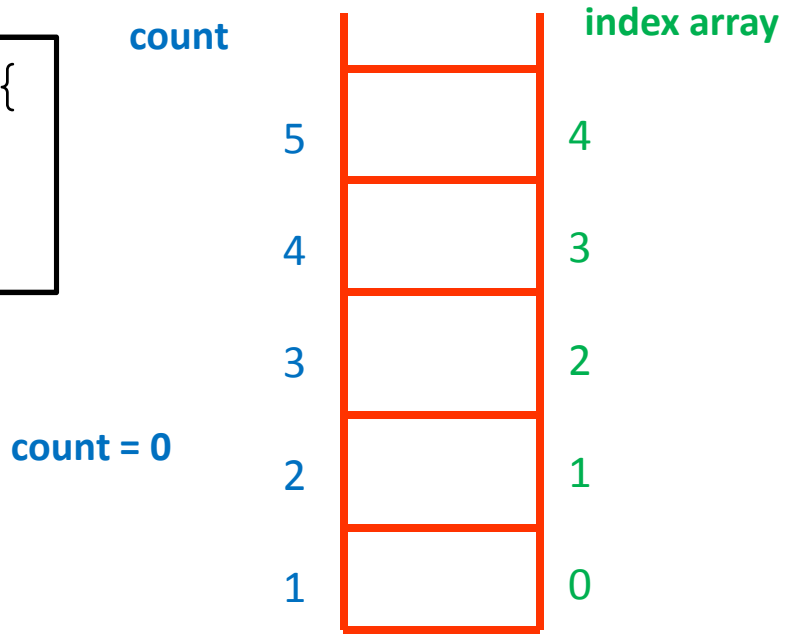
- Field pertama bertipe array untuk menyimpan elemen stack
- Field kedua bertipe integer untuk:
  - mencatat jumlah elemen
  - sebagai penunjuk lokasi elemen pada stack pada saat dilakukan operasi pop dan push.

```
#define MAX 5
typedef char itemType;
typedef struct {
    itemType data[MAX];
    int count;
} stack;
```

Operasi Inisialisasi

- Menginisialisasi stack saat pertama kali dibuat.
- Set jumlah elemen dengan 0 (count=0)

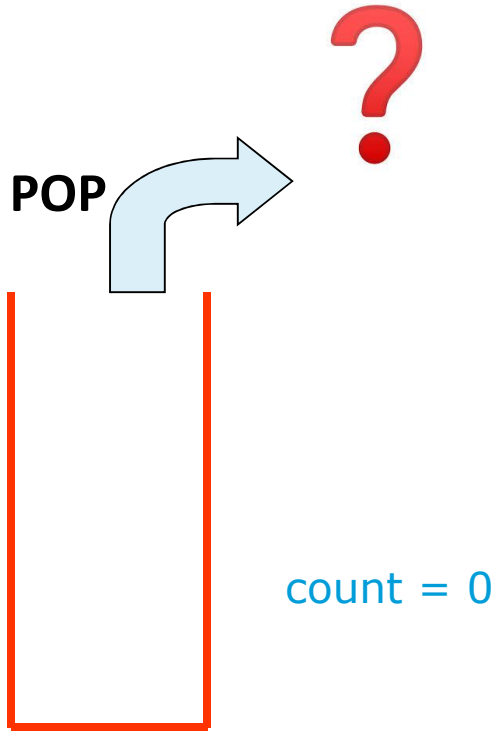
```
void inisialisasi (stack *s){
   s->count = 0;
}
```



Operasi Cek Stack Kosong

- Melakukan pengecekan apakah stack Kosong atau Tidak (Jika kosong return value=1, sebaliknya value=0)
- Digunakan bila melakukan operasi POP

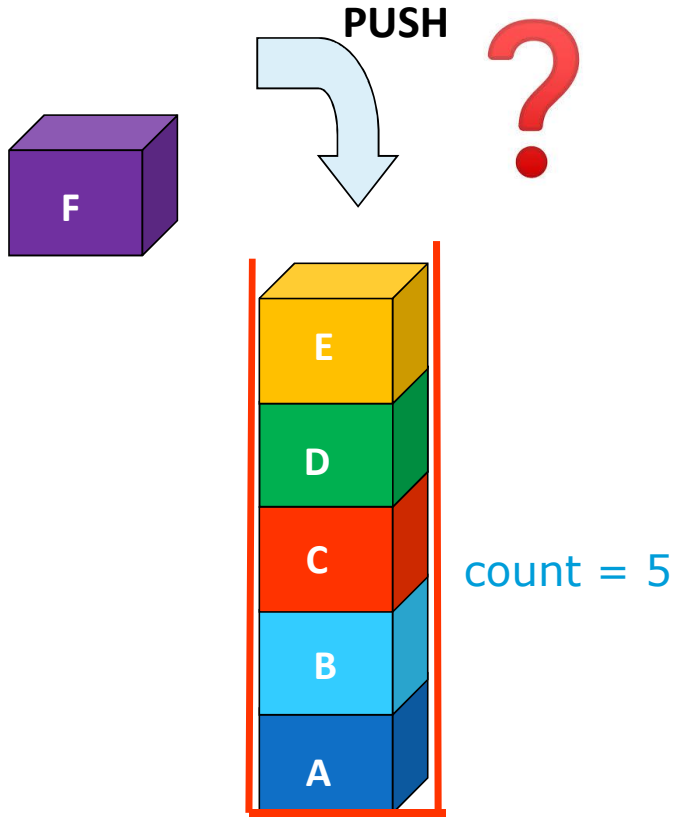
```
int kosong (stack *s){
   if(s->count==0)
        return(1);
   else
        return(0);
}
```



