TUGAS MODUL 3

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

class Stack:
    def __init__(self, capacity):
        self.top = None
        self.size = 0
        self.capacity = capacity

    def push(self, data):
        """Menambah elemen ke stack."""
        if self.is_full():
            print("Stack sudah penuh, tidak dapat menambah elemen.")
            return
        new_node = Node(data)
        new_node.next = self.top
        self.top = new_node
        self.size += 1
        print(f"Stack: {self.display_stack()}")

    def pop(self):
        """Menghapus elemen dari stack."""
        if self.is_empty():
            print("Stack kosong, tidak ada elemen yang dapat dihapus.")
            return None
        removed_data = self.top.data
        self.top = self.top.next
        self.size -= 1
        print(f"Elemen {removed_data} dihapus dari stack.")
        print(f"Stack: {self.display_stack()}")
        return removed_data

    def get_size(self):
        """Mencari tahu ukuran stack."""
        return self.size

    def peek(self):
        """Mencari tahu elemen puncak dari stack."""
        if self.is_empty():
            print("Stack kosong, tidak ada elemen puncak.")
            return None
        return self.top.data

    def is_full(self):
        """Mencari tahu apakah stack dalam kondisi penuh atau tidak."""
        return self.size == self.capacity

    def is_empty(self):
        """Mencari tahu apakah stack kosong."""
        return self.size == 0

    def display_stack(self):
        """Menampilkan elemen dalam stack."""
        elements = []
        current = self.top
        while current:
            elements.append(current.data)
            current = current.next
        return elements

def main():
    print("=====PROGRAM SEDERHANA UNTUK IMPLEMENTASI STACK DENGAN LINKED-LIST=====")
    capacity = int(input("Tentukan berapa kapasitas stack: "))
    stack = Stack(capacity)

    while True:
        print("\nPilih menu berikut ini:")
        print("1. Menambah isi stack")
        print("2. Menghapus isi stack")
        print("3. Cek Ukuran Stack saat ini")
        print("4. Cek Puncak Stack")
        print("5. Cek Stack Full")
        print("6. Keluar")

        choice = int(input("Masukkan pilihan anda: "))

        if choice == 1:
            data = int(input("Masukkan isi stack: "))
            stack.push(data)
            if not stack.is_full():
                continue_choice = input("Menambah isi Stack Pilih [Ya/Tidak]: ")
                if continue_choice.lower() != 'ya':
                    continue
        elif choice == 2:
            stack.pop()
        elif choice == 3:
            print(f"Ukuran stack saat ini: {stack.get_size()}")
        elif choice == 4:
            top_element = stack.peek()
            if top_element is not None:
                print(f"Elemen puncak: {top_element}")
        elif choice == 5:
            print(f"Apakah stack penuh? {'Ya' if stack.is_full() else 'Tidak'}")
        elif choice == 6:
            print("Keluar dari program.")
            break
        else:
            print("Pilihan tidak valid, silakan coba lagi.")

if __name__ == "__main__":
    main()
```

- class Node:
- def __init__(self, data):
- self.data = data
- self.next = None
- ```
  Kelas `Node` digunakan untuk merepresentasikan elemen dalam stack. Setiap node memiliki dua atribut: `data`, yang menyimpan nilai dari node, dan `next`, yang menunjuk ke node berikutnya dalam stack (diinisialisasi dengan `None`).

- class Stack:
- def __init__(self, capacity):
- self.top = None
- self.size = 0
- self.capacity = capacity
- ```
  Kelas `Stack` digunakan untuk merepresentasikan stack itu sendiri. Dalam metode `__init__`, kita menginisialisasi `top` (elemen teratas stack) dengan `None`, `size` (ukuran stack) dengan `0`, dan `capacity` (kapasitas maksimum stack) dengan nilai yang diberikan saat pembuatan objek stack.

- def push(self, data):
- """Menambah elemen ke stack."""
- if self.is_full():
- print("Stack sudah penuh, tidak dapat menambah elemen.")
- return
- ```
  Metode `push` digunakan untuk menambahkan elemen baru ke stack. Pertama, kita memeriksa apakah stack sudah penuh dengan memanggil metode `is_full()`. Jika penuh, kita mencetak pesan dan keluar dari metode.

- new_node = Node(data)
- new_node.next = self.top
- self.top = new_node
- self.size += 1
- print(f"Stack: {self.display_stack()}")
- ```
  Jika stack tidak penuh, kita membuat node baru dengan data yang diberikan. Kemudian, kita mengatur `next` dari node baru untuk menunjuk ke node yang saat ini menjadi `top`. Selanjutnya, kita memperbarui `top` untuk menjadi node baru dan meningkatkan ukuran stack (`size`) sebesar 1. Terakhir, kita mencetak isi stack dengan memanggil metode `display_stack()`.

- def pop(self):
- """Menghapus elemen dari stack."""
- if self.is_empty():
- print("Stack kosong, tidak ada elemen yang dapat dihapus.")

- return None
- ```
  Metode `pop` digunakan untuk menghapus elemen teratas dari stack. Pertama, kita memeriksa apakah stack kosong dengan memanggil metode `is_empty()`. Jika kosong, kita mencetak pesan dan keluar dari metode.

