

LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL KE-3

STACK PADA PYTHON



Disusun Oleh:

Nama : Oktario Mufti Yudha
NPM : 2320506044
Kelas : 04 (Empat)

Program Studi S1 Teknologi Informasi

Fakultas Teknik, Universitas Tidar

Genap 2023/2024

I. Tujuan Praktikum

Praktikum ini bertujuan untuk memperkenalkan mahasiswa pada konsep dasar stack serta memberikan pemahaman praktis dalam mengimplementasikannya menggunakan bahasa pemrograman Python. Praktikum ini bertujuan untuk memberikan mahasiswa pemahaman yang kuat tentang bagaimana stack beroperasi, termasuk konsep push (menambahkan elemen), pop (menghapus elemen), dan peek (melihat elemen paling atas tanpa menghapusnya)

II. Dasar Teori

struktur data stack adalah konsep LIFO (Last In, First Out), yang berarti elemen terakhir yang dimasukkan ke dalam tumpukan adalah elemen pertama yang akan dikeluarkan. Struktur data tumpukan mirip dengan tumpukan buku di meja; kita hanya dapat menambahkan atau mengambil buku dari atas tumpukan, bukan dari tengah atau dasarnya. Beberapa operasi dasar dalam tumpukan meliputi:.

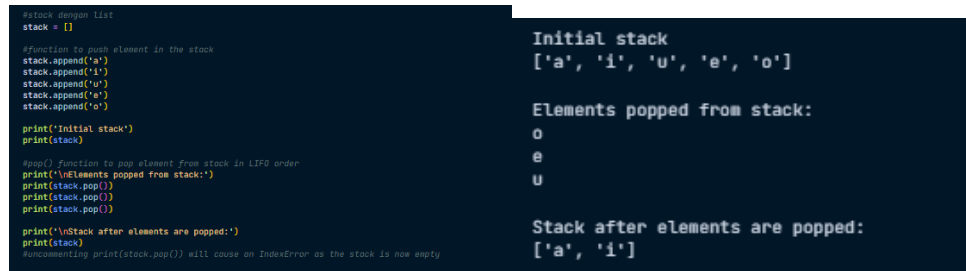
Operasi dasar yang biasa digunakan adalah:

1. Push: Menambahkan elemen baru ke dalam tumpukan.
2. Pop: Menghapus elemen teratas dari tumpukan.
3. Peek: Melihat nilai dari elemen teratas tanpa menghapusnya dari tumpukan

Tanda Tangan

III. Hasil dan Pembahasan

a. Stack dengan List



```
#Stack dengan list
stack = []

#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
['a', 'i', 'u', 'e', 'o']

Elements popped from stack:
o
e
u

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

Gambar 3.1: Stack dengan List dan Outputnya

1. `stack = []`: Membuat variabel `stack` yang merupakan tumpukan kosong.
2. `stack.append('a')`, `stack.append('i')`, dll.: Menambahkan elemen ke dalam tumpukan menggunakan metode `append()`.
3. `print('Initial stack')`, `print(stack)`: Mencetak pesan 'Initial stack' diikuti dengan isi tumpukan saat ini menggunakan `print()`.
4. `print(stack.pop())`, `print(stack.pop())`, dll.: Memanggil fungsi `pop()` untuk mengeluarkan elemen dari tumpukan dalam urutan LIFO (Last In, First Out).
5. `print("\nStack after elements are popped:')`: Mencetak pesan 'Stack after elements are popped:'
6. `print(stack)`: Mencetak isi tumpukan saat ini setelah beberapa elemen telah dihapus menggunakan `pop()`.
7. Komentar terakhir menjelaskan bahwa jika `print(stack.pop())` dihapus dari komentar, maka akan menghasilkan `IndexError` karena tumpukan sudah kosong setelah operasi `pop` dilakukan.

b. Stack dengan `collections.deque`

Tanda Tangan

```

#stack dengan collections.deque
from collections import deque

stack = deque()

stack.append('a')
stack.append('i')
stack.append('u')
stack.append('e')
stack.append('o')

print('Initial stack:')
print(stack)