Operasi Cek Stack Penuh

- Melakukan pengecekan apakah stack Penuh atau Tidak (Jika penuh return value=1, sebaliknya value=0)
- Digunakan bila melakukan operasi PUSH

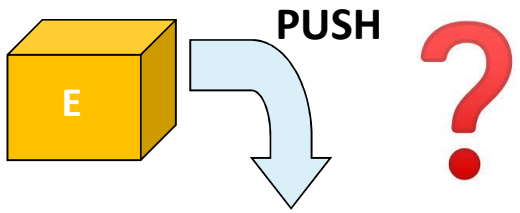
```
int penuh(stack *s){
   if(s->count==MAX)
      return(1);
   else
      return(0);
}
```

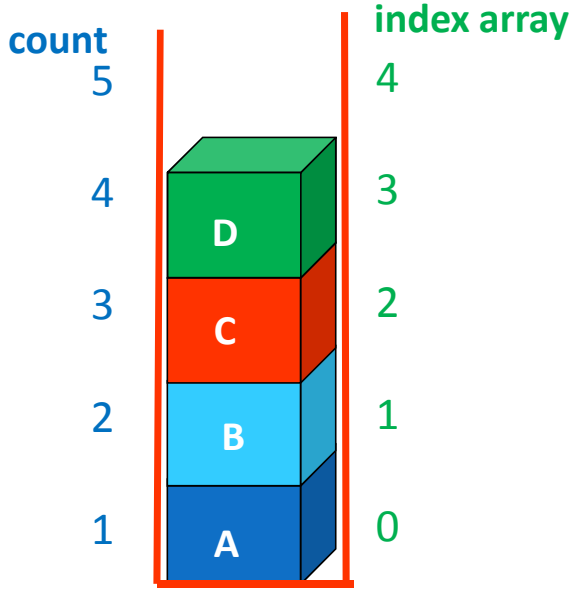


Operasi PUSH

- Untuk memasukkan dan menyimpan elemen pada posisi teratas
- Bila array penuh, tidak dapat melakukan operasi Push
- Setelah dilakukan penyimpanan, jumlah count di-increment

```
void push(itemType x, stack *s){
   if(penuh(s)){
       printf("Stack penuh, data tidak dapat
              disimpan.\n");
   }else{
      s->data[s->count]=x;
      s->count++;
   }
}
```



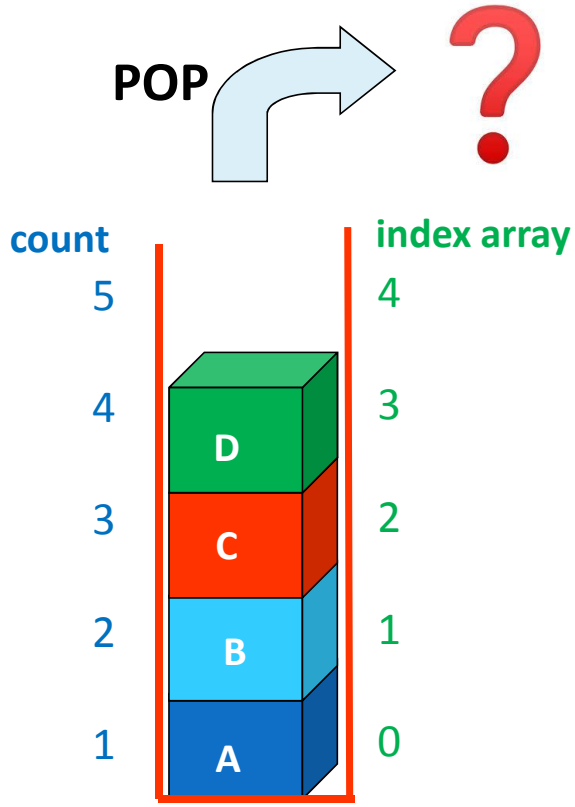


Operasi POP

- Mengambil data pada posisi teratas
- Bila array Kosong tidak dapat dilakukan operasi Pop
- Sebelum mengambil data, nilai count di-decrement

```
itemType pop(stack *s){
   itemType temp;

   if(kosong(s)){
      printf("Stack Kosong, tidak dapat
              mengambil data\n");
      return ' ';
   }else {
      --s->count;
      temp =s->data[s->count];
      return(temp);
   }
}
```



How To Code Stack Using Array ?

