MODUL I

**SINGLE LINKED LIST DAN ARRAY**

* 1. **TUJUAN**

Tujuan dari praktikum ini adalah:

* 1. Mahasiswa mengetahui bagaimana mengimplementasikan *array* dalam program.
  2. Mahasiswa memahami dan mampu membuat *single linked list.*
  3. Mahasiswa dapat mengimplementasikan berbagai operasi yang terdapat pada *linked list*.
  4. **DASAR TEORI**

**1.2.1*****Array***

*Array* atau larik adalah kelompok peubah tunggal atau multi-dimensi. *Array* atau larik merupakan sejumlah nilai-nilai bertipe sama yang bisa dirujuk menggunakan nama peubah yang sama diikuti indeksnya (posisi nilai tertentu dalam larik) [1].

*Array* menyimpan data secara berurutan pada memori komputer. Sekali *array* dideklarasikan (dibuat) maka akan dialokasikan sejumlah tempat di memori komputer yang letaknya selalu berdekatan (bersebelahan) [2].

Deklarasi *Array*:

|  |
| --- |
| tipe\_data nama\_array [jumlah\_elemen]; |

Pengaksesan *Array*:

|  |
| --- |
| nama\_array [indeks]; |

Jenis-jenis *array* antara lain sebagai berikut:

1. *Array* Satu Dimensi

*Array* satu dimensiadalah struktur data yang statis yang mempunyai 1 nama tetapi memiliki banyak tempat [2]. *Array* satu dimensi yaitu *array* yang terdiri atas satu baris dan banyak kolom. Pendeklarasian suatu *array* harus diikuti oleh suatu indeks yang menunjukan jumlah maksimum data yang disediakan [3].

Berikut pendeklarasian *array* satu dimensi:

|  |
| --- |
| type\_data[] nama\_array; |

1. *Array* Dua Dimensi

*Array* dua dimensi tersusun dalam bentuk baris dan kolom, dimana elemen pertama menunjukkan baris dan elemen kedua menunjukkan kolom [2].

Berikut pendeklarasian *array* dua dimensi:

|  |
| --- |
| type\_data[][] nama\_array = new type\_data[][]; |

1. *Array* Multi-Dimensi

*Array* multi-dimensi merupakan tipe data yang sering digunakan pada pendeklarasian variabel yang sama tapi memiliki lebih dari dua elemen yang berbeda, serta pengisian elemen *array* dilakukan melalui indeks. Indeks *array* secara *default* dimulai dari 0,0. Jumlah elemennya adalah elemen-1, elemen-2,...elemen-n [2].

Berikut pendeklarasian *array* multi-dimensi:

|  |
| --- |
| type\_data[][][] nama\_array = new type\_data[][][]; |

1. *Array* Statis

*Array* statis adalah model pendeklarasian *array* dimana tipe data yang digunakan mempunyai nilai yang tetap. Nilai yang digunakan untuk menentukan jangkauan pada umumnya bernilai *integer*. *Array* statis juga bisa disebut *array* dengan deklarasi tipe indeks *sub*-*range integer* [3].

1. *Array* Dinamis

*Array* dinamis adalah *array* yang tidak mempunyai suatu jangkauan atau ukuran yang tetap. Tetapi ketika program dijalankan maka memori untuk suatu *array* dinamis dialokasikan ketika menugaskan suatu nilai kepada *array* [3].

Berikut pendeklarasian *array* dinamis:

|  |
| --- |
| type\_data[][][] nama\_array = new type\_data[][][]; |

**1.2.2*****Single Linked List***

Sebuah *list* yang elemennya hanya menyimpan informasi elemen setelahnya (*next*) dan hanya memiliki satu buah pengait ke elemen berikutnya. Pengait ke elemen selanjutnya adalah sebuah *pointer* [1].

Contoh:



head

tail

data

pointer NULL

Gambar 1.1*Single linked list*

1. **Perbedaan *Array* dan *Single* *Linked* *List***

Berikut ini tabel perbedaan antara *array* dengan *single* *linked* *list* [2].

