LAPORAN PROYEK PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA 2024

QZY: SISTEM MANAJEMEN ANTRIAN BANK

Oleh Kelompok 38

Anggota:

Muhammad Iqbal Taufiq	**F1D02310080**
Yusri Abdi	**F1D02310098**
Yurian Fathur Fajar	**F1D02310097**
Lalu Maulana Rizki Hidayat	**F1D02310118**

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MATARAM

2024

LEMBAR PENGESAHAN LAPORAN PROYEK
PRAKTIKUM ALGORITMA DAN STRUKTUR DATA 2024

1. Kelompok : 38
2. Judul Proyek : Qzy : Sistem Manajemen Antrian Bank
3. Anggota Kelompok : Muhammad Iqbal Taufiq (F1D02310080)
   Yusri Abdi (F1D02310098)
   Yurian Fathur Fajar (F1D02310097)
   Lalu Maulana Rizki Hidayat (F1D02310118)

Laporan proyek ini disusun sesuai dengan kaidah penyusunan yang telah ditentukan dan dibuat sebagai syarat mata kuliah Algoritma dan Struktur Data2024.

Mataram, 22 Desember 2024

Telah diperiksa dan disahkan oleh:

Koordinator Asisten	**Asisten Pembimbing**
Aditya Rahmatdiyansyah	**Rizaldi Abyannata**
F1D022031	**F1D022158**

1.1 JUDUL

Judul proyek kami yaitu “Qzy : Sistem Manajemen Antrian Bank”

1.2 LATAR BELAKANG

Antrian di bank adalah salah satu masalah yang sering dihadapi oleh nasabah maupun pihak bank. Ketika jumlah nasabah yang datang lebih banyak daripada kapasitas layanan yang tersedia, waktu tunggu menjadi sangat panjang, sehingga menimbulkan ketidaknyamanan. Nasabah sering merasa frustrasi, sementara pihak bank menghadapi tantangan untuk menjaga kepuasan pelanggan.

Sistem antrian tradisional yang masih bergantung pada pengelolaan manual sering kali tidak cukup efisien. Akibatnya, banyak bank mencari solusi berbasis teknologi untuk mengelola antrian dengan lebih baik. Salah satu solusi tersebut adalah sistem manajemen antrian digital, yang memanfaatkan teknologi untuk menciptakan layanan yang lebih terorganisir dan efisien.

Kami mengembangkan “Qzy: Sistem Manajemen Antrian Bank” sebagai solusi untuk mengatasi masalah ini. Dengan memanfaatkan konsep algoritma dan struktur data, terutama algoritma *queue* (antrian), sistem ini mampu mengatur giliran nasabah dengan lebih efisien.

1.3 DESKRIPSI PROGRAM

Qzy adalah sebuah solusi berbasis teknologi yang dirancang untuk mengatasi masalah antrian di bank. Sistem ini memanfaatkan konsep algoritma dan struktur data, seperti *queue* (antrian), untuk menciptakan manajemen antrian yang lebih efisien dan terorganisir. Dengan pendekatan ini, nasabah dapat dilayani berdasarkan prioritas dan waktu kedatangan mereka, sehingga pengalaman layanan menjadi lebih nyaman dan tertata.

Dalam sistem ini, nasabah mendaftarkan diri melalui perangkat yang terintegrasi, di mana data mereka, termasuk layanan yang diinginkan, waktu masuk, dan tingkat prioritas, langsung tercatat secara digital. Sistem kemudian mengatur urutan antrian secara otomatis menggunakan algoritma *sorting* yang mempertimbangkan prioritas dan waktu masuk. Dengan demikian, nasabah dengan kebutuhan mendesak dapat didahulukan tanpa mengabaikan urutan kedatangan yang lain.

Struktur data yang digunakan juga mencakup mekanisme pencarian, pengurutan, dan manajemen antrian yang mendalam. Selain itu, sistem ini dilengkapi dengan fitur riwayat (*history*) dan undo untuk membantu operator dalam mengelola kesalahan operasional secara lebih fleksibel. Dengan menggunakan teknologi ini, bank tidak hanya

meningkatkan efisiensi layanan, tetapi juga memberikan pengalaman yang lebih memuaskan bagi nasabah.

Dengan desain yang modern dan fitur yang inovatif, Qzy menawarkan lebih dari sekadar pengelolaan antrian. Sistem ini menghadirkan pendekatan baru dalam dunia perbankan, mengintegrasikan efisiensi operasional dengan kemudahan teknologi, sehingga dapat menjadi solusi ideal untuk tantangan antrian di bank.

1.4 ALGORITMA

1. *Start*
2. Program menggunakan kelas Node untuk merepresentasikan data antrian, "`PriorityTreeNode`" untuk antrian berbasis prioritas, "`linkedlist`" untuk mengelola antrian dengan *linked list*, "`Stack`" untuk mencatat riwayat tindakan, dan "`Tree`" untuk mengelola antrian berdasarkan prioritas dengan struktur *binary tree*.
3. Program menampilkan menu utama dengan pilihan untuk menambah data, menampilkan antrian, mencari berdasarkan nama, mengurutkan berdasarkan prioritas, mengeluarkan antrian, dan melihat riwayat.
4. Pengguna memasukkan nomor antrian, nama, layanan, prioritas, dan waktu masuk yang otomatis dicatat, lalu data ditambahkan ke *linked list* dan *binary tree*, serta tindakan tersebut dicatat dalam *stack*.
5. Program menampilkan seluruh data antrian dengan traversal melalui *linked list* dan mencetaknya dalam format tabel.
6. Pengguna memasukkan nama untuk mencari data pelanggan yang sesuai melalui pencarian linear dalam *linked list*.
7. Program menggunakan struktur *binary tree* untuk mengurutkan dan menampilkan data berdasarkan prioritas pelanggan, dengan traversal *preorder*.
8. *Node* pertama dalam *linked list* dihapus menggunakan metode *dequeue*, dan tindakan penghapusan tersebut dicatat dalam *stack*.
9. Program menampilkan seluruh riwayat tindakan yang telah dilakukan, yang tersimpan dalam stack.
10. Program keluar dari *loop* utama dan menampilkan pesan "Keluar dari program" ketika pengguna memilih opsi keluar.
11. Program memeriksa apakah input pengguna valid dan jika tidak, memberikan pesan kesalahan dan meminta pengguna untuk mencoba lagi.
12. *End*

1.5 PENJELASAN CODE

1.5.1 *Class Node*

```
public class Node {
    int nomorAntrian;
    String nama;
    String layanan;
    String prioritas;
    String waktuMasuk;
    Node next = null;

    public Node(int nomorAntrian, String nama, String layanan,
String prioritas, String waktuMasuk) {
        this.nomorAntrian = nomorAntrian;
        this.nama = nama;
        this.layanan = layanan;
        this.prioritas = prioritas;
        this.waktuMasuk = waktuMasuk;
    }
}
```

Script di atas merupakan kelas "`Node`" yang merepresentasikan sebuah simpul dalam struktur *linked list*. Kelas ini memiliki lima atribut yaitu "`nomorAntrian`" bertipe data *integer*, "`nama`" bertipe data *string*, "`layanan`" bertipe data *string*, "`prioritas`" bertipe data *string*, serta "`waktuMasuk`" bertipe data *string*. Selain itu, terdapat atribut "`next`" yang merupakan referensi ke simpul berikutnya dalam *linked list*, yang diatur ke *null* sebagai nilai *default*. Konstruktor kelas ini digunakan untuk menginisialisasi kelima atribut dengan nilai yang diberikan pada saat objek "`Node`" dibuat.

