**TUGAS-2 STRUKTUR DATA**

Instruksi:

a. Kerjakan satu persatu soal berikut ini, diatas kertas double folio bergaris.

b. Semua jawaban program atau ADT ditulis tangan

c. Bukti test/percobaan di-copy dari edior Python berserta hasilnya.

d. WAKTU: harus dikumpulkan paling lambat pada pertemuan ke-15.

SOAL:

1. Pelajari kembali implementasi dari struktur data untaian ganda, “my\_DList.py”. Tambahkan sebuah fungsi ADT, beri nama “print\_Back()”, fungsi-nya adalah menampilkan isi untaian ganda mulai dari belakang (self.tail) hingga ke depan (self.head).

Source code:

def print\_Back(self):

current = self.tail

while current is not None:

print(current.data, end=" <=> ")

current = current.back

print("None")

def main():

dll = DoublyLinkedList()

dll.add\_first(1)

dll.add\_last(2)

dll.add\_last(3)

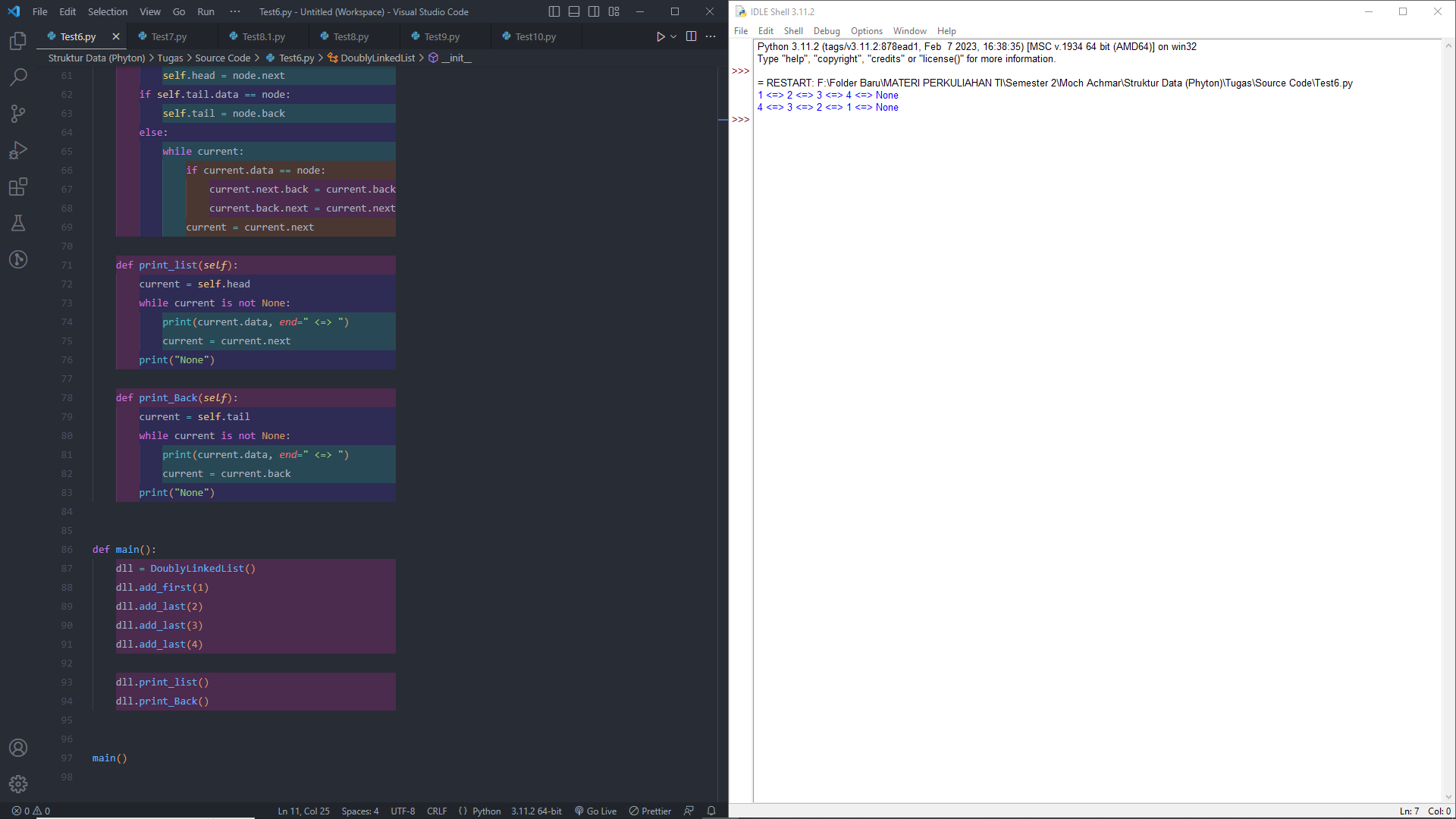
dll.add\_last(4)

