LAPORAN PROYEK PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA 2024

SISTEM PENGELOLAAN BARANG PADA GUDANG

Oleh Kelompok 23

Anggota:

Irfan Jayadi	**F1D02310011**
R. Rafi Yudi Pramana	**F1D02310132**
Syazwani	**F1D02310140**
Zahratu Syita	**F1D02310148**

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS MATARAM

2024

LEMBAR PENGESAHAN LAPORAN PROYEK
PRAKTIKUM ALGORITMA DAN STRUKTUR DATA 2024

1. Kelompok : 23
2. Judul Proyek : Sistem Pengelolaan Barang pada Gudang
3. Anggota Kelompok : Irfan Jayadi (F1D02310011)
   R. Rafi Yudi Pramana (F1D02310132)
   Syazwani (F1D02310140)
   Zahratu Syita (F1D02310148)

Laporan proyek ini disusun sesuai dengan kaidah penyusunan yang telah ditentukan dan dibuat sebagai syarat mata kuliah Algoritma dan Struktur Data 2024.

Mataram, 22 Desember 2024

Telah diperiksa dan disahkan oleh:

Koordinator Asisten	**Asisten Pembimbing**
Aditya Rahmatdiyansyah	**M. Ilham Abdul Shaleh**
F1D022031	**F1D022061**

1.1 Judul Proyek

Judul dari *project* praktikum algoritma dan struktur data ini adalah “Sistem Pengelolaan Barang pada Gudang”.

1.2 Latar Belakang

Dalam dunia bisnis, khususnya industri setiap petarung bisnis pasti mengorientasikan bisnisnya untuk mendapatkan keuntungan semaksimal mungkin dengan pengorbanan seminimal mungkin. Salah satu cara untuk mendapatkan keuntungan semaksimal mungkin adalah dengan melakukan mimalisisir penganggaran biaya pada proses operasi produksi nya salah satunya operasi pengecekan inventarisnya. Oleh karena itu, tiap instansi bisnis dituntut untuk memiliki suatu strategi optimasi untuk mengelola proses pengecekan gudangnya sebagai solusi pemecahan masalah.[1]

Maka pengelolaan stok barang adalah kegiatan yang penting bagi perusahaan, terlebih lagi masih menggunakan sistem manual seperti exel untuk mencatat data persediaan serta pengelolaan barang, hal ini akan menyulitkan administrasi gudang [2], serta proses tersebut belum berjalan secara optimal, informasi yang didapat masih menghambat kegiatan operasional dan informasi laporan manajemen pun menjadi simpang siur karena sering terlambatnya laporan data ke manajemen. [3] Maka diperlukan adalanya sistem informasi pengelolaan barang yang diharapkan mampu mengatasi permasalahan-permasalahan tersebut.

Dengan adanya sistem informasi “StokTrack,” proses pengelolaan barang menjadi lebih efisien dan efektif. Sistem ini dirancang untuk membantu pengguna dalam melakukan berbagai aktivitas yang berkaitan dengan manajemen barang, seperti penerimaan atau penambahan barang baru, mengganti data barang, melakukan pengiriman barang, serta menghapus data barang yang sudah tidak diperlukan. Dalam hal ini, "StokTrack" tidak hanya berfungsi sebagai alat untuk mencatat barang yang masuk dan keluar, tetapi juga memberikan kemudahan dalam memperbarui informasi barang secara *real-time.*

Salah satu tujuan utama dari sistem ini adalah untuk mempermudah pengguna dalam mengelola stok barang. Dengan fitur-fitur yang ada, pengguna dapat dengan cepat menambah atau mengubah informasi barang, seperti jumlah stok, harga satuan, atau kategori barang. Hal ini sangat berguna untuk memastikan bahwa data barang yang ada selalu akurat dan terbaru. Selain itu, proses pengiriman barang dapat dilakukan dengan lebih cepat dan terorganisir, sehingga tidak ada barang yang salah kirim atau tertinggal.

