

# Laporan Praktikum Struktur Data

Semester Gasal 2021/2022



<b>NIM</b>	<b>71200634</b>
<b>Nama Lengkap</b>	<b>Egi Granaldi Ginting</b>
<b>Minggu ke / Materi</b>	<b>07 / Stack</b>

SAYA MENYATAKAN BAHWA LAPORAN PRAKTIKUM INI SAYA BUAT DENGAN USAHA SENDIRI TANPA MENGGUNAKAN BANTUAN ORANG LAIN. SEMUA MATERI YANG SAYA AMBIL DARI SUMBER LAIN SUDAH SAYA CANTUMKAN SUMBERNYA DAN TELAH SAYA TULIS ULANG DENGAN BAHASA SAYA SENDIRI.

SAYA SANGGUP MENERIMA SANKSI JIKA MELAKUKAN KEGIATAN PLAGIASI, TERMASUK SANKSI TIDAK LULUS MATA KULIAH INI.

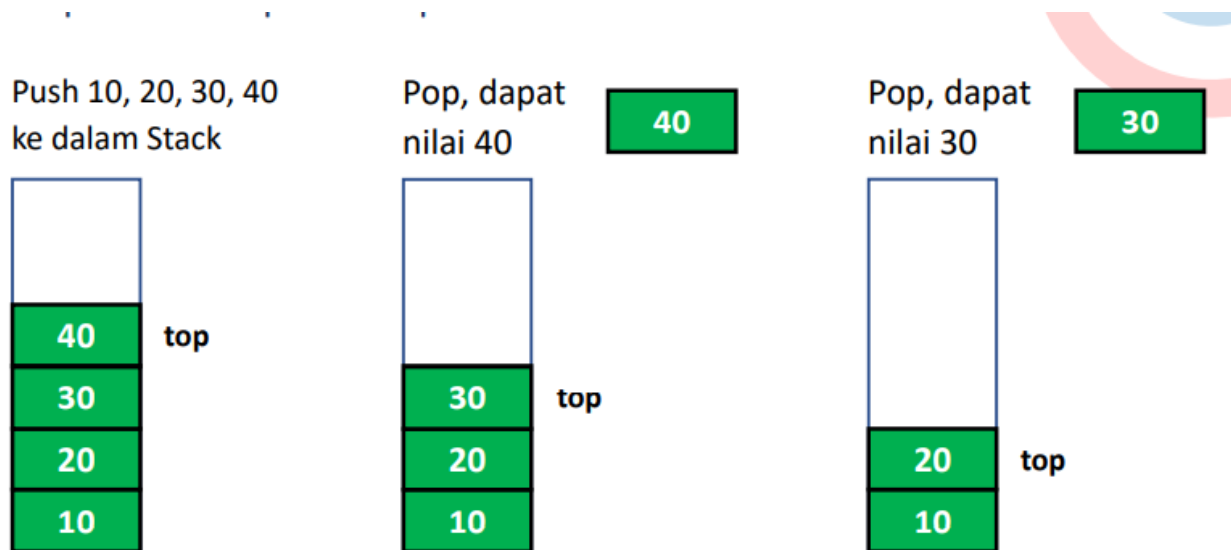
**PROGRAM STUDI INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI  
UNIVERSITAS KRISTEN DUTA WACANA  
YOGYAKARTA  
2021**

## STACK

Stack biasa disebut sebagai tumpukan. Stack atau tumpukan merupakan suatu struktur data yang terbentuk dari barisan hingga yang terurut dari satuan data. Pada stack, penambahan dan penghapusan elemennya hanya dapat dilakukan pada satu posisi, yaitu posisi akhir stack. Posisi tersebut biasa disebut Top dari stack. Salah satu struktur data dinamis yang memiliki aturan khusus pada penambahan maupun pengambilan data yang hanya bisa dilakukan di satu posisi / ujung saja. Prinsip kerja pada stack yaitu Last In First Out (LIFO). Dimana elemen yang terakhir disimpan didalam stack, menjadi elemen pertama yang diambil.

Adapun Operasi – operasi pada Stack adalah sebagai berikut:

- Push : memasukkan data ke dalam stack, data terbaru berada di posisi paling atas.
- Pop : mengeluarkan data yang berada di paling atas.
- Top : melihat nilai yang berada di posisi paling atas.
- Clear : Mengosongkan isi Stack.



Gambar 1 Operasi pada Stack

## MATERI 1

### Implementasi Stack

Implementasi pada Stack dapat kita gunakan dengan menggunakan List dan Linked List.