dll.print\_list()

dll.print\_Back()

main()

Hasil No. 1:



2. a. Jelaskan bagaimana sebuah untaian tunggal dapat digunakan sebagai antrian (queue).

Jawab:

Satu utas dapat digunakan sebagai antrian dengan menggunakan teknik yang disebut **pola produsen-konsumen** . Dalam pola ini, satu utas (produsen) menambahkan item ke antrean, dan utas lainnya (konsumen) menghapus item dari antrean. Thread produsen dan konsumen dapat berjalan secara bersamaan, tetapi mereka harus mengoordinasikan akses mereka ke antrean untuk menghindari kondisi balapan.

Untuk menerapkan pola produsen-konsumen menggunakan satu utas, kita dapat menggunakan struktur data sederhana yang disebut **buffer melingkar** . Buffer melingkar adalah array ukuran tetap yang digunakan untuk menyimpan item data. Utas produsen menambahkan item ke buffer melingkar, dan utas konsumen menghapus item dari buffer melingkar. Thread produsen dan konsumen harus mengoordinasikan akses mereka ke buffer melingkar dengan menggunakan kunci atau semaphore.

b. Melalui implementasi “my\_Untaian.py”, buat sebuah program test, yang membentuk antrian dari minimal 10 data. Tampilkan isi antrian, kemudian lakukan pelayanan sebanyak 3 kali, dan tampilkan isi antrian setelah pelayanan.

Source code:

queue = Untaian()

for i in range(1, 11):

queue.delete\_at\_tail(i)

print("Isi antrian sebelum pelayanan:")

queue.traverse()

for \_ in range(3):

