# 03 Catatan Kuliah Struktur Data
# Stack dan Queue

Dr. Eko Mulyanto Yuniarno

13 September 2023

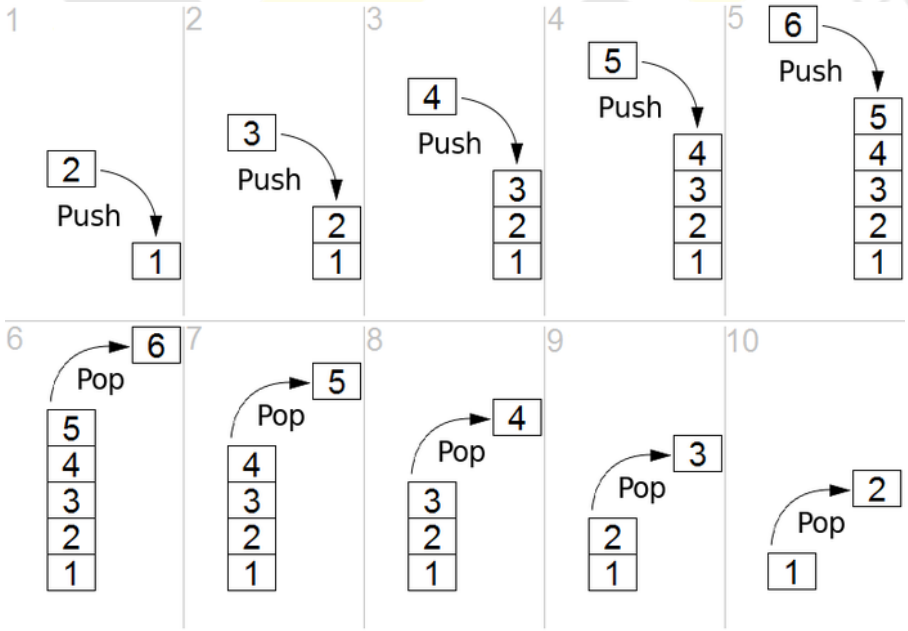
## 1 Stack

Stack adalah struktur data dengan prinsip Last-In-First-Out (LIFO), di mana elemen terakhir yang masuk akan pertama kali dikeluarkan, mirip dengan tumpukan piring

### 1.1 Stack ADT

Beberapa operasi dasar pada stack adalah:

1. Push: Menambahkan elemen ke atas stack.
2. Pop: Mengambil dan menghapus elemen dari atas stack.
3. Peek (atau Top): Melihat elemen teratas stack tanpa menghapusnya.
4. isEmpty: Memeriksa apakah stack kosong.
5. getSize (atau length): Mengembalikan jumlah elemen dalam stack.




Gambar 1

Link Program : Lihat Di Sini

```python
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return len(self.items) == 0

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.isEmpty():
            return self.items.pop()
        else:
            return "Stack is empty"

    def peek(self):
        if not self.isEmpty():
            return self.items[-1]
        else:
            return "Stack is empty"

    def getSize(self):
        return len(self.items)

# Contoh penggunaan stack

s = Stack()
print(s.isEmpty())  # Mengembalikan True karena stack masih kosong

s.push(1)
s.push(2)
s.push(3)

print(s.peek())  # Mengembalikan 3 karena 3 adalah elemen teratas
print(s.pop())   # Menghapus dan mengembalikan 3
print(s.getSize())  # Mengembalikan 2 karena tersisa dua elemen di stack
```

## 2 List

ADT (Abstract Data Type) List adalah kumpulan elemen-elemen dengan urutan tertentu.

Operasi dasar ADT List meliputi:

1. Penambahan elemen.
2. Penghapusan elemen.
3. Pencarian elemen.
4. Perubahan elemen.
5. Pemeriksaan ukuran.
6. Pengecekan keberadaan elemen.
7. Pembersihan list.

contoh program : Link Program : Lihat Di Sini

```python
# Inisialisasi list
lst = []

```

```python
# Penambahan elemen
lst.append(5)

# Penghapusan elemen
lst.remove(5)

# Pencarian elemen (mengembalikan indeks)
index = lst.index(5) if 5 in lst else None

# Pemeriksaan ukuran
size = len(lst)

# Pengecekan keberadaan elemen
exists = 5 in lst

# Pembersihan list
lst.clear()
```

# 3 Aplikasi

## 3.1 perhitungan ekspresi

Tiga Notasi Ekspresi:

1. Infix: Operator ditulis di antara dua operan (misalnya, A + B).
2. Prefix: Operator ditulis sebelum dua operan (misalnya, + A B).
3. Postfix: Operator ditulis setelah dua operan (misalnya, A B +).

