**MODUL II**

**DOUBLE LINKED LIST**

* 1. **TUJUAN**

Tujuan dari praktikum ini adalah:

* 1. Mahasiswa dapat memahami konsep *double linked list*.
  2. Mahasiswa dapat menerapkan jenis-jenis *double linked list* dalam program.
  3. Mahasiswa memahami dan dapat mengimplementasikan *double linked list* dalam program.
  4. **DASAR TEORI**
     1. ***Double Linked List***

*Double linked list* adalah suatu *linked list* yang mempunyai 2 penunjuk, yaitu penunjuk ke simpul sebelumnya dan ke simpul berikutnya [2]. Perhatikan gambar di bawah ini:

|  |
| --- |
| *head tail*  2  4  5  9 |

Gambar 2.1 *Double linked list*

Jadi satu simpul di *double linked list* adalah sebagai berikut :

|  |  |  |  |
| --- | --- | --- | --- |
| Kiri Info Kanan   |  |  |  | | --- | --- | --- | |  |  |  | |

Gambar 2.2 Simpul

Dari gambar di atas dapat dilihat bahwa untuk setiap simpul terdiri dari 3 buah *field*, yaitu medan sambungan kiri (*prev*), medan data (info), dan medan sambungan kanan (*next*) [4].

Contoh pendeklarasiannya:

|  |
| --- |
| struct simpul {  char nama[25];  int nrp;  struct simpul \*before;  struct simpul \*next;  };struct simpul \*baru; |

Ada dua macam *double linked list*, yaitu:

* 1. *Double Linked List Circular* (DLLC)

*Circular* artinya *pointer next* dan *prev*-nya menunjuk ke dirinya sendiri. Jadi, *Double Linked List Circular* (DLLC) adalah *linked list* dengan menggunakan *pointer*, dimana setiap *node* memiliki 3 *field*, yaitu 1 *field pointer* yang menunjuk *pointer* berikutnya (*next*), 1 *field* menunjuk *pointer* sebelumnya (*prev*), serta 1 *field* yang berisi data dengan *pointer next* dan *prev*-nya menunjuk ke dirinya sendiri secara *circular*.

C

B

A

Gambar 2.3 *Double Linked List Cilcular*

Deklarasi *node*:

|  |
| --- |
| Typedef struct Tnode {  int data;  Tnode \*next;  Tnode \*prev; }; |

1. *Double Linked List Non-Circular* (DLLNC)

*Non-Circular* artinya *pointer prev* dan *next*-nya akan menunjuk pada NULL. Jadi, *Double Linked List Non-Circular* (DLLNC) adalah *linked list* yang memiliki 2 buah *pointer*,yaitu *pointer next* dan *prev. Pointer next* menunjuk pada *node* setelahnya dan *pointer prev* menunjuk pada *node* sebelumnya [5].

A

A

A

NULL

NULL

Gambar 2.4 *Double Linked List Non-Cilcular*

Deklarasi *node* :

|  |
| --- |
| Typedef struct Tnode {  int data;  Tnode \*next;  Tnode \*prev; }; |

* + 1. **Operasi – Operasi Pada *Double Linked List***

Penyisipan *Node*

1. Penyisipan di depan/awal

Operasi ini berguna untuk menambahkan satu simpul baru di posisi pertama. Langkah pertama untuk penambahan data adalah pembuatan simpul baru dan mengisinya dengan data pada *field* info-nya.

Ada 2 kondisi yang harus diperhatikan dalam penambahan data di awal yaitu:

1. Ketika *linked list* masih kosong

Jika kondisi *linked list* masih kosong, maka simpul baru akan menjadi simpul awal dan sekaligus simpul akhir dari *double linked list*.

1. Ketika *linked list* tidak kosong

Pada kondisi ini, *pointer* awal akan dipindahkan ke simpul yang baru dengan menghubungkan medan sambungan kanan dari simpul yang ditunjuk oleh *pointer* baru ke simpul yang ditunjuk oleh *pointer* awal dan kemudian menghubungkan medan sambungan kiri (*prev*) dari simpul yang ditunjuk oleh *pointer* awal ke simpul yang baru.

1. Penyisipan di tengah

Operasi penyisipan data di tengah *linked list* adalah suatu operasi menambah data di posisi tertentu di dalam *linked list*. Karena *double linked list* memiliki dua *pointer* sambungan, maka penyisipan bisa dilakukan sebelum data tertentu atau sesudah data tertentu, berbeda dengan *single linked list* yang hanya memiliki satu *pointer* sambungan yaitu sambungan kanan (*next*).

Untuk proses tersebut ada 2 hal yang harus diperhatikan yaitu:

Kondisi *linked list* masih kosong

Prosesnya sama seperti penyisipan di depan/awal.

Ketika *linked list* tidak kosong

Pada kondisi ini program akan memeriksa apakah *head* sama dengan NULL yang artinya *list* tidak kosong, kemudian akan menjalankan perintah tertentu.

1. Penyisipan di belakang/akhir

Operasi ini berguna untuk menambahkan elemen baru di posisi akhir. Langkah pertama untuk penambahan data adalah pembuatan elemen baru dan pengisian nilai infonya.