- Menggunakan List
  - Top bisa diletakkan di depan atau di belakang
  - Relatif mudah karena bisa menggunakan fungsi bawaan dari list
  - Dua fungsi utama yaitu : Append dan pop (jika top diletakkan di belakang)

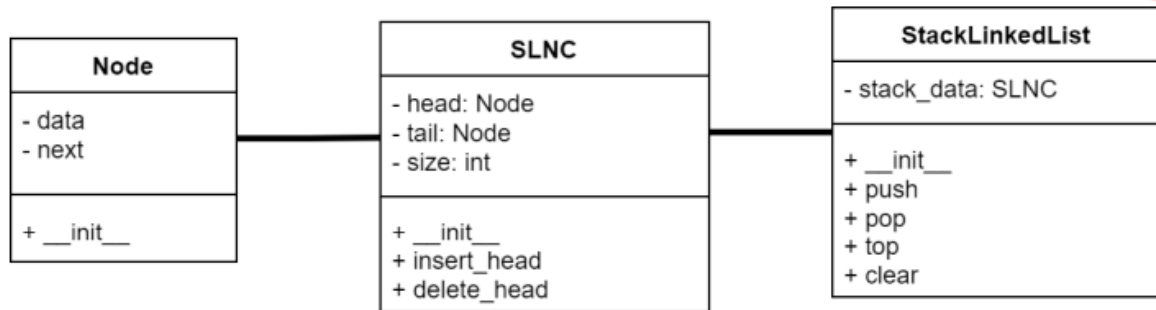
Stack dengan list menggunakan list bawaan dari python yang sudah efisien implementasinya. Adapun Kompleksitas untuk setiap operasi adalah sebagai berikut:

- Push : Menggunakan Fungsi append pada list,  $O(1)$
- Pop : Menggunakan fungsi pop pada list,  $O(1)$
- Top : Menggunakan index -1,  $O(1)$
- Clear : Menggunakan fungsi clear pada list,  $O(1)$

StackList
- stack_data: list
+ __init__ + push + pop + top + clear

*Gambar 2 Stack dengan List*

- Menggunakan Linked List
  - Lebih mudah jika Top diletakkan di posisi head
  - Bisa menggunakan Class Linked List yang sudah pernah dibuat sebelumnya.
  - Menggunakan Linked List, sehingga kompleksitasnya tergantung pada bentuk implementasinya. Top diletakkan di bagian head atau tail.
  - Operasi Push : jika head dianggap sebagai top, maka  $O(1)$  karena jumlah Langkah untuk insert head selalu tetap.
  - Operasi Pop : jika head dianggap sebagai top, maka  $O(1)$  jika tail dianggap top, maka kompleksitas dari pop adalah  $O(n)$
  - Operasi Top : posisi pada head/tail, maka  $O(1)$
  - Operasi Clear : Menghapus satu – persatu Node, maka  $O(n)$



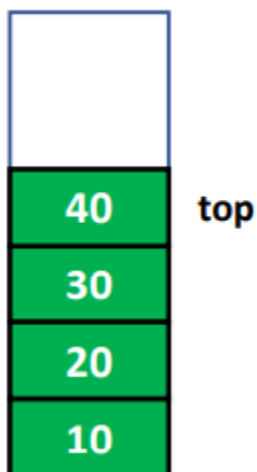
Gambar 3 Stack dengan Linked List

## MATERI 2

### Kasus – Kasus untuk Stack

#### Reversing Data

Stack bisa digunakan untuk membalik urutan data karena stack beroperasi berdasarkan prinsip LIFO. Sebagai contoh Push 10, 20, 30, 40 ke dalam stack. Jika melakukan operasi Pop, maka akan didapatkan urutan data menjadi terbalik. Jika pada awalan 10,20,30,40, maka akan menjadi 40,30,20,10.



Gambar 4 Reversing Data pada Stack

## Bracket Matching

Bracket Matching digunakan untuk melakukan pengecekan apakah semua bracket digunakan dengan benar, apakah sesuai dengan urutan ( pembuka, penutup), bisa juga nested. Apakah sesuai dengan pasangannya {}, (), [].

- **Algoritma Bracket Matching**

Untuk setiap karakter pada input string

- karakter bukan tanda kurung, abaikan
- karakter kurung buka ( , [ , atau { push ke dalam stack
- karakter kurung tutup ) , ] , atau }

Langkah selanjutnya pop stack, cek hasilnya. If hasilnya kurung buka yang tidak sesuai dengan kurung tutup, input bukan ekspresi yang benar. else lanjutkan baca karakter.

- If stack tidak kosong setelah baca seluruh input string, input bukan ekspresi yang benar.
- else stack kosong, input ekspresi benar.