1.3 Deskripsi Program

Program “StokTrack” adalah sebuah sistem informasi yang dirancang untuk mengelola dan memantau persediaan barang dalam sebuah organisasi atau perusahaan. Sistem ini mengotomatisasi berbagai proses terkait pengelolaan barang, seperti penerimaan barang baru, penggantian data barang, pengiriman barang, serta penghapusan data barang yang tidak diperlukan. Tujuan utama dari program StokTrack adalah untuk meningkatkan efisiensi dan efektivitas pengelolaan stok barang, sekaligus memastikan bahwa data yang digunakan selalu akurat dan *real-time*.

Fitur utama yang di tawarkan oleh sistem “StokTrack” adalah “manajemen barang” dimana fitur ini pengguna dapat melakukan tambah barang baru, edit data barang dan hapus barang yang tidak diperlukan, kemudian terdapat fitur “lihat daftar barang” pengguna dapat melihat semua daftar barang yang ada digudang serta informasi ID barang, nama barang, jumlah stok, harga per unit serta tanggal masuk atau dii terima nya, pengguna juga dapat mencari barang berdasarkan ID barang serta mengurutkan barang berdasarkan ID barang, kemudian terdapat fitur “pengiriman barang” pengguna dapat membuat pengiriman barang, melihat riwayat pengiriman barang serta mencari pengiriman berdasarkan ID barang.

Selain itu sistem ini dapat diakses dengan mudah dan *user-friendly* dengan menyediakan navigasi yang terstruktur akan mempermudah pengguna menemukan fitur yang ingin digunakan dan memudahkan pengguna dalam mengoperasikan sistem

1.4 Algoritma

1.5 Penjelasan Code

1.5.1 *Class Node*

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

    Node(Barang data) {
        this.data = data;
        this.next = null;
    }
}
```

Script di atas merupakan *class* "`Node`", dimana kelas ini menyimpan objek dari kelas barang dan referensi *next* untuk ke *node* berikutnya. Saat sebuah *node* dibuat, data di isi dengan objek "`Barang`", sedangkan referensi *next* di atur sebagai *null* karena nilai nya belum ditentukan.

1.5.2 *Class* Barang

```
public class Barang {
    String id;
    String nama;
    String kategori;
    int jumlah_stok;
    int harga_satuan;
    String tanggal_diterima;

    Barang(String id, String nama, String kategori, int
jumlah_stok, int harga_satuan, String tanggal_diterima) {
        this.id = id;
        this.nama = nama;
        this.kategori = kategori;
        this.jumlah_stok = jumlah_stok;
        this.harga_satuan = harga_satuan;
        this.tanggal_diterima = tanggal_diterima;
    }
}
```

Script di atas merupakan *class* "`Barang`", dimana kelas ini digunakan untuk merepresentasikan data barang. Setiap atribut barang diinisialisasi seperti "`id, nama, kategori, jumlah_stok, harga_satuan, tanggal_diterima`", kemudian *script* "`Barang(String id, String nama, String kategori, int jumlah_stok, int harga_satuan, String tanggal_diterima)`" adalah konstruktor untuk mengisi semua atribut atribut tersebut saat barang ditambahkan.

1.5.3 *Class LinkedList*

```
public void tambah(Barang barang) {
        Node newNode = new Node(barang);
        if (head == null) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
```

```
            }
            current.next = newNode;
        }
    }
```

Script di atas merupakan *class* "`LinkedList`", di mana digunakan untuk mengelola barang di gudang, *method* "`tambah`" digunakan untuk menambah barang ke daftar, jika daftar kosong, *node* baru langsung menjadi *node* pertama atau "`head`" sedangkan jika tidak kosong, maka akan memeriksa hingga *node* terakhir lalu menambah *node* baru di akhir.