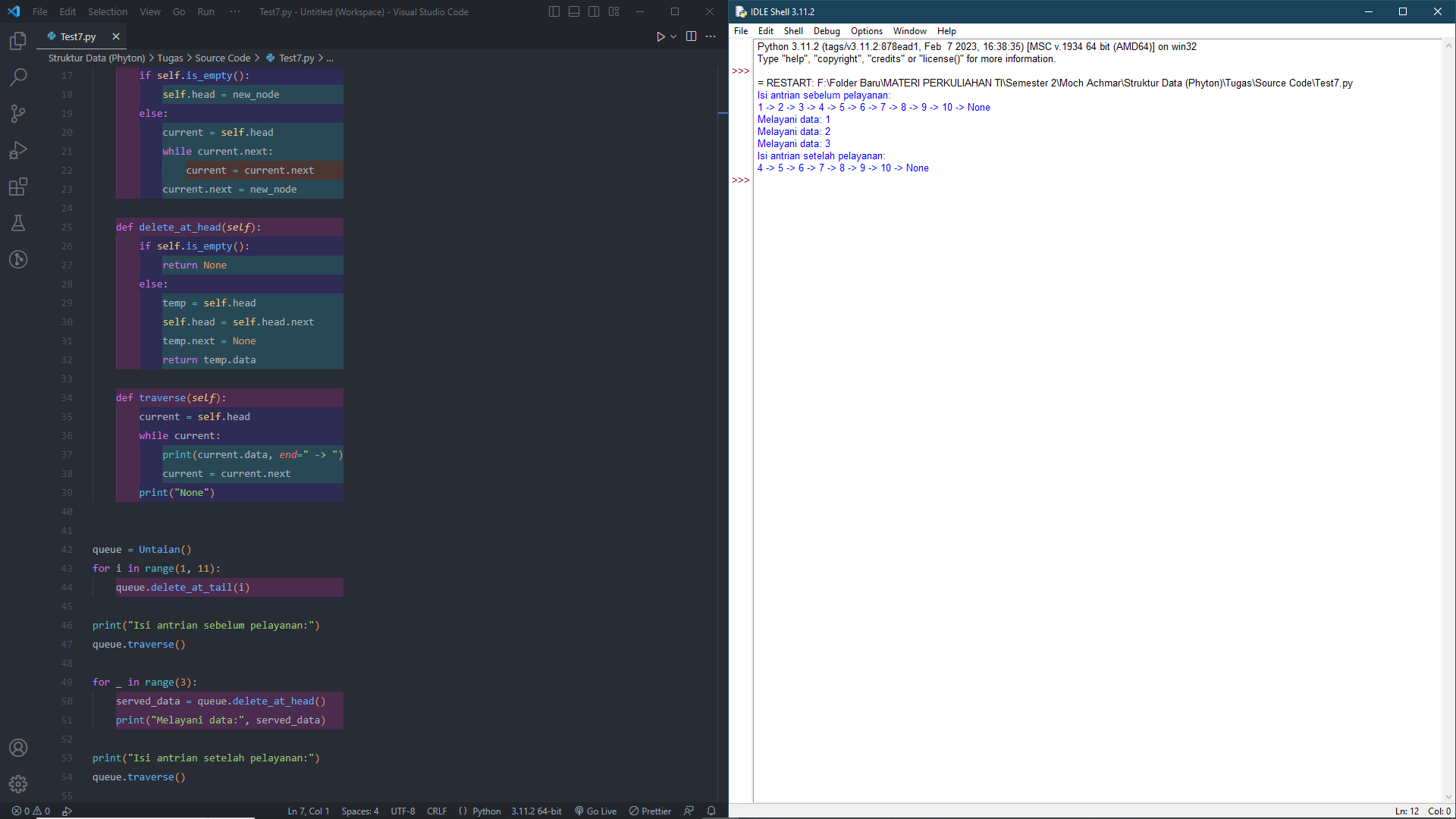
served\_data = queue.delete\_at\_head()

print("Melayani data:", served\_data)

print("Isi antrian setelah pelayanan:")

queue.traverse()

Hasil No. 2 Bagian B:



3. a. Jelaskan bagaimana cara menyisipkan sebuah Node baru diantara dua Node yang sudah ada dalam sebuah antrian melingkar.

Jawab:

Untuk menyisipkan simpul baru di antara dua simpul yang ada dalam antrean melingkar, Anda dapat mengikuti langkah-langkah berikut:

1. Buat node baru dan inisialisasi datanya dan pointer berikutnya.
2. Temukan dua simpul di mana Anda ingin menyisipkan simpul baru.
3. Tetapkan pointer berikutnya dari node baru ke pointer berikutnya dari node kedua.
4. Tetapkan pointer berikutnya dari node pertama ke node baru.
5. Tetapkan pointer berikutnya dari node baru ke node kedua.

b. Buat sebuah ADT tambahan pada implementasi untaian melingkar “my\_Circular\_LL.py”, yang fungsi-nya menempatkan Node secara berurutan (sorted) ke dalam untaian melingkar.