## Infix, Postfix, dan Prefix

- Infix notation :  $X + Y$   
 $A * (B + C) / D$
- Postfix notation ( Reverse Polish )  $X Y +$   
 $A B C + * D /$
- Prefix notation ( Polish ) :  $+ X Y$   
 $/* A + B C D$

## Algoritma Infix to Postfix

### Infix to Postfix

- Misalkan operasi:  $3 + 2 * 5$
- Operasi di atas disebut notasi infix, notasi infix tersebut harus diubah menjadi notasi postfix
- $3 + 2 * 5$  notasi postfixnya adalah  $3 2 5 * +$ 
  - $a+b*c-d$ 
    - Stack OP (kosong) dan Postfix (Output kosong)
  - Scan a
    - Postfix : a
  - Scan +
    - Stack OP: +
  - Scan b
    - Postfix: ab
  - Scan \*, karena  $\text{Tos}(+) < *$ , maka add ke stack
    - Stack OP: +\*
  - Scan C
    - Postfix: abc
  - Scan -, karena  $\text{ToS}(*) > -$ , maka pop Stack, dan add ke Postfix
    - Stack OP: +
    - Postfix: abc\*
    - Karena  $\text{ToS}(+) \geq -$ , maka pop Stack, dan add \* ke Postfix, karena Stack kosong, maka push - ke stack
    - Stack OP: -
    - Postfix: abc\*+
  - Scan d
    - Postfix: abc\*+d
  - Karena sudah habis, push  $\text{ToS}(-)$  stack ke Postfix
  - - Postfix: abc\*+d-

### Postfix Evaluator

- Jika berupa operand, maka masukkan ke Stack Hasil
- Jika berupa operator, maka:
  - Pop Nilai pertama dari stack hasil = y
  - Pop nilai kedua dari Stack Hasil = x
  - Lakukan operasi sesuai dengan operator yang didapat.

## BAGIAN 2: SOAL UNGUIDED

Pada bagian ini anda menuliskan jawaban dari soal-soal unguided yang diberikan. Jawaban anda harus disertai dengan source code, penjelasan dan screenshot output

### Unguided 1

```
class Konversi:
    def __init__(self, isi):
        self.top = -1
        self.isi = isi
        self.stack = []
        self.output = []
        self.precedence = {'+':2, '-':2, '*':3, '/':3, '^':4}

    def isEmpty(self):
        if self.top == -1:
            return True
        else:
            return False

    def peek(self):
        return self.stack[-1]

    def pop(self):
        if not self.isEmpty():
            self.top -= 1
            return self.stack.pop()
        else:
            return "$"

    def push(self, masuk):
        self.top += 1
        self.stack.append(masuk)

    def isOperand(self, ch):
        return ch.isalpha()

    def notGreater(self, i):
        try:
            a = self.precedence[i]
            b = self.precedence[self.peak()]
            if a <= b:
                return True
            else:
                False
```

```

except KeyError:
    return False

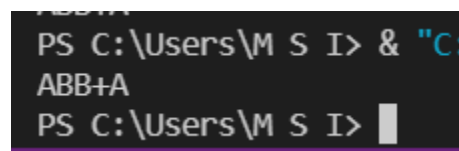
def infixToPostfix(self, exp):

    for i in exp:
        if self.isOperand(i):
            self.output.append(i)
        if i == '(':
            self.push(i)
        if i == ')':
            while( (not self.isEmpty()) and
                    self.peak() != '('):
                a = self.pop()
                self.output.append(a)
            if (not self.isEmpty() and self.peak() != '('):
                return -1
            else:
                self.pop()
        else:
            while(not self.isEmpty() and self.notGreater(i)):
                self.output.append(self.pop())
            self.push(i)
    while not self.isEmpty():
        self.output.append(self.pop())

    print ("".join(self.output))

exp = 'A+B'
obj = Konversi(len(exp))
obj.infixToPostfix(exp)

```



```

PS C:\Users\M S I> & "C:\Users\M S I>
ABB+A
PS C:\Users\M S I>

```

Pada soal diatas class akan menginisialisasi penghubung dalam class Konversi, setelah itu dia akan melihat variabel stacknya, dan membuat pengaturan prioritas. Pada fungsi def isEmpty, class akan memeriksa isi didalam stack apakah kosong atau tidak. Setelah itu dia akan mengembalikan nilai pada bagian atas Stack dengan menggunakan function def peek. Setelah itu akan melakukan pop dari dalam stack, untuk menghapus data teratas dari stack. Method Push pada soal diatas digunakan untuk menambahkan nilai/data baru pada stack. Lalu akan memeriksa karakter alfabet yang telah dimasukkan ke dalam stack dan memeriksa prioritas yang ada didalam stack. Setelah itu class akan melakukan perulangan While untuk mengecek kondisi, jika kondisi masih bernilai true, maka looping akan terus berlanjut.



Daftar Pustaka :

[http://sisfo.binadarma.ac.id/upload/materi/8868\\_SD4-STACK%20%28Tumpukan%29.pdf](http://sisfo.binadarma.ac.id/upload/materi/8868_SD4-STACK%20%28Tumpukan%29.pdf)

[ASD-Modul-4-STACK.pdf](#)