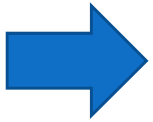
Penerapan stack

- Konversi notasi infix ke posfix
- Konversi basis bilangan
- Pengecekan palindrom

Penerapan stack: Konversi notasi infix ke posfix
INFIX

- Pada ekspresi operasi numerik, pemakaian tanda kurung mempengaruhi hasil akhir.
- Cara penulisan ungkapan tersebut disebut dengan notasi infix, yaitu operator ditulis diantara dua operand.

Contoh notasi infix



$$(A + B) * (C - D)$$

$$A + B * C - D$$

Penerapan stack: Konversi notasi infix ke posfix
PREFIX

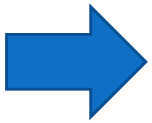
- Seorang ahli matematika, Jan Lukasiewicz mengembangkan cara penulisan ungkapan numeris yang disebut notasi polish atau notasi prefix,
- Notasi prefix: operator ditulis **sebelum** kedua operand yang akan disajikan.
- Notasi prefix tidak memerlukan tanda kurung pengelompokan.

Infix	Prefix
A + B	+ A B
A + B – C	- + A B C
(A + B) * (C – D)	* + A B – C D
A – B / (C * D ^ E)	- A / B * C ^ D E

Penerapan stack: Konversi notasi infix ke posfix
POSTFIX

- Notasi postfix: kebalikan dari notasi prefix, operator ditulis **sesudah** kedua operand yang akan disajikan.
- Notasi postfix tidak memerlukan tanda kurung pengelompokan

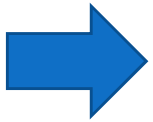
Contoh notasi postfix



(A + B) * (C – D)

[AB+] * [CD-]

p q



pq*

AB+CD–*

Penerapan stack: Konversi notasi infix ke posfix
CONTOH

Notasi Infix	Notasi Postfix
A + B	AB+
A*B+C	AB*C+
A*(B+C)	ABC+*
A*B+C/D	AB*CD/+
(A+B)*C-D/E	AB+C*DE/-
(A+B)*(C-D)^E/F	AB+CD-E^*F/
A+B*C-D^E/F	ABC*+DE^F/-

Derajat operator

- ‘(‘ berderajat 0
- ‘+’ dan’ –‘ berderajat 1
- ‘*’ dan ‘/’ berderajat 2
- ‘^’ (pangkat) berderajat 3

Algoritma Konversi Notasi Infix ke Postfix

1. Sediakan stack untuk menyimpan operator (tipe : char)
2. Siapkan derajat masing-masing operator.
3. Baca setiap karakter notasi infix dari awal i=0

   R=S[i], cek apakah nilai R adalah:
   1. operan : langsung dicetak.
   2. kurung buka ‘(‘ : **push** ke dalam tumpukan.
   3. kurung tutup ‘)’ : **pop** dan cetak semua isi stack sampai ujung stack = ‘(‘.
      pop juga tanda ‘(‘ ini, tetapi tidak dicetak.
   4. operator : jika stack kosong atau derajat operator lebih tinggi dibanding derajat ujung stack (TOS), **push** operator ke dalam stack.
      Jika tidak (derajat operator lebih rendah dibanding derajat ujung stack (TOS)) , **pop** ujung stack (TOS) dan cetak; kemudian ulangi pembandingan R dengan ujung stack (TOS). Kemudian R di-**push.**
4. Jika akhir notasi infix telah tercapai, dan stack masih belum kosong, **pop** semua isi stack dan cetak hasilnya.

Contoh
- (1 + 2) / ((3 – 4) * 5 ^ 6)
  1 2 + 3 4 – 5 6 ^ * /
- 1 + 2 ^ 3 * 4
  1 2 3 ^ 4 * +

Contoh 1: Ilustrasi Konversi Infix ke Postfix

A + B

Notasi infix	Stack	Cetak
A		A
+	+	A
B	+	AB
		AB+

Contoh 2: Ilustrasi Konversi Infix ke Postfix

(A + B) * C

Notasi infix	Stack	Cetak
(	(	
A	(	A
+	(+	A
B	(+	AB
)		AB+
*	*	AB+
C	*	AB+C
		AB+C*

Contoh 2: Ilustrasi Konversi Infix ke Postfix

A + B * C

Notasi infix	Stack	Cetak
A		A
+	+	A
B	+	AB
*	+*	AB
C	+*	ABC
		ABC*+

Ilustrasi Konversi Infix ke Postfix

A + B ^ C * D

Notasi infix	Stack	Cetak
A		A
+	+	A
B	+	AB
^	+^	AB
C	+^	ABC
*	+*	ABC^
D	+*	ABC^D
		ABC^D*+

Rangkuman

- Stack menyimpan elemen/item dengan konsep LIFO, dimana item yang terakhir masuk akan keluar terlebih dahulu
- Elemen pada stack terdiri dari:
  - item yang disimpan di penyimpan
  - count yang menyimpan jumlah elemen dalam stack sekaligus sebagai penunjuk top of stack
- Terdapat dua operasi dasar stack yaitu PUSH dan POP
- Terdapat tiga operasi tambahan stack yaitu INISIALISASI, PENUH, dan KOSONG
- Contoh penerapan stack adalah konversi dari infix ke postfix.