**MODUL III**

**STACK DAN QUEUE**

1. **TUJUAN**

Tujuan dari praktikum ini adalah:

1. Mahasiswa memahami konsep dari *stack* dan *queue*.
2. Mahasiswa dapat mengimplementasikan *stack* dan *queue* menggunakan *double linked list* dalam program.
3. **DASAR TEORI**

**3.2.1 *Stack***

*Stack* atau tumpukan adalah salah satu konsep struktur data yang memiliki sistem kerja yang terakhir masuk adalah yang pertama keluar (LIFO=*last in first out* ) [1].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | E | | D | | C | | B | | A | |

Gambar 3.1 Ilustrasi *stack*

Ada 2 operasi dasar yang bisa dilaksanakan pada *stack*, yaitu operasi menyisipkan data (*push*) dan menghapus data (*pop*).

1. Operasi *push*

Perintah *push* digunakan untuk memasukan data ke dalam *stack*. Misalkan kita mempunyai data-data 3, 10, 16, dalam *stack*, ketika kita ingin memasukan data 9, maka 9 diletakkan dibagian atas tumpukan [2].

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 9 | | 16 | | 10 | | 3 |  |  | | --- | | 16 | | 10 | | 3 |   *Push* 9 |

Gambar 3.2 Ilustrasi *push*

1. Operasi *pop*

Operasi *pop* adalah operasi untuk menghapus elemen yang terletak pada posisi paling atas dari sebuah *stack* [2].

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | 16 | | 10 | | 3 |  |  | | --- | | 9 | | 16 | | 10 | | 3 |   *Pop* 9 |

Gambar 3.3Ilustrasi *pop*

Adapun algoritma dari operasi-operasi pada *stack* yaitu sebagai berikut:

1. Algoritma operasi *push*:
   1. Tentukan kondisi tumpukan, dimana kondisi tumpukan masih kosong.
   2. Deklarasikan *node* baru atau data yang akan dimasukkan ke dalam tumpukan (variabel n).
   3. n adalah data pertama yang ada dalam *stack* atau tumpukan tersebut.
   4. Masukkan data baru.
   5. Posisi n tidak yang terakhir, tetapi digantikan oleh T, begitu seterusnya kalau ada atau *node* baru yang dimasukkan.
2. Algoritma operasi *pop*:
3. Tentukan kondisi tumpukan, dimana kondisi tumpukan sudah ada isinya.
4. Jika *stack* kosong, kerjakan langkah 5 jika tidak kerjakan langkah 3.
5. Ambil *node* yang paling atas.
6. Misal data atau *node* ada 2, maka posisi *node* terakhir bukan lagi *node* 2, tetapi *node* pertama.
7. Selesai.

**3.2.2 *Queue***

*Queue* adalah suatu kumpulan data yang penambahan elemennya hanya bisa dilakukan pada suatu ujung (disebut dengan sisi belakang atau *rear*), dan penghapusan atau pengambilan elemen dilakukan lewat ujung yang lain (disebut dengan sisi depan atau *front*) [1].

* + - 1. Implementasi *queue* dengan *array*

Karena *queue* merupakan suatu kumpulan data, maka tipe data yang sesuai untuk menyajikan *queue* adalah menggunakan *array* atau *list*.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | Keluar | A | B | C | D | E | Masuk | |

Gambar 3.4 Ilustrasi *queue*

*Queue* di atas berisi lima elemen yaitu A,B,C,D,E. Elemen A terletak dibagian depan dan E terletak dibagian belakang. Jika ada elemen baru yang masuk maka elemen tersebut diletakkan di sebelah kanan E. Dan jika ada elemen yang dihapus, maka A dihapus terlebih dahulu [1].

1. Implementasi *queue* dengan *pointer*

Keterbatasan pengimplementasian *queue* dengan menggunakan *array* memiliki keterbatasan antara lain, jumlah elemen yang bisa diisi dalam *queue* terbatas pada ukuran *array* yang digunakan. Untuk memanipulasi *queue* bisa digunakan dua buah variabel yang menyimpan posisi elemen paling depan dan elemen paling belakang. *Queue* diimplementasikan senarai berantai (*linked list*) menggunakan dua buah *pointer* yang ditempatkan pada elemen paling depan atau *head* dan elemen paling belakang atau *tail* [1]. Operasi yang dilakukan pada *queue* adalah memasukkan elemen dari sisi bagian belakang *queue* dan mengeluarkan elemen dari sisi depan *queue* [2].

Adapun algoritma dari operasi-operasi pada *stack* yaitu sebagai berikut:

1. Algoritma memasukkan elemen ke dalam *queue*:
2. Kondisi awal sebuah *queue* dan dibuat sebuah elemen baru yang akan dimasukkan ke *queue*.
3. Elemen baru dimasukkan ke dalam *queue.*
4. Petunjuk *last* diubah menunnjukan ke elemen baru sebagai elemen paling belakang.
5. Algoritma mengeluarkan elemen dari dalam *queue*:
6. Kondisi awal ada sebuah *queue* yang berisi elemen.
7. Petunjuk *first* diubah ke elemen di belakang elemen yang paling depan.
8. Elemen paling depan dikeluarkan dari elemen.
9. **PERMASALAHAN**
10. Sebuah bandara memiliki 2 buah hanggar (A dan B) berkapasitas masing masing 5 pesawat untuk memarkir pesawat yang tidak sedang beroperasi. Hanggar hanggar tersebut memiliki dinding yang sempit dan memiliki satu pintu, sehingga pengaturan pesawat keluar dan masuk menggunakan sistem LIFO. Pesawat yang tidak sedang terbang harus berada di dalam hanggar. Anda adalah petugas yang mengatur keberangkatan pesawat, dan mencatat jadwal dalam sebuah *linked list*. Tampilkan setiap proses perpindahan pesawat. Berikut adalah nama dan kode masing-masing pesawat

|  |  |
| --- | --- |
| Kode | Nama |
| 123 | Wantutri |
| 737 | Seventriseven |
| 666 | Sixsixsix |
| 369 | Trisixnain |
| 400 | Fourzerozero |
| 186 | Waneightsix |
| 648 | Sixfoureight |
| 221 | tutuwan |

