

# STACK

# PENGERTIAN STACK

- merupakan sebuah koleksi objek yang menggunakan prinsip LIFO (Last In First Out), yaitu data yang terakhir kali dimasukkan akan pertama kali keluar dari stack tersebut. Stack dapat diimplementasikan sebagai representasi berkait atau kontigu (dengan tabel fix)

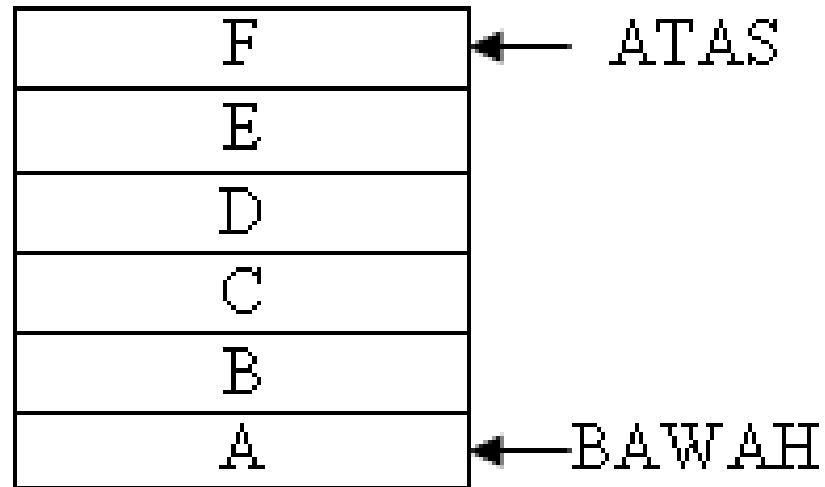
# Ciri Stack

- Elemen TOP (puncak) diketahui
- penyisipan dan penghapusan elemen selalu dilakukan di TOP
- LIFO

# Ilustrasi Stack

- Terdapat dua buah kotak yang ditumpuk, kotak yang satu akan ditumpuk diatas kotak yang lainnya. Jika kemudian stack 2 kotak tadi, ditambah kotak ketiga, keempat, kelima, dan seterusnya, maka akan diperoleh sebuah stack kotak yang terdiri dari  $N$  kotak.

# Ilustrasi Stack - Cont.




# OPERASI PADA STACK

- Push (input E : typeelmt, input/output data : stack):  
menambahkan sebuah elemen ke stack
- Pop (input/output data : stack, output E : typeelmt ) :  
menghapus sebuah elemen stack
- IsEmpty ()
- IsFull ()
- dan beberapas selektor yang lain

# OPERASI PADA STACK – Cont.

- Operasi Push

Push (34)



9
25
3

34
9
25
3

# OPERASI PADA STACK

## 1. buat stack (stack) - **create**

- membuat sebuah stack baru yang masih kosong
- **spesifikasi:**
  - tujuan : mendefinisikan stack yang kosong
  - input : stack
  - syarat awal : tidak ada
  - output stack : - (kosong)
  - syarat akhir : stack dalam keadaan kosong



# OPERASI PADA STACK

## 2. stack kosong (stack) - **empty**

- fungsi untuk menentukan apakah stack dalam keadaan kosong atau tidak
- **spesifikasi:**
  - tujuan : mengecek apakah stack dalam keadaan kosong
  - input : stack
  - syarat awal : tidak ada
  - output : boolean
  - syarat akhir : stack kosong bernilai true jika stack dalam keadaan kosong

# OPERASI PADA STACK

## 3. stack penuh (stack) - **full**

- fungsi untuk memeriksa apakah stack yang ada sudah penuh
- **spesifikasi:**
  - tujuan : mengecek apakah stack dalam keadaan penuh
  - input : stack
  - syarat awal : tidak ada
  - output : boolean
  - syarat akhir : stack penuh bernilai true jika stack dalam keadaan penuh

# OPERASI PADA STACK

## 4. **push** (stack, info baru)

- menambahkan sebuah elemen kedalam stack.
- **spesifikasi:**
  - tujuan : menambahkan elemen, info baru pada stack pada posisi paling atas
  - input : stack dan Info baru
  - syarat awal : stack tidak penuh
  - output : stack
  - syarat akhir : stack bertambah satu elemen

# OPERASI PADA STACK

## 5. **pop** (stack, info pop)

- mengambil elemen teratas dari stack
- **spesifikasi:**
  - tujuan : mengeluarkan elemen dari stack yang berada pada posisi paling atas
  - input : stack
  - syarat awal : stack tidak kosong
  - output : stack dalam info pop
  - syarat akhir : stack berkurang satu elemen

# CONTOH PEMANFAATAN STACK

- **Notasi Infix Prefix**
- **Notasi Infix Postfix**

Pemanfaatan stack antara lain untuk menulis ungkapan dengan menggunakan notasi tertentu.

Contoh :

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

Tanda kurung selalu digunakan dalam penulisan ungkapan numeris untuk mengelompokkan bagian mana yang akan dikerjakan terlebih dahulu.

Dari contoh  $(A + B)$  akan dikerjakan terlebih dahulu, kemudian baru  $(C - D)$  dan terakhir hasilnya akan dikalikan.

$$A + B * C - D$$

$B * C$  akan dikerjakan terlebih dahulu, hasil yang didapat akan berbeda dengan hasil notasi dengan tanda kurung.

# Notasi Infix Prefix

Cara penulisan ungkapan yaitu dengan menggunakan notasi infix, yang artinya operator ditulis diantara 2 operator.

Seorang ahli matematika bernama Jan Lukasiewicz mengembangkan suatu cara penulisan ungkapan numeris yang disebut prefix, yang artinya operator ditulis sebelum kedua operand yang akan disajikan.

Contoh :

Proses konversi

dari infix ke prefix :

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

$$= [ + A B ] * [ - C D ]$$

$$= * [ + A B ] [ - C D ]$$

$$= * + A B - C D$$

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

# Notasi Infix Postfix

Cara penulisan ungkapan yaitu dengan menggunakan notasi postfix, yang artinya operator ditulis sesudah operand.

Contoh :

Proses konversi

dari infix ke postfix :

$$= ( 6 - 2 ) * ( 5 + 4 )$$

$$= [ 6 2 - ] * [ 5 4 + ]$$

$$= [ 6 2 - ] [ 5 4 + ] *$$

$$= 6 2 - 5 4 + *$$

Infix	Postfix
$16 / 2$	$16 2 /$
$( 2 + 14 ) * 5$	$2 14 + 5 *$
$2 + 14 * 5$	$2 14 5 * +$
$( 6 - 2 ) * ( 5 + 4 )$	$6 2 - 5 4 + *$

# Contoh :

Penggunaan notasi postfix dalam stack, misal :

$$2 \ 14 \ + \ 5 \ * \ = \ 80$$

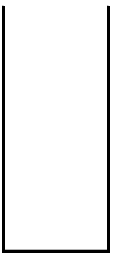
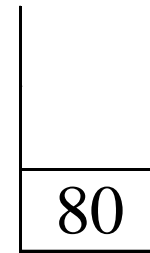
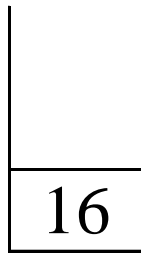
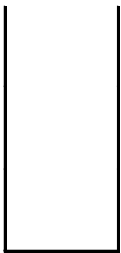
push 2  
push 14

pop 14  
pop 2  
push 2 +  
14

push 5

pop 5  
pop 16  
push 16  
\* 5

pop 80





**TERIMA KASIH**