Ada 2 kondisi yang harus diperhatikan dalam penambahan data di akhir yaitu:

Kondisi *linked list* masih kosong

Pada kondisi ini, prosesnya sama seperti penyisipan di depan/awal.

Ketika *linked list* sudah mempunyai data

Pada kondisi ini, *node* baru akan dihubungkan dengan *tail*, lalu *tail* dipindah ke *node* baru. Setelah itu, variabel penunjuk *prev* dan *next* pada *tail* akan dijadikan NULL.

* + - 1. Penghapusan *Node*

1. Penghapusan di awal

Operasi ini berguna untuk menghapus data pada posisi pertama. Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu:

Kondisi *linked list* masih kosong

Untuk proses penghapusan data pada *linked list* yang masih kosong tidak bisa dilakukan karena tidak ada data yang bisa dihapus dalam *linked list* tersebut.

Kondisi *linked list* hanya memiliki 1 data

Langkah yang perlu dilakukan ketika ingin melakukan proses penghapusan *linked list* yang hanya memiliki 1 data adalah dengan langsung menghapus data dari memori, kemudian *pointer* awal dan akhir diberi harga NULL.

Penghapusan di tengah

Untuk melakukan proses penghapusan di tengah *linked list*, ada 3 kondisi yang perlu diperhatikan yaitu:

1. Kondisi ketika *linked list* masih kosong atau ketika posisi hapus lebih kecil dari 1

Ketika kondisi ini terjadi, maka proses penghapusan tidak bisa dilakukan karena data masih kosong atau karena posisi hapus di luar jangkauan *linked list* (posisi kurang dari 1).

1. Kondisi ketika *linked list* memiliki satu simpul atau posisi hapus sama dengan satu (hapus data pertama)

Ketika kondisi ini terjadi, maka proses yang dilakukan adalah proses penghapusan di posisi awal (hapus\_awal).

1. Kondisi ketika *list* lebih dari satu simpul atau posisi hapus lebih besar dari satu

Variabel penunjuk akan mencari *node* yang akan dihapus kemudian menghubungkan *node* sebelumnya dengan setelahnya, begitu juga sebaliknya. Setelah itu *node* yang ditunjuk akan dihapus.

Penghapusan di akhir

Operasi ini berguna untuk menghapus data pada posisi terakhir. Ada 3 keadaan yang mungkin terjadi ketika akan melakukan proses hapus yaitu:

1. Kondisi *linked list* masih kosong.

Jika kondisi ini terjadi, maka proses penghapusan data tidak bisa dilakukan karena *linked list* masih kosong.

1. Kondisi *linked list* hanya memiliki satu data atau satu simpul. Penghapusan di akhir prosesnya sama seperti penghapusan di depan.
2. Kondisi *linked list* memiliki lebih dari satu data atau lebih dari 1 simpul.

Untuk operasi penghapusan data di posisi terakhir pada *double linked list* yang mempunyai data lebih dari 1 buah yaitu dengan membuat variabel penunjuk pada *tail*, kemudian *tail* dipindah ke *node* sebelumnya lalu variabel penunjuknya dijadikan NULL. Selanjutnya variabel penunjuk akan dihapus [6] .

* 1. **PERMASALAHAN**

1. Membuat program *Circular Double Linked List* yang dapat melakukan:
2. Menambah *Node*
3. Menghapus *Node* di posisi tertentu
4. Menambah *Node* di posisi tertentu
5. Menampilkan *Circular Double Linked List* (dari *head* ke *head*)
6. Membuat program *Double Linked List* berdasarkan kejadian berikut:

Brute Force adalah algoritma untuk memecahkan masalah dengan cara mencoba semua kemungkinan yang ada untuk memecahkan masalah. Pada suatu hari Bambang sang ahli *hacker*, sedang mencoba untuk membobol pin gembok yang mempunyai 5 digit angka, namun Bambang tidak tahu menahu soal pin gembok, karena pin gembok itu memiliki nilai random. Bambang kemudian mencoba memecahkan pin gembok dengan memasukkan angka random untuk nilai 5 digit awal, Bambang kemudian mengeluarkan sebuah alat yang secara otomatis akan mengubah nilai awal menjadi pin gembok, dengan menggunakan Brute Force dimana pengecekan nilai akan dilakukan secara bertahap dari awal ke akhir dan berulang sampai pin gembok terbuka.

* 1. **HASIL DAN PEMBAHASAN**
     1. **Program membuat *circular double linked list*.**