1. Pesawat 123, 666, 737, 369 berada di hanggar A.
2. Pesawat 648, 400, berada di hanggar B.
3. Pesawat 186, 221 sedang terbang.
4. Pukul 14.00 Pesawat 666 lepas landas.
5. Pukul 15.00 Pesawat 648 lepas landas.
6. Pukul 16.00 Pesawat 186 masuk hanggar B.
7. Pukul 21.00 Pesawat 666 masuk hanggar B.
8. Pukul 22.00 Pesawat 221 masuk hanggar A.
9. Pukul 22.22 Pesawat 648 hilang kontak.
10. Beberapa orang *demigod* membuat barisan berdasarkan abjad ketika menunggu dipanggil sebagai berikut:

Jason(1) - Meg (4) - Piper (10) - Thalia (1)

Ketika Jason sudah dipanggil dan membentuk barisan baru yaitu barisan orang-orang yang sudah dipanggil, beberapa orang terlambat datang dan masuk ke dalam barisan yaitu:

Percy(3) - Will (7) - Connor (12) - Travis (12) Nico (13)

Setelah kedatangan beberapa orang tersebut, aturan diubah yaitu yang memiliki nomor kabin terkecil dipanggil terlebih dahulu. Sehingga ketika *demigod* dengan nomor kabin terkecil berada di tengah, *demigod* lain membentuk barisan baru dengan urutan yang sama, dengan cara FIFO sehingga *demigod* yang di tengah dapat dikeluarkan (dengan cara FIFO juga), Setelah semua *demigod* terpanggil, tampilkan barisan yang terbentuk.

1. Ubahlah pernyataan berikut: (A + B) \* C - (D - E) \* (F + G) ke dalam bentuk *prefix* dan *postfix*.
2. **HASIL DAN PEMBAHASAN**
3. **Program *Stack* Pesawat**
4. Algoritma
5. Membuat Fungsi Penambahan Data.
   * + 1. Memberikan parameter sebuah data kepada *method*.
       2. Memeriksa apakah tumpukan kosong atau tidak, jika kosong maka *head* dan *tail* dari tumpukan adalah data yang baru.
       3. Jika tidak kosong, maka data baru dijadikan *head* dari tumpukan.
6. Membuat Fungsi Penghapusan Data.
7. Membuat sebuah variabel bantu untuk menyimpan *head* dari tumpukan.
8. Melewatkan *head* dari tumpukan dengan cara mengganti *head* dengan elemen yang ada setelahnya.
9. Mengubah *head* menjadi *null*.
10. Membuat Fungsi Menampilkann Data.
11. Megecek apakah *linked list* kosong atau tidak.
12. Jika *linked list* kosong, memberi keterangan bahwa *linked list* kosong.
13. Menampilkan data dari *linked list*, jika *linked list* tidak kosong.
14. Melakukan perulangan unutk menampilkan semua data pada *linked list* hingga data terakhir.
15. Menambahkan data dengan menggunakan *method*  penambahan data.
16. Menampilkan data yang telah dimasukkan dengan menggunakan *method*  untuk menampilkan data.
17. Menghapus data yang ingin dihapus dengan menggunakan *method* pengurangan data.
18. Menampilkan kembali data setelah melakukan penghapusan data.
19. *Source code*

|  |
| --- |
| package PRAKTIKUM;  import java.util.Scanner;  public class JURNAL\_1 {  class Node {  public int data;  String nama;  public Node next;  public Node(int d,String nama) {  data = d;  this.nama=nama;  next = null;  }  }  public Node head, tail;  int size = 0;    public boolean isEmpty() {  return head == null;  }  public void Push(int newData,String nama) {  Node newNode = new Node(newData,nama);  if (isEmpty()) {  head = tail = newNode;  }else {  newNode.next = head;  head = newNode;  }  size++;  }  public void Pop() {  Node temp = head;  head= head.next;  temp=null;  }  void list(){  System.out.print(head.data + " " + head.nama + " Lepas Landas ");  }  public void display() {  Node current = head;  while (current != null) {  System.out.print(current.data + " ");  System.out.print(current.nama+ " " );  current = current.next;  }  }  public static void main(String[] args) {  JURNAL\_1 list =new JURNAL\_1();  JURNAL\_1 list1 =new JURNAL\_1();  JURNAL\_1 list2 =new JURNAL\_1();  System.out.println("Hanggar A");  list.Push(123,"wantutri");  list.Push(737,"seventriseven");  list.Push(369,"trisixnine");  list.Push(666,"sixsixsix");  list.display();  System.out.println("");  System.out.println("Hanggar B");  list1.Push(400,"fourzerozero");  list1.Push(648,"sixfoureight");  list1.display();  System.out.println("");  System.out.println("yang sedang terbang");  System.out.println("Pukul 14.00 : ");  list.list();  list.Pop();  System.out.println();  System.out.println("Pukul 15.00 : ");  list1.list();  list1.Pop();  System.out.println();  System.out.println("Pukul 16.00 Pesawat 186 Masuk hanggar B ");  list1.Push(186,"waneightsix");  System.out.println("Pukul 16.00 Pesawat 666 Masuk hanggar B ");  list1.Push(666,"sixsixsix");  System.out.println("Pukul 16.00 Pesawat 221 Masuk hanggar A ");  list.Push(221,"tutuwan");  System.out.println("Pesawat Pada Hanggar A : ");  list.display();  System.out.println();  System.out.println("Pesawat Pada Hanggar B : ");  list1.display();  System.out.println();  }  } |

1. Hasil program

|  |
| --- |
| X:\Kampus\Kuliah Angga\Semester 3\Praktikum\Algodat\Modul 3 Stack & Queue\WhatsApp Image 2019-09-30 at 14.51.51.jpeg |

Gambar 3.5 Hasil *run* program *stack* pesawat

Pada **Gambar 3.5** dapat dilihatprogram yang menampilkan sebuah *stack* pada hanggar yang menampilkan data setiap pesawat yang ada pada setiap hanggar, posisi pesawat pada setiap hanggar akan berubah setiap ada keberangkatan dan kedatangan pesawat.