Source code:

def add\_node(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

self.tail = new\_node

else:

self.tail.next = new\_node

self.tail = new\_node

self.tail.next = self.head

def print\_list(self):

current\_node = self.head

if current\_node is None:

print("Circular Linked List is empty.")

else:

while True:

print(current\_node.data)

current\_node = current\_node.next

if current\_node == self.head:

break

def delete\_node(self, list, data):

if list.head is None:

return

current\_node = list.head

previous\_node = None

while current\_node is not None:

if current\_node.data == data:

if previous\_node is None:

list.head = current\_node.next

else:

previous\_node.next = current\_node.next

if current\_node == list.tail:

list.tail = previous\_node

break

previous\_node = current\_node

current\_node = current\_node.next

def insert\_node\_sorted(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

self.tail = new\_node

new\_node.next = self.head

elif data <= self.head.data:

new\_node.next = self.head

self.head = new\_node

self.tail.next = self.head

elif data >= self.tail.data:

self.tail.next = new\_node

self.tail = new\_node

self.tail.next = self.head

else:

current\_node = self.head

while current\_node.next.data < data:

current\_node = current\_node.next

new\_node.next = current\_node.next

current\_node.next = new\_node

if \_\_name\_\_ == "\_\_main\_\_":

my\_list = CircularLinkedList()

print("add 3 nodes")

my\_list.add\_node(1)

my\_list.add\_node(2)

my\_list.add\_node(3)

print("Insert a node in the middle")

my\_list.insert\_node\_in\_middle(my\_list, 2, 4)

my\_list.print\_list()

print("Insert a node in sorted order")

my\_list.insert\_node\_sorted(0)

my\_list.insert\_node\_sorted(5)

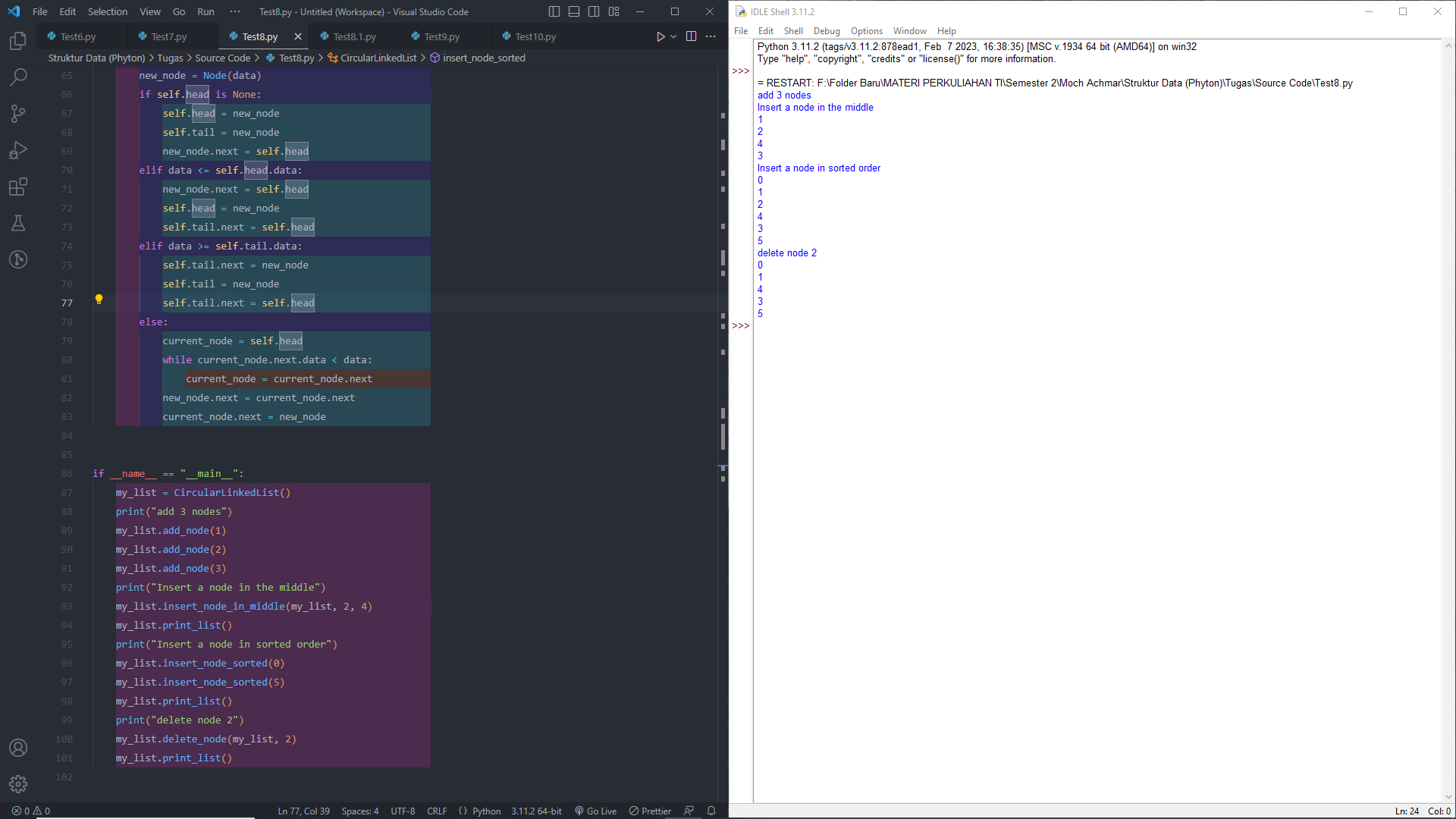
my\_list.print\_list()