Tabel 1.1Perbedaan *array* dengan *single linked list*

|  |  |
| --- | --- |
| *Array* | *Single* *Linked* *List* |
| Bersifat statis. | Bersifat dinamis. |
| Elemen data bisa menggunakan *record*. | Elemen data harus menggunakan *record*. |
| Volumenya selalu tetap, tidak tergantung pada jumlah data. | Ukurannya berubah – ubah disesuaikan dengan kebutuhan. |
| Alokasi memori dilakukan pada saat *array* didefinisikan. | Alokasi memori ditentukan pada saat data baru dibuat. |
| Pembebasan memori dilakukan saat program berhenti. | Pembebasan memori dilakukan pada setiap ada penghapusan data. |
| Cara akses bersifat random dengan menggunakan nomor indeks. | Cara akses ke masing – masing *class* data dilakukan secara linier (selalu dimulai dari elemen pertama). |

* + - 1. **Operasi – Operasi yang terdapat pada *Single* *Linked List***

Beberapa operasi – operasi dasar pada *single* *linked* *list* antara lain:

*Insert*

*Insert first*, merupakan operasi penyisipan atau penambahan simpul (*node*) sebagai simpul pertama atau depan.

|  |
| --- |
| void insertashead(list insert){  insert->next=head;  head=insert;  } |

*Insert last (add back),* merupakan operasi penyisipan atau penambahan simpul (*node*) sebagai simpul akhir.

|  |
| --- |
| void insertastail(list insert){  list tail;tail=head;  do  tail=tail->next;  while(tail->next!=NULL);  tail->next=insert;  tail=tail->next;  } |

*Insert after*, merupakan operasi penyisipan atau penambahan simpul (*node*) setelah simpul tertentu.

|  |
| --- |
| void insertafternode(int x,list insert){  list after;after=head;  do  after=after->next;  while(after->datalist!=x);  insert->next=after->next;  after->next=insert;  } |

*Insert before*, merupakan operasi penyisipan atau penambahan simpul (*node*) sebelum simpul tertentu.

|  |
| --- |
| void insertbeforenode(int x,list insert){  list before;prevbefore;  if(head->datalist=x)  insertashead(insert);  else{  before=head;  do  prevbefore=before;  before=before->next;  while(before->datalist!=x);  insert->next=before;  prevbefore->next=insert;}  } |

Contoh kode program dan notasi pada pada operasi *insert*:

1. Operasi Tambah Depan

|  |
| --- |
| public void tambahDepan(int d){  Node baru = new Node(d);  if(head==null){  head = baru;  tail = baru;  }  else{  baru.next = head;  head = baru;  }  } |

Notasi dalam bentuk *node*:

* + - * 1. (“head = null”)

|  |
| --- |
| *“*null” ← *“head”* |

* + - * 1. Masuk data baru, misalnya 5

|  |
| --- |
|  |

Gambar 1.2 *Node* memasukkan data baru sebesar 5

* + - * 1. Masuk data baru, misalnya 20 (penambahan di depan)

|  |
| --- |
|  |

Gambar 1.3 *Node* memasukkan data baru sebesar 20

1. Operasi Tambah Belakang:

|  |
| --- |
| public void tambahBelakang(int d){  Node baru = new Node(d);  if(head==null){  head = baru;  tail = baru;  }  else{  tail.next = baru;  tail = baru;  }  } |

Notasi dalam bentuk *node*:

|  |
| --- |
|  |

Gambar 1.4 *Node* pada operasi tambah belakang

*Delete*

*Delete first/front*, merupakan operasi penghapusan simpul (*node*) pada awal *linked list*.

*Delete last/back*, merupakan operasi penghapusan simpul (*node*) pada akhir *linked list.*

*Delete after*, merupakan operasi penghapusan simpul (*node*) setelah simpul tertentu pada *linked list.*

*Delete before*, merupakan operasi penghapusan simpul *(node*) sebelum simpul tertentu pada *linked list.*

*Delete all*, merupakan operasi penghapusan semua *linked list* yang ada.

Contoh kode program dan notasi pada pada operasi *insert*:

* + - 1. Operasi Hapus Depan

|  |
| --- |
| public Node hapusDepan(){  if(head!=null){  node temp = head;  head = head.next;  temp.next = null;  return temp;  }  else{  System.out.println(“list kosong”);  return null;  }  } |

Notasi dalam bentuk *node*:

|  |
| --- |
|  |

Gambar 1.5 *Node* pada operasi hapus depan

* + - 1. Operasi Hapus Belakang:

|  |
| --- |
| public Node hapusBelakang(){  if(head!=null){  Node bantu, temp;  if(head.next==null){  temp = head;  head = tail null;  }  else{  bantu = head;  while(bantu.next!=tail){  bantu = bantu.next;  }  temp = tail;  tail = bantu;  tail.next = null;  }  else{  System.out.println(“list kosong”);  return null;  }  }  } |

Notasi dalam bentuk *node*:

|  |
| --- |
|  |

Gambar 1.6*Node* pada operasi hapus belakang

* 1. **PERMASALAHAN**

1. Membuat program untuk menghitung hasil penjumlahan matriks menggunakan *array* dua dimensi (ukuran *array* dimasukkan secara dinamis).
2. Membuat program berdasarkan kejadian berikut:

Narto dan Saskeh adalah seorang sahabat, suatu hari ia dan 18 teman lainnya pergi ke sebuah gunung angker. Narto dan temannya berjalan membentuk sebuah barisan sejumlah 20 orang, Narto berada di barisan ketiga dan Saskeh berada di barisan kedua. Di tengah perjalanan tiba-tiba orang yang berada di barisan paling depan dan paling belakang menghilang dan akhirnya menyebabkan barisan terbelah menjadi 2 kubu, kubu pertama dipimpin Narto terdiri dari orang yang berada pada barisan ganjil dan sisa nya berada di kubu Saskeh. Kemudian mereka melanjutkan perjalanan pada gunung tersebut secara terpisah kemudian tanpa disadari terdapat seorang di kubu Narto pindah ke kubu Saskeh dan sebaliknya, akibat hal tersebut kubu Narto dan Saskeh mulai mengakami kesusahan untuk melanjutkan perjalanan karena konflik semakin melebar, mereka mulai putus asa hingga akhirnya Narto dan Saskeh memutuskan untuk bergabung untuk melakukan pendakian kembali, dalam perjalananya mereka menemukan 2 orang yang hilang dalam perjalanan tadi, kemudian 2 orang tersebut kembali masuk ke tengah barisan dan akhirnya mereka berhasil mencapai puncak gunung tersebut dengan anggota lengkap.

* 1. **HASIL DAN PEMBAHASAN**
     1. **Program penjumlahan *array* dua dimensi**

1. Algoritma
   1. Menyiapkan variabel-variabel untuk menyimpan banyak baris dan kolom matriks.
   2. Membuat *array* di dalam perulangan untuk menyimpan matriks-matriks yang akan dijumlahkan dan matriks hasil penjumlahan.
   3. Memasukkan banyak nilai baris yang akan dimasukan.
   4. Memasukkan banyak nilai kolom yang akan dimasukan.
   5. Memasukkan nilai bilangan-bilangan pada indeks-indeks matriks pertama.
   6. Menampilkan hasil pada matriks pertama.
   7. Memasukkan nilai bilangan-bilangan pada indeks-indeks matriks pertama.
   8. Menampilkan hasil pada matriks kedua.
   9. Melakukan operasi penjumlahan pada matriks pertama dan kedua.
2. *Source code*

|  |
| --- |
| import java.util.Scanner;  public class JumlahMatriks {  public static void main(String[] args) {  int baris, kolom, b, k;  Scanner in = new Scanner(System.in    System.out.println("Penjumlahan Matriks");  System.out.println("Masukkan ordo matriks");  System.out.println("=====================");  System.out.print("\nMasukkan jumlah baris: "); baris = in.nextInt();  System.out.print("Masukkan jumlah kolom: "); kolom = in.nextInt();    int A[][] = new int[baris][kolom];  int B[][] = new int[baris][kolom];  int jumlah[][] = new int[baris][kolom];    System.out.print("\n\nMasukkan angka-angka ke dalam matriks A");  System.out.print("\n=======================\n");  for ( b = 0 ; b < baris ; b++ ){  for ( k = 0 ; k < kolom ; k++ ){  System.out.print("[" +(b+1)+ "][" +(k+1)+ "]:");  A[b][k] = in.nextInt();  }  }  System.out.print("Matriks A");  for ( b = 0 ; b < baris ; b++ ){  System.out.print("\n");  for ( k = 0 ; k < kolom ; k++ ){  System.out.print(" " + A[b][k]);  }  }    System.out.print("\n\nMasukkan angka-angka ke dalam matriks B");  System.out.print("\n=======================\n");  for ( b = 0 ; b < baris ; b++ ){  for ( k = 0 ; k < kolom ; k++ ){  System.out.print("[" +(b+1)+ "][" +(b+1)+ "]:");  B[b][k] = in.nextInt();  }  }  System.out.print("Matriks B");  for ( b = 0 ; b < baris ; b++ ){  System.out.print("\n");  for ( k = 0 ; k < kolom ; k++ ){  System.out.print(" " + B[b][k]);  }  }    for ( b = 0 ; b < baris ; b++){  for ( k = 0 ; k < kolom ; k++){  jumlah[b][k] = A[b][k] + B[b][k];  }  }    System.out.println("\nHasil penjumlahan matriks");  System.out.println("=========================");  for ( b = 0 ; b < baris ; b++ ){  System.out.print("\n");  for ( k = 0 ; k < kolom ; k++ ){  System.out.print(" " + jumlah[b][k] + "\t");  }  }  }  } |