- removed_data = self.top.data
- self.top = self.top.next
- self.size -= 1
- print(f"Elemen {removed_data} dihapus dari stack.")
- print(f"Stack: {self.display_stack()}")
- return removed_data
- ```
  Jika stack tidak kosong, kita menyimpan nilai dari elemen teratas (`self.top.data`) ke dalam variabel `removed_data`. Kemudian, kita memperbarui `top` untuk menunjuk ke node berikutnya (`self.top.next`) dan mengurangi ukuran stack (`size`) sebesar 1. Kita mencetak informasi tentang elemen yang dihapus dan isi stack yang baru, lalu mengembalikan nilai yang dihapus.

- def get_size(self):
- """Mencari tahu ukuran stack."""
- return self.size
- ```
- - Metode `get_size` mengembalikan ukuran stack saat ini.
-
- ```python
- def peek(self):
- """Mencari tahu elemen puncak dari stack."""
- if self.is_empty():
- print("Stack kosong, tidak ada elemen puncak.")
- return None
- ```
  Metode `peek` digunakan untuk melihat nilai elemen teratas dari stack tanpa menghapusnya. Jika stack kosong, kita mencetak pesan dan mengembalikan `None`.

- return self.top.data

  Jika stack tidak kosong, kita mengembalikan nilai dari elemen teratas (`self.top.data`).

- def is_full(self):
- """Mencari tahu apakah stack dalam kondisi penuh atau tidak."""
- return self.size == self.capacity
- ```
  Metode `is_full` mengembalikan `True` jika ukuran stack sama dengan kapasitas maksimum, dan `False` jika tidak.

- def is_empty(self):

- """Mencari tahu apakah stack kosong."""
- return self.size == 0
- ```
- - Metode `is_empty` mengembalikan `True` jika ukuran stack adalah 0 (kosong), dan `False` jika tidak.

- def display_stack(self):
- """Menampilkan elemen dalam stack."""
- elements = []
- current = self.top
- while current:
- elements.append(current.data)
- current = current.next
- return elements
- ```
  Metode `display_stack` digunakan untuk menampilkan semua elemen dalam stack. Kita membuat list kosong `elements`, kemudian menggunakan loop untuk menelusuri setiap node dalam stack, menambahkan nilai dari setiap node ke dalam list `elements`. Setelah selesai, kita mengembalikan list tersebut.

- def main():
- print("=====PROGRAM SEDERHANA UNTUK IMPLEMENTASI STACK DENGAN LINKED-LIST=====")
- capacity = int(input("Tentukan berapa kapasitas stack: "))
- stack = Stack(capacity)
- ```
  Fungsi `main` adalah titik masuk program. Di sini, kita mencetak judul program dan meminta pengguna untuk menentukan kapasitas stack. Kemudian, kita membuat objek `Stack` dengan kapasitas yang diberikan.

- while True:
- print("\nPilih menu berikut ini:")
- print("1. Menambah isi stack")
- print("2. Menghapus isi stack")
- print("3. Cek Ukuran Stack saat ini")
- print("4. Cek Puncak Stack")
- print("5. Cek Stack Full")
- print("6. Keluar")
- ```
  Kita memulai loop `while` yang akan terus berjalan hingga dihentikan. Di dalam loop, kita mencetak menu pilihan untuk pengguna.

- choice = int(input("Masukkan pilihan anda: "))
- ```
  Kita meminta pengguna untuk memasukkan pilihan menu yang diinginkan.

- ```
  if choice == 1:
  data = int(input("Masukkan isi stack: "))
  stack.push(data)
  if not stack.is_full():
  continue_choice = input("Menambah isi Stack Pilih [Ya/Tidak]: ")
  if continue_choice.lower() != 'ya':
  continue
  ```
  Jika pengguna memilih untuk menambah isi stack, kita meminta mereka untuk memasukkan data yang ingin ditambahkan. Kita kemudian memanggil metode `push` untuk menambahkan data tersebut ke stack. Jika stack tidak penuh, kita menanyakan kepada pengguna apakah mereka ingin menambah lebih banyak elemen. Jika tidak, kita melanjutkan ke iterasi berikutnya.

- ```
  elif choice == 2:
  stack.pop()
  ```
- - Jika pengguna memilih untuk menghapus isi stack, kita memanggil metode `pop` untuk menghapus elemen teratas dari stack.

- ```
  elif choice == 3:
  print(f"Ukuran stack saat ini: {stack.get_size()}")
  ```
  Jika pengguna memilih untuk memeriksa ukuran stack, kita memanggil metode `get_size` dan mencetak ukuran saat ini.

- ```
  elif choice == 4:
  top_element = stack.peek()
  if top_element is not None:
  print(f"Elemen puncak: {top_element}")
  ```
  Jika pengguna memilih untuk memeriksa elemen puncak stack, kita memanggil metode `peek` dan mencetak elemen puncak jika tidak kosong.

- elif choice == 5:
- print(f"Apakah stack penuh? {'Ya' if stack.is_full() else 'Tidak'}")
  ```
  Jika pengguna memilih untuk memeriksa apakah stack penuh, kita memanggil metode `is_full` dan mencetak hasilnya.

- elif choice == 6:
- print("Keluar dari program.")
- break
- ```
  - Jika pengguna memilih untuk keluar dari program, kita mencetak pesan dan menghentikan loop dengan `break`.
  ```

- else:
- print("Pilihan tidak valid, silakan coba lagi.")
  ```

  - Jika pengguna memasukkan pilihan yang tidak valid, kita mencetak pesan untuk meminta mereka mencoba lagi.

- if __name__ == "__main__":
- main()
  ```

  - Baris ini memastikan bahwa fungsi `main` hanya akan dijalankan jika skrip ini dijalankan sebagai program utama, bukan jika diimpor sebagai modul.