1. **Program Pengurutan Barisan Berdasarkan Abjad**
2. Algoritma
3. Membuat Operasi Menambahkan Data.
4. Menentukan kondisi antrian, apakah antrian dalam keadaan kosong atau tidak.
5. Jika kosong maka mendeklarasikan data baru yang akan dimasukkan ke dalam antrian.
6. Melakukan perulangan untuk memasukkan data antrian dengan cara menambahkan data di belakang data sebelumnya.
7. Membuat Fungsi Penghapusan Data.
   * + 1. Pembuatan variabel bantu untuk memudahkan menentukan *front linked list*.
       2. Melakukan pemindahan letak *front* dari *linked list*.
       3. *Front* dikembalikan kekelas *orang*.
8. Membuat *method* mengurutkan dari kabin terkecil.
9. Melakukan pengecekan antrian, apakah data sudah ada atau belum (data masih kosong).
10. Melakukan perulangan untuk menentukan letak data baru.
11. Saat data kabin pertama lebih besar dari data kabin baru maka letak kabin baru didepan kabin pertama.
12. Saat data kabin terakhir lebih kecil atau sama dengan data kabin baru maka data kabin baru menjadi data terakhir.
13. Membuat Fungsi Menampilkann Data.
14. Memberikan penanda pada *head*.
15. Megecek apakah *linked list* kosong atau tidak.
16. Jika *linked list* kosong, memberi keterangan bahwa *linked list* kosong.
17. Menampilkan data dari *linked list*, jika *linked list* tidak kosong.
18. Melakukan perulangan menampilkan data penanda *head*.
19. Menampilkan data yang telah dimasukkan dengan menggunakan *method*  untuk menampilkan data.
20. Menghapus data yang ingin dihapus dengan menggunakan *method* pengurangan data.
21. Menampilkan data dari data kabin yang terkecil dengan mengguankan perulangan *for* yang ada pada *method prior*.
22. *Source code*

|  |
| --- |
| class Orang{  String nama;  int kabin;  Orang next;  public Orang (String nama,int kabin){  this.nama=nama;  this.kabin=kabin;  }  }  class Queue{  Orang front,rear;  void enqueue(String nama, int kabin){  Orang baru = new Orang(nama,kabin);  if(front==null){  front=rear=baru;  } else {  rear.next = baru;  rear = baru;  }  }  public Orang dequeue(){  Orang temp = front;  front= front.next;  return temp;  }  public void prior(String nama, int kabin) {  Orang baru = new Orang(nama, kabin);  if (front==null) {  front = rear = baru;  } else {  Orang cur = front;  while (cur != null) {  // penambahan data bagian depan  if (front.kabin > baru.kabin) {  baru.next = front;  front = baru;  break;  // penambahan data dibelakang  } else if (rear.kabin <= baru.kabin) {  rear.next = baru;  rear = baru;  break;  // penyisipan ditengah-tengah  } else {  if (baru.kabin >= cur.kabin && cur.next.kabin > baru.kabin) {  baru.next = cur.next;  cur.next = baru;  break;  }  }  cur = cur.next;  }  }  }  public void display() {  Orang current = front;  while (current != null) {  System.out.print(current.nama + "(" + current.kabin + ") ");  current = current.next;  }  }  }  public class BarisPrioritas {  public static void main(String[] args) {  Queue baris1 = new Queue();  Queue baris2 = new Queue();  Queue baris3 = new Queue();  baris1.enqueue("Jason",1);  baris1.enqueue("Meg",4);  baris1.enqueue("Piper",10);  baris1.enqueue("Thalia",1);  baris1.display();  System.out.println();  System.out.println("Jason dipanggil");  baris1.dequeue();  baris1.display();  System.out.println();  System.out.println("Barisan Telat");  baris2.enqueue("Percy",3);  baris2.enqueue("Will",7);  baris2.enqueue("Connor",12);  baris2.enqueue("Travis",12);  baris2.enqueue("Nico",13);  baris2.display();  System.out.println();  Orang tampung = new Orang(" ", 0);  for (int i=1; i<6;i++){  tampung=baris2.dequeue();  baris1.enqueue(tampung.nama, tampung.kabin);  }  System.out.println();  System.out.println("Barisan Belum dipanggil");  baris1.display();  System.out.println();  for (int i=1; i<9;i++){  tampung=baris1.dequeue();  baris2.prior(tampung.nama, tampung.kabin);  }  System.out.println("Barisan Setelah diurutkan Sesuai Nomor Kabin Terkecil");  baris2.display();}} |

1. Hasil program

|  |
| --- |
| X:\Kampus\Kuliah Angga\Semester 3\Praktikum\Algodat\Modul 3 Stack & Queue\WhatsApp Image 2019-09-30 at 14.51.52.jpeg |

Gambar 3.6 Hasil *run* program barisan berdasarkan abjad

Pada **Gambar 3.6** dapat dilihatprogram barisan berdasarkan abjad, barisan yang semulanya terdiri dari empat orang anak, yang kemudian anak pada barisan pertama dipanggil maka anak-anak yang berada dibelakangnya akan maju, kemudian barisan yang belum dipanggil akan ditempatkan didepan barisan yang telat, dan setelah itu barisan akan diurutkan berdasarkan kabin terkecil.