1. Algoritma
2. Mendeklarasikan *class node* sebagai tempat untuk menjadi konstraktor dan penyimpanan data untuk data yang akan diolah.
3. *isEmpty*
4. Memberikan kondisi jika elemen pertama pada *linked list* masih kosong maka bernilai benar dan sebaliknya.
5. Mendeklarasikan *method setCirc* yang dilakukan sebagai *method* untuk menampilkan *circular* yang dimana pada data dalam *list* akan menampilkan *list* data secara *circular*.
6. Memberi perintah kepada *tail* untuk menunjuk ke *head* sehingga *list* setelah *tail* adalah *head.*
7. Memberi perintah kepada head untuk menunjuk ke *tail* sehingga *list* sebelum *head* adalah *tail.*
8. Mendeklarasikan *method addLast* yang dilakukan untuk penambahan *node* dengan penambahan belakang.
9. Mendeklarasikan *node* baru sebagai *node* yang akan dipakai
10. Mengkondisikan jika *head* masih kosong maka *node* baru tersebut akan menjadi *head* sekaligus *tail* pada *list* data.
11. Jika kondisi yang diberikan tidak sesuai, maka sambungkan *tail* dengan *node* baru.
12. Memindahkan posisi tail ke *node* baru.
13. Mendeklarasikan *addposition* yang dilakukan sebagai penambahan *node* dalam posisi tertentu. Maksudnya ialah memasukkan data ke dalam *list* data yang dimana memiliki 2 parameter sebagai acuannya yakni data yang ingin dimasukkan dan setelah indeks keberapa data itu akan dimasukkan.
14. Mengkondisikan jika *head* masih kosong maka *node* tidak bisa ditambahkan.
15. Mendeklarasikan *node* baru sebagai *node* yang ingin di tambahkan pada *list.*
16. Membuat *node* bantu untuk menunjuk *head.*
17. Mencari posisi yang dituju dengan memberikan perulangan yang dimulai dari angka 1 hingga kurang dari posisi yang ingin dituju dimana setiap perulangan *node* bantu akan berpindah ke satu *node* yang berada di depannya.
18. Mengkondisikan jika posisisi node *bantu* saat ini ada di *tail* maka dilakukan penambahan pada *node* paling belakang dengan memanggil *method addLast*.
19. Jika tidak terpenuhi, sisipkan *node* baru ke indeks posisi yang dimaksud.
20. Lalu, membuat *list* data menjadi bentuk *circular* dengan memanggil method *SetCirc.*
21. Mendeklarasikan *method removeposition* yang digunakan untuk menghapus *node* pada saat dalam posisi tertentu. Maksudnya ialah menghapus data pada indeks tertentu yang ada pada dalam *list* data.
22. Mengkondisikan jika *head* masih kosong maka tidak ada *list* data yang dapat dihapus.
23. Mendeklarasikan *node* bantu yang menunjuk ke *head.*
24. Mencari posisi yang dituju dengan memberikan perulangan yang dimulai dari angka 1 hingga kurang dari posisi yang ingin dituju dimana setiap perulangan *node* bantu akan berpindah ke satu *node* yang berada di depannya.
25. Jika *node* bantu berada pada *head* maka pindah *head* ke satu *node* di depannya, kemudian buat menjadi bentuk *circular* dengan memanggil *method SetCirc.*
26. Jika *node* bantu berada di *tail* maka pindah *tail* ke satu elemen dibelakangnya kemudian tunjuk *head* menggunakan *tail* sehingga *head* ada didepan *tail.*
27. Jika selain itu maka elemen setelah bantu *pointer* belakangnya menuju ke satu *node* sebelum bantu, kemudian elemen sebelum *node* bantu *pointer* depannya menunjuk ke satu *node* ke depan bantu.
28. Lalu, membuat *list* data menjadi bentuk *circular* dengan memanggil method *SetCirc*
29. Mendeklarasikan *method print* yang yang berfungsi untuk menampilkan isi *node*. Menampilkan secara urut *node-node* sebuah rantai dari *head* hingga *tail*.
30. Membuat *node* bantu menunjuk ke *head.*
31. Lakukan perulangan selama satu *node* di depan bantubukan sama dengan *head* maka tampilkan isi dari *node bantu* dan pindah *bantu* ke satu *node*  di depannya.
32. Tampilkan isi dari *node* bantu.
33. Tampilkan isi dari *node* yang ada di depan bantu.
34. Memasukkan angka pada program secara dinamis dan memanggil *method-method* yang dibutuhkan dalam menjalankan program.
35. Setelah itu, menampilkan angka-angka yang sudah di deklarasikan ke dalam program.
36. *Source code*

|  |
| --- |
| import java.util.Scanner;  class node{  int data;  node next;  node prev;  public node(int data){  this.data = data;  }  }  class circ{  node head;  node tail;  int size = 0;  boolean isEmpty(){  return head == null;  }  void setCirc(){  head.prev = tail;  tail.next = head;  }    void addlast(int data){  node baru = new node(data);  if(isEmpty()){  head = tail = baru;  }else{  tail.next = baru;  baru.prev = tail;  tail = baru;  }  setCirc();  size++;  }  void addposition(int data, int position){  if(isEmpty()){  return;  }  node baru = new node(data);  node bantu = head;  for(int i = 1; i<position-1; i++){  bantu = bantu.next;  }  if(bantu == tail){  addlast(data);  return;  }  baru.next = bantu.next;  bantu.next.prev = baru;  baru.prev = bantu;  bantu.next = baru;  setCirc();  size++;    }  void print(){  node bantu = head;  // = = = = =  while(bantu.next != head){  System.out.print(bantu.data + " ");  bantu = bantu.next;  }  System.out.print(bantu.data+" ");  System.out.println(bantu.next.data);  }  void removeposition(int position){  if(isEmpty()){  return;  }  node bantu = head;  for(int i = 1; i<position; i++){  bantu = bantu.next;  }  //= = = =  if(bantu == head){  head = head.next;  setCirc();  return;  }  if(bantu == tail){  tail = tail.prev;  tail.next = head;  return;  }  bantu.next.prev = bantu.prev;  bantu.prev.next = bantu.next;  setCirc();  }  }  public class CLL{  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  circ list = new circ();  int pilih = -1;  int posisi = -1;  while(pilih != 0){  System.out.println("Menu : ");  System.out.println("1.Tambah data");  System.out.println("2.Tambah di posisi tertentu ");  System.out.println("3.Hapus di posisi tertentu ");  System.out.println("0.keluar ");  System.out.print("pilih : ");  pilih = in.nextInt();  switch(pilih){  case 1 :  System.out.print("Masukkan banyak data yang ingin di tambah : ");  int jml = in.nextInt();  for(int i = 1; i<=jml; i++){  System.out.print("Masukkan nilai : ");  list.addlast(in.nextInt());  }  break;  case 2 :  System.out.print("Tambah pada posisi ke : ");  posisi = in.nextInt();  System.out.print("Masukkan data : ");  int data = in.nextInt();  list.addposition(data, posisi);  break;  case 3 :  System.out.print("hapus pada posisi ke : ");  posisi = in.nextInt();  list.removeposition(posisi);  break;  case 0 :  break;  default:  System.out.println("kembali masukin pilihan");  }  System.out.print("Data : ");  list.print();  }  System.out.println("Terimakasih");    }  } |

1. Hasil program

|  |
| --- |
|  |

Gambar 2.5 Hasil *run* program *circular double linked list*

Pada **Gambar 2.5** menunjukkan hasil *output* dari permasalahan 1, dapat dilihat bahwa angka yang tertera pada **Gambar 2.5** menunjukkan angka yang keluar merupakan angka 1 sampai 5. Tetapi, setelah angka 3 tersebut ada angka 30 yang ditambahkan ke dalam *list*. lalu mengahpus angka 2 pada list, Hal ini dikarenakan oleh kondisi pada saat dideklarasikan di dalam program, terdapat perintah untuk menghapus dan menambahkan pada saat kondisi tertentu.

* + 1. **Program *Brute Force***

1. Algoritma
2. Membuat kelas untuk simpul, kelas “methode”, dan kelas utama.
3. Mendeklarasikan *method-method* yang dibutuhkan oleh program, pada kelas “methode”.
4. Mendeklarasikan *method addLast* yang dilakukan untuk penambahan *node* dengan penambahan belakang. Maksudnya ialah data yang ada di dalam *list* akan terus bertambah, melalui penambahan belakang. Namun, jika data masih kosong maka data yang dimasukkan sama dengan data pada bagian *list* awal dan akhir.
5. Mendeklarasikan *node* baru sebagai *node* yang akan dipakai
6. Mengkondisikan jika *head* masih kosong maka *node* baru tersebut akan menjadi *head* sekaligus *tail* pada *list* data.
7. Jika kondisi yang diberikan tidak sesuai, maka sambungkan *tail* dengan *node* baru.
8. Memindahkan posisi tail ke *node* baru.
9. Membuat *method display* pada kelas *node* yang berfungsi untuk menampilkan isi *node*. Menampilkan secara urut *node-node* sebuah rantai dari *head* hingga *tail*.
10. Membuat *node* bantu menunjuk ke *head.*
11. Lakukan perulangan selama satu *node* di depan bantubukan sama dengan *head* maka tampilkan isi dari *node bantu* dan pindah *bantu* ke satu *node*  di depannya.
12. Tampilkan isi dari *node* bantu.
13. Tampilkan isi dari *node* yang ada di depan bantu.
14. Mendeklarasikan 2 *object* yang digunakan untuk terhubung dengan *class* methode dan dapat memanggil *method-method* yang ada pada kelas tersebut.
15. Memanggil method *addlast* yang digunakan untuk menampilkan nilai *random* yang angka ditampilkannya dari 0-9.
16. Memberi kondisi jika nilai dari keluaran tebakan *random*-nya lebih besar dari *password*-nya maka nilai *random*-nya akan terus berkurang sehingga sama dengan *password*-nya.
17. Memberi kondisi jika nilai dari keluaran tebakan *random*-nya sama dengan *password*-nya maka nilai yang menampung banyak digit nya terus bertambah yang dimana nilai digitnya dibatasi sama dengan 5.
18. Memberi kondisi jika nilai dari keluaran tebakan *random*-nya lebih kecil dari *password*-nya maka nilai *random*-nya akan terus bertambah sehingga sama dengan *password*-nya.
19. Menampilkan *password*-nya yang dihasilkan dari nilai *random* dan menampilkan tebakan *random* sehingga bisa sama dengan *password*-nya.
20. *Source code*

|  |
| --- |
| import java.util.Random;  import java.util.Scanner;  class simpul{  int data;  simpul next;  simpul prev;  public simpul(int data){  this.data = data;  }  }  class methode {  simpul head;  simpul tail;  int size = 0;  boolean isEmpty() {  return head == null;  }  void addlast(int data) {  simpul baru = new simpul(data);  if (isEmpty()) {  head = tail = baru;  } else {  tail.next = baru;  baru.prev = tail;  tail = baru;  }  size++;  }  void display() {  simpul bantu = head;  // = = = = =  while (bantu.next != null) {  System.out.print(bantu.data + " ");  bantu = bantu.next;  }  System.out.print(bantu.data + " ");  System.out.println();  }  }  public class jurnal2bljr {  public static void main(String[] args) {  methode met = new methode();  methode tebak = new methode();  Random ran = new Random();  Scanner In = new Scanner(System.in);  int tanda = 0;  int time = 0;  System.out.println(" Input password : ");  for (int i = 0; i < 5; i++) {  met.addlast(ran.nextInt(10));  tebak.addlast(ran.nextInt(10));  }  while (true) {  simpul bantu2 = met.head;  simpul bantu1 = tebak.head;  tanda = 0;  while (bantu1 != null) {  if (bantu1.data > bantu2.data) {  bantu1.data--;  } else if (bantu1.data == bantu2.data) {  tanda++;  } else {  bantu1.data++;  }  time++;  bantu1 = bantu1.next;  bantu2 = bantu2.next;  }  if(tanda == 5){  break;  }  System.out.print("Password asli\t: ");  met.display();  System.out.print("Tebakan\t\t\t: ");  tebak.display();  }  }  } |

1. Hasil program

|  |
| --- |
|  |

Gambar 2.6 Hasil *run* program *Brute Force*

Pada **Gambar 2.6** menunjukkan hasil *output* dari permasalahan 2, dapat dilihat bahwa program telah meng-*input* pin, kemudian pada tampilan berikutnya menunjukkan pin yang diacak sampai pin tersebut menunjuk atau menghasilkan pin yang sebenarnya.

* 1. **ANALISA**
     1. **Program Membuat *Circular Double Linked List***

|  |
| --- |
| import java.util.scanner; |

*Source code* “import java.util.Scanner;” berfungsi agar fungsi *Scanner* pada *source code* bisa digunakan atau di jalankan.

|  |
| --- |
| class Node{  public Node next;  public Node prev;  int data;  public Node(int data){  this.data=data;}  } |

*Source code* “class Node {” adalah pembuatan kelas bernama “Node”, menampung pendeklarasian variabel “int data;”, “node next;” dan “node prev;”.

Pada *code* “public Node (int data)” merupakan sebuah konstruktor dari kelas “Node” yang di dalamnya terdapat inisialisasi variabel “this.data=data;” yang dimana dimaksudkan untuk memberi inisialisasi terhadap parameter “data” pada “Node” sama dengan variabel “data” yang ada pada *class* “Node”.

|  |
| --- |
| class circ{  node head;  node tail;  int size = 0; |

*Source code* “class circ{” merupakan kelas bernama “circ”, dimana nantinya terdapat *method main* serta *method-method* lainnya di deklarasikan di dalamnya. Pada *Code* “Node head;” dan “Node tail” merupakan pendeklarasian variabel yang bertipe data “Node” dan variabel “size=0” adalah pendeklarasian variabel bertipe data “integer” yang memiliki nilai sama dengan 0.

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

*Script* “public boolean isEmpty(){” adalah suatu *method* yang memeriksa apakah suatu *list* kosong atau menampung suatu data di dalamnya. Kondisi yang membuat *list* dikatakan kosongadalah apabila awal atau “head”dari *list* tidak menampung data apapun atau bisa disebut “null” yang berarti kosong.

|  |
| --- |
| void setCirc(){  head.prev = tail;  tail.next = head;  } |

*Script* “void setCirc(){” adalah suatu *method* yang digunakan untuk memberi kondisi bahwa dalam *list* bisa menjadi bentuk yang *circular* dan pada *method*  ini dilakukan inisialisasi bahwa “head.prev=tail;” lalu “tail.next=head” yang berarti data *list* tersebut merupakan list yang *circular*.

|  |
| --- |
| void addlast(int data){  node baru = new node(data);  if(isEmpty()){  head = tail = baru;  }else{  tail.next = baru;  baru.prev = tail;  tail = baru;  }  setCirc();  size++;  } |

*Script* “public void addlast(int data){” adalah suatu *method* yang berfungsi menambahkan data baru di akhir *list* tetapi harus diberikan kondisi terlebih dahulu, yaitu kondisi jika *list* kosong maka “head” dan “tail” akan diisikan oleh data “baru”, kondisi lainnya yaitu jika *list* tidak kosong maka “baru” akan menjadi “tail” dari *list*. Lalu, memanggil *method* “setCirc” dan memberikan variabel “size” nilai menjadi ditambah 1.

|  |
| --- |
| void addposition(int data, int position){  if(isEmpty()){  return;  }  node baru = new node(data);  node bantu = head;  for(int i = 1; i<position-1; i++){  bantu = bantu.next;  }  if(bantu == tail){  addlast(data);  return;  }  baru.next = bantu.next;  bantu.next.prev = baru;  baru.prev = bantu;  bantu.next = baru;  setCirc();  size++;    } |

*Script*  “public void addposition(int data){” adalah suatu *method* yang berfungsi menambahkan data baru di indeks terentu pada *list* tetapi harus diberikan kondisi terlebih dahulu, yaitu kondisi jika *list* kosong program akan “return”, lalu ada pendeklarasian baru yang diberi nama “bantu” yang bertipe data “node”. Lalu ada kondisi dimana melakukan pengulangan atau *looping* menggunnakan fungsi “for” yang akan berhenti jika “i<position-1”. Jika “bantu” sama dengan “tail” maka akan dipanggil *method* “addlast(data)”. Sehingga nantinya “bantu.next” akan sama dengan “baru”. Lalu, memanggil *method* “setCirc” dan memberikan variabel “size” nilai menjadi ditambah 1.

|  |
| --- |
| void print(){  node bantu = head;  // = = = = =  while(bantu.next != head){  System.out.print(bantu.data + " ");  bantu = bantu.next;  }  System.out.print(bantu.data+" ");  System.out.println(bantu.next.data);  } |

*Source* *code* “void print(){” berfungsi membuat sebuah *method* yang bernama “print” yang bertipe *void* artinya tidak memiliki nilai *return* atau pengembalian. *Method* ini berfungsi untuk menampilkan nilai semua *node*  yang sudah dibuat. *Method* memiliki perulangan dimana “bantu.next” akan tidak sama dengan “head” sehingga variabel “bantu” sama dengan “bantu.next” dan akan menampilkan data dari “bantu.data” dan “bantu.next.data”.

|  |
| --- |
| void removeposition(int position){  if(isEmpty()){  return;  }  node bantu = head;  for(int i = 1; i<position; i++){  bantu = bantu.next;  }  if(bantu == head){  head = head.next;  setCirc();  return;}  if(bantu == tail){  tail = tail.prev;  tail.next = head;  return;}  bantu.next.prev = bantu.prev;  bantu.prev.next = bantu.next;  setCirc();  }  } |

*Source* *code* “remove position(){” berfungsi membuat sebuah *method* yang bernama “remove position” yang bertipe *void* artinya tidak memiliki nilai *return* atau pengembalian. *Method* ini berfungsi untuk menghapus nilai di suatu indeks pada *list*. Tetapi harus diberikan kondisi terlebih dahulu, yaitu kondisi jika *list* kosong program akan “return”, lalu ada pendeklarasian baru yang diberi nama “bantu” yang bertipe data “node” dan “bantu” sama dengan “head”. Lalu ada kondisi dimana melakukan pengulangan atau *looping* menggunnakan fungsi “for” yang akan berhenti jika “i<position” maka variabel “bantu” sama dengan “bantu.next”. Jika “bantu” sama dengan “head” maka variabel “head” sama dengan “head.next” lalu akan dipanggil *method* “setCirc()”. Jika “bantu” sama dengan “tail” maka “tail” sama dengan “tail.prev” lalu “tail.next” sama dengan “head”. Maka variabel “bantu.prev.next” akan menunjuk variabel “bantu.next”. Lalu, memanggil *method* “setCirc”.

|  |
| --- |
| public class CLL{  public static void main(String[] args) {  Scanner in = new Scanner(System.in);  circ list = new circ();  int pilih = -1;  int posisi = -1; |

*Source code* “public class CLL {” merupakan kelas utama bernama “CLL”, dimana nantinya kelas ini menjadi kelas utama dalam program. Pada *Code* “circ list = new circ()” merupakan pendeklarasian bahwa “list” menjadi penghubung dengan *class* “circ” . sertavariabel “pilih=-1” dan “posisi=-1” adalah pendeklarasian variabel bertipe data “integer” yang memiliki nilai sama dengan -1.

|  |
| --- |
| while(pilih != 0){  System.out.println("Menu : ");  System.out.println("1.Tambah data");  System.out.println("2.Tambah di posisi tertentu ");  System.out.println("3.Hapus di posisi tertentu ");  System.out.println("0.keluar ");  System.out.print("pilih : ");  pilih = in.nextInt(); |

*Source code* “while(pilih != 0)” merupakan proses perulangan dimana variabel “pilih” tidak sama dengan 0. Serta pada program akan ditampilkan menu yang berisi “1. Tambah data 2.Tambah di posisi tertentu 3.Hapus di posisi tertentu 0.keluar dan pilih : ”. Variabel “pilih” dideklarasikan sebagai variabel penyimpan *input*-an.

|  |
| --- |
| case 1 :  System.out.print("Masukkan banyak data yang ingin di tambah : ");  int jml = in.nextInt();  for(int i = 1; i<=jml; i++){  System.out.print("Masukkan nilai : ");  list.addlast(in.nextInt());  }  break;  case 2 :  System.out.print("Tambah pada posisi ke : ");  posisi = in.nextInt();  System.out.print("Masukkan data : ");  int data = in.nextInt();  list.addposition(data, posisi);  break;  case 3 :  System.out.print("hapus pada posisi ke : ");  posisi = in.nextInt();  list.removeposition(posisi);  break;  case 0 :  break;  default:  System.out.println("kembali masukin pilihan");  }  System.out.print("Data : ");  list.print();  }  System.out.println("Terimakasih");    }  } |