print("delete node 2")

my\_list.delete\_node(my\_list, 2)

my\_list.print\_list()

Hasil No. 3 Bagian B:



c. Buat sebuah program uji / test dengan memasukkan Node secara tidak ber-urutan, kemudian ketika di-tampilkan (print\_list()) isi Node sudah ber-urutan. (contoh masukkan digit dari nomer mahaiswa anda, kemudian print hasilnya.

Source code:

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def insert(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

current = self.head

while current.next:

current = current.next

current.next = new\_node

def print\_list(self):

current = self.head

while current:

print(current.data, end=" ")

current = current.next

print()

def sort\_list(self):

if self.head is None:

return

sorted\_list = None

current = self.head

while current:

next\_node = current.next

sorted\_list = self.sorted\_insert(sorted\_list, current)

current = next\_node

self.head = sorted\_list

def sorted\_insert(self, sorted\_list, new\_node):

if sorted\_list is None or sorted\_list.data >= new\_node.data:

new\_node.next = sorted\_list

sorted\_list = new\_node

else:

current = sorted\_list

while current.next and current.next.data < new\_node.data:

current = current.next

new\_node.next = current.next

current.next = new\_node

return sorted\_list

linked\_list = LinkedList()

input\_string = input("Masukkan angka: ")

numbers = [int(digit) for digit in input\_string if digit.isdigit()]

for num in numbers:

linked\_list.insert(num)

print("Linked List sebelum diurutkan:")