```
public void hapus(String id) {
        if (head == null) {
            return;
        }
        if (head.data.id.equals(id)) {
            head = head.next;
            return;
        }
        Node current = head;
        while (current.next != null) {
            if (current.next.data.id.equals(id)) {
                current.next = current.next.next;
                return;
            }
            current = current.next;
        }
    }
```

Script "`hapus(String id)`" menghapus *node* dalam *linked list* berdasarkan ID. Pertama, jika *head null*, fungsi langsung keluar dan tidak ada yang perlu dihapus. Jika ID *node* pertama sesuai, *head* diperbarui menjadi *node* berikutnya. Selanjutnya, fungsi melintasi *linked list* menggunakan *pointer current*. Saat menemukan *node* dengan ID yang sesuai, ia memodifikasi

```
public void terima(Barang barang) {
        Node current = head;
        while (current != null) {
            if (current.data.id.equals(barang.id)) {
                        current.data.jumlah_stok           +=
barang.jumlah_stok;
                return;
            }
            current = current.next;
        }
        tambah(barang);
    }
```

Script "`terima(Barang barang)`" digunakan untuk menambah data barang, Pertama, ia memulai dari *node* pertama (*head*) dan berjalan melalui *linked list* menggunakan *pointer* "`current`". Selama traversal, ia memeriksa apakah ID barang yang diterima "`barang.id`" sudah ada di dalam *linked list*. Jika ditemukan, stok barang pada *node* yang sesuai diperbarui dengan menambahkan nilai

"`jumlah_stok`" dari barang yang diterima, lalu fungsi keluar. Jika ID tidak ditemukan setelah traversal selesai, fungsi "`tambah(barang)`" dipanggil untuk menambahkan barang sebagai *node* baru ke dalam *linked list*

```
public void update(String id, String field, Object newValue) {
        Node current = head;

        while (current != null) {
            if (current.data.id.equals(id)) {
                switch (field.toLowerCase()) {
                    case "nama":
                        if (newValue instanceof String) {
                              current.data.nama    =    (String)
newValue;
                        } else {
                                              throw              new
IllegalArgumentException("Inputan  salah,  ulangi  dengan  nilai
yang benar.");
                        }
                        break;
                    case "stok":
                        if (newValue instanceof Integer) {
                             current.data.jumlah_stok  =  (int)
newValue;
                        } else {
                                              throw              new
IllegalArgumentException("Inputan  salah,  ulangi  dengan  nilai
yang benar.");
                        }
                        break;
                    case "tanggal":
                        if (newValue instanceof String) {
                              current.data.tanggal_diterima   =
(String) newValue;
                        } else {
                                              throw              new
IllegalArgumentException("Inputan  salah,  ulangi  dengan  nilai
yang benar.");
                        }
                        break;
                    default:
                                           throw                 new
IllegalArgumentException("Field  tidak  valid!  Gunakan  'nama',
'stok', atau 'tanggal'.");
                }
                return;
            }
            current = current.next;
        }

          throw    new    RuntimeException("ID    Barang    tidak
ditemukan.");
    }
```

Script "`update(String id, String field, Object newValue)`" digunakan untuk memperbarui data barang dalam *linked list* berdasarkan ID. Fungsi mencari *node* dengan ID yang cocok, lalu memeriksa "`field`" yang akan diubah menggunakan "`switch-case`". Pembaruan dilakukan hanya jika tipe data

nilai baru "newValue" sesuai "String" untuk "nama" dan "tanggal", serta "Integer" untuk stok. Jika ID tidak ditemukan, fungsi memanggil "RuntimeException" untuk memberi pesan IDE tidak ditemukan, dan jika "field" atau tipe data tidak *valid*, fungsi memangil "IllegalArgumentException" untuk memberi validasi *input* yang dimasukkan *valid* atau tidak.

```
public void tampilkan() {
        System.out.printf("+------------+------------------+---
---------+----------------+------------------+\n");
        System.out.printf("| ID Barang | Nama Barang      | Stok
Barang | Harga Per Unit | Tanggal Diterima |\n");
        System.out.printf("+------------+------------------+---
---------+----------------+------------------+\n");

        Node current = head;
        while (current != null) {
            System.out.printf("| %-10s | %-16s | %-10d | %-14d
| %-16s |\n",
                    current.data.id,     current.data.nama,
current.data.jumlah_stok,
                                    current.data.harga_satuan,
current.data.tanggal_diterima);
            current = current.next;
        }
        System.out.printf("+------------+------------------+---
---------+----------------+------------------+\n");
    }
```

Scipt "tampilkan()" digunakan untuk menampilkan data barang. Format tabel dibuat untuk menampilkan data barang yang di *input* pengguna dengan menggunakan "System.out.printf", kemudian fungsi berjalan dengan *pointer* "current" dan mencetak setiap *node* dalam format baris tabel dengan data barang.

```
public void sort(String field) {
        if (head == null || head.next == null) {
            return;
        }
        head = mergeSort(head, field);
    }
```

Script "sort(String field)" digunakan untuk mengurutkan data berdasarkan kolom yang ditentukan oleh parameter "field". Jika data kosong atau hanya memiliki satu *node*, fungsi langsung keluar. Jika tidak, fungsi memanggil metode "mergeSort" untuk mengurutkan *node* dan memperbarui kepala "head" dengan hasil yang terurut

```
private Node mergeSort(Node head, String field) {
        if (head == null || head.next == null) {
            return head;
        }
```

```
        Node middle = getMiddle(head);
        Node nextOfMiddle = middle.next;
        middle.next = null;

        Node left = mergeSort(head, field);
        Node right = mergeSort(nextOfMiddle, field);

        return sortedMerge(left, right, field);
    }
```

Script "`mergeSort(Node head, String field)`" adalah implementasi algoritma *merge sort* untuk mengurutkan *linked list* berdasarkan kolom yang ditentukan oleh parameter "`field`". fungsi apakah *list* kosong atau hanya memiliki satu *node*, jika ya, kembalikan *node* tersebut. Jika tidak, fungsi mencari *node* tengah menggunakan "`getMiddle()`", membagi *list* menjadi dua bagian, dan kemudian memanggil "`mergeSort`" secara rekursif untuk mengurutkan kedua bagian (kiri dan kanan). Setelah itu, fungsi menggabungkan kedua bagian yang sudah terurut menggunakan metode "`sortedMerge()`" dan mengembalikan hasilnya.

```
private Node getMiddle(Node head) {
        if (head == null) {
            return head;
        }
        Node slow = head, fast = head.next;
        while (fast != null) {
            fast = fast.next;
            if (fast != null) {
                slow = slow.next;
                fast = fast.next;
            }
        }
        return slow;
    }
```

Script "`getMiddle(Node head)`" mencari *node* tengah *linked list* dengan menggunakan dua pointer yaitu "`slow`" yang bergerak satu langkah dan "`fast`" yang bergerak dua langkah. Ketika "`fast`" mencapai akhir, "`slow`" berada di tengah dan dikembalikan sebagai hasil. Jika *linked list* kosong, fungsi mengembalikan *null.*

```
private Node sortedMerge(Node left, Node right, String field) {
        if (left == null) {
            return right;
        }
        if (right == null) {
            return left;
        }

        Node result;
        if (compare(left.data, right.data, field) <= 0) {
            result = left;
            result.next = sortedMerge(left.next, right, field);
```

```
        } else {
            result = right;
            result.next = sortedMerge(left, right.next, field);
        }
        return result;
    }
```

Script "`sortedMerge(Node left, Node right, String field)`" menggabungkan dua *list* yang sudah terurut berdasarkan kolom yang ditentukan oleh *field*. Fungsi membandingkan data pada *node left* dan *right* menggunakan metode "`compare()`", dan memilih *node* dengan nilai yang lebih kecil untuk menjadi bagian dari hasil yang terurut. Fungsi ini dipanggil secara rekursif untuk menggabungkan sisa *node* yang belum diproses, hingga semua *node* digabungkan dan mengembalikan data yang terurut.

