

# **LAPORAN PRAKTIKUM STRUKTUR DATA**

**MODUL KE-03**

**STACK DALAM PYTHON**



**Disusun Oleh:**

**Nama** : Restu Wibisono  
**NPM** : 2340506061  
**Kelas** : 03 (Tiga)

**Program Studi S1 Teknologi Informasi**

**Fakultas Teknik, Universitas Tidar**

**Genap 2023/2024**

## **I. Tujuan Praktikum**

Praktikum ini akan memberikan pemahaman dan latihan dalam struktur data stack menggunakan singel linked list dalam python. Serta memahami konsep dasar OOP (Object-Oriented Program) dengan mendefinisikan kelas 'Node' serta kelas 'Stack' sebagai struktur data. Selain itu juga memberikan dalam manipulasi, seperti menambahkan elemen ke dalam stack (push), menghapus elemen (pop) , dan melihat elemen teratas (peek).

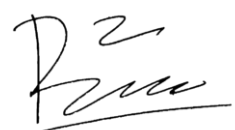
## **II. Dasar Teori**

Stack adalah struktur data yang digunakan dalam pemrograman komputer untuk menyimpan serta mengelola data secara berurutan. Prinsip utama dari struktur data stack yaitu Last In, First Out (LIFO), yang berarti elemen yang terakhir dimasukkan ke dalam stack bisa menjadi elemen pertama yang diambil (pop). Stack biasanya memiliki dua operasi utama: push, dengan menambahkan elemen ke dalam stack, dan pop, yang menghapus dan mengembalikan elemen teratas dari stack.

Dalam implementasinya, stack dapat diwujudkan menggunakan banyak struktur data, salah satunya adalah linked list. Linked list adalah struktur data linier yang terdiri dari serangkaian simpul yang saling terhubung. Dalam konteks stack, linked list juga sering digunakan karena akan memungkinkan operasi push dan pop yang dilakukan dengan cepat, terutama saat menambahkan atau menghapus elemen dari ujung linked list.

Dengan di pahami konsep stack dan linked list, programmer mampu mengembangkan solusi untuk berbagai masalah dalam pengembangan perangkat lunak, seperti manajemen memori, pengolahan ekspresi matematika, dan penjadwalan tugas.

Tanda Tangan



### III. Hasil dan Pembahasan

#### A. Stack dengan List

```
# Python program to demonstrate stack implementation using list
stack = []

# append() function to push element in the stack
stack.append('a')
stack.append('i')
stack.append('u')
stack.append('e')
stack.append('o')

print('initial stack')
print(stack)

# pop() function to pop element form stack in LIFO order
print('\nElements popped form stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)

# uncommenting print(stack.pop()) will cause an IndexError as the stack is now empty

initial stack
['a', 'i', 'u', 'e', 'o']

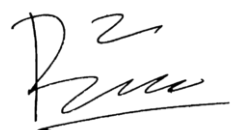
Elements popped form stack:
o
e
u

Stack after elements are popped:
['a', 'i']
```

(Gambar 3.)

- Membuat sebuah variabel dengan nama 'stack' yang merupakan list kosong. Ini akan berfungsi sebagai stack untuk menyimpan elemen-elemen.
- Membuat 'stack.append()' yang dimasukkan elemen 'a', 'i', 'u', 'i', 'o' kedalam stack menggunakan 'append' yang membuat 'a' berada dipaling bawah dan 'o' berada di top atau atas.
- Mencetak string 'initial stack' yang akan membuat tanda bahwa ini adalah stack awal.
- Mencetak isi dari stack yang akan dicetak dari top hingga elemen yang paling bawah.

Tanda Tangan



- Mencetak pesan “Elemen popped from stack”, memberi tahu bahwa elemen akan popped dari beberapa elemen stack.
- Memanggil metode ‘pop()’ untuk mengambil dan mencetak tiga elemen teratas sesuai dengan urutan LIFO. Dan setiap ‘pop()’ akan menghapus elemen dari stack.
- Mencetak pesan “Stack after elements are popped”,
- Mencetak isi dari stack setelah beberapa elemen stack dipop.

#### B. Stack dengan Collections.deque

```
# Python program to demonstrate stack implementation using collection.deque
from collections import deque

stack = deque()

# append() function to push element in the stack
stack.append('a')
stack.append('i')
stack.append('u')
stack.append('e')
stack.append('o')

print('initial stack')
print(stack)

# pop() function to pop element from stack in LIFO order
print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)

# uncommenting print(stack.pop()) will cause an IndexError as the stack is now empty

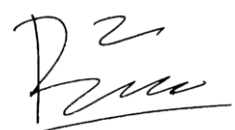
initial stack
deque(['a', 'i', 'u', 'e', 'o'])

Elements popped from stack:
o
e
u

Stack after elements are popped:
deque(['a', 'i'])
```

(Gambar 3.2)

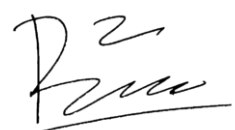
Tanda Tangan



- Pertama mengimpor class 'deque' dari modul 'collections'. Deque ini mirip dengan lis, tetapi ditambahkan dengan operasi menambah dan menghapus diujungnya.
- Membuat objek 'deque' yang disebut 'stack' yang berfungsi digunakan untuk stack.
- Baris selanjutnya menambahkan beberapa elemen ke dalam stack dengan metode 'append()' dari object 'deque', elemen yang ditambahkan akan sesuai dengan urutan pemanggilan 'append()'.
- Mencetak string 'initial stack' yang akan membuat tanda bahwa ini adalah stack awal.
- Kemudian program akan mencetak pada objek 'deque', maka seluruh dari elemen stack akan dicetak.
- Selanjutnya mencetak "Elemen popped from stack" dan diikuti dengan pemanggilan 'pop()' untuk mengambil dan mencetak yang sesuai dengan urutan LIFO.
- Baris berikutnya menggunakan metode 'pop()' dari object 'deque' untuk menghapus dan mencetak elemen dari stack dan mengembalikan elemennya.
- Kemudian mencetak pesan "Stack after elements are popped:" ke layar, yang menandakan bahwa kita akan menampilkan stack setelah beberapa elemen telah dipop.
- Terakhir mencetak isi dari stack setelah tiga elemen dipop. Ini akan mencetak stack yang tersisa setelah operasi pop dilakukan.

C. Stack dengan queue.LifoQueue

Tanda Tangan



```

# Python program to demonstrate stack implementation using queue module
from queue import LifoQueue

# Initializing a stack
stack = LifoQueue(maxsize=5)

# qsize() show the number of elements in the stack
print(stack.qsize())

# put() function to push element in the stack
stack.put('a')
stack.put('i')
stack.put('u')
stack.put('e')
stack.put('o')

print('Full: ', stack.full())
print('Size: ', stack.qsize())

# get() function to pop element form stack in LIFO order
print('\nElements popped form the stack')
print(stack.get())
print(stack.get())
print(stack.get())

print('\nEmpty: ', stack.empty())

0
Full:  True
Size:  5

Elements popped form the stack
o
e
u

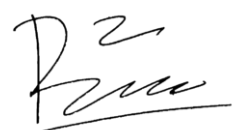
Empty:  False

```

(Gambar 3.3)

- Pertama mengimpor 'LifoQueue' dari kelas 'queue'. 'LifoQueue' adalah struktur data yang mempresentasikan stack.
- Membuat objek 'stak' yang merupakan 'LifoQueue' dengan kapasitas maksimum 5.
- Baris selanjutnya akan mencetak jumlah elemen dalam stack menggunakan metode 'qsize()' dari 'stack'.
- Membuat tambahan beberapa elemen ke dalam stack menggunakan metode 'put()' dari 'stack'.
- Mencetak stack menggunakan metode 'full()' dan ukuran stack menggunakan metode 'qsize()'

Tanda Tangan



- Selanjutnya menggunakan metode 'get()' dari object 'stack'. Karena LIFO Queue, elemen ini diambil dengan urutan LIFO.
- Terakhir mencetak stack yang sudah kosong dengan 'empty()', jika kosong mencetak 'True' jika tidak maka mencetak 'False'.

#### D. Stack dengan Singel Lingked List

```
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class Stack:

    # Initalize the stack, use a dummy node, which is easier for handling edge cases.
    def __init__(self):
        self.head = Node("head")
        self.size = 0

    # String representation of the stack
    def __str__(self):
        cur = self.head.next
        out = ""
        while cur:
            out += str(cur.value) + "->"
            cur = cur.next
        return out[:-2]

    # Get the current size of the stack
    def getSize(self):
        return self.size

    # Check if the stack is empty
    def isEmpty(self):
        return self.size == 0

    # Get the top item of the stack
    def peek(self):

        # Sanitary check to see if we are peeking an empty stack
        if self.isEmpty():
            raise Exception("Peeking from an empty stack")
        return self.head.next.value

    # Push a value into the stack
    def push(self, value):
        node = Node(value)
        node.next = self.head.next
        self.head.next = node
        self.size += 1

    # Remove a value from the stack and return
    def pop(self):
        if self.isEmpty():
            raise Exception("Popping from an empty stack")
        remove = self.head.next
        self.head.next = self.head.next.next
        self.size -= 1
        return remove.value

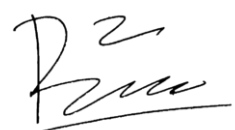
# Driver code
if __name__ == "__main__":
    stack = Stack()
    for i in range(1, 11):
        stack.push(i)
    print(f"Stack: {stack}")

    for _ in range(1, 6):
        remove = stack.pop()
        print(f"Pop: {remove}")
    print(f"Stack: {stack}")

Stack: 10->9->8->7->6->5->4->3->2->1
Pop: 10
Pop: 9
Pop: 8
Pop: 7
Pop: 6
Stack: 5->4->3->2->1
```

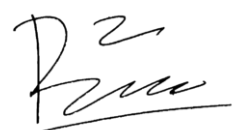
(Gambar 3.4)

Tanda Tangan



- Pertama mendefinisikan kelas 'Node' yang setiap node dalam linked list memiliki dua atribut yaitu 'value' untuk menyimpan nilai dan 'next' untuk menunjukkan ke node berikutnya.
- Selanjutnya mendefinisikan kelas 'Stack' yang mengimplementasikan stack memakai linked list.
- Membuat konstruktor kelas 'Stack' dengan sebuah node sentinel 'head' yang mengandung nilai dan mengatur 'size' menjadi 0.
- Menunjang seriap node dalam stack, mulai dari node pertama setelah 'head' dan menambahkan nilai ke dalam string 'out'
- Untuk mendapatkan jumlah elemen dari stack.
- Lalu program akan mengecek apakah stack kosong atau tidak.
- Kemudian 'peek' berfungsi sebagai melihat elemen teratas dari stack tanpa menghapusnya. Jika kosong, menimbulkan pengecualian.
- Membuat metode untuk menambahkan elemen baru ke stack. Kemudian mengatur 'next' ke node yang menjadi elemen teratas dari stack lalu mengatur 'head.next' ke node baru.
- Menghapus node yang menjadi elemen teratas dari stack, mengatur 'head.next' ke node berikutnya, mengurangi ukuran stack, dan mengembalikan nilai dari node yang dihapus.
- Blok kode terakhir untuk menguji kelas 'Stack' yang telah didefinisikan sebelumnya. Dan membuat objek 'stack' menambahkan angka dari 1 sampai 10 dalam stack, kemudian menghapus elemen teratas menggunakan 'pop()' dan mencetak stack.

Tanda Tangan



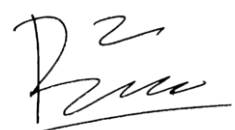


#### **IV. Kesimpulan**

Secara keseluruhan, praktikum implementasi struktur data stack menggunakan single linked list dalam bahasa pemrograman Python ini memberikan pemahaman yang baik tentang konsep dasar struktur data stack dan penggunaannya dalam pemrograman. Dalam praktikum ini, kita belajar tentang bagaimana membuat stack menggunakan linked list, termasuk bagaimana menambahkan elemen baru ke dalam stack (push), menghapus elemen dari stack (pop), dan melihat elemen teratas dari stack (peek).

Selain itu, praktikum ini juga membahas penanganan kasus khusus, seperti menangani stack kosong saat melakukan operasi pop atau peek. Dengan pemahaman yang diperoleh dari praktikum ini, kita dapat mengimplementasikan stack dalam berbagai aplikasi, termasuk dalam manajemen memori, pengolahan ekspresi matematika, dan penjadwalan tugas. Kesimpulannya, praktikum ini memberikan landasan yang kuat dalam memahami dan mengimplementasikan struktur data stack, yang merupakan keterampilan penting dalam pengembangan perangkat lunak.

Tanda Tangan

A handwritten signature in black ink, appearing to be 'P. Z. Rana', is written over a light blue grid background.