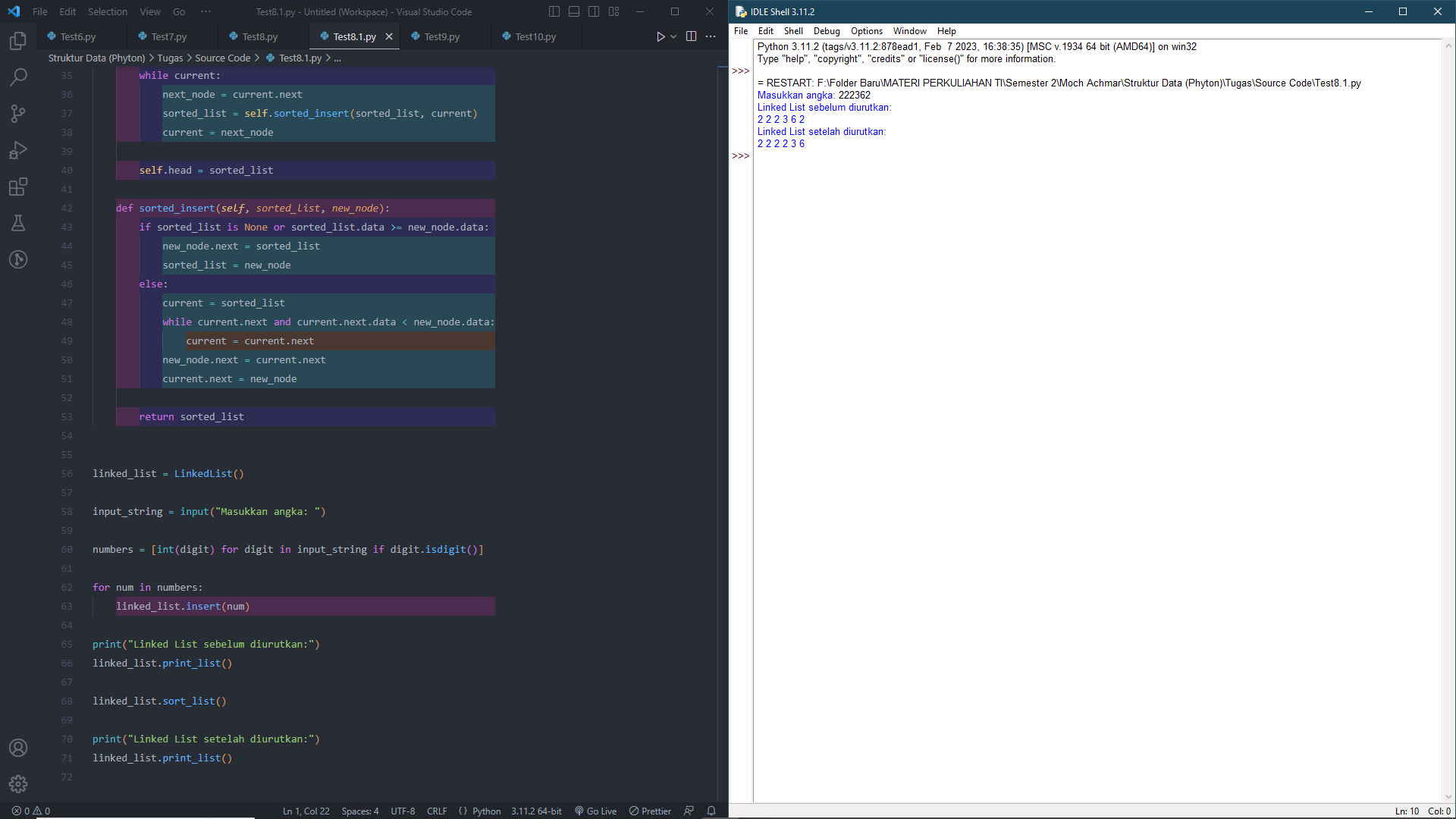
linked\_list.print\_list()

linked\_list.sort\_list()

print("Linked List setelah diurutkan:")

linked\_list.print\_list()

Hasil No. 3 Bagian C:



4. Pelajari kembali implementasi pohon biner “BTNode.py”, kemuadian masukkan setiap huruf yang ada dalam kata berikut:

a. KADABRA, gambarkan hasil pohon binermya. dan telusuri pohon ini, (pre, in, post) order

b. MINAKJINGGO, gambarkan hasil pohon binernya, dan telusuri pohon ini (pre, in, post) order

Source code:

import sys

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

def children\_count(self):

if self.left is None and self.right is None:

return None

cnt = 0

if self.left:

cnt += 1

if self.right:

cnt += 1

return cnt

def insert(self, data):