```
private int compare(Barang a, Barang b, String field) {
        switch (field.toLowerCase()) {
            case "nama":
                return a.nama.compareTo(b.nama);
            case "stok":
                    return      Integer.compare(a.jumlah_stok,
b.jumlah_stok);
            case "tanggal":
                                                          return
a.tanggal_diterima.compareTo(b.tanggal_diterima);
            default:
                throw new IllegalArgumentException("Field tidak
valid! Gunakan 'nama', 'stok', atau 'tanggal'.");
        }
    }
```

Script "`compare(Barang a, Barang b, String field`" membandingkan dua objek barang berdasarkan kolom yang ditentukan oleh parameter *field*. Fungsi menggunakan *switch* untuk memilih kolom yang sesuai nama, stok, atau tanggal dan membandingkan nilai dari kedua objek barang sesuai dengan tipe data masing-masing. *Script* "`CompareTo()`" untuk String nama dan tanggal dan "`Integer.compare()`" untuk "`int`" (stok).

```
public Barang searchById(String id) {
        Node current = head;
        while (current != null) {
            if (current.data.id.equals(id)) {
                return current.data;
            }
            current = current.next;
        }
        return null;
```

Script "`searchById(String id)`" encari objek Barang berdasarkan. Fungsi ini berjalan melalui setiap *node*, memeriksa apakah ID pada *node* tersebut sesuai dengan parameter "`id`". Jika ditemukan, objek barang yang sesuai maka nilai akan

dikembalikan. Jika tidak ditemukan setelah memeriksa semua *node*, fungsi akan mengembalikan *null*.

```
public int getSize() {
        int size = 0;
        Node current = head;
        while (current != null) {
            size++;
            current = current.next;
        }
        return size;
    }
}
```

Script "`getSize`" digunakan untuk menghitung jumlah *node*. Fungsi ini dimulai dengan variabel "`size`" yang diinisialisasi ke 0, lalu berjalan melalui setiap *node* dengan pointer "`current`". Setiap kali melintasi satu *node*, variabel "`size`" akan bertambah satu. Setelah mencapai akhir *linked list*, fungsi mengembalikan nilai "`size`" yang menunjukkan jumlah total *node*.

1.5.4 *Class Queue*

```
void enqueue(Barang barang) {
        Node newNode = new Node(barang);
        if (rear == null) {
            front = rear = newNode;
            return;
        }
        rear.next = newNode;
        rear = newNode;
  }
```

Script "`enqueue (Barang barang)`" menambahkan objek barang ke dalam *queue*. Jika *queue* kosong, *node* baru menjadi elemen pertama dan terakhir. Jika tidak, *node* baru ditambahkan di akhir dan "`rear`" diperbarui untuk menunjuk ke *node* baru sebagai elemen terakhir.

```
Barang dequeue() {
        if (front == null) return null;
        Barang barang = front.data;
        front = front.next;
        if (front == null) rear = null;
        return barang;
 }
```

Script "`dequeue`" menghapus dan mengembalikan objek barang dari elemen depan *queue*. Jika *queue* kosong, fungsi mengembalikan *null*. Jika tidak, data pada *node* depan disalin, kemudian pointer "`front`" diperbarui ke *node* berikutnya.

```
void tampilkan() {
        if (front == null) {
            System.out.println("Antrian kosong.");
            return;
        }
        Node temp = front;
        int posisi = 1;
```

```
        while (temp != null) {
            System.out.println(posisi + ". " + temp.data.nama +
" (ID: " + temp.data.id + ")");
            temp = temp.next;
            posisi++;
        }
    }
```

Script "`tampilkan()`" menampilkan seluruh elemen dalam *queue*. Jika *queue* kosong, fungsi akan mencetak pesan "`Antrian kosong`". Jika tidak, fungsi akan berjalan melalui setiap *node* mulai dari "`front`", menampilkan informasi nama dan ID barang pada setiap posisi antrian. Posisi antrian ditampilkan dengan menghitung urutan *node* yang sedang diperiksa, dan setelah mencetak setiap elemen, pointer "`temp`" bergerak ke *node* berikutnya hingga akhir *queue*.

