**LAPORAN PRAKTIKUM**

**STRUKTUR DATA**

**MODUL KE- 2**

**LINKED LIST DALAM PYTHON**



**Disusun Oleh:**

**Nama :** Restu Wibisono

**NPM :** 2340506061

**Kelas :** 03 (Tiga)

**Program Studi S1 Teknologi Informasi**

**Fakultas Teknik, Universitas Tidar**

**Genap 2023/2024**

1. **Tujuan Praktikum**

Praktikum ini bertujuan untuk memahami konsep dasar dan lanjutan dari struktur data linked list serta mengimplementasikannya dalam pemrograman Python. Dengan diharapkan dapat memahami konsep dasar linked list, seperti pembuatan node, penambahan serta penghapusan node, juga serta pengelolaan linked list secara umum. Selain itu, juga bertujuan agar keterampilan pemrograman terlatih, pemahaman algoritma, serta kemampuan analisis dan pemecahan masalah.

Tujuan lain dari praktikum ini adalah memperluas pemahaman tentang struktur data linked list, khususnya doubly linked list, serta akan melatih kemampuan dalam mengimplementasikan konsep ini dalam pemrograman Python. Selain itu, diharapkan dapat memahami perbedaan antara singly linked list serta doubly linked list, dan keunggulan mauoun kelemahan dari masing-masing jenis linked list tersebut.

Praktikum ini juga mencakup pembelajaran konsep-konsep baru yang terkait dengan doubly linked list, seperti penambahan dan penghapusan node dari kedua ujung linked list, serta navigasi maju dan mundur di dalam linked list. Dengan demikian, praktikum ini memberikan landasan yang kokoh bagi untuk mengembangkan keterampilan pemrograman dan pemrosesan data menggunakan struktur data linked list dalam lingkungan pemrograman Python.

1. **Dasar Teori**

Linked list adalah salah satu struktur data linear yang terdiri dari sejumlah simpul (node) yang terhubung satu sama lain secara sekuensial. Setiap node mempunyai dua bagian utama, yaitu data (dataval) yang menyimpan informasi, dan pointer (nextval) yang berfungsi untuk menunjuk ke node berikutnya dalam linked list. Linked list bisa dibagi dalam beberapa jenis, seperti singly linked list, doubly linked list, dan circular linked list. Singly linked list adalah jenis linked list di mana setiap node hanya memiliki satu pointer yang menunjuk ke node berikutnya. Konsep dasar yang terlibat dalam implementasi linked list meliputi pembuatan node, penambahan dan penghapusan node di berbagai posisi, serta pencarian dan pengurutan data.

Doubly linked list, di sisi lain, adalah jenis struktur data linked list di mana setiap node memiliki dua pointer, yaitu nextval yang menunjuk ke node berikutnya, dan prevval yang menunjuk ke node sebelumnya. Dengan pointer prevval, doubly linked list memungkinkan navigasi maju dan mundur dalam linked list dengan mudah, jadi memberikan keunggulan dibandingkan dengan singly linked list. Konsep dasar ini terlibat dalam implementasi doubly linked list mirip dengan singly linked list, tetapi dengan penambahan pengaturan pointer prevval yang sesuai.

Praktikum ini bertujuan untuk memberikan pemahaman mendalam tentang konsep dasar dan lanjutan dari struktur data linked list, dan akan melatih kemampuan dalam mengimplementasikan konsep ini dalam pemrograman Python. Melalui praktikum ini, diharapkan bisa memahami perbedaan dari singly linked list dan doubly linked list, dan keunggulan serta kelemahan dari masing-masing jenis linked list tersebut.

1. **Hasil dan Pembahasan**
   1. Membuat Node

class Node:

  def \_\_init\_\_(self, dataval=None):

    self.dataval = dataval

    self.nextval = None

n1 = Node("Januari")

n2 = Node("Febuari")

n3 = Node("Maret")

print (n1.dataval)

print (n2.dataval)

print (n3.dataval)

Januari

Febuari

Maret

(Gambar 3.1)

* Mendefinisikan class yang bernama ‘Node’.
* Fungsi ‘\_\_init\_\_’ untuk constructor yang berguna untuk instalasi objek ‘Node’ dengan data ‘dataval’ dan ditetapkan nilai awal ‘nextval’ ke ‘Node’.
* Variabel ‘dataval’ akan menyimpan nilai data dari node, sedangkan ‘nextval’ untuk menunjukkan node berikutnya dalam Linked list.
* Lalu membuat tiga instance baru dari class ‘Node’ dengan data “Januari”, “Febuari”, dan “Maret” yang disimpan dalam variabel ‘n1’, ‘n2’, dan ‘n3’.
* Program akan menampilkan data dari setiap node dengan ‘print’ statement.
  1. Memuat Class Linked List

class LinkedList:

  def \_\_init\_\_(self):

    self.headval = None

Li = LinkedList()

Li.headval = n1

(Gambar 3.2.1)

# Link frist Node to second node

Li.headval.nextval = n2

# Link second Node to third node

n2.nextval = n3

(Gambar 3.2.2)

* Pertama mendefinisikan sebuah class bernama ‘LinkedList’.
* Constructor berfungsi untuk instalasi object ‘LinkedList’ dengan ‘headval’ yang di hubungkan ke ‘None’
* Variabel ‘headval’ akan menunjukkan node pertama pada Linked list.
* Lalu membuat instance baru yang diambil dari class ‘LinkedList’ dan menyimpannya di dalam variabel ‘Li’.
* Mengatur node pertama ‘headval’ untuk dihubungkan dengan ‘n1’, yang merupakan node pertama dalam Linked list.
* Selanjutnya mengatur ‘nextval’ dari ‘n2’ untuk menunjuk ‘n3’, yang akan membuat node kedua dengan node ketiga terhubung.
  1. Menulusi Linked List

class Node:

  def \_\_init\_\_(self, dataval=None):

    self.dataval = dataval

    self.nextval = None

class LinkedList:

  def \_\_init\_\_(self):

    self.headval = None

  def listprint(self):

    printval = self.headval

    while printval is not None:

      print (printval.dataval)

      printval = printval.nextval

Li = LinkedList()

Li.headval = n1

(Gambar 3.3.1)

# Link frist Node to second node

Li.headval.nextval = n2

# Link second Node to third node

n2.nextval = n3

Li.listprint()

Januari

Febuari

Maret

(Gambar 3.3.2)

* Mendefinisikan sebuah class ‘Node’ dengan dua atribut yaitu, atribut ‘dataval’ berfungsi untuk menyimpan nilai data dari node, dan ‘nextval’ untuk menunjuk node berikutnya dalam Linked list.
* Selanjutnya mendefinisikan sebuah class ‘LinkedList’ dengan atribut ‘headval’ berfungsi untuk menunjuk node pertama di dalam Linked list.
* Kemudian fungsi ‘listprint’ adalah metode untuk menampilkan isi Linked list.
* Program akan menganalisis variabel ‘printval’ dengan ‘headval’, kemudian akan melakukan perulangan ‘while’ untuk mencetak nilai ‘dataval’ dari setiap node sampai ‘printval’ menjadi ‘None’.
* Membuat instance baru dari class ‘LinkedList’ dan akan menyimpannya di dalam variabel ‘Li’.
* Lalu menghubungkan node pertama dengan node kedua dengan mengatur ‘nextval’ dari ‘n1’ untuk menunjuk ke ‘n2’.
* Menghubungkan node kedua dengan node ketiga dengan mengatur ‘nextval’ dari ‘n2’ untuk menunjuk ke ‘n3’.
* Terakhir memanggil fungsi ‘listprint’ dari objek ‘Li’ untuk menampilkan isi dari Linked list.
  1. Penyisipan di Awal

­class Node:

  def \_\_init\_\_(self, dataval=None):

    self.dataval = dataval

    self.nextval = None

# create node

n1 = Node("Januari")

n2 = Node("Febuari")

n3 = Node("Maret")

class LinkedList:

  def \_\_init\_\_(self):

    self.headval = None

# Print the Linked list

  def listprint(self):

    printval = self.headval

    while printval is not None:

      print (printval.dataval)

      printval = printval.nextval

# create atribut add begining to Update

# the new nodes next val to existing node

  def AddBegining(self,newdata):

    NewNode = Node(newdata)

    NewNode.nextval = self.headval

    self.headval = NewNode

Li = LinkedList()

Li.headval = n1

# Link frist Node to second node

Li.headval.nextval = n2

# Link second Node to third node

n2.nextval = n3

Li.AddBegining("Start")

Li.listprint()

Start

Januari

Febuari

Maret

(Gambar 3.4)

* Mendefinisikan sebuah class ‘Node’ dengan dua atribut yaitu, atribut ‘dataval’ berfungsi untuk menyimpan nilai data dari node, dan ‘nextval’ untuk menunjuk node berikutnya dalam Linked list.
* Membuat tiga instance baru dari class ‘Node’ dengan data “Januari”, “Febuari”, dan “Maret” dan disimpan di variabel ‘n1’, ‘n2’, dan‘n3’.
* Selanjutnya mendefinisikan sebuah class ‘LinkedList’ dengan atribut ‘headval’ untuk menunjuk node pertama dalam Linked list.
* Program akan menganalisis variabel ‘printval’ dengan ‘headval’, kemudian akan melakukan perulangan ‘while’ untuk mencetak nilai ‘dataval’ dari setiap node sampai ‘printval’ menjadi ‘None’.
* Fungsi ‘AddBegining’ adalah metode untuk menambahkan data baru di awal lingket list.
* Kemudian membuat instance baru dari class ‘Node’ dengan berisikan data baru, kemudian mengatur ‘nextval’ dari node baru untuk menunjuk ke ‘headval’, dan mengubah akan ‘headval’ menjadi node baru tersebut.
* Membuat instance baru dari class ‘LinkedList’ dan akan menyimpannya di dalam variabel ‘Li’.
* Lalu menghubungkan node pertama dengan node kedua dengan mengatur ‘nextval’ dari ‘n1’ untuk menunjuk ke ‘n2’.
* Menghubungkan node kedua dengan node ketiga dengan mengatur ‘nextval’ dari ‘n2’ untuk menunjuk ke ‘n3’.
* Memanggil method ‘AddBegining’ untuk menambahkan node baru dengan data “Start” pada awal Linked list.
* Terakhir memanggil fungsi ‘listprint’ dari objek ‘Li’ untuk menampilkan isi dari Linked list.
  1. Penyisipan di Tengah

class Node:

  def \_\_init\_\_(self, dataval=None):

    self.dataval = dataval

    self.nextval = None

# create node

n1 = Node("Januari")

n2 = Node("Febuari")

class LinkedList:

  def \_\_init\_\_(self):

    self.headval = None

# Print the Linked list

  def listprint(self):

    printval = self.headval

    while printval is not None:

      print (printval.dataval)

      printval = printval.nextval

# Function to add node in the middle

  def AddInbetween(self,mid\_node,newdata):

    if mid\_node is None:

      print("The mentioned node is absent")

      return

    NewNode = Node(newdata)

    NewNode.nextval = mid\_node.nextval

    mid\_node.nextval = NewNode

Li = LinkedList()

Li.headval = n1

# Link frist Node to second node

Li.headval.nextval = n2

# add node after n1

Li.AddInbetween(n1, "Middle")

Li.listprint()

Januari

Middle

Febuari

(Gambar 3.5)

* Mendefinisikan sebuah class ‘Node’ dengan dua atribut yaitu, atribut ‘dataval’ berfungsi untuk menyimpan nilai data dari node, dan ‘nextval’ untuk menunjuk node berikutnya dalam Linked list.
* Membuat instance baru dari class ‘Node’ dengan data tiap-tiap “Januari” dan “Febuari”
* Selanjutnya mendefinisikan sebuah class ‘LinkedList’ dengan atribut ‘headval’ untuk menunjuk node pertama dalam Linked list.
* Program akan menganalisis variabel ‘printval’ dengan ‘headval’, kemudian akan melakukan perulangan ‘while’ untuk mencetak nilai ‘dataval’ dari setiap node sampai ‘printval’ menjadi ‘None’.
* Fungi ‘AddInBetween’ berfungsi untuk menambahkan node baru di tengah Linked list setelah node (‘mid\_node’).
* Membuat instance baru dari class ‘Node’ dengan data baru, selanjutnya mengatur ‘nextval’ dari node baru untuk menunjukkan ke node dari yang sebelumnya ditunjuk oleh ‘mid\_node’, lalu mengubah ‘nextval’ dari ‘mid\_node’ untuk menunjuk ke node baru tersebut.
* Membuat instance baru dari class ‘LinkedList’ dan akan menyimpannya di dalam variabel ‘Li’.
* Lalu menghubungkan node pertama dengan node kedua dengan mengatur ‘nextval’ dari ‘n1’ untuk menunjuk ke ‘n2’.
* Menghubungkan node kedua dengan node ketiga dengan mengatur ‘nextval’ dari ‘n2’ untuk menunjuk ke ‘n3’.
* Memanggil method ‘AddInBetween’ untuk menambah node baru dengan data “Middle” setelah node ‘n1’.
* Terakhir memanggil fungsi ‘listprint’ dari objek ‘Li’ untuk menampilkan isi dari Linked list.
  1. Penyisipan di Akhir

class Node:

  def \_\_init\_\_(self, dataval=None):

    self.dataval = dataval

    self.nextval = None

# create node

n1 = Node("Januari")

n2 = Node("Febuari")

class LinkedList:

  def \_\_init\_\_(self):

    self.headval = None

# Print the Linked list

  def listprint(self):

    printval = self.headval

    while printval is not None:

      print (printval.dataval)

      printval = printval.nextval

# Function to add newnode in the end

  def AddInEnd(self, newdata):

    NewNode = Node(newdata)

    if self.headval is None:

      self.headval = NewNode

      return

      last = self.headval

      while(last.nextval):

        last = last.nextval

        last.nextval = NewNode

Li = LinkedList()

Li.headval = n1

# Link frist Node to second node

Li.headval.nextval = n2

# add node after n1

Li.AddInEnd("The Last")

Li.listprint()

Januari

Febuari

(Gambar 3.6)

* Mendefinisikan sebuah class ‘Node’ dengan dua atribut yaitu, atribut ‘dataval’ berfungsi untuk menyimpan nilai data dari node, dan ‘nextval’ untuk menunjuk node berikutnya dalam Linked list.
* Membuat instance baru dari class ‘Node’ dengan data tiap-tiap “Januari” dan “Febuari”
* Selanjutnya mendefinisikan sebuah class ‘LinkedList’ dengan atribut ‘headval’ untuk menunjuk node pertama dalam Linked list.
* Program akan menganalisis variabel ‘printval’ dengan ‘headval’, kemudian akan melakukan perulangan ‘while’ untuk mencetak nilai ‘dataval’ dari setiap node sampai ‘printval’ menjadi ‘None’.
* Fungsi ‘AddInEnd’ berfungsi untuk menambahkan instance baru dari class ‘Node’ dengan data baru.
* Jika Linked list masih kosong (‘headval’ adalah ‘None’), maka node baru akan menjadi ‘headval’.
* Jika sudah terisi maka program akan mencari node terakhir Linked list dan menambahkan node baru tersebut setelah node terakhir.
* Membuat instance baru dari class ‘LinkedList’ dan akan menyimpannya di dalam variabel ‘Li’.
* Lalu menghubungkan node pertama dengan node kedua dengan mengatur ‘nextval’ dari ‘n1’ untuk menunjuk ke ‘n2’.
* Menghubungkan node kedua dengan node ketiga dengan mengatur ‘nextval’ dari ‘n2’ untuk menunjuk ke ‘n3’.
* Memanggil method ‘AddInEnd’ untuk menammbahkan node baru dengan data “The Last” pada akhir list Linked.
* Terakhir memanggil fungsi ‘listprint’ dari objek ‘Li’ untuk menampilkan isi dari Linked list.
  1. Menghapus Node pada Linked List

1. Hapus Node dengan Kata Kunci

class Node:

  def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class LinkedList:

  def \_\_init\_\_(self):

      self.headval = None

# Print the linked list

  def listprint(self):

      printval = self.headval

      while printval is not None:

          print(printval.dataval)

          printval = printval.nextval

# Function to add node at the beginning

  def AddBeginning(self, newdata):

      NewNode = Node(newdata)

      # Update the new node's next val to existing node

      NewNode.nextval = self.headval

      self.headval = NewNode

# Function to remove node by key

  def RemoveNode(self, Removekey):

      HeadVal = self.headval

      if HeadVal is not None:

          if HeadVal.dataval == Removekey:

              self.headval = HeadVal.nextval

              HeadVal = None

              return

      while HeadVal is not None:

          if HeadVal.dataval == Removekey:

              break

          prev = HeadVal

          HeadVal = HeadVal.nextval

      if HeadVal == None:

          return

      prev.nextval = HeadVal.nextval

      HeadVal = None

(Gambar 3.7.1.1)

# created linked list named Li

Li = LinkedList()

# add node

Li.AddBeginning("April")

Li.AddBeginning("Maret")

Li.AddBeginning("Februari")

print("Sebelum dilakukan remove")

Li.listprint()

print("==========================================”)

# remove node by key = "Maret"

Li.RemoveNode("Maret")

print("Setelah dilakukan remove dengan kunci 'Maret'")

Li.listprint()

Sebelum dilakukan remove

Febuari

Maret

April

==========================================

Setelah dilakukan remove dengan kunci ‘Maret’

Febuari

April

(Gambar 3.7.1.2)

* Mendefinisikan sebuah class ‘Node’ dengan dua atribut yaitu, atribut ‘dataval’ berfungsi untuk menyimpan nilai data dari node, dan ‘nextval’ untuk menunjuk node berikutnya dalam Linked list.
* Selanjutnya mendefinisikan sebuah class ‘LinkedList’ dengan atribut ‘headval’ berfungsi untuk menunjuk node pertama di dalam Linked list.
* Kemudian fungsi ‘listprint’ adalah metode untuk menampilkan isi Linked list.
* Program akan menganalisis variabel ‘printval’ dengan ‘headval’, kemudian akan melakukan perulangan ‘while’ untuk mencetak nilai ‘dataval’ dari setiap node sampai ‘printval’ menjadi ‘None’.
* Fungsi ‘AddBegining’ adalah metode untuk menambahkan data baru di awal lingket list.
* Kemudian membuat instance baru dari class ‘Node’ dengan berisikan data baru, kemudian mengatur ‘nextval’ dari node baru untuk menunjuk ke ‘headval’, dan mengubah akan ‘headval’ menjadi node baru tersebut.
* Membuat fungsi ‘RemoveNode’ untuk menghapus node berdasarkan kunci (‘Removekey’).
* Program akan memeriksa apakah node yang akan dihapus adalah node pertama, jika ya maka akan mengubah ‘headval’ menjadi node setelahnya dan node pertama akan terhapus.
* Jika bukan node pertama, program akan mencari node dengan kata kunci yang sesuai dan menghapusnya dengan mengubah ‘nextval’ dari node dengan kata kunci yang sesuai untuk menunjuk node setelahnya.
* Membuat instance baru dari class ‘LinkedList’ dan akan menyimpannya di dalam variabel ‘Li’.
* Memanggil ‘AddBegining’ untuk menambahkan tiga node baru (“April, “Maret”, dan “Febuari”) pada awal Linked list.
* Selanjutnya memanggil method ‘listprint’ untuk mencetak isi Linked list sebelum dilakukan penghapusan.
* Kemudian ‘RemoveNode’ akan berjalan dan menghapus node “Maret” dengan kunci “Maret”.
* Terakhir memanggil method ‘listprint’ untuk mencetak isi Linked list setelah dilakukan penghapusan.

1. Hapus Node di Awal

class Node:

  def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

(Gambar 3.7.2.1)

class LinkedList:

  def \_\_init\_\_(self):

      self.headval = None

# Print the linked list

  def listprint(self):

      printval = self.headval

      while printval is not None:

          print(printval.dataval)

          printval = printval.nextval

# Function to add node at the beginning

  def AddBeginning(self, newdata):

      NewNode = Node(newdata)

# Update the new node's next val to existing node

      NewNode.nextval = self.headval

      self.headval = NewNode

# Function to remove frist node

  def RemoveFrist(self):

    afterhead = self.headval

    self.headval = afterhead.nextval

# created linked list named Li

Li = LinkedList()

# add node

Li.AddBeginning("April")

Li.AddBeginning("Maret")

Li.AddBeginning("Februari")

print("Sebelum dilakukan remove")

Li.listprint()

print("==========================================")

# remove frist node

Li.RemoveFrist()

print("Setelah dilakukan remove node pertama")

Li.listprint()

Sebelum dilakukan remove

Febuari

Maret

April

==========================================

Setelah dilakukan remove node pertama

Maret

April

(Gambar 3.7.2.2)

* Mendefinisikan sebuah class ‘Node’ dengan dua atribut yaitu, atribut ‘dataval’ berfungsi untuk menyimpan nilai data dari node, dan ‘nextval’ untuk menunjuk node berikutnya dalam Linked list.
* Selanjutnya mendefinisikan sebuah class ‘LinkedList’ dengan atribut ‘headval’ berfungsi untuk menunjuk node pertama di dalam Linked list.
* Kemudian fungsi ‘listprint’ adalah metode untuk menampilkan isi Linked list.
* Program akan menganalisis variabel ‘printval’ dengan ‘headval’, kemudian akan melakukan perulangan ‘while’ untuk mencetak nilai ‘dataval’ dari setiap node sampai ‘printval’ menjadi ‘None’.
* Fungsi ‘AddBegining’ adalah metode untuk menambahkan data baru di awal lingket list.
* Kemudian membuat instance baru dari class ‘Node’ dengan berisikan data baru, kemudian mengatur ‘nextval’ dari node baru untuk menunjuk ke ‘headval’, dan mengubah akan ‘headval’ menjadi node baru tersebut.
* Membuat fungsi ‘RemoveFrist’ untuk menghapus node pertama dari Linked list.
* Program akan mengatur ‘headval’ menjadi node berikutnya dari node pertama yang ada, sehingga node pertama akan terhapus.
* Membuat instance baru dari class ‘LinkedList’ dan akan menyimpannya di dalam variabel ‘Li’.
* Memanggil ‘AddBegining’ untuk menambahkan tiga node baru (“April, “Maret”, dan “Febuari”) pada awal Linked list.
* Selanjutnya memanggil method ‘listprint’ untuk mencetak isi Linked list sebelum dilakukan penghapusan.
* Kemudian ‘RemoveFrist’ akan mengapus node pertama dalam Linked list.
* Terakhir memanggil method ‘listprint’ untuk mencetak isi Linked list setelah dilakukan penghapusan.

1. Hapus Node di Akhir

class Node:

  def \_\_init\_\_(self, dataval=None):

      self.dataval = dataval

      self.nextval = None

class LinkedList:

  def \_\_init\_\_(self):

      self.headval = None

# Print the linked list

  def listprint(self):

      printval = self.headval

      while printval is not None:

          print(printval.dataval)

          printval = printval.nextval

# Function to add newnode in the end

  def AddInEnd(self, newdata):

    NewNode = Node(newdata)

    if self.headval is None:

      self.headval = NewNode

      return

    last = self.headval

    while(last.nextval):

      last = last.nextval

    last.nextval = NewNode

# Function to remove frist node

  def RemoveEnd(self):

    last = self.headval

    while(last is not None):

      if last.nextval == None:

        break

      prev = last

      last = last.nextval

    if (last == None):

(Gambar 3.7.3.1)

      return

    prev.nextval = last.nextval

    last = None

# created linked list named Li

Li = LinkedList()

# add node

Li.AddInEnd("April")

Li.AddInEnd("Maret")

Li.AddInEnd("Februari")

print("Sebelum dilakukan remove")

Li.listprint()

print("======================================================================")

# remove frist node

Li.RemoveEnd()

print("Setelah dilakukan remove node terkahir")

Li.listprint()

Sebelum dilakukan remove

April

Maret

Febuari

==========================================

Setelah dilakukan remove node pertama

April

Maret

(Gambar 3.7.3.2)

* Mendefinisikan sebuah class ‘Node’ dengan dua atribut yaitu, atribut ‘dataval’ berfungsi untuk menyimpan nilai data dari node, dan ‘nextval’ untuk menunjuk node berikutnya dalam Linked list.
* Selanjutnya mendefinisikan sebuah class ‘LinkedList’ dengan atribut ‘headval’ berfungsi untuk menunjuk node pertama di dalam Linked list.
* Kemudian fungsi ‘listprint’ adalah metode untuk menampilkan isi Linked list.
* Program akan menganalisis variabel ‘printval’ dengan ‘headval’, kemudian akan melakukan perulangan ‘while’ untuk mencetak nilai ‘dataval’ dari setiap node sampai ‘printval’ menjadi ‘None’.
* Fungsi ‘AddInEnd’ untuk menambahkan node baru pada akhir Linked list.
* Membuat instance baru dari class ‘None’ dengan data baru, Jika Linked list masih kosong maka ‘headval’ adalah ‘None’, maka node baru akan menjadi ‘headval’.
* Jika tidak kosong maka program ini akan mencari node terakhir dari Linked list dan menambahkan node baru tersebut seyelah node terakhir.
* Membuat fungsi ‘RemoveEnd’ yang berfungsi untuk menghapus node terakhir.
* Method ini akan mencari node terakhir dan node sebelumnya, kemudian mengubah ‘nextval’ dari node sebelumnya untuk menunjuk ke ‘None’, sehingga node terakhir akan terhapus.
* Membuat instance baru dari class ‘LinkedList’ dan akan menyimpannya di dalam variabel ‘Li’.
* Memanggil ‘AddInEnd’ untuk menambahkan tiga node baru (“April, “Maret”, dan “Febuari”) pada akhir Linked list.
* Selanjutnya memanggil method ‘listprint’ untuk mencetak isi Linked list sebelum dilakukan penghapusan.
* Kemudian ‘RemoveEnd’ akan mengapus node terakhir dalam Linked list.
* Terakhir memanggil method ‘listprint’ untuk mencetak isi Linked list setelah dilakukan penghapusan.

1. **Latihan**
   * + 1. Double Linked List

class Node:

    def \_\_init\_\_(self, dataval=None):

        self.dataval = dataval

        self.nextval = None

        self.prevval = None

class DoublyLinkedList:

    def \_\_init\_\_(self):

        self.headval = None

        self.tailval = None

    def listprint(self):

        printval = self.headval

        while printval is not None:

            print(printval.dataval)

            printval = printval.nextval

    def InsertAtMiddle(self, prevkey, newdata):

        NewNode = Node(newdata)

        if self.headval is None:

            self.headval = NewNode

            return

        current = self.headval

        while current.dataval != prevkey:

            current = current.nextval

            if current is None:

                print("Node sebelumnya dengan kata kunci yang diberikan tidak ditemukan.")

                return

        NewNode.nextval = current.nextval

        NewNode.prevval = current

        if current.nextval is not None:

            current.nextval.prevval = NewNode

        else:

            self.tailval = NewNode

        current.nextval = NewNode

    def RemoveNodeByKey(self, Removekey):

        current = self.headval

        while current is not None:

            if current.dataval == Removekey:

(Gambar 4.1.1)

                if current.prevval is not None:

               current.prevval.nextval=current.nextval

                else:

                    self.headval = current.nextval

                if current.nextval is not None:

               current.nextval.prevval=current.prevval

                else:

                    self.tailval = current.prevval

                return

            current = current.nextval

dlist = DoublyLinkedList()

node1 = Node("Jhonson")

node2 = Node("Laila")

node3 = Node("Argus")

node1.nextval = node2

node2.prevval = node1

node2.nextval = node3

node3.prevval = node2

dlist.headval = node1

dlist.tailval = node3

print("Linked list sebelum diapa-apain:")

dlist.listprint()

print("=========================================")

dlist.InsertAtMiddle("Jhonson", "Oddet")

print("Linked list jika ditambah diantara:")

dlist.listprint()

print("=========================================")

dlist.RemoveNodeByKey("Laila")

print("Linked list setelah diban 'Laila':")

dlist.listprint()

Lingked list sebelum diapa-apain:

Jhonson

Laila

Argus

=========================================

Lingked list jika ditambah diantara:

(Gambar 4.1.2)

Jhonson

Oddet

Laila

Argus

=========================================

Lingked list setelah diban 'Laila':

Jhonson

Oddet

Argus

(Gambar 4.1.3)

* Kelas ‘Node’ berfungsi untuk membuat simpul dalam linked list. Setiap simpul memili dua atribut yaitu ‘dataval’ yang mentimpan data dan ‘nexuval’ serta ‘prevval’ yang menjadi referensi ke simpul berikutnya juga sebelumnya.
* Kelas ‘DoublyLinkedList’ berfungsi untuk mengelola operasi-operasi pada linked list, misalnya menyisipkan simpul di tengah (InsertAtMiddle) serta menghapus simpul berdasarkan kunci (RemoveNodeByKey).
* Metode ‘InsertAtMiddle’ dengan memasukkan simpul baru di antara dua simpul yang ada. Metode ini akan mencari simpul dengan kunci tertentu (prevkey) serta selanjutnya menyisipkan simpul baru setelah simpul tersebut.
* Metode ‘RemoveNodeByKey’ berfungsi menghapus simpul dengan kunci yang diberikan. Metode ini mencari simpul dengan kunci yang sesuai dan lalu mengatur referensi sebelum dan sesudah simpul tersebut agar tidak lagi menunjuk ke simpul yang akan dihapus.
* Program kemudian membuat objek dlist sebagai instance dari kelas ‘DoublyLinkedList’ dan tiga objek ‘node1’, ‘node2’, dan ‘node3’ sebagai simpul-simpul yang akan dimasukkan ke dalam linked list. Kemudian dilakukan pengaturan hubungan antara simpul-simpul tersebut dengan mengatur ‘nextval’ dan ‘prevval’.
* Setelah linked list dan semua simpul dibuat, dilakukan serangkaian operasi seperti mencetak isi linked list sebelum dan setelah operasi penyisipan serta penghapusan.
  + - 1. Circular Linked List

class Node:

    def \_\_init\_\_(self, dataval=None):

        self.dataval = dataval

        self.nextval = None

class CircularLinkedList:

    def \_\_init\_\_(self):

        self.headval = None

    def listprint(self):

        if self.headval is None:

            print("Lingked list Circular kosong")

            return

        printval = self.headval

        while True:

            print(printval.dataval)

            printval = printval.nextval

            if printval == self.headval:

                break

    def InsertAtEnd(self, newdata):

        NewNode = Node(newdata)

        if self.headval is None:

            self.headval = NewNode

            NewNode.nextval = self.headval

            return

        last = self.headval

        while last.nextval != self.headval:

            last = last.nextval

        last.nextval = NewNode

        NewNode.nextval = self.headval

    def RemoveLast(self):

        if self.headval is None:

            print("Lingked list Circular kosong, tidak ada yang dihapus.")

(Gambar 4.2.1)

            return

        if self.headval.nextval == self.headval:

            self.headval = None

            return

        current = self.headval

        while current.nextval.nextval != self.headval:

            current = current.nextval

        current.nextval = self.headval

clist = CircularLinkedList()

node1 = Node("Lancelot")

node2 = Node("Clint")

node3 = Node("Helcrut")

node1.nextval = node2

node2.nextval = node3

node3.nextval = node1

clist.headval = node1

print("Lingked list Circlar sebelum penyisipan diakhir dan penghapusan diakhir:")

clist.listprint()

print("=========================================")

clist.InsertAtEnd("Hanabi")

print("Lingked list Circular setelah penyisipan diakhir:")

clist.listprint()

print("=========================================")

clist.RemoveLast()

print("Lingked list Circular setelah penghapusan node terakhir:")

clist.listprint()

Lingked list Circlar sebelum penyisipan diakhir dan penghapusan diakhir:

Lancelot

Clint

Helcrut

=========================================

Lingked list Circular setelah penyisipan diakhir:

Lancelot

Clint

(Gambar 4.2.2)

Helcrut

Hanabi

=========================================

Lingked list Circular setelah penghapusan node terakhir:

Lancelot

Clint

Helcrut

(Gambar 4.2.3)

* Kelas ‘’Node berfungsi untuk membuat simpul-simpul dalam linked list. Setiap simpul memiliki dua atribut ‘dataval’ yang menyimpan data dan ‘nextval’ yang merupakan referensi ke simpul berikutnya.
* Kelas ‘CircularLinkedList’ berfungsi untuk mengelola operasi-operasi pada linked list, seperti penyisipan di akhir (InsertAtEnd) dan penghapusan elemen terakhir (RemoveLast).
* Metode ‘InsertAtEnd’ berfungsi memasukkan simpul baru di akhir linked list. Lalu jika linked list kosong, simpul baru diatur sebagai kepala dan dihubungkan ke dirinya sendiri. Jika tidak, maka akan mencari simpul terakhir dan menghubungkan simpul baru ke simpul terakhir serta mengatur kembali koneksi simpul terakhir ke kepala.
* Metode ‘RemoveLast’ berfungsi menghapus simpul terakhir dari linked list. Jika linked list itu kosong atau hanya memiliki satu simpul, simpul tersebut akan dihapus. Jika tidak, metode akan mencari simpul sebelum simpul terakhir, kemudian menghubungkannya kembali ke kepala.
* Program kemudian membuat objek clist sebagai instance dari kelas ‘CircularLinkedList’ dan tiga objek ‘node1’, ‘node2’, dan ‘node3’ sebagai simpul-simpul yang akan dimasukkan ke dalam linked list. Kemudian dilakukan pengaturan hubungan antara simpul-simpul tersebut dengan mengatur nextval.
* **S**etelah linked list dan simpul-simpulnya dibuat, dilakukan serangkaian operasi seperti mencetak isi linked list sebelum dan setelah operasi penyisipan serta penghapusan.

1. **Kesimpulan**

Praktikum ini memberikan pemahaman yang tentang struktur data linked list, dalam bentuk singly linked list maupun juga dalam doubly linked list. Melalui praktikum ini, tidak hanya mempelajari konsep dasar dari linked list, selain itu, praktikum ini juga memperluas pengetahuan dengan memperkenalkan konsep baru, seperti navigasi maju dan mundur di dalam linked list pada doubly linked list. Dengan demikian, bisa dapat diperoleh keterampilan yang lebih luas dalam pemrograman dan pemrosesan data, serta juga meningkatkan kemampuan dalam menganalisis dan memecahkan masalah terkait struktur data. Praktikum ini juga bisa untuk mengembangkan aplikasi yang memanfaatkan struktur data linked list secara efisien dan efektif dalam pemrograman Python.

1. **Referensi**

Goodrich, M. T., *et-al*, 2013. *Data structures and algorithms in Python*. Wiley Publishing.s

Saha, S. 2023. *Data Structures and Algorithms Using Python*. Cambridge University Press.