1.5.2 *Class* PriorityTreeNode

```
public class PriorityTreeNode {
    int nomorAntrian;
    String nama;
    String layanan;
    String prioritas;
    String waktuMasuk;
    PriorityTreeNode left = null;
    PriorityTreeNode right = null;

    public PriorityTreeNode(int nomorAntrian, String nama,
String layanan, String prioritas, String waktuMasuk) {
        this.nomorAntrian = nomorAntrian;
        this.nama = nama;
        this.layanan = layanan;
        this.prioritas = prioritas;
        this.waktuMasuk = waktuMasuk;
    }
}
```

Script di atas merupakan kelas "`PriorityTreeNode`" yang merepresentasikan sebuah simpul dalam struktur pohon biner. Kelas ini memiliki lima atribut yaitu "`nomorAntrian`" bertipe data *integer*, "`nama`" bertipe data

string, "`layanan`" bertipe data *string*, "`prioritas`" bertipe data *string*, serta "`waktuMasuk`" bertipe data *string*. Selain itu, terdapat dua atribut tambahan, yaitu "`left`" dan "`right`", yang masing-masing merupakan referensi ke anak kiri dan kanan dalam pohon biner. Kedua atribut ini diatur ke *null* sebagai nilai *default*. Konstruktor kelas ini digunakan untuk menginisialisasi kelima atribut dengan nilai yang diberikan pada saat objek "`PriorityTreeNode`" dibuat.

1.5.3 *Class Linkedlist*

```
public class linkedlist {
    Node head = null;
    Node tail = null;

    public void AddNode(int nomorAntrian, String nama, String
layanan, String prioritas, String waktuMasuk) {
        Node newNode = new Node(nomorAntrian, nama, layanan,
prioritas, waktuMasuk);

        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            if (nomorAntrian < head.nomorAntrian) {
                newNode.next = head;
                head = newNode;
            } else {
                Node current = head;
                while (current.next != null &&
current.next.nomorAntrian < nomorAntrian) {
                    current = current.next;
                }
                newNode.next = current.next;
                current.next = newNode;

                if (current == tail) {
                    tail = newNode;
                }
            }
        }
    }
```

Script di atas merupakan kelas "`linkedlist`" yang merepresentasikan sebuah *linked list* untuk menyimpan data antrian. Kelas ini memiliki dua atribut utama, "`head`" dan "`tail`", yang berfungsi sebagai referensi ke simpul pertama dan terakhir. Method "`AddNode`" digunakan untuk menambahkan simpul baru dengan mengurutkannya berdasarkan nomor antrian. Jika *linked list* kosong, simpul baru menjadi "`head`" dan "`tail`". Jika tidak, simpul baru disisipkan pada posisi yang tepat, dan "`tail`" akan diperbarui jika simpul baru berada di akhir *linked list*.

```
public void display() {
   Node current = head;
   if (head == null) {
        System.out.println("Mohon maaf, data masih kosong!");
   } else {
        System.out.println("==================================
====================");
       System.out.println("| No. Antrian | Nama | Layanan |
Prioritas | Waktu Masuk |");
       System.out.println("----------------------------------
--------------------");
       while (current != null) {
           System.out.println("| " + current.nomorAntrian + " |
" + current.nama + " | " + current.layanan + " | " +
current.prioritas + " | " + current.waktuMasuk + " |");
           System.out.println("------------------------------
------------------------");
           current = current.next;
        }
    }
}
```

Script di atas merupakan. Metode "`display`" digunakan untuk menampilkan seluruh data yang tersimpan pada *linked list*. Pertama, method ini memeriksa apakah linked list kosong dengan mengecek apakah "`head`" bernilai null. Jika kosong, pesan "`Mohon maaf, data masih kosong!`" akan ditampilkan. Namun, jika terdapat data, *header* tabel ditampilkan terlebih dahulu dengan struktur kolom seperti "`NomorAntrian, Nama, Layanan, Prioritas, WaktuMasuk`". Selanjutnya, dengan menggunakan *loop while*, setiap simpul dalam *linked list* ditelusuri dari awal "`head`" hingga simpul terakhir. Data pada setiap simpul, yang terdiri dari atribut "`nomorAntrian, nama, layanan, prioritas, dan waktuMasuk`", ditampilkan dalam baris tabel yang rapi. Setelah data dari satu simpul selesai ditampilkan, penunjuk "`current`" diperbarui ke simpul berikutnya hingga seluruh data selesai diproses.

```
public Node searchByName(String name) {
    Node current = head;
    while (current != null) {
          if (current.nama.equalsIgnoreCase(name)) {
              return current;
          }
          current = current.next;}
      return null;}
```

Script di atas merupakan metode "`searchByName`" yang mencari simpul berdasarkan nama. Metode ini menelusuri *linked list* dari simpul pertama "`head`", membandingkan atribut nama dengan parameter menggunakan "`equalsIgnoreCase`". Jika ditemukan, simpul dikembalikan. Jika tidak, metode mengembalikan *null*.

```
    public void sortByPriority() {
        Node current = head;
        while (current != null) {
            Node nextNode = current.next;
            while (nextNode != null) {
                if (current.prioritas.equalsIgnoreCase("Iya{
                   int tempNomorAntrian = current.nomorAntrian;
                    String tempNama = current.nama;
                    String tempLayanan = current.layanan;
                    String tempPrioritas = current.prioritas;
                    String tempWaktuMasuk = current.waktuMasuk;

                  current.nomorAntrian = nextNode.nomorAntrian;
                    current.nama = nextNode.nama;
                    current.layanan = nextNode.layanan;
                    current.prioritas = nextNode.prioritas;
                    current.waktuMasuk = nextNode.waktuMasuk;

                    nextNode.nomorAntrian = tempNomorAntrian;
                    nextNode.nama = tempNama;
                    nextNode.layanan = tempLayanan;
                    nextNode.prioritas = tempPrioritas;
                    nextNode.waktuMasuk = tempWaktuMasuk;
                                    if   (current.nomorAntrian   <
current.next.nomorAntrian) {
                                        int  temppNomorAntrian  =
current.nomorAntrian;
                          String temppNama = current.nama;
                          String temppLayanan = current.layanan;
                        String temppPrioritas = current.prioritas;
                                       String  temppWaktuMasuk  =
current.waktuMasuk;

                                          current.nomorAntrian  =
nextNode.nomorAntrian;
                          current.nama = nextNode.nama;
                          current.layanan = nextNode.layanan;
                          current.prioritas = nextNode.prioritas;
                       current.waktuMasuk = nextNode.waktuMasuk;

                       nextNode.nomorAntrian = temppNomorAntrian;
                          nextNode.nama = temppNama;
                          nextNode.layanan = temppLayanan;
                          nextNode.prioritas = temppPrioritas;
                        nextNode.waktuMasuk = temppWaktuMasuk;}}
                nextNode = nextNode.next;}
            current = current.next;}}
```