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

else:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

def lookup(self, data, parent=None):

if data == self.data:

return self, parent

elif data < self.data:

if self.left is None:

return None, None

return self.left.lookup(data, self)

else:

if self.right is None:

return None, None

return self.right.lookup(data, self)

def delete(self, data):

node, parent = self.lookup(data)

if node is not None:

children\_count = node.children\_count()

if children\_count == 0:

if parent.left is node:

parent.left = None

else:

parent.right = None

del node

elif children\_count == 1:

if node.left:

n = node.left

else:

n = node.right

if parent:

if parent.left is node:

parent.left = n

else:

parent.right = n

del node

else:

self.data = n.data

self.left = n.left

self.right = n.right

else:

parent = node

successor = node.right

while successor.left:

parent = successor

successor = successor.left

node.data = successor.data

if parent.left == successor:

parent.left = successor.right

else:

parent.right = successor.right

del node

def preorder(self):

def preorder\_traversal(node):

if node is not None:

sys.stdout.write(str(node.data) + " -> ")

preorder\_traversal(node.left)

preorder\_traversal(node.right)

preorder\_traversal(self)

print("None")

def inorder(self):

def inorder\_traversal(node):

if node is not None:

inorder\_traversal(node.left)

sys.stdout.write(str(node.data) + " -> ")

inorder\_traversal(node.right)

inorder\_traversal(self)

print("None")

def postorder(self):

def postorder\_traversal(node):

if node is not None:

postorder\_traversal(node.left)

postorder\_traversal(node.right)

sys.stdout.write(str(node.data) + " -> ")

postorder\_traversal(self)

print("None")

def main():

# KADABRA

kadabra = Node('K')

kadabra.insert('A')

kadabra.insert('D')

kadabra.insert('A')

kadabra.insert('B')

kadabra.insert('R')

kadabra.insert('A')

print('Inorder Traversal for KADABRA:')

kadabra.inorder()

print()

print('Preorder Traversal for KADABRA:')

kadabra.preorder()

print()

print('Postorder Traversal for KADABRA:')

kadabra.postorder()

print()

# MINAKJINGGO

minakjinggo = Node('M')

minakjinggo.insert('I')

minakjinggo.insert('N')

minakjinggo.insert('A')

minakjinggo.insert('K')

minakjinggo.insert('J')

minakjinggo.insert('I')

minakjinggo.insert('N')

minakjinggo.insert('G')

minakjinggo.insert('G')

minakjinggo.insert('O')

print('Inorder Traversal for MINAKJINGGO:')

minakjinggo.inorder()

print()

print('Preorder Traversal for MINAKJINGGO:')

minakjinggo.preorder()

print()

print('Postorder Traversal for MINAKJINGGO:')