1. **Program Konversi Notasi**
2. Algoritma
3. Membuat *method* untuk mengubah ekspresi menjadi *postfix* dengan cara:
4. Baca notasi dari kiri, kita misalkan R.
5. Jika R *operand*, maka langsung cetak.
6. Jika R adalah ‘(‘, maka *push* R ke dalam *stack.*
7. Jika R adalah ‘)’, maka lakukan *pop* *stack* dan cetak sampai bertemu tanda ‘)’. kemudian *pop stack.*
8. Jika R adalah *operator* maka, *push* R ke tumpukan.
9. Membuat *method* untuk mengubah ekspresi menjadi *postfix* dengan cara:
10. Baca notasi dari kiri, kita misalkan R.
11. Jika R *operator*, maka langsung cetak.
12. Jika R adalah ‘(‘, maka *push* R ke dalam *stack.*
13. Jika R adalah ‘)’, maka lakukan *pop* *stack* dan cetak sampai bertemu tanda ‘)’. kemudian *pop stack.*
14. Jika R adalah *operand* maka, *push* R ke tumpukan.
15. Membuat *method* untuk menampilkan *stack* yang dalam hal ini berupa ekspresi hasil konversi ke *postfix* dan *prefix*.

2. *Source code*

|  |
| --- |
| package javaapplication7;  import java.util.Scanner;  class HmmNode {  public Object data;  public HmmNode next;  public HmmNode () {  data =' ';  }  public HmmNode (Object val) {  data = val;  }  }  public class jurnal3quasi {  private HmmNode top;  public jurnal3quasi() {  top = null;  }  public boolean empty(){  return top == null;  }  public boolean full(){  return false;  }  public void push(Object e){  HmmNode tmp = new HmmNode(e);  tmp.next = top;  top = tmp;  }  public Object pop(){  Object x = top.data;  top = top.next;  return x;  }  public Object check(){  Object x = top.data;  return x;  }  public void postfix(String x){  String output="";  jurnal3quasi S=new jurnal3quasi();  for(int i=0;i<x.length();i++){  char c=x.charAt(i);  if(c==('+')||c==('\*')||c==('-')||c==('/')){  while(!S.empty()&&prior(S.check())>=prior(c))  output+=S.pop();  S.push(c);  }  else if(c=='('){  S.push(c);  }  else if(c==')'){  while(!S.check().equals('('))  output+=S.pop();  S.pop();  }  else  output+=c;  }  while(!S.empty())  output+=S.pop();  System.out.println("Notasi Infix : "+x);  System.out.println("Notasi Postfix : "+output);  }  public void prefix(String x){  String output="";  jurnal3quasi S=new jurnal3quasi();  int i;  for(i=0;i<x.length();i++){  char c=x.charAt(i);  if(c==('+')||c==('\*')||c==('-')||c==('/')){  while(!S.empty() && prioritas(S.check()) <= prioritas(c))  output+=S.pop();  S.push(c);  }  }  while(!S.empty())  output+=S.pop();  for(i=0;i<x.length();i++){  char c=x.charAt(i);  if(c==('+')||c!=('\*')||c!=('-')||c!=('/')){  while(!S.empty() && prioritas(S.check()) >= prioritas(c))  output+=S.pop();  S.push(c);  }  else  output+=c;  if((c=='(')||(c==')')){  S.push(c);  }  if(c==('+')||c==('\*')||c==('-')||c==('/')){  S.pop();  }  }  while(!S.empty())  output+=S.pop();  System.out.println("Notasi Prefix : "+output);  }  public int prior(Object x){  if(x.equals('+')||x.equals('-'))  return 1;  else if(x.equals('\*')||x.equals('/'))  return 2;  else  return 0;  }  public int prioritas(Object x){  if(x.equals('+')||x.equals('-'))  return 2;  else if(x.equals('\*')||x.equals('/'))  return 1;  else  return 0;  }  public static void main(String args[]){  jurnal3quasi s=new jurnal3quasi();  System.out.println();  System.out.println("-----------------------------");  System.out.println("Notasi Infix: \"(A+B)\*C-(D-E)\*(F+G)\"");  System.out.println("-----------------------------");  s.postfix("(A+B)\*C-(D-E)\*(F+G)");  s.prefix("(A+B)\*C-(D-E)\*(F+G)");  System.out.println("-----------------------------");  }  } |

1. Hasil program

|  |
| --- |
|  |

Gambar 3.7 Hasil *run* program konversi nilai

Pada **Gambar 3.7** dapat dilihathasil konversi dari notasi *infix*menjadi *postfix* dan *prefix* dimana dapat terlihat pebedaan dalam notasi-notasi tersebut. Notasi *infix* meletakkan *operator* di tengah-tengah *operand*-nya, sementara notasi *postfix* meletakkan *operator* setelah *operand* dan notasi *prefix* meletakkan *operator* sebelum *operand*-nya.

* 1. **ANALISA**
     1. **Program *Stack* Pesawat**

|  |
| --- |
| public class JURNAL\_1 {  public Node head, tail;  int size = 0; |

*Script*  “public class JURNAL\_1 {” berfungsi sebagai kelas yang digunakan untuk menampung *method-method* yang ada didalamnya yang menggunakan *modifier* “public” sehingga kelas ini nantinya dapat diakses dengan mudah, dalam kelas “JURNAL\_1” dideklarasikan variabel “head” dan “tail” yang dengan *modifier* “public”, dan juga dideklarasikan variabel “size” yang bertipe data *integer* dengan nilai awal yang sama dengan “0”.

|  |
| --- |
| class Node {  public int data;  String nama;  public Node next;  public Node(int d,String nama) {  data = d;  this.nama=nama;  next = null;  }  } |

*Script* “class node{” berfungsi sebagai kelas dengan nama kelas “Node”, kelas ini memiliki variabel-variabel yang telah dideklarasikan seperti, variabel “data” dengan tipe data *integer* dan variabel “nama” dengan tipe data *string*. Kelas ini juga memiliki *constructor* yang fungsinya untuk memberikan nilai awal pada saat suatu objek dibuat, *constructor* pada kelas ini memiliki 2 buah parameter dengan tipe data *integer* dan *string* yang dimana *integer* fungsinya digunakan untuk menyimpan nilai berupa suatu angka dan *string* digunakan untuk menyimpan suatu nilai baik berupa angka, karakter maupun keduanya. “c data = d;”, *script* tersebut digunakan untuk mengembalikan nilai dari data yang ditunjuk, biasanya ini dijumpai pada sebuah *constructor* untuk mengembalikan nilai dari suatu variabel.

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

*Script* “public boolean kosong(){” berfungsi sebagai *method* yang berfungsi untuk melakukan pengecekan pada nilai “head”, apabila nilai “head” masih sama dengan “null”.

|  |
| --- |
| public void Push(int newData,String nama) {  Node newNode = new Node(newData,nama);  if (isEmpty()) {  head = tail = newNode;  }else {  newNode.next = head;  head = newNode;  }  size++;  } |

*Script* “public void Push(int newData,String nama) {” merupakan *method* yang menggunakan 2 buah parameter yaitu “newData” dengan tipe data *integer* dan “nama” dengan tipe data *String*. Kelas ini berfungsi untuk memasukkan data kedalam suatu *node*, dengan cara memasukkan data ke *node* baru yang diwakili variabel “newNode” terlebih dahulu, dan kemudian kelas ini akan melakukan pengecekan kondisi, apabila kondisi terpenuhi maka *statement* dari *method* ini akan dieksekusi, “size++” berfungsi untuk melakukan penambahan atau mengukur panjang *linked list* setiap dimasukkannya sebuah *node* kedalam *linked* *list* itu sendiri.

|  |
| --- |
| public void Pop() {  Node temp = head;  head= head.next;  temp=null;  } |

*Script* “public void Pop() {” merupakan sebuah *method* yang berfungsi untuk melakukan penghapusan data di awal linked list atau dapat disebut juga untuk melakukan penghapusan data pada *node* “head”, untuk melakukan penghapusan, *method* ini akan mendeklarasikan bahwa *node* “temp” sama dengan “head”, kemudian *node* “head” akan dipindahkan ke *node* selanjutnya, setelah itu *node* “temp” akan diubah nilainya yaitu sama dengan “null”, hal ini dikarenakan *node* “temp” hanya digunakan sebagai variabel bantu dikarenakan dalam proses pemindahan *node* “head”, *node* sebelumnya tidak boleh dikosongkan.

|  |
| --- |
| void list(){  System.out.print(head.data + " " + head.nama + " Lepas Landas ");  } |

*Script* “void list(){” merupakan *method* yang digunakan untuk menampilkan nama dari pesawat apa saja yang telah melakukan lepas landas.

|  |
| --- |
| public void display() {  Node current = head;  while (current != null) {  System.out.print(current.data + " ");  System.out.print(current.nama+ " " );  current = current.next;  }} |

*Script* “public void display() {” merupakan suatu *method* yang berfungsi untuk menampilkan data, *method* ini akan menampilkan data dengan cara melakukan perulangan, pertama pada *method* ini dideklarasikan sebuah variabel bantu dengan nama variabel “current”, pada perulangan apabila nilai variabel ini tidak sama dengan “null” maka *statement* dari perulangan tersebut akan dieksekusi secara terus menerus hingga kondisi dari perulangan tersebut sudah tidak terpenuhi lagi, data-data yang ditampilkan merupakan hasil dari perulangan yang telah dilakukan tadi.

|  |
| --- |
| public static void main(String[] args) {  JURNAL\_1 list =new JURNAL\_1();  JURNAL\_1 list1 =new JURNAL\_1();  JURNAL\_1 list2 =new JURNAL\_1();  System.out.println("Hanggar A");  list.Push(123,"wantutri");  list.Push(737,"seventriseven");  list.Push(369,"trisixnine");  list.Push(666,"sixsixsix");  list.display(); |

*Script* “public static void main(String[] args) {” merupakan *main class* dimana nantinya *method-method* yang telah dibuat akan dipanggil melalui kelas ini. Potongan kode “JURNAL\_1 list =new JURNAL\_1();” berfungsi untuk mendeklarasikan objek baru “JURNAL\_1”, pendeklarasian objek sendiri berfungsi agar objek tersebut nantinya dapat mengakses kelas yang dituju sehingga *method-method* yang ada didalam kelas tersebut dapat dipanggil ke kelas utama. Pada potongan kode “list.Push(123,"wantutri");” merupakan proses pemanggilan *method* menggunakan objek yang telah dibuat sebelumnya, *method* yang dipanggilpun memiliki data beserta nilai yang nantinya ditampilkan oleh *method* itu sendiri.

|  |
| --- |
| System.out.println("Hanggar B");  list1.Push(400,"fourzerozero");  list1.Push(648,"sixfoureight");  list1.display();  System.out.println("");  System.out.println("yang sedang terbang"); |

*Script* “list1.Push(400,"fourzerozero");” merupakan pemanggilan suatu *method* yang difungsikan untuk memasukkan data yang sesuai dengan data yang ada didalam kurung pada *method* itu sendiri, nantinya data-data yang telah dipanggil ditampilkan melalui *method* “display”.

|  |
| --- |
| System.out.println("Pukul 14.00 : ");  list.list();  list.Pop();  System.out.println(); |

*Script* “System.out.println("Pukul 14.00 : ");” merupakan suatu *script* yang memuat *method* “list.list()” untuk melakukan penampilan data pesawat yang akan lepas landas, sedangkan *method* “list.pop()” digunakan untuk menghapus data yang terletak diawal *linked list*.

|  |
| --- |
| System.out.println("Pukul 15.00 : ");  list1.list();  list1.Pop();  System.out.println(); |

*Script* “System.out.println("Pukul 15.00 : ");” merupakan suatu *script* yang memuat *method* “list.list()” untuk melakukan penampilan data pesawat yang akan lepas landas, sedangkan *method* “list.pop()” digunakan untuk menghapus data yang terletak diawal *linked list*.

|  |
| --- |
| System.out.println("Pukul 16.00 Pesawat 186 Masuk hanggar B ");  list1.Push(186,"waneightsix"); |

*Script* “list1.Push(186,"waneightsix");” berfungsi untuk melakukan proses penambahan data berupa data “(186,"waneightsix")” ke dalam suatu *node*.

|  |
| --- |
| System.out.println("Pukul 16.00 Pesawat 666 Masuk hanggar B ");  list1.Push(666,"sixsixsix"); |

*Script* “list1.Push(666,"sixsixsix");” berfungsi untuk melakukan proses penambahan data berupa data “(666,"sixsixsix")” ke dalam suatu *node* pada *linked list*.

|  |
| --- |
| System.out.println("Pukul 16.00 Pesawat 221 Masuk hanggar A ");  list.Push(221,"tutuwan"); |

*Script* “list.Push(221,"tutuwan");” berfungsi untuk melakukan proses penambahan data berupa data “(221,"tutuwan");” ke dalam suatu *node* pada *linked list*.

|  |
| --- |
| System.out.println("Pesawat Pada Hanggar A : ");  list.display();  System.out.println(); |

*Script* “list.display();” merupakan suatu *method* yang difungsikan untuk menampilkan data pesawat apa saja yang menempati “Hanggar A”.

|  |
| --- |
| System.out.println("Pesawat Pada Hanggar B : ");  list1.display();  System.out.println();  }  } |

*Script* “list.display();”merupakan suatu *method* yang difungsikan untuk menampilkan data pesawat apa saja yang menempati “Hanggar B”.

* + 1. **Program Pengurutan Berdasarkan Abjad**

|  |
| --- |
| class Orang{  String nama;  int kabin;  Orang next;  public Orang (String nama,int kabin){  this.nama=nama;  this.kabin=kabin;  }  } |

*Script* “class Orang{” berfungsi sebagai kelas dengan nama kelas “Orang”, kelas ini memiliki variabel-variabel yang telah dideklarasikan seperti, variabel “data” dengan tipe data *integer* dan variabel “nama” dengan tipe data *string*. Kelas ini juga memiliki *constructor* yang fungsinya untuk memberikan nilai awal pada saat suatu objek dibuat, *constructor* pada kelas ini memiliki 2 buah parameter dengan tipe data *integer* dan *string* yang dimana *integer* fungsinya digunakan untuk menyimpan nilai berupa suatu angka dan *string* digunakan untuk menyimpan suatu nilai baik berupa angka, karakter maupun keduanya.

|  |
| --- |
| class Queue{  Orang front,rear;  void enqueue(String nama, int kabin){  Orang baru = new Orang(nama,kabin);  if(front==null){  front=rear=baru;  } else {  rear.next = baru;  rear = baru;  }  } |

*Script* “class Queue{” merupakan suatu kelas yang memuat variabel “front” dan “rear”, kelas ini juga memiliki suatu *method* yaitu “enqueue”. *Method* ini memiliki 2 buah paramater yang memiliki fungsinya masing-masing, penambahan data dalam *method* ini dilakukan dengan cara terlebih awal *object* yang bernama baru untuk memanggil *node*-*node* yang ada pada *class* orang, selanjutnya dilakukan pengecekan terhadap *statement if* apabila kondisinya terpenuhi maka dibuatlah suatu pernyataan bahwa “front”, “rear” dan “baru” akan menempati *node* yang sama. Bila pengecekan dilakukan pada *statement else* maka program akan melakukan penambahan *node*.

|  |
| --- |
| Public Orang dequeue(){  Orang temp = front;  front= front.next;  return temp;  } |

*Script* “Public Orang dequeue(){” merupakan suatu *method* yang digunakan untuk melakukan penghapusan data yang terletak diawal, “Orang temp = front;” *script* ini berfungsi untuk menentukan lokasi *front* dengan menggunakan variabel bantu *temp*, kemudian “front = front.next;” berfungsi untuk memindahkan *front* ke orang berikutnya, “return temp;” berfungsi untuk mengembalikan nilai yang dihapus ke *class* orang.

|  |
| --- |
| public void prior(String nama, int kabin) {  Orang baru = new Orang(nama, kabin);  if (front==null) {  front = rear = baru;  } else {  Orang cur = front;  while (cur != null) {  // penambahan data bagian depan  if (front.kabin > baru.kabin) {  baru.next = front;  front = baru;  break;  // penambahan data dibelakang  } else if (rear.kabin <= baru.kabin) {  rear.next = baru;  rear = baru;  break;  // penyisipan ditengah-tengah  } else {  if (baru.kabin >= cur.kabin && cur.next.kabin > baru.kabin) {  baru.next = cur.next;  cur.next = baru;  break;  }  }  cur = cur.next;  }  }  } |

*Script* “public void prior(String nama, int kabin) {” merupakan suatu *method* yang digunakan untuk melakukan proses pengurutan data pada *linked list*, ini dapat terlihat melalui *statement* kondisi yang ada pada *method* ini, *method* juga menggunakan *statement looping* yang digunakan untuk melakukan perulangan apabila kondisi dari *statement* tersebut terpenuhi hingga kondisinya sudah tidak memenuhi lagi, penyisipan data akan dilakukan ketika masing-masing kondisi memenuhi syarat untuk dilakukannya penyisipan.

|  |
| --- |
| public void display() {  Orang current = front;  while (current != null) {  System.out.print(current.nama + "(" + current.kabin + ") ");  current = current.next;  }  }  } |

*Script* “public void display() {” merupakan suatu *method* yang difungsikan untuk menampilkan data-data yang telah dimuat dalam program. Data ditampilkan ketika kondisi *while* memenuhi syarat, maka data akan dikeluarkan menggunakan fungsi “System.out.print”, sedangkan “current = current.next;” difungsikan untuk melakukan proses pemindahan ke data selanjutnya.

|  |
| --- |
| public static void main(String[] args) {  Queue baris1 = new Queue();  Queue baris2 = new Queue();  Queue baris3 = new Queue(); |

*Script* “public static void main(String[] args) {” merupakan kelas utama pada program. *Script* “Queue baris1 = new Queue();” digunakan untuk mendeklarasikan sebuah objek baru yang nantinya objek tersebut digunakan untuk memanggil *method* pada kelas “Queue”. Hal ini juga berlaku terhadap *script* yang serupa dengannya.

|  |
| --- |
| baris1.enqueue("Jason",1);  baris1.enqueue("Meg",4);  baris1.enqueue("Piper",10);  baris1.enqueue("Thalia",1);  baris1.display();  System.out.println(); |

*Script* “baris1.enqueue("Jason",1);” merupakan pemanggilan *method* yang berfungsi untuk melakukan penambahan data *linked list*, data yang ditambah merupakan data yang dimuat didalam *method* tersebut. Data-data yang telah dipanggil nantinya akan ditampilkan menggunakan *method* “baris1.display()”.

|  |
| --- |
| System.out.println("Jason dipanggil");  baris1.dequeue();  baris1.display();  System.out.println(); |

*Script* “System.out.println("Jason dipanggil");” merupakan suatu *method* yang dibuat untuk menghapus data pada *linked list* untuk sementara. Penghapusan dilakukan oleh *method dequeue* yang ada pada baris *line* di program tersebut.

|  |
| --- |
| System.out.println("Barisan Telat");  baris2.enqueue("Percy",3);  baris2.enqueue("Will",7);  baris2.enqueue("Connor",12);  baris2.enqueue("Travis",12);  baris2.enqueue("Nico",13);  baris2.display();  System.out.println(); |

*Script* “baris2.enqueue("Percy",3);” merupakan proses pemanggilan *method* yang difungsikan untuk menambahkan “Percy” dan teman-temannya ke dalam barisan. Hasil penambahan kemudian ditampilkan menggunakan *method* “baris2.display();”.

|  |
| --- |
| Orang tampung = new Orang(" ", 0);  for (int i=1; i<6;i++){  tampung=baris2.dequeue();  baris1.enqueue(tampung.nama, tampung.kabin);  }  System.out.println(); |

*Script* “for (int i=1; i<6;i++){” merupakan suatu *statement looping* yang difungsikan untuk membantu penghapusan data pada *linked list* yang kedua, yang kemudian dilakukan penambahan data pada *linked list* pertama dengan menggunakan 2 parameter yang ada.

|  |
| --- |
| System.out.println("Barisan Belum dipanggil");  baris1.display();  System.out.println(); |

*Script* “System.out.println("Barisan Belum dipanggil");” merupakan *script* yang digunakan untuk menunjukkan “Barisan Belum dipanggil”. *Method* yang digunakan untuk menampilkan data tersebut adalah “baris1.display();”

|  |
| --- |
| for (int i=1; i<9;i++){  tampung=baris1.dequeue();  baris2.prior(tampung.nama, tampung.kabin);  } |

*Script* “for (int i=1; i<9;i++){” merupakan *statement looping* yang digunakan untuk melakukan pemindahan barisan. *Script* “baris2.prior(tampung.nama, tampung.kabin);” merupakan data yang sudah ditampung tadi akan dimaskukin ke *method* “prior” yang berfungsi untuk mengurutkan data dari kabin terkecil.

|  |
| --- |
| System.out.println("Barisan Setelah diurutkan Sesuai Nomor Kabin Terkecil");  baris2.display();  } |

*Script* “baris2.display();” merupakan *method* yang digunakan untuk menampilkan “Barisan Setelah diurutkan Sesuai Nomor Kabin Terkecil” ke dalam fungsi utama.

* + 1. **Program Konversi Nilai**

|  |
| --- |
| class HmmNode {  public Object data;  public HmmNode next;  public HmmNode () {  data =' ';  }  public HmmNode (Object val) {  data = val;  }  } |

*Script* “class HmmNode” adalah pembuatan *method constructor* agar setiap data dapat memiliki *pointer*. Pada *class* tersebut ada tiga pendeklarasian data yang memiliki *modifier* bertipe *public*, dan juga ada *constructor* yang memiliki parameter *object* “val” , dimana data akan sama dengan “val” .

|  |
| --- |
| public void push(Object e){  HmmNode tmp = new HmmNode(e);  tmp.next = top;  top = tmp;  } |

*Script* “public void push(Object e){” adalah *script* untuk membuat *method* dengan nama “push” yang digunakan untuk memasukkan karakter-karakter ekspresi ke dalam *stack* dan menjadikan setiap karakter yang baru masuk menjadi *top* dari *stack*.

|  |
| --- |
| public Object pop(){  Object x = top.data;  top = top.next;  return x;  } |

*Script* “public void pop(){” adalah *script* untuk membuat *method* dengan nama “pop” yang digunakan untuk mengeluarkan karakter-karakter ekspresi yang ada di dalam *stack* dan menjadikan karakter yang berada pada posisi setelah *top* menjadi *top* dari *stack*.

|  |
| --- |
| public void postfix(String x){  String output="";  jurnal3quasi S=new jurnal3quasi();  for(int i=0;i<x.length();i++){  char c=x.charAt(i); |

*Script* “public void postfix(String x){” adalah *code* untuk membuat *method* bernama “postfix” yang digunakan untuk mengkonversi ekxpresi *infix* menjadi *postfix*.

|  |
| --- |
| if(c==('+')||c==('\*')||c==('-')||c==('/')){  while(!S.empty() && prior(S.check())>= prior(c))  output+=S.pop();  S.push(c);  }  else if(c=='('){  S.push(c);  }  else if(c==')'){  while(!S.check().equals('('))  output+=S.pop();  S.pop();  }  else{  output+=c;  }  } |

*Script* “if(c==('+')||c==('\*')||c==('-')||c==('/')){” merupakan *script* yang akan bekerja selama dilakukannya pengecekan karakter, diberikan kondisi dalam perulangan dimana apabila karakter yang ditemukan merupakan simbol *operator*, maka akan dimasukkan ke dalam *stack* terlebih dahulu sampai bertemu dengan *operator* lain. Apabila karakter yang ditemukan berupa *operand*, maka langsung ditampilkan tanpa perlu dimasukkan ke dalam *stack*.

|  |
| --- |
| while(!S.empty()){  output+=S.pop();  System.out.println("Notasi Infix : "+x);  System.out.println("Notasi Postfix : "+output);  } |

*Script*  “while(!S.empty()){” berfungsi untuk memeriksa apakah karakter yang akan diubah sudah habis atau tidak, jika sudah maka semua karakter yang ditampung dalam *stack* akan langsung dikeluarkan semuanya.

|  |
| --- |
| public void prefix(String x){  String output="";  jurnal3quasi S=new jurnal3quasi();  int i;  for(i=0;i<x.length();i++){  char c=x.charAt(i); |

*Script* “public void prefix(String x){” adalah *code* untuk membuat *method* bernama “prefix” yang digunakan untuk mengkonversi ekxpresi *infix* menjadi *prefix*.

|  |
| --- |
| if(c==('+')||c==('\*')||c==('-')||c==('/')){  while(!S.empty() && prioritas(S.check()) <= prioritas(c))  output+=S.pop();  S.push(c);  }  }  while(!S.empty())  output+=S.pop(); |

*Script*“if(c==('+')||c==('\*')||c==('-')||c==('/')){” Diberikan kondisi dimana apabila karakter yang ditemukan berupa *operand* maka akan langsung dikeluarkan dari *stack*.

|  |
| --- |
| for(i=0;i<x.length();i++){  char c=x.charAt(i);  if(c==('+')||c!=('\*')||c!=('-')||c!=('/')){  while(!S.empty() && prioritas(S.check()) >= prioritas(c))  output+=S.pop();  S.push(c);  }  else  output+=c;  if((c=='(')||(c==')')){  S.push(c);  }  if(c==('+')||c==('\*')||c==('-')||c==('/')){  S.pop();  }  } |

*Script*“for(i=0;i<x.length();i++){” adalah *script* yang digunakan untuk melakukan perulangan dimana terdapat variabel “c” adalah untuk menentukan apakah karakter yang ditemukan berupa *operand* atau *operator* sehingga akan dikeluarkan dari *stack* menurut urutan prioritas yang terdapat dalam proses konversi menjadi *prefix*.

|  |
| --- |
| while(!S.empty())  output+=S.pop();  System.out.println("Notasi Prefix : "+output);  } |

*Script*“while(!S.empty())” berfungsi untuk mengeluarkan semua isi dari *stack* yang tertampung setelah pengecekan *operand* sudah tidak ditemukan dan langsung menampilkan *output* berupa notasi *prefix* dari ekspresi.

|  |
| --- |
| public int prior(Object x){  if(x.equals('+')||x.equals('-'))  return 1;  else if(x.equals('\*')||x.equals('/'))  return 2;  else  return 0;  } |

*Script* “public int prior(Object x){” merupakan *method* yang digunakan untuk menentukan prioritas dari simbol-simbol yang digunakan dalam ekspresi *infix* untuk dijadikan ketentuan prioritas dalam *method* “postfix”.

|  |
| --- |
| public int prioritas(Object x){  if(x.equals('+')||x.equals('-'))  return 2;  else if(x.equals('\*')||x.equals('/'))  return 1;  else  return 0;  } |

*Script* “public int prioritas(Object x){” merupakan *method* yang digunakan untuk menentukan prioritas dari simbol-simbol yang digunakan dalam ekspresi *infix* untuk dijadikan ketentuan prioritas dalam *method* “prefix”.

|  |
| --- |
| public static void main(String args[]){  jurnal3quasi s=new jurnal3quasi();  System.out.println();  System.out.println("------------------------------------");  System.out.println("Notasi Infix: \"(A+B)\*C-(D-E)\*(F+G)\"");  System.out.println("------------------------------------");  s.postfix("(A+B)\*C-(D-E)\*(F+G)");  s.prefix("(A+B)\*C-(D-E)\*(F+G)");  System.out.println("------------------------------------");  } |

*Script* “public static void main(String args[]){” *Method* utama dari program berisi pemanggilan *method* “postfix” yang diberikan parameter berupa ekspresi yang ada di permasalahan untuk kemudian dikonversi menjadi notasi *postfix*. *Method* utama dari program juga berisi pemanggilan *method* “prefix” yang diberikan parameter berupa ekspresi yang ada di permasalahan untuk kemudian dikonversi menjadi notasi *prefix.*

1. **KESIMPULAN**

Berdasarkan hasil praktikum yang telah dilaksanakan, dapat disimpulkan bahwa:

1. *Stack* adalah struktur data yang memiliki sistem LIFO (*Last In First Out*), dimana data yang terakhir ditambahkan akan menjadi yang pertama kali dikeluarkan atau dihapus atau diproses. *Queue* adalah struktur data yang memiliki sistem FIFO (*First In First Out*), dimana data yang pertama ditambahkan akan menjadi yang pertama dikeluarkan atau diproses.
2. *Stack* dan *Queue* dibentuk menggunakan konsep *Double Linked List*. Pada *Stack*, penambahan dilakukan di depan (*add front*) dan penghapusan juga dilakukan di depan(*delete front*). Pada *Queue*, penambahan dilakukan di belakang (*add last*) dan penghapusan dilakukan di depan (*delete front*).