Script di atas merupakan implementasi metode "`sortByPriority`" yang digunakan untuk mengurutkan simpul dalam *linked list* berdasarkan prioritas. Metode ini menelusuri setiap simpul menggunakan dua *loop* bersarang: *loop* pertama untuk simpul saat ini "`current`" dan *loop* kedua untuk simpul berikutnya "`nextNode`". Jika atribut prioritas dari simpul saat ini bernilai "`Iya`", data simpul tersebut ditukar dengan simpul berikutnya menggunakan variabel sementara

untuk menyimpan nilai-nilai atribut seperti "`nomorAntrian, nama, layanan, prioritas, dan waktuMasuk`".

```
public void displaySortedQueue() {
    Node current = head;
    System.out.println("Hasil Sorting Berdasarkan Prioritas:");
    System.out.println("----------------------------------------
----------------");
     System.out.printf("%-15s  %-10s  %-20s  %-10s\n",  "Nomor
Antrian", "Nama", "Layanan", "Prioritas");
    System.out.println("----------------------------------------
----------------");
    while (current != null) {
    System.out.printf("%-15d %-10s %-20s %-10d\n",
            current.nomorAntrian,
            current.nama,
            current.layanan,
            current.prioritas);
            current = current.next;
        }
    System.out.println("----------------------------------------
----------------");}
```

Script di atas merupakan metode "`displaySortedQueue`" yang digunakan untuk menampilkan data antrian yang sudah diurutkan berdasarkan prioritas. Metode ini dimulai dengan mencetak judul "`Hasil Sorting Berdasarkan Prioritas`" dan "`header`" tabel yang berisi kolom "`Nomor Antrian, Nama, Layanan, dan Prioritas`".

```
public void sequentialSort() {
        if (head == null) return;

        Node current = head;
        while (current != null) {
            Node minNode = current;
            Node next = current.next;

            while (next != null) {
              if (next.waktuMasuk.compareTo(minNode.waktuMasuk)
< 0) {
                    minNode = next;
                }
                next = next.next;
            }

            if (current != minNode) {
                swapNodes(current, minNode);
            }
            current = current.next;
        }
    }
```

Script di atas merupakan metode "`sequentialSort`" yang mengurutkan simpul dalam *linked list* berdasarkan atribut "`waktuMasuk`" menggunakan algoritma *selection sort*. Metode ini mencari simpul dengan "`waktuMasuk`"

terkecil di antara simpul yang belum diproses dan menukarnya dengan simpul saat ini. Proses ini diulang hingga seluruh *linked list* terurut.

```
    public void swapNodes(Node a, Node b) {
        if (a == b) {
            return;}
        Node prevA = null, prevB = null;
        Node current = head;
        while (current != null) {
            if (current.next == a) {
                prevA = current;
            }
            if (current.next == b) {
                prevB = current;
            }
            current = current.next;
        }
        if (a == head) {
            head = b;
        }
        else if (b == head) {
            head = a;
        }
        if (prevA != null) {
            prevA.next = b;}
        if (prevB != null) {
            prevB.next = a;}
        Node temp = a.next;
        a.next = b.next;
        b.next = temp;}
```

Script di atas merupakan metode "`swapNodes`" yang digunakan untuk menukar posisi dua simpul dalam *linked list*. Metode ini pertama-tama memeriksa apakah kedua simpul yang diberikan sama, jika ya, maka tidak ada yang perlu dilakukan. Kemudian, metode mencari simpul-simpul sebelumnya "`prevA`" dan "`prevB`" dari simpul yang ingin ditukar dengan menelusuri *linked list*.

```
    public void enqueue(int nomorAntrian, String nama, String
layanan, String prioritas, String waktuMasuk) {
        Node newNode = new Node(nomorAntrian, nama, layanan,
prioritas, waktuMasuk);

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

Script di atas merupakan metode "`enqueue`" yang digunakan untuk menambahkan simpul baru ke dalam *linked list*, khususnya untuk antrian. Metode ini menerima parameter berupa "`nomorAntrian, nama, layanan, prioritas, dan waktuMasuk`" yang kemudian digunakan untuk membuat simpul baru "`newNode`".

```
    public Node dequeue(Stack stack) {
        if (head == null) {
            return null;
        } else {
            Node temp = head;
            head = head.next;
            if (head == null) {
                tail = null;
            }
            stack.push("Hapus Antrian - Nama: " + temp.nama);
             System.out.println("Data dengan nama " + temp.nama
+ " berhasil dihapus.");
            return temp;
        }
    }
```

Script di atas merupakan metode "`dequeue`" yang digunakan untuk menghapus simpul pertama "`antrian`" dari *linked list* dan memindahkannya ke dalam *stack*. Jika *linked list* kosong yaitu "`head = null`", metode akan mengembalikan *null*. Namun, jika tidak, simpul pertama disimpan dalam variabel sementara "`temp`", kemudian "`head`" diperbarui untuk menunjuk ke simpul berikutnya. Jika setelah penghapusan *linked list* menjadi kosong, "`tail`" juga diatur menjadi *null*.

1.5.4 *Class Stack*

```
public class Stack {

    private static class HistoryNode {
        String action;
        HistoryNode next;

        public HistoryNode(String action) {
            this.action = action;
            this.next = null;
        }
    }

    private HistoryNode top;

    public Stack() {
        this.top = null;
    }
```

Script di atas merupakan kelas "`Stack`" yang merepresentasikan struktur data tumpukan (*stack*) berbasis *linked list* untuk menyimpan data riwayat (*history*). Kelas ini memiliki kelas *nested* "`HistoryNode`" untuk merepresentasikan simpul, dengan atribut "`action`" untuk menyimpan data, "`next`" untuk menunjuk ke simpul berikutnya, dan konstruktor untuk inisialisasi. Atribut "`top`" menunjuk ke simpul paling atas, yang bernilai *null* jika *stack* kosong. Konstruktor "`Stack`" menginisialisasi stack dengan top bernilai *null*. Struktur ini bekerja dengan prinsip

LIFO (*Last In, First Out*), di mana elemen terakhir yang dimasukkan adalah elemen pertama yang diambil.

```
public void push(String action) {
    HistoryNode newNode = new HistoryNode(action);
    if (top == null) {
        top = newNode;
    } else {
        newNode.next = top;
        top = newNode;}
```

Script di atas merupakan metode "`push`" yang digunakan untuk menambahkan elemen baru ke dalam struktur data tumpukan (*stack*) berbasis *linked lis*t. Metode ini membuat simpul baru menggunakan objek "`HistoryNode`" dengan parameter "`action`" sebagai data yang akan disimpan. Jika *stack* kosong ("`top == null`"), simpul baru langsung menjadi elemen paling atas dengan mengatur "`top = newNode`". Jika tidak, simpul baru menunjuk ke elemen paling atas saat ini melalui atribut "`next`", kemudian "`top`" diperbarui untuk menunjuk ke simpul baru.

```
public String pop() {
     if (top == null) {
         return null;
     } else {
         String action = top.action;
         top = top.next;
         return action;}}
```

Script di atas merupakan metode "`pop`" yang digunakan untuk menghapus dan mengambil elemen paling atas dari struktur data tumpukan (*stack*). Jika *stack* kosong ("`top == null`"), metode akan mengembalikan *null*, menandakan tidak ada elemen yang bisa dihapus. Jika *stack* tidak kosong, data pada elemen paling atas ("`top.action`") disimpan dalam variabel "`action`", kemudian referensi "`top`" diperbarui untuk menunjuk ke elemen berikutnya dalam stack ("`top = top.next`"), sehingga elemen paling atas sebelumnya dihapus. Akhirnya, metode mengembalikan nilai "`action`", yaitu data dari elemen yang dihapus.

```
public String peek() {
     if (top == null) {
         return null;
     } else {
         return top.action;
     }}
 public boolean isEmpty() {
     return top == null}
```

Script di atas berisi metode "`peek`" dan "`isEmpty`" untuk operasi dasar pada struktur data tumpukan (*stack*) berbasis *linked list*. Metode "`peek`" digunakan

untuk melihat data pada elemen paling atas tanpa menghapusnya, dengan mengembalikan *null* jika *stack* kosong ("`top == null`") atau nilai "`action`" dari elemen paling atas jika *stack* tidak kosong. Metode "`isEmpty`" digunakan untuk memeriksa apakah *stack* kosong, mengembalikan *true* jika atribut "`top`" bernilai *null*, atau *false* jika terdapat elemen dalam *stack*. Kedua metode ini mendukung operasi pengaksesan data dan pemeriksaan status *stack*.

```
public void displayHistory() {
        if (top == null) {
            System.out.println("History is empty.");
        } else {
            System.out.println("Action History:");
            HistoryNode current = top;
            while (current != null) {
                System.out.println(current.action);
                current = current.next;
            }
        }}
    public void clearHistory() {
        top = null;
    }
}
```

Script di atas berisi dua metode, "`displayHistory`" dan "`clearHistory`", untuk mengelola elemen dalam struktur data tumpukan (*stack*) berbasis *linked list*. Metode "`displayHistory`" digunakan untuk menampilkan semua data yang tersimpan dalam *stack*, dimulai dari elemen paling atas hingga elemen terakhir. Jika *stack* kosong ("`top == null`"), metode akan mencetak pesan "`History is empty`". Jika tidak, metode mencetak "`Action History:`" diikuti oleh setiap data dalam *stack* menggunakan iterasi melalui simpul-simpul *stack*. Metode "`clearHistory`" digunakan untuk mengosongkan seluruh elemen dalam *stack* dengan mengatur referensi "`top`" menjadi *null*, yang secara efektif menghapus semua data dalam *stack*. Kedua metode ini memungkinkan pengguna untuk melihat dan mengelola riwayat yang tersimpan dalam *stack*.

1.5.5 *Class Tree*

```
public class Tree {
    private PriorityTreeNode root;

    public Tree() {
        this.root = null;}
```

Script di atas merupakan bagian dari kelas "`Tree`" yang merepresentasikan struktur data *tree* dengan simpul prioritas. Kelas ini memiliki atribut "`root`", yaitu simpul utama atau akar dari pohon, yang diinisialisasi dengan nilai "`null`" melalui

konstruktor *tree*. Konstruktor ini memastikan bahwa pohon dimulai dalam keadaan kosong tanpa simpul apa pun.

```
public void addCustomer(int nomorAntrian, String nama, String
layanan, String prioritas, String waktuMasuk) {
        root = addCustomerRec(root, nomorAntrian, nama, layanan,
prioritas, waktuMasuk);
    }
```

Script di atas merupakan metode "`addCustomer`" dalam kelas "`Tree`" yang digunakan untuk menambahkan simpul baru ke dalam pohon prioritas. Metode ini menerima parameter berupa data pelanggan seperti "`nomorAntrian, nama, layanan, prioritas, dan waktuMasuk`".

```
public void push(String item) {
         StackNode newNode = new StackNode(item);
         newNode.next = top;
         top = newNode;}
```

Script di atas merupakan metode "`push`" yang menambahkan elemen baru ke dalam *stack*. Metode ini membuat objek "`StackNode`" baru dengan data item yang diberikan. Simpul baru tersebut "`newNode`" kemudian diarahkan ke simpul teratas saat ini "`top`" melalui atribut "`next`". Setelah itu, "`top`" diperbarui untuk menunjuk ke simpul baru, menjadikannya elemen teratas di *stack*.

```
private PriorityTreeNode addCustomerRec(PriorityTreeNode node,
int nomorAntrian, String nama, String layanan, String prioritas,
String waktuMasuk) {

       if (node == null) {
              return new PriorityTreeNode(nomorAntrian, nama,
layanan, prioritas, waktuMasuk);
       }

       if (prioritas.equalsIgnoreCase("Iya")) {
           node.left = addCustomerRec(node.left, nomorAntrian,
nama, layanan, prioritas, waktuMasuk);

       } else {
           node.right = addCustomerRec(node.right, nomorAntrian,
nama, layanan, prioritas, waktuMasuk);
       }
        return node;
}
```

Script di atas merupakan metode rekursif "`addCustomerRec`" yang digunakan untuk menambahkan simpul baru ke dalam pohon prioritas berdasarkan atribut "`prioritas`". Jika simpul saat ini "`node = null`", metode membuat simpul baru menggunakan data yang diberikan.

```
public void displayByPriority() {
      System.out.println("| No. Antrian | Nama | Layanan |
Prioritas | Waktu Masuk |");
```

```
      System.out.println("----------------------------------
--------------------");
      preorderTraversal(root);}
```

Script di atas merupakan metode "displayByPriority" yang digunakan untuk menampilkan data dalam pohon prioritas. Metode ini memulai dengan mencetak *header* tabel yang berisi kolom seperti "No. Antrian, Nama, Layanan, Prioritas, dan Waktu Masuk". Setelah itu, metode memanggil "preorderTraversal" dengan argumen "root", yaitu akar pohon, untuk menampilkan data secara "pre-order traversal" kunjungan ke simpul akar diikuti oleh subpohon kiri dan subpohon kanan. Hal ini memastikan data ditampilkan sesuai urutan prioritas yang diatur dalam struktur pohon.

```
private void preorderTraversal(PriorityTreeNode node) {
    if (node != null) {
        System.out.println("| " + node.nomorAntrian + " | " +
node.nama + " | " + node.layanan + " | " + node.prioritas + " |
" + node.waktuMasuk + " |");
        preorderTraversal(node.left);
        preorderTraversal(node.right);
    }
}
```

Script di atas merupakan metode "preorderTraversal" yang digunakan untuk menelusuri dan menampilkan data dari pohon prioritas menggunakan algoritma "pre-order traversal". Method ini menerima simpul awal "node" sebagai parameter.

1.5.6 *Class Main*

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        linkedlist linkedList = new linkedlist();
        Stack stack = new Stack();
        Tree tree = new Tree();
        Scanner scanner = new Scanner(System.in);
```

Script di atas merupakan bagian dari program utama yang mengimpor beberapa kelas bawaan Java seperti "LocalDateTime" dan "DateTime Formatter" untuk menangani waktu, serta "Scanner" untuk menerima *input* dari pengguna. Dalam metode main, dibuat beberapa objek untuk berbagai struktur data, yaitu "linkedList" untuk mengelola *linked list*, "stack" untuk mengelola tumpukan (*stack*), dan "tree" untuk mengelola struktur pohon (*tree*). Selain itu, objek *Scanner* diinisialisasi untuk membaca input dari pengguna melalui konsol,

memungkinkan program berinteraksi dengan pengguna secara dinamis. Bagian ini berfungsi sebagai dasar untuk menghubungkan berbagai struktur data dan logika program lainnya.

```
int choice;
  do {
       System.out.println("====================================
====================");
       System.out.println("|                                  Qzy
                  ");
       System.out.println("====================================
====================");
      System.out.println("========== Menu ==========");
      System.out.println("1. Tambah Data Antrian");
      System.out.println("2. Tampilkan Antrian");
      System.out.println("3. Cari Antrian Berdasarkan Nama");
        System.out.println("4.  Urutkan  Antrian  Berdasarkan
Prioritas");
      System.out.println("5. Keluarkan Antrian");
      System.out.println("6. Tampilkan History");
      System.out.println("0. Keluar");
      System.out.print("Pilih menu: ");
      choice = scanner.nextInt();
      scanner.nextLine();
```

Script di atas adalah bagian dari program utama yang menampilkan menu interaktif untuk pengguna, memungkinkan mereka memilih operasi yang akan dijalankan pada sistem antrian. Menggunakan struktur "`do-while`", program terus menampilkan menu hingga pengguna memilih untuk keluar dengan memasukkan angka 0. Menu ini mencakup berbagai opsi: menambah data antrian (1), menampilkan seluruh antrian (2), mencari antrian berdasarkan nama (3), mengurutkan antrian berdasarkan prioritas (4), menghapus antrian (5), dan menampilkan riwayat operasi (6). Setelah menampilkan menu, program meminta input pengguna menggunakan "`scanner.nextInt()`" untuk membaca pilihan, diikuti oleh "`scanner.nextLine()`" untuk mengatasi sisa *input newline*.

```
switch (choice) {
 case 1:
     System.out.print("Masukkan Nomor Antrian: ");
     int nomorAntrian = scanner.nextInt;
     scanner.nextLine();
     System.out.print("Masukkan Nama: ");
     String nama = scanner.nextLine();
     System.out.print("Masukkan Layanan: ");
     String layanan = scanner.nextLine();
     System.out.print("Apakah Prioritas (Iya/Tidak): ");
     String prioritas = scanner.nextLine();
     String  waktuMasuk  =  LocalDateTime.now().format(DateTime
Formatter.ofPattern("HH:mm:ss yyyy-MM-dd"));
     linkedList.AddNode(nomorAntrian, nama, layanan, prioritas,
waktuMasuk);
      tree.addCustomer(nomorAntrian, nama, layanan, prioritas,
waktuMasuk);
```

```
    stack.push("Tambah Antrian - Nama: " + nama);
    System.out.println("Data berhasil ditambahkan.");
    break;
```

Script di atas adalah bagian dari program yang menangani penambahan data antrian ketika pengguna memilih opsi tertentu dalam menu. Program meminta input pengguna berupa nomor antrian, nama, layanan, dan status prioritas ("`Iya`" atau "`Tidak`"), lalu secara otomatis mencatat waktu masuk menggunakan "`LocalDateTime`" dengan format "`HH:mm:ss yyyy-MM-dd`". Data yang dimasukkan ditambahkan ke tiga struktur data: *linked list* melalui metode "`AddNode`", *tree* melalui metode "`addCustomer`", dan *stack* melalui metode "push" untuk mencatat riwayat tindakan dengan deskripsi "`Tambah Antrian - Nama: [Nama]`". Setelah semua data berhasil ditambahkan, program mencetak pesan konfirmasi bahwa data telah tersimpan. Bagian ini memungkinkan pengelolaan antrian secara terintegrasi dalam berbagai struktur data.

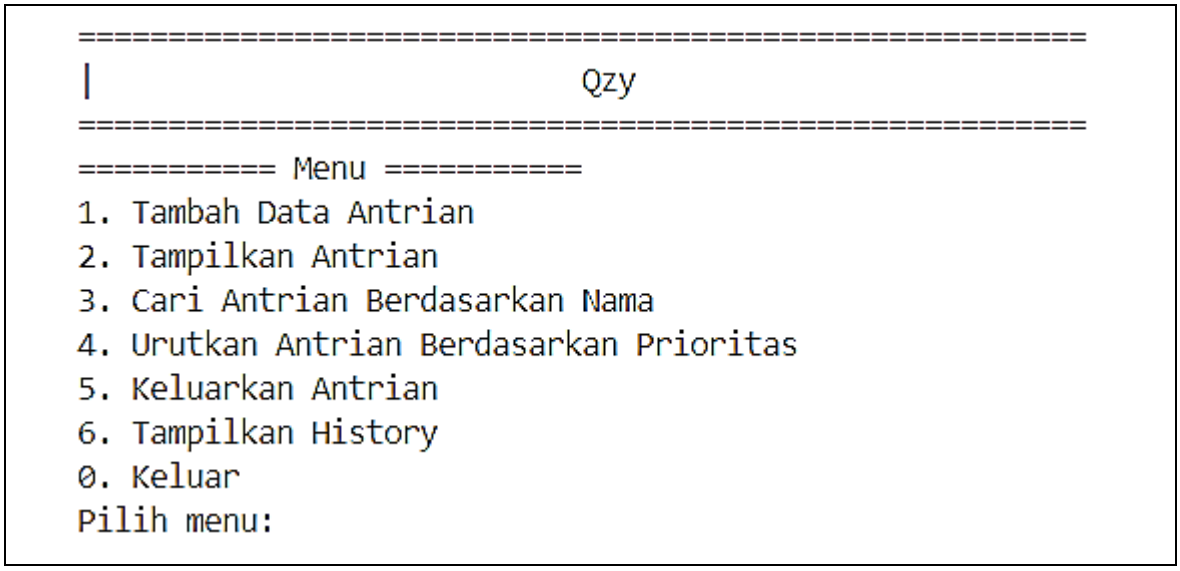
```
case 2:
 linkedList.display();
 break;

case 3:
 System.out.print("Masukkan Nama untuk Pencarian: ");
 String searchName = scanner.nextLine();
 Node foundNode = linkedList.searchByName(searchName);
 if (foundNode != null) {
     System.out.println("Data ditemukan: \n" +
       "Nomor Antrian: " + foundNode.nomorAntrian + "\n" +
             "Nama: " + foundNode.nama + "\n" +
             "Layanan: " + foundNode.layanan + "\n" +
             "Prioritas: " + foundNode.prioritas + "\n" +
             "Waktu Masuk: " + foundNode.waktuMasuk);
 } else {
     System.out.println("Data tidak ditemukan.");
 }
 break;
```

Script di atas adalah bagian dari program yang menangani operasi berdasarkan pilihan pengguna untuk menampilkan semua data atau mencari data tertentu dalam *linked list*. Pada *case* 2, program memanggil metode *display* dari *linked list* untuk menampilkan seluruh data antrian yang tersimpan. Pada *case* 3, program meminta input nama yang ingin dicari, lalu menggunakan metode "`searchByName`" dari *linked list* untuk mencari data berdasarkan nama. Jika data ditemukan, program mencetak informasi lengkap dari *node* terkait, termasuk nomor antrian, nama, layanan, status prioritas, dan waktu masuk. Jika tidak ditemukan, program mencetak pesan "`Data tidak ditemukan`". Bagian ini memungkinkan pengguna untuk melihat atau mencari data antrian secara spesifik.

```
case 4:
    System.out.println("Menampilkan     Antrian     Berdasarkan
Prioritas:");
    tree.displayByPriority();
    break;
case 5:
    System.out.println("Mengeluarkan Antrian:");

System.out.println("========================================
==============");
    Node removedNode = linkedList.dequeue(stack);
    linkedList.display();
    break;
case 6:
    System.out.println("\n========== History ==========");
    stack.displayHistory();
    break;
case 0:
    System.out.println("Keluar dari program.");
    break;

default:
    System.out.println("Pilihan   tidak   valid.   Silakan   coba
lagi.");
}

} while (choice != 0);

scanner.close();
}
```

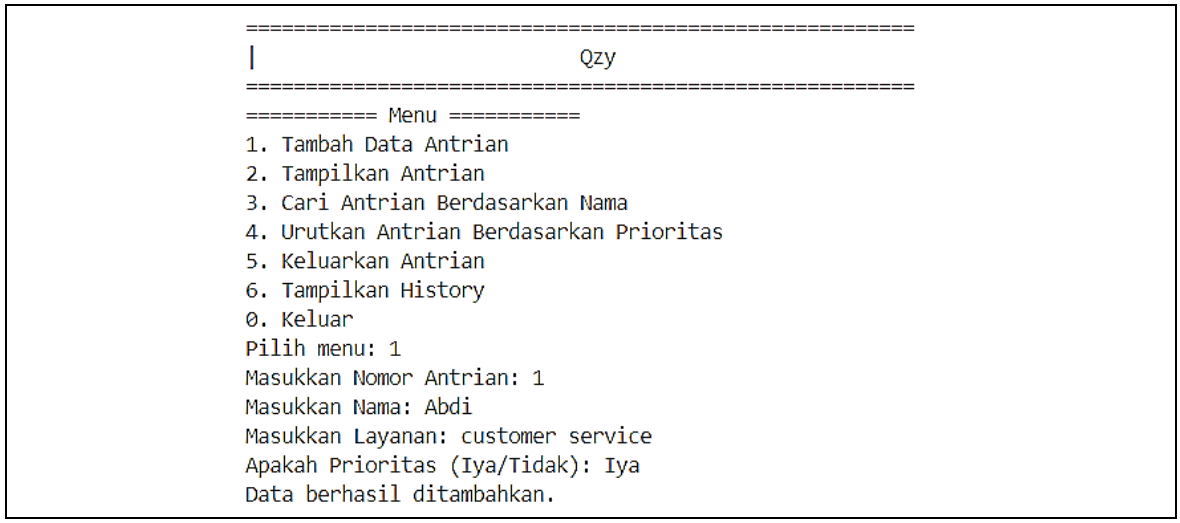
Script di atas adalah bagian akhir dari program yang menangani beberapa pilihan pengguna untuk operasi yang berbeda pada sistem antrian hingga pengguna memilih untuk keluar. Pada *case* 4, program memanggil metode "`displayByPriority`" dari struktur data *tree* untuk menampilkan antrian berdasarkan prioritas. Pada *case* 5, program menghapus elemen dari antrian menggunakan metode *dequeue* pada *linked list*, mencatat tindakan ke dalam *stack*, dan kemudian menampilkan daftar antrian yang tersisa. Pada *case* 6, program memanggil metode "`displayHistory`" dari stack untuk menampilkan riwayat operasi yang dilakukan. Jika pengguna memilih *case* 0, program mencetak pesan bahwa proses akan dihentikan dan keluar dari *loop*. Untuk input yang tidak valid, program mencetak pesan kesalahan. Setelah *loop* selesai, *scanner* ditutup untuk mengakhiri pembacaan input. Bagian ini memungkinkan pengguna mengelola antrian dengan berbagai operasi hingga memilih untuk keluar dari program.

1.6 OUTPUT PROGRAM



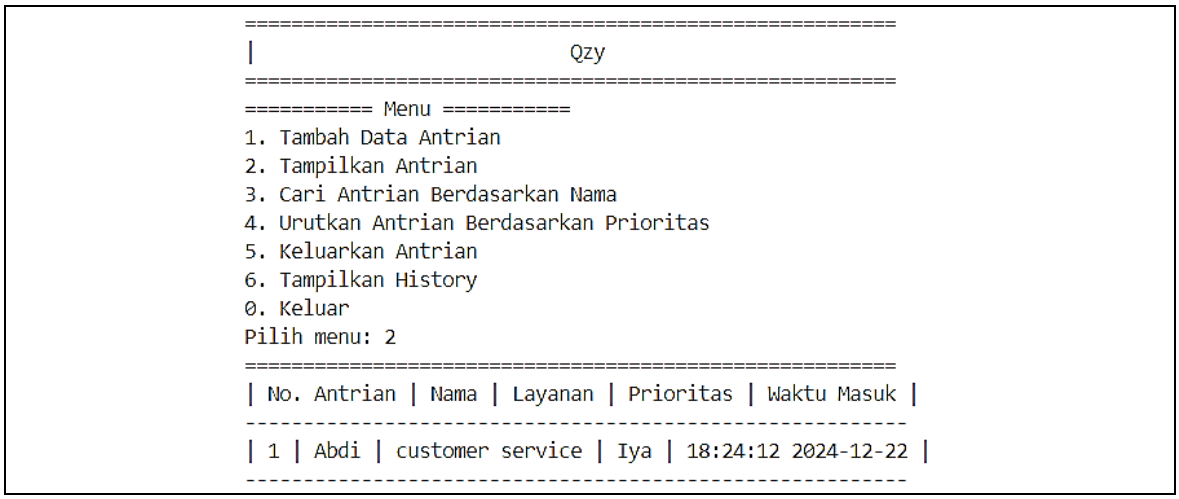
Gambar 1.1 Menu Utama

Berdasarkan **Gambar 1.1** di atas, dapat dilihat pada menu utama terdapat tujuh pilihan yang bisa dipilih oleh admin. Pilihan pertama digunakan untuk menambah nasabah dalam antrian, pilihan kedua akan menampilkan antrian yang masih ada, pilihan ketiga digunakan untuk mencari yang terdapat dalam antrian berdasarkan nama, pilihan keempat akan menampilkan daftar antrian berdasarkan prioritas, pilihan kelima digunakan untuk mengeluarkan nasabah dari antrian, keenam akan menampilkan histori apa saja yang dilakukan pada program, dan terakhir keluar digunakan untuk keluar dari program.