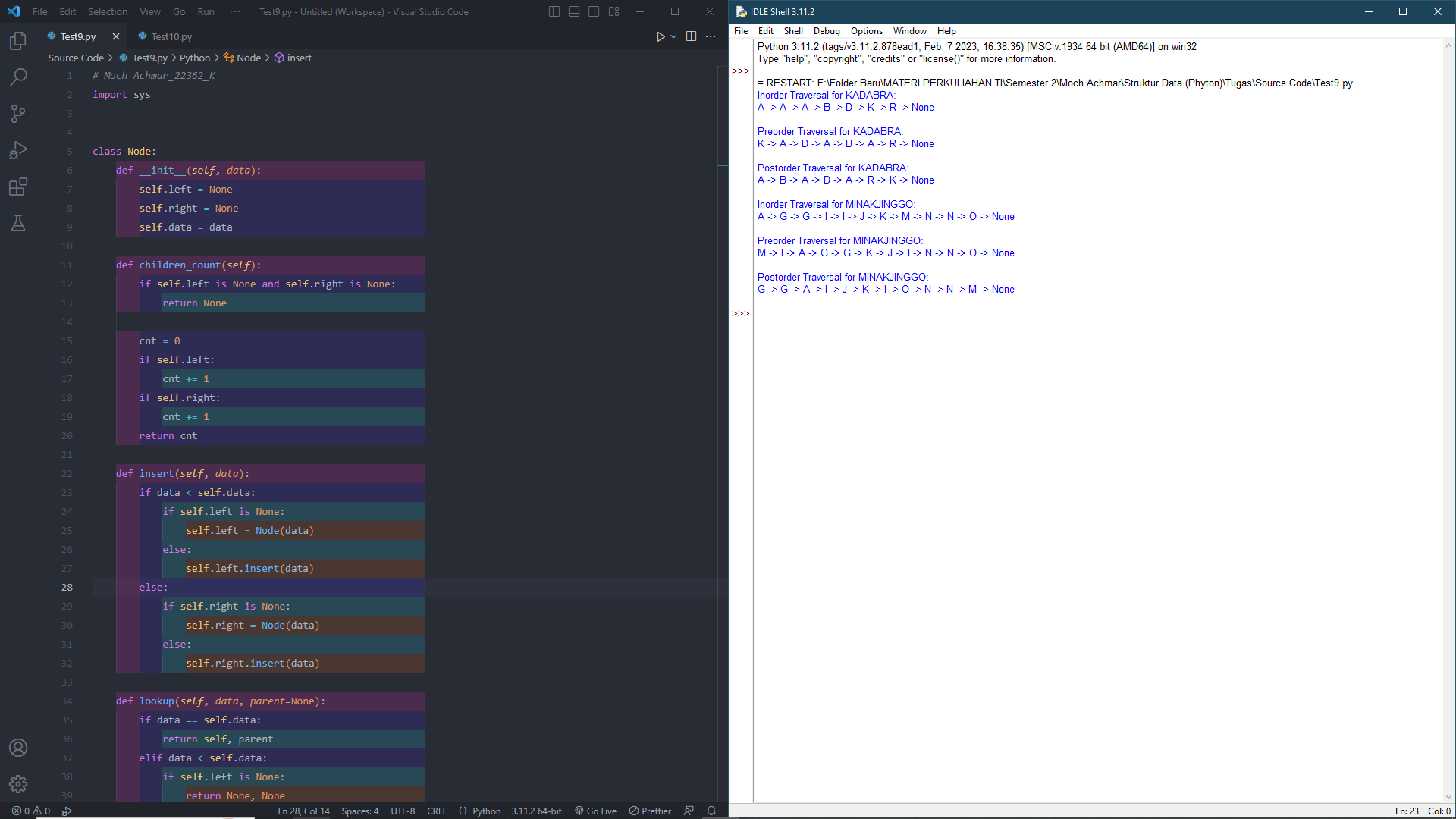
minakjinggo.postorder()

print()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Hasil No. 4:



Hasil Pohon Biner A:

K

/ \

A R

/ \

A D

/ \

A B

Hasil Pohon Biner B:

M

/ \

I N

/ / \

A J O

/ \

G N

5. Andaikan sebuah struktur data hashing akan dibentuk melalui “linear probing”, memakai fungsi : posisi = data Mod 11

Gambarkan hasil penempatan data apabila urutan data yang dimasukkan adalah:

13 15 17 19 24 35 47.

Source code:

class LinearProbingHash:

def \_\_init\_\_(self, size):

self.size = size

self.hash\_table = [None] \* size

def hash\_function(self, data):

return data % self.size

def insert(self, data):

hash\_value = self.hash\_function(data)

if self.hash\_table[hash\_value] is None:

self.hash\_table[hash\_value] = data

else:

next\_slot = (hash\_value + 1) % self.size

while self.hash\_table[next\_slot] is not None and next\_slot != hash\_value:

next\_slot = (next\_slot + 1) % self.size

if self.hash\_table[next\_slot] is None:

self.hash\_table[next\_slot] = data

else:

print("Tabel hash penuh. Tidak dapat memasukkan data.")

def display(self):

for index, data in enumerate(self.hash\_table):

print(f"Slot {index}: {data}")

def linear\_probing(data):

array = [None] \* 11

for num in data:

position = num % 11

while array[position] is not None:

position = (position + 1) % 11

array[position] = num

return array

hash\_table = LinearProbingHash(11)

data\_list = [13, 15, 17, 19, 24, 35, 47]

for data in data\_list:

hash\_table.insert(data)

hash\_table.display()

result = linear\_probing(data\_list)

print("\nArray:")

print(result)

Hasil No. 5:

