Name : Azaria Cindy Sahasika
Number Id : 2341760169 / 06
Class : 1G – Business Information System
Lesson : Algorithm and Data Structure
Material : Material 9 – Linked List
Github Link : https://github.com/azariacindy/algorithm-ds

JOBSHEET IX

LINKED LIST

1. Tujuan Praktikum

Setelah melakukan materi praktikum ini, mahasiswa mampu:

1. Membuat struktur data linked list
2. Membuat linked list pada program
3. Membedakan permasalahan apa yang dapat diselesaikan menggunakan linked list

2. Praktikum

2.1 Pembuatan Linked List

Waktu percobaan: 50 menit

Didalam praktikum ini, akan dilakukan implementasi pembuatan linked list menggunakan array dan penambahan node ke dalam linked list

1. Buat folder baru Praktikum09
2. Tambahkan class-class berikut:
    a. Node.java
    b. LinkedList.java
    c. SLLMain.java
3. Deklarasikan class Node yang memiliki atribut data untuk menyimpan elemen dan atribut next bertipe Node untuk menyimpan node berikutnya. Tambahkan constructor berparameter untuk mempermudah inisialisasi

```
public class Node {
    int data;
    Node next;

    public Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
}
```

4. Deklarasikan class LinkedList yang memiliki atribut head. Atribut head menyimpan node pertama pada linked list

```
public class LinkedList {
    Node head;
}
```

5. Sebagai langkah berikutnya, akan diimplementasikan method-method yang terdapat pada class LinkedList.
6. Tambahkan method **isEmpty()**

```
public boolean isEmpty() {
    return (head == null);
}
```

7. Implementasi method print() untuk mencetak dengan menggunakan proses traverse.

```
public void print() {
    if (!isEmpty()) {
        System.out.print("Isi linked list: ");
        Node currentNode = head;

        while (currentNode != null) {
            System.out.print(currentNode.data + "\t");
            currentNode = currentNode.next;
        }

        System.out.println("");
    } else {
        System.out.println("Linked list kosong");
    }
}
```

8. Implementasikan method **addFirst()** untuk menambahkan node baru di awal linked list

```
public void addFirst(int input) {
    Node newNode = new Node(input, null);

    if (isEmpty()) {
        head = newNode;
    } else {
        newNode.next = head;
        head = newNode;
    }
}
```

9. Implementasikan method **addLast()** untuk menambahkan node baru di akhir linked list

```
public void addLast(int input) {
    Node newNode = new Node(input, null);

    if (isEmpty()) {
        head = newNode;
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            currentNode = currentNode.next;
        }

        currentNode.next = newNode;
    }
}
```

10. Implementasikan method **insertAfter()** menambahkan node baru pada posisi setelah node yang berisi data tertentu (key)

```
public void insertAfter(int key, int input) {
    Node newNode = new Node(input, null);

    if (!isEmpty()) {
        Node currentNode = head;

        do {
            if (currentNode.data == key) {
                newNode.next = currentNode.next;
                currentNode.next = newNode;
                break;
            }

            currentNode = currentNode.next;
        } while (currentNode != null);
    } else {
        System.out.print("Linked list kosong");
    }
}
```

11. Pada class SLLMain, buatlah fungsi **main**, kemudian buat object myLinkedList bertipe LinkedList. Lakukan penambahan beberapa data. Untuk melihat efeknya terhadap object myLinkedList, panggil method print()

```
public static void main(String[] args) {
    LinkedList myLinkedList = new LinkedList();
    myLinkedList.print();
    myLinkedList.addFirst(800);
    myLinkedList.print();
    myLinkedList.addFirst(700);
    myLinkedList.print();
    myLinkedList.addLast(500);
    myLinkedList.print();
    myLinkedList.insertAfter(700, 300);
    myLinkedList.print();
}
```

2.1.1 Verifikasi Hasil Percobaan

Cocokkan hasil run program Anda dengan output berikut ini.

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
```

```
Linked List Kosong!
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
```

2.1.2 Pertanyaan

1. Mengapa class LinkedList tidak memerlukan method isFull() seperti halnya Stack dan Queue?

   ➔ Karena struktur data tersebut tidak memiliki batasan max dalam jumlah jumlah elemen yang dapat disimpan, seperti pada stack dan queue yang menggunakan array sebagai penyimpanan. LinkedList ini menggunakan node yang dihubungkan satu sama lain, sehingga dapat menambahkan atau menghapus elemen secara dinamis tanpa batasan max.

2. Mengapa class LinkedList hanya memiliki atribut head yang menyimpan informasi node pertama? Bagaimana informasi node kedua dan lainnya diakses?

   ➔ Linked list hanya memiliki atribut 'head' yang dapat menyimpan informasi node pertama karena setiap node dalam LinkedList saling terhubung melalui referensi 'next'. Untuk mengakses informasi node kedua dan seterusnya, jadi perlu mengikuti referensi 'next' dari node pertama(head) hingga ke node terakhir.

3. Pada langkah, jelaskan kegunaan kode berikut

```
if (currentNode.data == key) {
    newNode.next = currentNode.next;
    currentNode.next = newNode;
    break;
}
```

➔ Kode tersebut sedang mencari node dengan nilai data tertentu 'key'. Jika nilai data pada node saat ini 'currentNode.data' sama dengan nilai 'key', maka sisipkan kode baru 'newNode' setelah node saat ini. Kode 'newNode.next = currentNode.next' mengatur referensi next dari node baru (newNode) untuk menunjuk ke node berikutnya setelah currentNode. Kemudian, 'currentNode.next = newNode' mengatur referensi next dari currentNode untuk menunjuk ke node baru (newNode). Maka, node baru berhasil disisipkan di antara currentNode dan node berikutnya.

4. Implementasikan method insertAt(int index, int key) dari tugas mata kuliah ASD (Teori)

jobsheet9_LinkedList > pract1 > SLLMain06.java > ...

Click here to ask Blackbox to help you code faster

```
1  package jobsheet9_LinkedList.pract1;
2
3  public class SLLMain06 {
       Run | Debug
4      public static void main(String[] args) {
5          LinkedList06 myLinkedList = new LinkedList06();
6          myLinkedList.print();
7          myLinkedList.addFirst(input:800);
8          myLinkedList.print();
9          myLinkedList.addFirst(input:700);
10         myLinkedList.print();
11         myLinkedList.addLast(input:500);
12         myLinkedList.print();
13         myLinkedList.insertAfter(key:700, input:300);
14         myLinkedList.print();
15         myLinkedList.insertAt(index:1, input:300); // Inserting 300 at index 1
16         myLinkedList.print();
17     }
18 }
```

```
Linked List Kosong!
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
Data inserted at index 1
Isi linked list: 700    300     300     800     500
```

```
jobsheet9_LinkedList > pract1 > J LinkedList06.java > ...
  3  public class LinkedList06 {
 73      public void insertAt(int index, int input) {
 75
 76          if (isEmpty()) {
 77              head = newNode;
 78          } else {
 79              Node06 currentNode = head;
 80              Node06 previousNode = null;
 81              int currentIndex = 0;
 82
 83              // Traverse the linked list to find the position to insert the new node
 84              while (currentNode != null && currentIndex < index) {
 85                  previousNode = currentNode;
 86                  currentNode = currentNode.next;
 87                  currentIndex++;
 88              }
 89
 90              // If the index is valid, insert the new node at the specified position
 91              if (currentIndex == index) {
 92                  if (previousNode == null) { // Insertion at the beginning
 93                      newNode.next = head;
 94                      head = newNode;
 95                  } else { // Insertion at other positions
 96                      newNode.next = currentNode;
 97                      previousNode.next = newNode;
 98                  }
 99                  System.out.println("Data inserted at index " + index);
100              } else {
101                  System.out.println(x:"Invalid index! Insertion failed.");
102              }
103          }
104      }
```

2.2 Mengakses dan menghapus node pada Linked List

Waktu percobaan: 50 menit

Didalam praktikum ini, kita akan mengimplementasikan method untuk melakukan pengaksesan dan penghapusan data pada linked list

2.2.1 Langkah-langkah Percobaan

1. Tambahkan method getData() untuk mengembalikan nilai elemen di dalam node pada index tertentu

```
public int getData(int index) {
    Node currentNode = head;

    for (int i = 0; i < index; i++) {
        currentNode = currentNode.next;
    }

    return currentNode.data;
}
```

2. Tambahkan method indexOf() untuk mengetahu index dari node dengan elemen tertentu

```
public int indexOf(int key) {
    Node currentNode = head;
    int index = 0;

    while (currentNode != null && currentNode.data != key) {
        currentNode = currentNode.next;
        index++;
    }

    if (currentNode == null) {
        return -1;
    } else {
        return index;
    }
}
```

3. Tambahkan method removeFirst() untuk menghapus node pertama pada linked list

```
public void removeFirst() {
    if (!isEmpty()) {
        head = head.next;
    } else {
        System.out.println("Linked list kosong");
    }
}
```

4. Tambahkan method removeLast() untuk menghapus node terakhir pada linked list

```
public void removeLast() {
    if (isEmpty()) {
        System.out.println("Linked list kosong");
    } else if (head.next == null) {
        head = null;
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            if (currentNode.next.next == null) {
                currentNode.next = null;
                break;
            }

            currentNode = currentNode.next;
        }
    }
}
```

5. Method remove() digunakan untuk mengapus node yang berisi elemen tertentu

```
public void remove(int key) {
    if (isEmpty()) {
        System.out.println("Linked list kosong");
    } else if (head.data == key) {
        removeFirst();
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            if (currentNode.next.data == key) {
                currentNode.next = currentNode.next.next;
                break;
            }

            currentNode = currentNode.next;
        }
    }
}
```

6. Kemudian, coba lakukan pengaksesan dan penghapusan data di method main pada class SLLMain dengan menambahkan kode berikut

```
System.out.println("Data pada index ke-1: " + myLinkedList.getData(1));
System.out.println("Data 300 berada pada index ke: " + myLinkedList.indexOf(300));

myLinkedList.remove(300);
myLinkedList.print();
myLinkedList.removeFirst();
myLinkedList.print();
myLinkedList.removeLast();
myLinkedList.print();
```

7. Compile dan run program kemudian amati hasilnya

2.2.2 Verifikasi Hasil Percobaan

Cocokkan hasil run program dengan output berikut ini.

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
Data pada index ke-1: 300
Data 300 berada pada index ke: 1
Isi linked list: 700    800     500
Isi linked list: 800    500
Isi linked list: 800
```

```
Linked List Kosong!
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
Data pada index ke-1: 300
Data 300 berada pada index ke: 1
Isi linked list: 700    800     500
Isi linked list: 800    500
Isi linked list: 800
```

2.2.3 Pertanyaan

1. Jelaskan maksud potongan kode di bawah pada method remove()

```
if (currentNode.next.data == key) {
    currentNode.next = currentNode.next.next;
    break;
}
```

➔ Kode diatas digunakan untuk mengecek apakah nilai pada node selanjutnya sama dengan nilai key yang ingin dihapus. Jika iya, maka node tersebut akan dilewati dengan mengatur referensi 'next' dari 'currentNode' untuk menunjukkan ke node setelahnya sehingga node yang mengandung nilai kunci akan dihapus dari linked list.

2. Jelaskan maksud if-else block pada method indexOf() berikut

```
if (currentNode == null) {
    return -1;
} else {
    return index;
}
```

➔ Kode diatas digunakan untuk menentukan apakah pencarian nilai key berhasilatau tidak. Jika 'currentNode' berakhir pada 'null', artinya nilai key tidak ditemukan dalam linked list, sehingga method mengembalikannilai '-1'. Jika tidak, maka nilai key ditemukan dan method mengembalikan indeks node tempat nilai key ditemukan.

3. Error apa yang muncul jika argumen method getData() lebih besar dari jumlah node pada linked list? Modifikasi kode program untuk menghandle hal tersebut.

```java
public int getData(int index) {
    Node06 currentNode = head;
    int currentIndex = 0;

    while (currentNode != null && currentIndex < index) {
        currentNode = currentNode.next;
        currentIndex++;
    }

    if (currentNode == null) {
        System.out.println(x:"Index out of bounds! Returning default value 0.");
        return 0; // Default value or throw an exception according to the requirements
    } else {
        return currentNode.data;
    }
}
```

4. Apa fungsi keyword break pada method remove()? Bagaimana efeknya jika baris tersebut dihapus?

   ➔ Digunakan untuk menghentikan iteraso atau looping setelah node dengan nilai key ditemukan dan dihapus. Jika baris tersebut dihapus, looping akan terus berlanjut sampai akhir linked list, meskipun node yang ingin dihapus sudah ditemukan dan dihapus. Ini akan menyebabkan waktu komputasi yang tidak efisien, karena tidak perlu mencari lebih lanjut setelah node ditemukan dan dihapus.

3. Tugas

Waktu pengerjaan: 50 menit

1. Implementasikan method-method berikut pada class LinkedList:
   a. insertBefore() untuk menambahkan node sebelum keyword yang diinginkan

```java
public void insertBefore(int key, int input) {
    Node06 newNode = new Node06(input, next:null);

    if (!isEmpty()) {
        Node06 currentNode = head;
        Node06 previousNode = null;

        while (currentNode != null) {
            if (currentNode.data == key) {
                if (previousNode == null) { // Insertion at the beginning
                    newNode.next = head;
                    head = newNode;
                } else { // Insertion at other positions
                    newNode.next = currentNode;
                    previousNode.next = newNode;
                }
                break;
            }

            previousNode = currentNode;
            currentNode = currentNode.next;
        }
    } else {
        System.out.println(x:"Linked list kosong!");
    }
}
```

b. insertAt(int index, int key) untuk menambahkan node pada index tertentu

```
73      public void insertAt(int index, int input) {
74          Node06 newNode = new Node06(input, next:null);
75
76          if (isEmpty()) {
77              head = newNode;
78          } else {
79              Node06 currentNode = head;
80              Node06 previousNode = null;
81              int currentIndex = 0;
82
83              // Traverse the linked list to find the position to insert the new node
84              while (currentNode != null && currentIndex < index) {
85                  previousNode = currentNode;
86                  currentNode = currentNode.next;
87                  currentIndex++;
88              }
89
90              // If the index is valid, insert the new node at the specified position
91              if (currentIndex == index) {
92                  if (previousNode == null) { // Insertion at the beginning
93                      newNode.next = head;
94                      head = newNode;
95                  } else { // Insertion at other positions
96                      newNode.next = currentNode;
97                      previousNode.next = newNode;
98                  }
99                  System.out.println("Data inserted at index " + index);
100             } else {
101                 System.out.println(x:"Invalid index! Insertion failed.");
102             }
103         }
104     }
```

c. removeAt(int index) untuk menghapus node pada index tertentu

```
212     public void removeAt(int index) {
213         if (!isEmpty()) {
214             if (index == 0) {
215                 removeFirst();
216             } else {
217                 Node06 currentNode = head;
218                 Node06 previousNode = null;
219                 int currentIndex = 0;
220
221                 while (currentNode != null && currentIndex < index) {
222                     previousNode = currentNode;
223                     currentNode = currentNode.next;
224                     currentIndex++;
225                 }
226
227                 if (currentNode != null) {
228                     previousNode.next = currentNode.next;
229                 } else {
230                     System.out.println(x:"Invalid index! Removal failed.");
231                 }
232             }
233         } else {
234             System.out.println(x:"Empty list, cannot remove at index.");
235         }
236     }
```

2. Dalam suatu game scavenger hunt, terdapat beberapa point yang harus dilalui peserta untuk menemukan harta karun. Setiap point memiliki soal yang harus dijawab, kunci jawaban, dan pointer ke point selanjutnya. Buatlah implementasi game tersebut dengan linked list.