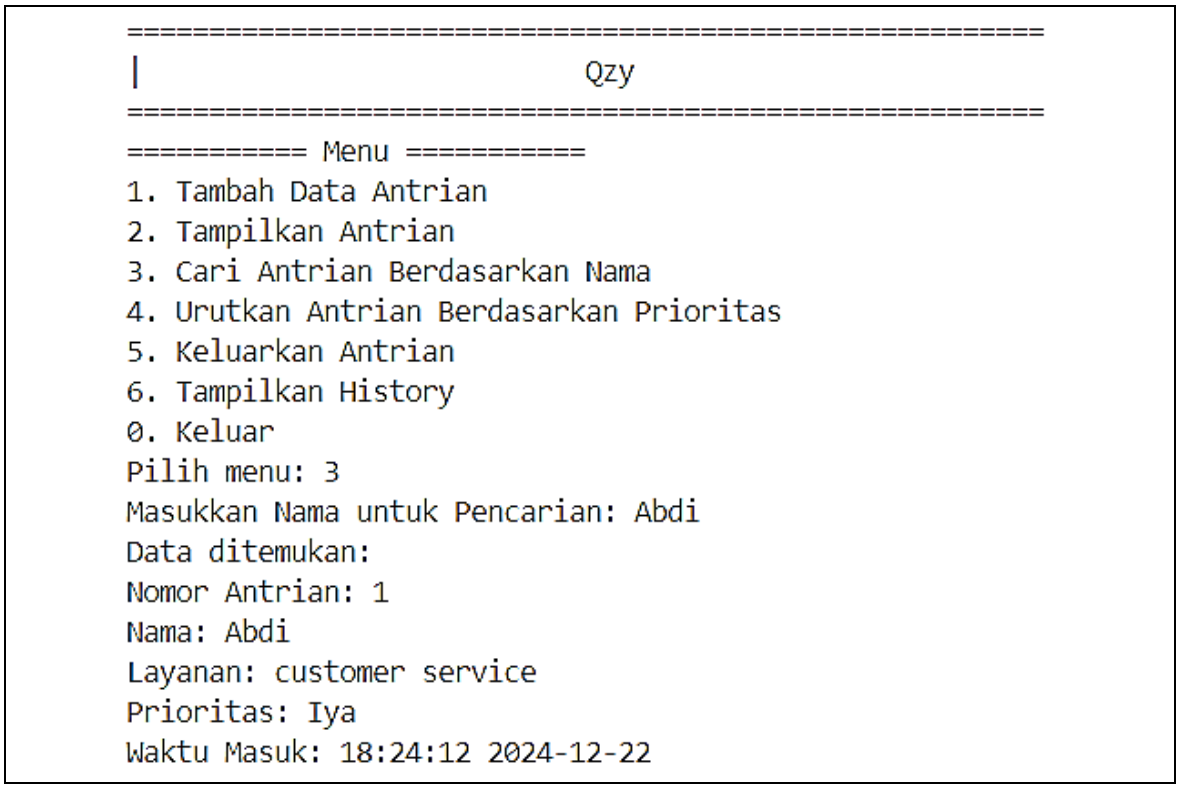
Gambar 1.2 Memilih Menu Pertama

Berdasarkan **Gambar 1.2** di atas, jika pengguna memilih pilihan pertama maka program akan meminta untuk memasukkan "`nomor antrian`", "`nama`", "`layanan`", dan apakah nasabah prioritas, serta program akan menampilkan pesan "`data berhasil ditambahkan`".



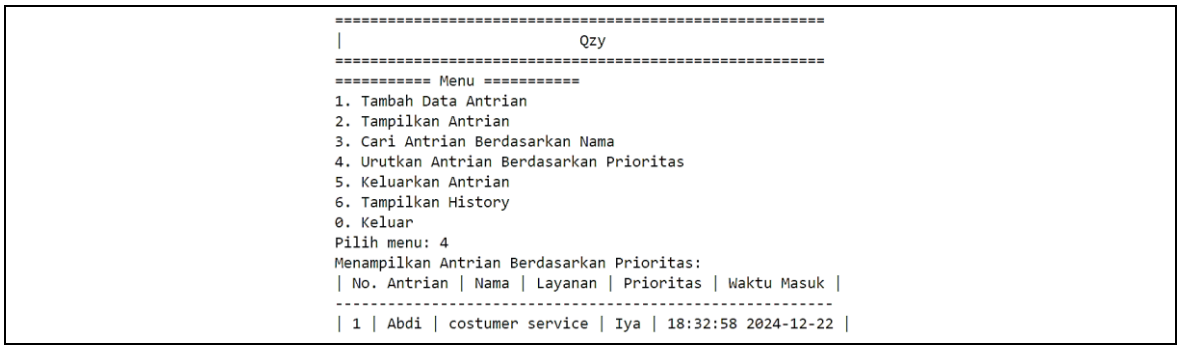
Gambar 1.3 Memilih Menu Kedua

Berdasarkan **Gambar 1.3** di atas, jika pengguna memilih menu kedua maka program akan menampilkan nasabah-nasabah yang berada dalam daftar antrian berdasarkan nomor antrian yang sudah didapatkan.



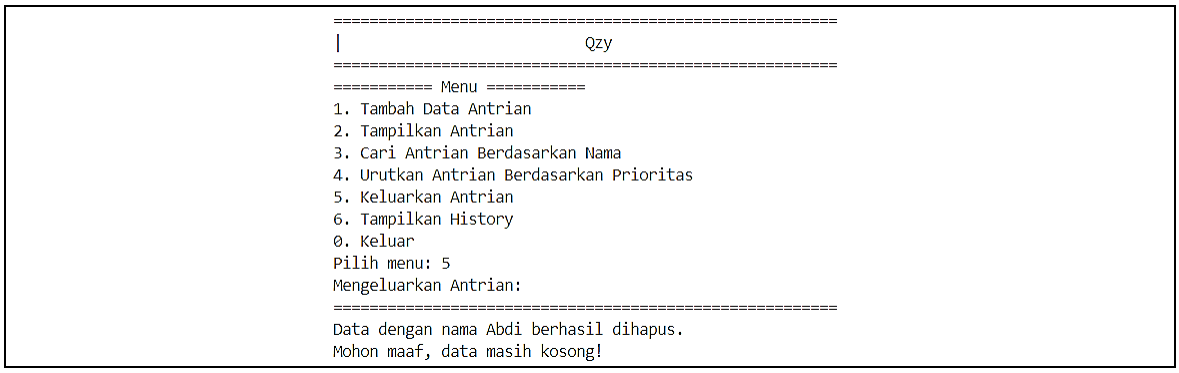
Gambar 1.4 Memilih Menu Ketiga

Berdasarkan **Gambar 1.4** di atas, jika pengguna memilih menu ketiga maka program akan meminta untuk memasukkan nama yang ingin dicari dan program akan menampilkan data-data yang dimasukkan pada menu pertama jika nama yang dicari ada pada antrian.