*Source Code* “case 1” merupakan opsi pertama yang dimana menyimpan nilai input dari variabel “pilih” yang dimana pada opsi pertama akan memanggil *method* “addlast()” yang digunakan untuk menambahkan data pada list. *Code* “case 2” merupakan opsi kedua yang dimana menyimpan nilai input dari variabel “pilih” yang dimana pada opsi kedua akan memanggil *method* “addposition()” yang digunakan untuk menambahkan data pada indeks tertentu. *Code* “case 3” merupakan opsi ketiga yang dimana menyimpan nilai input dari variabel “pilih” yang dimana pada opsi ketiga akan memanggil *method* “removeposition()” yang digunakan untuk menghapus data pada indeks tertentu. Lalu diluar dari beberapa opsi tadi akan ditampilkan pada program tentang data yang ada pada list dengan memanggil *method* “print()” dan menampilkan kalimat “Terima kasih” pada program.

**2.5.2 Program Brute Force**

|  |
| --- |
| import java.util.Random; |

Potongan kode “import java.util.Random;” berfungsi untuk dapat menggunakan perintah “Random”.

|  |
| --- |
| import java.util.scanner; |

*Source code* “import java.util.Scanner;” berfungsi agar fungsi *Scanner* pada *source code* bisa digunakan atau di jalankan.

|  |
| --- |
| class simpul{  int data;  simpul next;  simpul prev;  public simpul(int data){  this.data = data;  }  } |

*Source code* “class simpul {” adalah pembuatan kelas bernama “simpul”, menampung pendeklarasian variabel “int data;”, “simpul next;” dan “simpul prev;”. Pada *code* “public simpul (int data)” merupakan sebuah konstruktor dari kelas “simpul” yang di dalamnya terdapat inisialisasi variabel “this.data=data;” yang dimana dimaksudkan untuk memberi inisialisasi terhadap parameter “data” pada “simpul” sama dengan variabel “data” yang ada pada *class* “simpul”.

|  |
| --- |
| class methode {  simpul head;  simpul tail;  int size = 0; |

*Source code* “class methode{” merupakan kelas bernama “methode”, dimana nantinya terdapat *method-method* lainnya di deklarasikan di dalamnya. Pada *Code* “simpul head;” dan “simpul tail” merupakan pendeklarasian variabel yang bertipe data “simpul” dan variabel “size=0” adalah pendeklarasian variabel bertipe data “integer” yang memiliki nilai sama dengan 0.

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

*Script* “public boolean isEmpty(){” adalah suatu *method* yang memeriksa apakah suatu *list* kosong atau menampung suatu data di dalamnya. Kondisi yang membuat *list* dikatakan kosongadalah apabila awal atau “head”dari *list* tidak menampung data apapun atau bisa disebut “null” yang berarti kosong.

|  |
| --- |
| void addlast(int data) {  simpul baru = new simpul(data);  if (isEmpty()) {  head = tail = baru;  } else {  tail.next = baru;  baru.prev = tail;  tail = baru;  }  size++;  } |

*Script* “public void addlast(int data){” adalah suatu *method* yang berfungsi menambahkan data baru di akhir *list* tetapi harus diberikan kondisi terlebih dahulu, yaitu kondisi jika *list* kosong maka “head” dan “tail” akan diisikan oleh data “baru”, kondisi lainnya yaitu jika *list* tidak kosong maka “baru” akan menjadi “tail” dari *list*. Lalu, memberikan variabel “size” nilai menjadi ditambah 1.

|  |
| --- |
| void display() {  simpul bantu = head;  // = = = = =  while (bantu.next != null) {  System.out.print(bantu.data + " ");  bantu = bantu.next;  }  System.out.print(bantu.data + " ");  System.out.println();  }  } |

*Source* *code* “void print(){” berfungsi membuat sebuah *method* yang bernama “display” yang bertipe *void* artinya tidak memiliki nilai *return* atau pengembalian. *Method* ini berfungsi untuk menampilkan nilai semua *node*  yang sudah dibuat. *Method* memiliki perulangan dimana “bantu.next” akan tidak sama dengan “head” sehingga variabel “bantu” sama dengan “bantu.next” dan akan menampilkan data dari “bantu.data”.

|  |
| --- |
| public class jurnal2bljr {  public static void main(String[] args) {  methode met = new methode();  methode tebak = new methode();  Random ran = new Random();  Scanner In = new Scanner(System.in);  int tanda = 0;  int time = 0; } |

*Source* *code* “public class jurnal2bljr” digunakan untuk mendeklarasikan *class* “jurnal2bljr” yang digunakan untuk menjadi *class* utamanya. Pada *code* “methode met = new methode();, methode tebak = new methode();” digunakan untuk mendeklarasikan penunjukyang terhubung dengan *class* “methode” yang bernama “met” dan “tebak” yang dimana nantinya digunakan untuk menunjuk *class* “methode” . Pada *code* “random ran = new Random();” dan “Scanner In = new Scanner(System.in);” digunakan untuk melakukan pendeklarasian terhadap fungsi *Scanner* dan fungsi *Random*. Serta *code* “int tanda=0” dan “int time=0” berfungsi untuk mendeklarasikan variabel “tanda” dan “time” sama dengan 0.

|  |
| --- |
| System.out.println(" Input password : ");  for (int i = 0; i < 5; i++) {  met.addlast(ran.nextInt(10));  tebak.addlast(ran.nextInt(10));  } |

*Source* *code* “for (int i = 0; i < 5; i++) {” digunakan untuk melakukan perulangan dengan mendeklarasikan variabel “i=0” yang dimana variabel “i<5” dan variabel “i” melakukan operasi *increment* dimana “i++”. Pada *code* “met.addlast(ran.nextInt(10));” menggunakan penunjuk “met” untuk memanggil *method* “addlast” yang berfungsi untuk mencetak data-data random yang kurang dari 10 yakni dari angka 1 sampai 9. Pada *code* “tebak.addlast(ran.nextInt(10);” menggunakan penunjuk “tebak” untuk memanggil *method* “addlast” yang berfungsi untuk mencetak data-data random yang kurang dari 10 yakni dari angka 1 sampai 9.

|  |
| --- |
| while (true) {  simpul bantu2 = met.head;  simpul bantu1 = tebak.head;  tanda = 0; |

*Source* *code* “while (true){ ” digunakan untuk melakukan perulangan yang dimana perulangan tersebut akan berhenti jika sudah pada kondisi *true*. Pada *code* “simpul bantu2 = met.head” dan “simpul bantu1 = tebak.head” yang digunakanuntuk mendeklarasikan variabel “bantu2” yang sama dengan “met.head” dan “bantu1” sama dengan “simpul tebak.head” yang dimana pendeklarasian tersebut berguna untuk nantinya mengecek data yang *random* dan mencari data *random* sampai nantinya tebakan nya sama dengan data *random* itu sendiri.

|  |
| --- |
| while (bantu1 != null) {  if (bantu1.data > bantu2.data) {  bantu1.data--;  } else if (bantu1.data == bantu2.data) {  tanda++;  } else {  bantu1.data++;  }  time++;  bantu1 = bantu1.next;  bantu2 = bantu2.next;  }  if(tanda == 5){  break;  } |

*Source* *code* “while (bantu1 != null){ ” digunakan untuk melakukan perulangan yang dimana perulangan tersebut akan berhenti jika data sudah berada pada kondisi *null*. Pada *code* “if(bantu1.data > bantu2.data)” merupakan pemberian suatu kondisi dimana jika “(bantu1.data)” lebih besar dari “(bantu2.data)” maka akan dijalankan perintah “bantu1.data--” yang dimana dilakukan proses pengurangan sebanyak -1. Pada *code* “ else if(bantu1.data == bantu2.data)” merupakan pemberian suatu kondisi yang lain dimana jika “(bantu1.data)” sama dengan “(bantu2.data)” maka akan dijalankan perintah “tanda++” yang dimana variabel “tanda” melakukan proses penjumlahan sebanyak +1. Pada *code* “ else {” merupakan pemberian suatu kondisi yang lain dari kondisi yang sudah diberikan yang dimana akan dijalankan perintah “bantu1.data++” yang dimana dilakukan proses penjumlahan sebanyak +1. Pada *code* “if(tanda==5)” merupakan pemberian suatu kondisi diluar perulangan “while (bantu1 != null){ ” dimana jika variabel “tanda” sama dengan 5 maka proses perulangan “while (true){ ” akan diberhentikan oleh perintah “break;” dan kembali lagi untuk mengecek ke jumlah digit data yang kurang dari 5.

|  |
| --- |
| System.out.print("Password asli\t: ");  met.display();  System.out.print("Tebakan\t\t\t: ");  tebak.display();  }  }  } |

*Source code* “System.out.print("Password asli\t: ");” digunakan untuk menampilkan data *password* *random* yang asli dan “met.display” akan menampilkan data yang sudah diolah sebelumnya untuk menampilkan data *random*.Pada *code* “System.out.print("Tebakan \t \t \t : ");” digunakan untuk menampilkan data tebakan akan data *random* yang asli dan “tebak.display” akan menampilkan data yang sudah diolah sebelumnya untuk menampilkan tebakan pada data *random.*

**2.6 KESIMPULAN**

Dari praktikum yang telah di laksanakan dapat diambil beberapa kesimpulan, yaitu sebagai berikut:

1. *Double linked list* tiap-tiap *node* nya saling terhubung dan oleh karena itu dengan *double linked list* pengolahan untuk mengurutkan data jadi lebih mudah dan efisien.
2. *Double linked list* memiliki jenis-jenis yakni *Circular* *double linked list* dan *Non*-*Circular* *double linked list*, yang dimana jika menggunakan *Circular DLL* berarti *node* pada *list* saling terhubung langsung antara *head* dan *tail* nya. Sedangkan *Non-Circular DLL* berarti *node* pada *list* hanya dimulai dari *head* hingga *tail* lalu setelah *tail* ada *node* yang *null*.
3. Pengimplementasian *double linked list* dalam sebuah program yaitu dengan menggunakan Operasi-Operasi yang ada pada *double linked list* itu sendiri, dimana pada praktikum yang sudah dilakukan Operasi yang digunakan ialah menambah data, menghapus dan menampilkan data.