print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

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

```

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: Stack dengan collections.deque dan Outputnya

1. `from collections import deque`: Mengimpor kelas `deque` dari modul `collections`.
2. `stack = deque()`: Membuat objek tumpukan menggunakan kelas `deque`. Ini menginisialisasi tumpukan kosong.
3. `stack.append('a')`, `stack.append('i')`, dll.: Menambahkan elemen ke dalam tumpukan menggunakan metode `append()`.
4. `print('Initial stack:')`, `print(stack)`: Mencetak pesan 'Initial stack:' diikuti dengan isi tumpukan saat ini menggunakan `print`.
5. `print('\nElements popped from stack:')`: Mencetak pesan 'Elements popped from stack:' untuk menandakan bahwa operasi `pop` akan dilakukan.
6. `print(stack.pop())`, `print(stack.pop())`, dll.: Memanggil metode `pop()` untuk mengeluarkan elemen dari tumpukan dalam urutan LIFO (Last In, First Out).
7. `print('\nStack after elements are popped:')`: Mencetak pesan 'Stack after elements are popped:' untuk menandakan bahwa operasi `pop` telah dilakukan.
8. `print(stack)`: Mencetak isi tumpukan saat ini setelah beberapa elemen telah dihapus menggunakan `pop()`.

Tanda Tangan

c. Stack dengan queue.LifoQueue

```
#Stack dengan queue.LifoQueue
from queue import LifoQueue

stack = LifoQueue(maxsize = 5)

print(stack.qsize())

stack.put('a')
stack.put('i')
stack.put('u')
stack.put('e')
stack.put('o')

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

print('\nElements popped from the stack')
print(stack.get())
print(stack.get())
print(stack.get())

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

```
8
Full: True
Size: 5

Elements popped from the stack
o
e
u

Empty: False
```

Gambar 3.3: Stack dengan queue.LifoQueue dan output

1. `from queue import LifoQueue`: Mengimpor kelas `LifoQueue` dari modul `queue`.
2. `stack = LifoQueue(maxsize = 5)`: Membuat objek tumpukan (`stack`) dengan memanggil konstruktor `LifoQueue()`.
3. `print(stack.qsize())`: Mencetak ukuran tumpukan saat ini menggunakan metode `qsize()`.
4. `stack.put('a')`, `stack.put('i')`, dll.: Memasukkan elemen ke dalam tumpukan menggunakan metode `put()`.
5. `print('Full: ', stack.full())`, `print('Size: ', stack.qsize())`: Mencetak status penuh (`full`) dan ukuran tumpukan saat ini.
6. `print('\nElements popped from the stack')`: Mencetak pesan 'Elements popped from the stack' untuk menandakan bahwa operasi pop akan dilakukan.
7. `print(stack.get())`, `print(stack.get())`, dll.: Memanggil metode `get()` untuk mengeluarkan elemen dari tumpukan dalam urutan LIFO (Last In, First Out).
8. `print('\nEmpty: ', stack.empty())`: Mencetak status tumpukan apakah kosong atau tidak setelah beberapa elemen telah dihapus menggunakan `get()`.

Tanda Tangan

d. Penyisipan di awal

```
#stack dengan singel linked list
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class Stack:
    #menginisialisasi stack
    def __init__(self):
        self.head = Node("head")
        self.size = 0

    #string merepresentasikan stack
    def __str__(self):
        cur = self.head.next
        out = ""
        while cur:
            out += str(cur.value) + "→"
            cur = cur.next
        return out[:-2]

    #mendapatkan panjang stack
    def getSize(self):
        return self.size

    #mengecek apakah stack kosong
    def isEmpty(self):
        return self.size == 0

    #mendapatkan elemen teratas dari stack
    def peek(self):
        if self.isEmpty():
            raise Exception("Peeking from an empty stack")
        return self.head.next.value

    #menambahkan elemen ke stack
    def push(self, value):
        node = Node(value)
        node.next = self.head.next
        self.head.next = node
        self.size += 1

    #menghapus elemen dari stack
    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

#menjalankan kode
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}")
```

Gambr 3.4: Stack dengan Single Linked List dan Output

1. class Node:: Mendefinisikan sebuah kelas bernama Node.
2. def __init__(self, value):: Mendefinisikan konstruktor untuk kelas Node. Ketika sebuah objek Node dibuat, konstruktor ini akan dijalankan untuk menginisialisasi objek tersebut dengan nilai yang diberikan.
3. self.value = value: Membuat atribut value pada objek Node yang akan menyimpan nilai dari node tersebut.
4. self.next = None: Membuat atribut next pada objek Node yang akan menunjukkan ke node selanjutnya dalam struktur data.
5. class Stack:: Mendefinisikan sebuah kelas bernama Stack.
6. def __init__(self):: Mendefinisikan konstruktor untuk kelas Stack. Konstruktor ini akan dijalankan ketika objek Stack dibuat. Ini menginisialisasi kepala (head) dari stack dengan nilai "head" dan mengatur ukuran stack menjadi 0.
7. self.head = Node("head"): Membuat sebuah objek Node dengan nilai "head" dan menjadikannya sebagai kepala (head) dari stack.
8. self.size = 0: Mengatur ukuran stack menjadi 0 karena pada awalnya stack kosong.
9. def __str__(self):: Mendefinisikan metode untuk merepresentasikan stack dalam bentuk string.

Tanda Tangan

10. `cur = self.head.next`: Menginisialisasi variabel `cur` untuk menunjuk ke node pertama setelah kepala (`head`) dalam stack.
11. `out = ""`: Inisialisasi string kosong `out` yang akan digunakan untuk menyimpan representasi string dari stack.
12. `while cur::`: Memulai loop `while` yang akan terus berjalan selama `cur` bukan `None`.
13. `out += str(cur.value) + "->"`: Menambahkan nilai dari node saat ini ke dalam string `out` dengan format "nilai->"
14. `cur = cur.next`: Memindahkan `cur` ke node berikutnya dalam stack.
15. `return out[:-2]`: Mengembalikan string `out`, tetapi menghapus dua karakter terakhir dari string tersebut (tanda panah dan spasi terakhir).
16. `def getSize(self)::` Mendefinisikan metode untuk mendapatkan ukuran stack.
17. `return self.size`: Mengembalikan nilai dari atribut `size` yang menunjukkan jumlah elemen dalam stack.
18. `def isEmpty(self)::` Mendefinisikan metode untuk memeriksa apakah stack kosong.
19. `return self.size == 0`: Mengembalikan `True` jika ukuran stack adalah 0 (stack kosong), dan `False` jika sebaliknya.
20. `def peek(self)::` Mendefinisikan metode untuk mendapatkan nilai elemen teratas dari stack tanpa menghapusnya.
21. `if self.isEmpty():`: Memeriksa apakah stack kosong. Jika iya, raise `Exception`.
22. `return self.head.next.value`: Mengembalikan nilai dari node setelah kepala (`head`) dalam stack.
23. `def push(self, value)::` Mendefinisikan metode untuk menambahkan elemen baru ke dalam stack.
24. `node = Node(value)`: Membuat objek `Node` baru dengan nilai yang diberikan.
25. `node.next = self.head.next`: Menjadikan node baru sebagai node berikutnya setelah kepala (`head`) dalam stack.

Tanda Tangan

26. `self.head.next = node`: Menjadikan node baru sebagai node pertama setelah kepala (`head`) dalam stack.
27. `self.size += 1`: Menambahkan 1 ke ukuran stack karena telah menambahkan satu elemen baru.
28. `def pop(self)::` Mendefinisikan metode untuk menghapus dan mengembalikan elemen teratas dari stack.
29. `if self.isEmpty():` Memeriksa apakah stack kosong. Jika iya, raise Exception.
30. `remove = self.head.next`: Menyimpan node berikutnya setelah kepala (`head`) dalam variabel `remove`.
31. `self.head.next = self.head.next.next`: Menghapus node berikutnya setelah kepala (`head`) dengan menggeser pointer `next`.
32. `self.size -= 1`: Mengurangi ukuran stack karena telah menghapus satu elemen.
33. `return remove.value`: Mengembalikan nilai dari node yang dihapus.
34. `if __name__ == "__main__":` Memeriksa apakah script dijalankan sebagai program utama.
35. `stack = Stack()`: Membuat objek stack baru.
36. `for i in range(1, 11):` Looping dari 1 hingga 10.
37. `stack.push(i)`: Menambahkan nilai dari `i` ke dalam stack.
38. `print(f"Stack: {stack}")`: Mencetak stack setelah nilai-nilai ditambahkan.
39. `for _ in range(1, 6):` Looping dari 1 hingga 5.
40. `remove = stack.pop()`: Menghapus elemen dari stack dan menyimpannya dalam variabel `remove`.
41. `print(f"Pop: {remove}")`: Mencetak nilai yang dihapus dari stack.
42. `print(f"Stack: {stack}")`: Mencetak stack setelah operasi `pop` dilakukan.

Tanda Tangan

IV. Kesimpulan

Praktikum stack membantu kita memahami konsep dasar tumpukan dalam pemrograman dengan cara yang menyenangkan. Kita belajar tentang cara menambahkan dan menghapus elemen dari tumpukan, serta bagaimana tumpukan bekerja menggunakan aturan LIFO (Last In, First Out). Dengan berbagai contoh implementasi, termasuk menggunakan kelas dan objek, kita dapat melihat bagaimana tumpukan digunakan dalam pemrograman sehari-hari. Praktikum ini memberikan pemahaman yang kuat tentang struktur data ini dan bagaimana kita bisa menggunakan mereka dalam memecahkan masalah komputasi.

Tanda Tangan