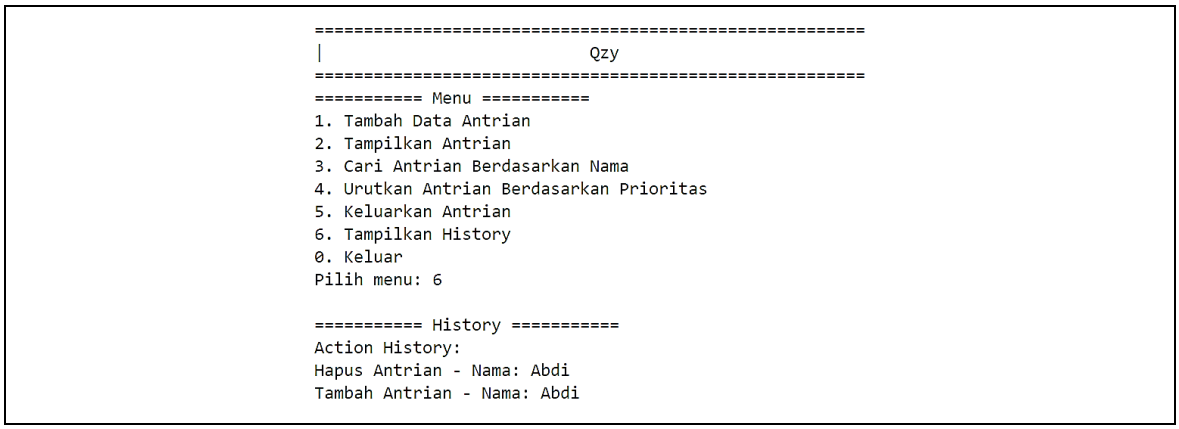
Gambar 1.5 Memilih Menu Keempat

Berdasarkan **Gambar 1.5** di atas, jika pengguna memilih opsi untuk menu 4 maka kode kan menampilkan antrian berdasarkan prioritas sesuai dengan metode pada *class tree* yang telah dibuat sebelumnya.



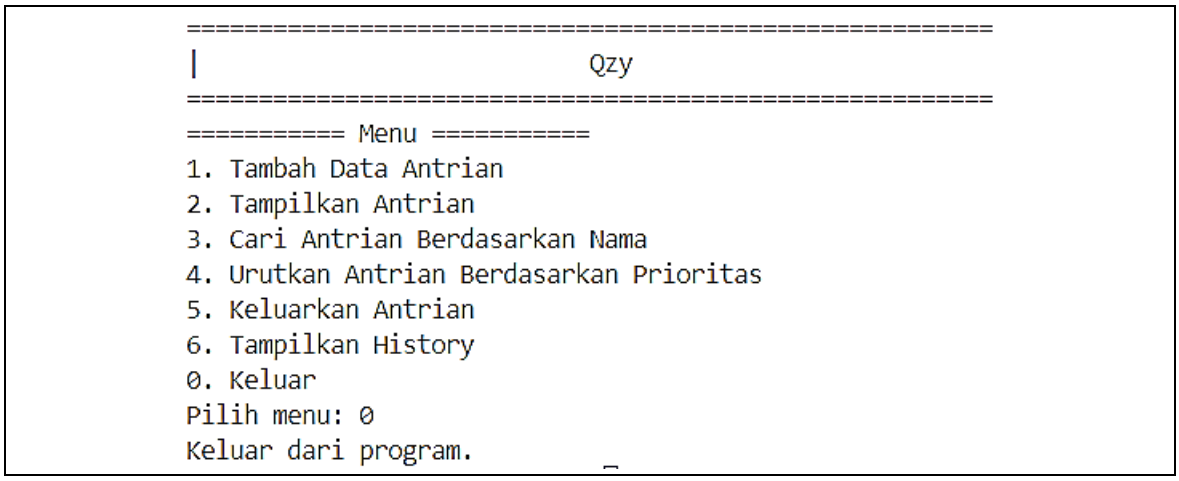
Gambar 1.6 Memilih Menu Kelima

Berdasarkan **Gambar 1.6** di atas, jika pengguna memilih menu kelima maka program akan mengeluarkan nasabah yang berada pada antrian paling depan dan program akan menempatkannya ke histori.



Gambar 1.7 Memilih Menu Keenam

Berdasarkan **Gambar 1.7** di atas, jika pengguna memilih opsi untuk menu 6 maka kode akan histori yang telah dilakukan oleh pengguna misalnya jika pengguna menambahkan dan ataupun menghapus antrian.



Gambar 1.8 Melakukan Redo

Berdasarkan **Gambar 1.8** di atas, jika pengguna memilih menu terakhir maka pengguna akan keluar dari program dan program akan menampilkan pesan "`Keluar dari program`" dan program berhenti berjalan.

1.7 KESIMPULAN

Berdasarkan hasil proyek praktikum Algoritma dan Struktur Data, dapat disimpulkan bahwa program Qzy: Sistem Manajemen Antrian Bank berhasil mengintegrasikan berbagai struktur data dan algoritma untuk menciptakan sistem pengelolaan antrian yang efisien dan terorganisir. Penggunaan struktur data seperti *queue, priority tree, linked list, stack,* dan *tree* memungkinkan sistem untuk mengatur antrian nasabah berdasarkan prioritas dan waktu kedatangan secara otomatis, sekaligus menyediakan fitur pencarian dan riwayat untuk fleksibilitas operasional. Dengan pendekatan ini, sistem tidak hanya meningkatkan efisiensi layanan bank, tetapi juga memberikan pengalaman yang lebih nyaman dan memuaskan bagi nasabah, menjadikan Qzy sebagai solusi ideal untuk permasalahan antrian di dunia perbankan.

1.8 REFERENSI

[1] R. Munir, Algoritma dan Pemrograman dalam Bahasa Java, 2nd ed. Bandung: Informatika, 2020, pp. 45-78.

[2] E. Komara, Struktur Data dan Algoritma. Jakarta: Erlangga, 2019, pp. 123-158.

[3] Elyzabeth. (2014). Aplikasi Model Antrian Multiserver dengan Vacation Pada Sistem Antrian di Bank BCA Cabang Ujung Berung. Universitas Pendidikan Indonesia