Prioritas operator adalah :

| | | |
|---|---|---|
| Exponential operator | ^ | Highest precedence |
| Multiplication/Division | *, / | Next precedence |
| Addition/Subtraction | +, - | Least precedence |

contoh :

1. $(A + B) * C - D$
   - Prefix :
     $- * + ABCD$
   - expresi infix:
     $(((A + B) * C) - D)$
   - expresi postfix :
     $AB + C * D -$

## 3.2 Konversi Infix me Postfix

Algoritma Konversi Infix ke Postfix:

```
1. Inisialisasi:
   * Buat stack 's_stack' untuk menyimpan operator dan tanda kurung.
   * Buat stack 's_output'  untuk menyimpan hasil konversi.
2. Pengolahan Ekspresi:
   * Ulangi untuk setiap karakter char dalam ekspresi exp:
     *Jika 'char' adalah operan (angka atau huruf):
        Tambahkan char ke 's_output'.
```

```
   * Jika 'char' adalah tanda kurung buka '(':
        Push 'char' ke 's_stack'.
   * Jika 'char' adalah tanda kurung tutup ')':
       * Pop operator dari 's_stack' dan tambahkan ke 's_output' sampai tanda kurung
         buka '(' ditemukan di puncak 's_stack'.
       * Pop tanda kurung buka '(' dari 's_stack'.
   * Jika 'char' adalah operator (mis. +, -, *, /):
     * Selama 's_stack' tidak kosong dan prioritas operator di puncak 's_stack' lebih
        tinggi atau sama dengan char:
              * Pop operator dari stack dan tambahkan ke output.
     * Push char ke s_stack.
3. Penyelesaian:
   Selama 's_stack' tidak kosong:
   Pop operator dari 's_stack' dan tambahkan ke 's_output'.
Hasil:
Kembalikan output sebagai ekspresi postfix.
```

Fungsi Prioritas: Fungsi prioritas digunakan untuk menentukan prioritas operator. Misalnya, * dan / memiliki prioritas lebih tinggi daripada + dan -

Contoh :

1. Notasi infix ke postfix $P = 4 * 3 + 1$

| **char** | **s_stack** | **s_output** |
|---|---|---|
| 4 | empty | 4 |
| * | * | 4 |
| 3 | * | 43 |
| + | + | 43* |
| 1 | + | 43*1 |
| empty | empty | 43*1+ |

   Notasi posfix :$43 * 1+$

2. Notasi infix ke postfix `P=A+(B/C-(D*E^F)+G)*H`

P = A + ( B / C - ( D * E ^ F ) + G ) * H

| *Character scanned* | *Stack* | *Postfix Expression (Q)* |
|---|---|---|
| A | ( | A |
| + | ( + | A |
| ( | ( + ( | A |
| B | ( + ( | A B |
| / | ( + ( / | A B |
| C | ( + ( / | A B C |
| – | ( + ( - | A B C / |
| ( | ( + ( - ( | A B C / |
| D | ( + ( - ( | A B C / D |
| * | ( + ( - ( * | A B C / D |
| E | ( + ( - ( * | A B C / D E |
| ^ | ( + ( - ( * ^ | A B C / D E |
| F | ( + ( - ( * ^ | A B C / D E F |
| ) | ( + ( - | A B C / D E F ^ * |
| + | ( + ( + | A B C / D E F ^ * - |
| G | ( + ( + | A B C / D E F ^ * - G |
| ) | ( + | A B C / D E F ^ * - G + |
| * | ( + * | A B C / D E F ^ * - G + |
| H | ( + * | A B C / D E F ^ * - G + H |
| ) | | A B C / D E F ^ * - G + H * + |

3.3 title