1. Hasil program

|  |
| --- |
|  |

Gambar 1.7Hasil program penjumlahan *array* dua dimensi

Pada **Gambar 1.7**, dapat dilihat hasil dari program penjumlahan matriks yang yang dibuat menggunakan bahasa Java dimana pengguna mula-mula diminta untuk memasukkan ordo matriks berupa jumlah bari dan jumlah kolom matriks. Setelah memasukkan ordo matriks, dilanjutkan dengan memasukkan nilai-nilai ke dalam matriks kemudian menekan tombol *enter* pada *keyboard* dan dapat dilihat hasil penjumlahan matriks yang sudah dibuat dengan menggunakan bahasa Java.

* + 1. **Program Narto Saskeh**

1. Algoritma
2. Deklarasi nama *class*.
3. Deklarasikan *Private Node head* dan *numNode* bertipe data *integer* yang bersifat statis.
4. Deklarasikan *Public* *class* berisi *method*-*method* yang akan digunakan.
5. Menyiapkan kondisi awal barisan yang berjumlah 20 orang.
6. Menampilkan barisan dimana Narto berada pada barisan ketiga dan Saskeh berada pada barisan kedua.
7. Menampilkan kondisi barisan yang berjumlah 18 dengan menghapus orang pertama dan yang terakhir.
8. Menampilkan kondisi barisan yang terdiri dari 2 kubu, masing-masing berjumlah 9.
9. Menampilkan barisan dimana terdapat angota kubu Narto pindah ke kubu Saskeh begitupun sebaliknya.
10. Menampilkan kondisi barisan dimana kedua kubu bersatu kembali.
11. *Source code*

|  |
| --- |
| public class NartoSaskeh {  public Node head,tail;  int size=0;  public boolean isEmpty(){  return head==null;  System.out.println("Barisan kosong");  }    public void addfrist(int D1){  Node ace= new Node(D1);  if (isEmpty()){  head=tail=ace;  }  else {  ace.next=head;  head=ace;  }  size++;  }  public void addlast(int D1){  Node ace= new Node (D1);  if (isEmpty()){  head=tail=ace;  }  else {  tail.next=ace;  tail=ace;  }  size++;  }  public void addafter(int bru , int before){  if (isEmpty()){  System.out.println("Maaf, List kosong. Coba lagi setelah anda memasukkan data");  addfrist(bru);  }  else {  Node current=head;  while (current !=null){  if(current.data== before){break;}  current=current.next;}  if (current!= null){  Node ace=new Node (before);  ace.next=current.next;  current.next=ace;  }  else {  addlast(before);  }  }  size++;  }  public void removeData(int hapusData){  if (isEmpty()){  System.out.println("Maaf, List kosong. Coba lagi setelah anda memasukkan data");  }  else {  Node current=head;  Node temp=null;  while (current.next!=null ){  if (current.next!=null && current.next.data==hapusData){  temp=current.next;  break;  }  current=current.next;  }  if (temp==null){  System.out.println("Data yang ingin dihapus tidak valid");  }  else if (temp.data==tail.data){  tail=current;  tail.next=null;  }  else {  current.next=temp.next;  temp=null;  }  }  size--;  }  public void removeFirst(){  Node current=head;  head=current.next;  current=null;  size--;  }  public void display(){  Node current=head;  while (current != null){  System.out.print(current.data+ " ");  current =current.next;  }  }  public void tuker(Node ace){  Node current = head;  while (current.next!=tail){ current= current.next;}  current.next=ace;  ace.next=null;  tail=ace;  }  }  class Node{  public int data;  public Node next;  public Node(int a){  data=a;  next=null;  }  public static void main(String[]args){  MethodNarto rombongan = new MethodNarto();  MethodNarto genap= new MethodNarto();  MethodNarto ganjil= new MethodNarto();  int a=20;    System.out.println("Banyak Rombongan: ");  for (int i=1; i<=a; i++)  {rombongan.addlast(i);}  rombongan.display();  System.out.println(" ");  System.out.println("Hilang satu paling depan :");  rombongan.removeFirst();  rombongan.display();  System.out.println(" ");  System.out.println("Hilang satu paling belakang: ");  rombongan.removeData(a);  rombongan.display();  System.out.println(" ");  for (int j=2 ; j<=19; j++){  if (j%2==0){  genap.addlast(j);  }  else {  ganjil.addlast(j);}  }    System.out.println("Rombongan genap : ");  genap.display();  System.out.println(" ");  System.out.println("Rombongan ganjil : ");  ganjil.display();  System.out.println(" ");    System.out.println(" Rombongan yang tertukar : ");  Node saya = genap.tail;  Node saya1= ganjil.tail;  genap.tuker(saya1);  ganjil.tuker(saya);  genap.display();  System.out.println(" ");  ganjil.display();  System.out.println(" ");  System.out.println("Rombongan kembali: ");  ganjil.addlast(1);  ganjil.addlast(20);  ganjil.tail.next=genap.head;  ganjil.display();  }  } |

1. Hasil program

|  |
| --- |
|  |

Gambar 1.8Hasil program Narto Saskeh

Pada **Gambar 1.8**, dapat dilihat rombongan dari Narto dan Saskeh yang kemudian berpisah menjadi dua kubu dimana kedua kubu tersebut terbagi atas urutan angka genap dan ganjil. Kedua kubu yang terpisah tersebut kehilangan masing-masing salah satu anggotanya yang kemudian masuk ke kubu berlawanan tetapi kemudian kembali menjadi satu rombongan.

1. **ANALISA**
2. **Program penjumlahan *array* dua dimensi**

|  |
| --- |
| import java.util.Scanner; |

Potongan kode“import java.util.Scanner;” berfungsi agar kelas *Scanner* pada *source code* bisa digunakan atau dijalankan.

|  |
| --- |
| Public class JumlahMatriks { |

Potongan kode“Public class JumlahMatriks{” berfungsi membuat kelas dengan nama “JumlahMatriks” dimana kelas ini nantinya berisikan *method-method* yang akan digunakan untuk membuat program.

|  |
| --- |
| public static void main(String[] args){ |

Potongan kode“public static void main(String[] args){” merupakan *method* utama dalam program atau dapat dikatakan *method* ini berfungsi sebagai otak pada program. Semua *statement* yang berada dalam fungsi ini yang akan diproses untuk menghasilkan keluaran.

|  |
| --- |
| Scanner x = new Scanner(System.in);  int baris, kolom, b, k;  System.out.print("Banyak baris: ");  baris = x.nextInt();  System.out.print("Banyak kolom: ");  kolom = x.nextInt(); |

Potongan kode“Scanner x = new Scanner(System.in); “ berfungsi membuat objek untuk memproses *input* suatu nilai dengan menggunakan fungsi *Scanner* akan meminta *input* dari *user* yang nantinya nilai dari *input* tersebut akan mengisi variabel “baris” dan “kolom”.

|  |
| --- |
| int A[][] = new int[baris][kolom];  int B[][] = new int[baris][kolom];  int jumlah[][] = new int[baris][kolom]; |

Potongan kode“int A[][] = new int[baris][kolom];” adalah baris pendeklarasian *array* bertipe data *integer* dengan indeks “baris” dan “kolom” untuk tiap dimensinya. Potongan kode “int B[][] = new int[baris][kolom];” merupakan pendeklarasian array bertipe data *integer* dengan indeks “baris” dan “kolom” untuk tiap dimensinya. Potongan kode“int jumlah[][]= new int[baris][kolom];” merupakan pendeklarasian *array* bertipe data *integer* dengan indeks “baris” dan “kolom” untuk tiap dimensinya.

|  |
| --- |
| System.out.print("\n\nMasukkan angka-angka ke dalam matriks A");  System.out.print("\n=======================\n");  for ( b = 0 ; b < baris ; b++ ){  for ( k = 0 ; k < kolom ; k++ ){  System.out.print("[" +(b+1)+ "][" +(k+1)+ "]:");  A[b][k] = in.nextInt();  }  } |

Potongan kode“for ( b = 0 ; b < baris ; b++ ){” berfungsi untuk melakukan perulangan atau *looping* menggunakan fungsi “for” yang akan terus berulang selama “b < baris” dan juga terdapat fungsi “for” lain di dalamnya berupa “for ( k = 0 ; k < kolom ; k++ ){”, yang akan terus berulang selama “k < kolom”. Pada fungsi tersebut perulangan yang dilakukan adalah menampilkan nilai dari variabel “b” dan “k” pada layar dan juga proses inisialisasi pada variabel “A[b][k] = in.nextInt();”.

|  |
| --- |
| System.out.print("Matriks A");  for ( b = 0 ; b < baris ; b++ ){  System.out.print("\n");  for ( k = 0 ; k < kolom ; k++ ){  System.out.print(" " + A[b][k]);  }  } |

Potongan kode “System.out.print("Matriks A"); “ berfungsi untuk menampilkan isi dari matriks A pada layar dengan perulangan atau *looping* menggunakan fungsi “for” yang akan terus berulang selama “b < baris” dan juga terdapat fungsi “for” lain di dalamnya, akan terus berulang selama “k < kolom”.

|  |
| --- |
| System.out.print("\n\nMasukkan angka-angka ke dalam matriks B");  System.out.print("\n=======================\n");  for ( b = 0 ; b < baris ; b++ ){  for ( k = 0 ; k < kolom ; k++ ){  System.out.print("[" +(b+1)+ "][" +(k+1)+ "]:");  B[b][k] = in.nextInt();  }  } |

Potongan kode “System.out.print("\n\nMasukkan angka-angka ke dalam matriks B");” berfungsi untuk melakukan perulangan atau *looping* menggunakan fungsi “for” yang akan terus berulang selama “b < baris” dan juga terdapat fungsi “for” lain di dalamnya, akan terus berulang selama “k < kolom”. Pada fungsi tersebut perulangan yang dilakukan adalah menampilkan nilai dari variabel “i” dan “j” pada layar dan juga proses inisialisasi pada variabel “B[b][k] = in.nextInt();”.

|  |
| --- |
| System.out.print("Matriks B");  for ( b = 0 ; b < baris ; b++ ){  System.out.print("\n");  for ( k = 0 ; k < kolom ; k++ ){  System.out.print(" " + B[b][k]);  }  } |

Potongan kode yang diawali “System.out.print("Matriks B");”berfungsi untuk menampilkan isi dari matriks B pada layar dengan perulangan atau *looping* menggunakan fungsi “for” yang akan berhenti jika “b < baris” dan juga terdapat fungsi “for” lain di dalamnya, akan berhenti jika “k < kolom”.

|  |
| --- |
| for ( i = 0 ; i < baris ; i++){  for ( j = 0 ; j < kolom ; j++){  jumlah[i][j] = A[b][k] + B[b][k];  }  } |

Potongan kode “jumlah[i][j] = A[b][k] + B[b][k];” berfungsi untuk melakukan operasi penjumlahan matriks dengan bantuan *nested for* untuk melakukan perulangan agar penjumlahan yang dilakukan sesuai dengan indeks.

|  |
| --- |
| System.out.println("\nHasil penjumlahan matriks");  System.out.println("=========================");  for ( b = 0 ; b < baris ; b++ ){  System.out.print("\n");  for ( k = 0 ; k < kolom ; k++ ){  System.out.print(" " + jumlah[b][k] + "\t");  }  } |

Potongan kode “Hasil penjumlahan matriks” berfungsi untuk menampilkan tampilan pada layar lalu perulangan atau *looping* menggunakan fungsi “for” yang akan terus berulang selama “b < baris” dan juga terdapat fungsi “for” lain di dalamnya, akan terus berulang selama “k < kolom”. Dan menampilkan nilai yang ada pada *array* “jumlah[b][k]”.

1. **Program Narto Saskeh**

|  |
| --- |
| Public class NartoSaskeh { |

Potongan kode“Public class NartoSaskeh{” berfungsi membuat kelas dengan nama “NartoSaskeh” dimana kelas ini nantinya berisikan *method-method* yang akan digunakan untuk membuat program.

|  |
| --- |
| public Node head,tail;  int size=0; |

Baris “public Node head,tail;” adalah baris deklarasi suatu simpul untuk menunjuk awal dan akhir dari suatu *linked list*, sehingga kita bisa membuat *linked list* melintas atau memulai dari “head” yang direpresentasikan sebagai awal *list*, dan berakhir di “head” yang menjadi akhir dari *list*. “size” adalah suatu variabel yang menyatakan ukuran atau panjang *list* yang terbentuk setelah melalui beberapa *method*.

|  |
| --- |
| public boolean isEmpty(){  return head==null;  System.out.println("Barisan kosong");  } |

Potongankode “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.

|  |
| --- |
| public void addfrist(int D1){  Node ace= new Node(D1);  if (isEmpty()){  head=tail=ace;  }  else {  ace.next=head;  head=ace;  }  size++;  } |

Potongan kode “public void addfrist(int D1){” adalah suatu *method* yang berfungsi menambahkan data baru di awal *list* tetapi harus diberikan kondisi terlebih dahulu, yaitu kondisi jika *list* kosong maka “head” dan “tail” akan diisikan oleh data “ace”, kondisi lainnya yaitu jika *list* tidak kosong maka “ace” akan menjadi “head” dari *list*.

|  |
| --- |
| public void addlast(int D1){  Node ace= new Node (D1);  if (isEmpty()){  head=tail=ace;  }  else {  tail.next=ace;  tail=ace;  }  size++;  } |

Potongan kode “public void addlast(int D1){” 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 “ace”, kondisi lainnya yaitu jika *list* tidak kosong maka “ace” akan menjadi “tail” dari *list*.

|  |
| --- |
| public void addafter(int bru , int before){  if (isEmpty()){  System.out.println("Maaf, List kosong. Coba lagi setelah anda memasukkan data");  addfrist(bru);  }  else {  Node current=head;  while (current !=null){  if(current.data== before){break;}  current=current.next;}  if (current!= null){  Node ace=new Node (before);  ace.next=current.next;  current.next=ace;  }  else {  addlast(before);  }  }  size++;  } |

Potongan kode “public void addafter(int bru , int before){...” adalah suatu *method* yang berfungsi menambahkan data baru di tengah-tengah *list* dengan menggunakan kondisi-kondisi *if* untuk memeriksa apakah data bisa ditambahkan melalui parameter-parameter bertipe data *integer* “bru” dan “before” untuk menunjuk data baru yang ingin ditambahkan ke tempat sebelum data lama yang sudah ada. Terdapat juga variabel “size++” yang nilainya akan bertambah jika terdapat data baru yang dimasukkan.

|  |
| --- |
| public void removeData(int hapusData){  if (isEmpty()){  System.out.println("Maaf, List kosong. Coba lagi setelah anda memasukkan data");  }  else {  Node current=head;  Node temp=null;  while (current.next!=null ){  if (current.next!=null && current.next.data==hapusData){  temp=current.next;  break;  }  current=current.next;  }  if (temp==null){  System.out.println("Data yang ingin dihapus tidak valid");  }  else if (temp.data==tail.data){  tail=current;  tail.next=null;  }  else {  current.next=temp.next;  temp=null;  }  }  size--;  } |

Potongan kode “public void removeData(int hapusData){...” adalah suatu *method* yang berfungsi menghapus data baru di tengah-tengah *list* dengan menggunakan kondisi-kondisi *if* untuk memeriksa apakah data bisa dihapus melalui parameter-parameter bertipe data *integer* “hapusData” untuk menunjuk data yang ingin dihapus. Terdapat juga variabel “size--” yang nilainya akan berkurang jika terdapat data yang dihapus.

|  |
| --- |
| public void removeFirst(){  Node current=head;  head=current.next;  current=null;  size--;  } |

Potongan kode “public void removefrist(){” adalah suatu *method* yang berfungsi menghapus data di awal, setelah data di awal terhapus maka “head” akan diisikan oleh data “current” yang merupakan data yang ada setelah “head”.

|  |
| --- |
| public int size (){  return size;  } |

Potongan kode “public int size(){” digunakan agar variabel “size” dapat dipanggil di semua *method* atau kelas yang membutuhkan.

|  |
| --- |
| class Node{  public int data;  public Node next;  public Node(int a){  data=a;  next=null;  }  } |

“class Node{...” adalah sebuah kelas yang berisi vaiabel-variabel dan *node-node* yang digunakan untuk menjalankan *­method*-*method* yang ada di dalam program.

|  |
| --- |
| public static void main(String[]args){  MethodNarto rombongan = new MethodNarto();  MethodNarto genap= new MethodNarto();  MethodNarto ganjil= new MethodNarto();  int a=20; |

“public static void main(String[]args){...” adalah fungsi utama dari program Narto Saskeh ini dimana *method*-*method* yang sudah dibuat sebelumnya akan dijalankan untuk menghasilkan tampilan sesuai dengan permasalahan yang diberikan.

|  |
| --- |
| System.out.println("Banyak Rombongan: ");  for (int i=1; i<=a; i++)  {rombongan.addlast(i);}  rombongan.display();  System.out.println(" "); |

Potongan kode “System.out.println("Banyak Rombongan: ");...” adalah potongan kode yang akan menampilkan semua anggota dari rombongan Narto dan Saskeh mulai dari orang pertama sampai orang terakhir saaat masih lengkap. Proses menampilkan rombongan menggunakan *method* “addlast” karena rombongan memiliki susunan yang urut dari 1 – 20, sehingga menggunakan *method* “addlast” merupakan pilihan yang lebih efisien karena hanya perlu mengganti “tail” dari *linked list* yang dalam hal ini adalah barisan rombongan.

|  |
| --- |
| System.out.println("Hilang satu paling depan :");  rombongan.removeFirst();  rombongan.display();  System.out.println(" "); |

Potongan kode “System.out.println("Hilang satu paling depan : ");...” adalah potongan kode yang akan menampilkan barisan dari rombongan Narto dan Saskeh setelah orang paling depan dalam barisan menghilang menggunakan *method* “removeFirst()” karena yang menghilang atau dihapus dari barisan merupakan “head” dari barisan rombongan tersebut.

|  |
| --- |
| System.out.println("Hilang satu paling belakang: ");  rombongan.removeData(a);  rombongan.display();  System.out.println(" "); |

Potongan kode “System.out.println("Hilang satu paling belakang: ");...” adalah potongan kode yang akan menampilkan barisan dari rombongan Narto dan Saskeh setelah orang paling depan dan paling belakang dalam barisan menghilang menggunakan *method* “removeData(a)” metode ini digunakan karena parameter yang ada di dalam *method* sudah dideklarasikan terlebih dahulu.

|  |
| --- |
| for (int j=2 ; j<=19; j++){  if (j%2==0){  genap.addlast(j);  else {  ganjil.addlast(j);}  } |

Perulangan “for (int j=2 ; j<=19; j++){...” berfungsi untuk menampilkan data node yang akan berpisah sehingga terbagi menjadi dua bagian yakni data node ganjil dan genap. Perulangan digunakan untuk memisahkan anggota rombongan yang genap dan yang ganjil dibantu dengan menggunakan *statement control if* yang memiliki kondisi “j%2==0” akan menjadi bagian dari rombongan genap, sementara anggota rombongan yang tidak sesuai dengan kondisi genap, akan masuk ke kondisi lainnya dan menjadi anggota rombongan ganjil.

|  |
| --- |
| System.out.println("Rombongan genap : ");  genap.display();  System.out.println(" ");  System.out.println("Rombongan ganjil : ");  ganjil.display();  System.out.println(" "); |

Potongan kode “System.out.println("Rombongan genap : ");” dan “System.out.println("Rombongan genap : ");” adalah potongan kode yang menampilkan keterangan dari rombongan genap dan rombongan ganjil yang sudah terpisah sebelumnya dengan memanggil *method* “genap.display();” dan “ganjil.display();” yang sudah dijalankan sebelumnya.

|  |
| --- |
| System.out.println(" Rombongan yang tertukar : ");  Node saya = genap.tail;  Node saya1= ganjil.tail;  genap.tuker(saya1);  ganjil.tuker(saya);  genap.display();  System.out.println(" ");  ganjil.display();  System.out.println(" "); |

Potongan kode “System.out.println(" Rombongan yang tertukar : ");” adalah potongan kode yang menampilkan kedua rombongan ganjil dan genap setelah anggota-anggota yang hilang masuk ke rombongan yang tidak sesuai dengan urutannya saat berada di situasi awal barisan dan tertukar. Proses penukaran anggota menggunakan *method* “genap.tuker(saya1);” dan “ganjil.tuker(saya);” dimana “saya1” dan “saya” adalah *node* yang berisi data “tail” dari masing-masing rombongan.

|  |
| --- |
| System.out.println("Rombongan kembali: ");  ganjil.addlast(1);  ganjil.addlast(20);  ganjil.tail.next=genap.head;  ganjil.display(); |

Potongan kode “System.out.println("Rombongan kembali: ");...” adalah potongan kode yang akan menampilkan semua anggota dari rombongan Narto dan Saskeh yang bersatu kembali. Proses menampilkan rombongan menggunakan *method* “addlast” yang digunakan pada *node* “ganjil” karena urutan angka dimulai dari angka ganjil walaupun itu semua terserah pada *programmer* untuk menentukan urutan. *Method* “ganjil.addlast” akan menambahkan data baru pada “tail” *node* “ganjil”, dan *node* “genap” akan menjadi “tail” dari node “ganjil”.

* 1. **KESIMPULAN**

Berdasarkan praktikum yang telah dilakukan, dapat disimpulkan bahwa:

1. *Array* pada struktur data dapat menyimpan banyak data yang tempat penyimpanannya berurutan sehingga bisa diakses menggunakan indeks.
2. *Linked list* pada struktur data yang di mana penyimpanannya tidak berurutan, melainkan dihubungkan dengan tiap alamat *node* selanjutnya.
3. *Single linked list* diimplementasikan dalam bahasa pemrograman Java menggunakan *class*. Setiap *node* dalam *linked list* memiliki tempat penyimpanan data dan tempat penyimpanan alamat *node* selanjutnya. Untuk melakukan operasi-operasi terhadap *linked list*, perlu dibuat metode-metode tertentu, seperti “add()”, “deletefirst()”, “deletelast()”, dan “display()”.