```
boolean hapus(String idBarang) {
        if (front == null) return false;
        if (front.data.id.equals(idBarang)) {
            front = front.next;
            if (front == null) rear = null;
            return true;
        }
        Node current = front;
        while (current.next != null &&
!current.next.data.id.equals(idBarang)) {
            current = current.next;
        }
        if (current.next == null) return false;
        current.next = current.next.next;
        if (current.next == null) rear = current;
        return true;
    }
}
```

Script "`hapus(String idBarang)`" menghapus barang dari *queue* berdasarkan ID. Jika *queue* kosong, fungsi mengembalikan *false*. Jika barang yang akan dihapus berada di depan antrian "`front`", maka "`front`" diperbarui ke *node* berikutnya, dan jika *queue* menjadi kosong, "`rear`" juga diatur ke *null*. Jika tidak, fungsi mencari barang dengan ID yang sesuai, menghapusnya dari antrian, dan memperbarui pointer "`rear`" jika elemen yang dihapus adalah yang terakhir. Fungsi mengembalikan *true* jika barang berhasil dihapus, dan *false* jika barang dengan ID yang diberikan tidak ditemukan.

1.5.5 *Class Stack*

```
public class Stack {
    private Node top;

    private class Node{
        Barang data;
        Node next;

        Node(Barang data){
```

```
            this.data = data;
            this.next = null;
        }
    }

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

Script di atas merupakan *class* "`Stack`", dimana kelas ini digunakan untuk menyimpan dan mengelola riwayat pengiriman barang dengan operasi LIFO (*Last In First Out*). Variabel "`top`" sebagai penunjuk elemen teratas. Di dalamnya terdapat kelas "*`Node`*" untuk merepresentasikan elemen *stack*, yang menyimpan data barang "`(Barang data)`" dan referensi ke *node* berikutnya "`(Node next)`". Konstruktor "`Stack()`" menginisialisasi *stack* sebagai *null* dengan "`top = null`".

```
public void push(Barang barang){
   Node newNode = new Node(barang);
   if(top == null){
      top = newNode;
   }else{
      newNode.next = top;
      top = newNode;
     }
      }
```

Script "`push(Barang barang`", menambahkan barang baru ke atas *stack*. Fungsi membuat *node* baru dengan data barang yang diberikan, dan jika *stack* kosong "`(top == null)`", *node* tersebut menjadi elemen pertama. Jika tidak, *node* baru ditempatkan di atas *stack* dengan menghubungkannya ke *node* yang sebelumnya berada di posisi top, lalu memperbarui top ke *node* baru.

```
public Barang pop(){
        if(top == null){
            System.out.println("Stack kosong, tidak ada barang
yang dikeluarkan");
            return null;
        }
        Barang barang = top.data;
        top = top.next;
        return barang;
    }
```

Script "`pop()`" menghapus dan mengembalikan barang dari atas stack. Jika stack kosong "`(top == null)`", fungsi mencetak pesan "`Stack kosong, tidak ada barang yang dikeluarkan`" dan mengembalikan *null*. Jika tidak, data dari *node* teratas "`(top.data)`" disimpan dalam variabel barang, pointer top diperbarui ke *node* berikutnya, dan data barang yang dihapus dikembalikan.

```
public void tampilkan(){
        if(top == null){
            System.out.println("Kosong");
            return;
```

```
        }
        Node current = top;
        System.out.println("Riwayat Pengiriman dan Penerimaan
Barang ");
        while(current != null){
            System.out.println("ID: " + current.data.id + ",
nama: " + current.data.nama +
            ", kategori: " + current.data.kategori +
            ", jumlah stok: " + current.data.jumlah_stok +
            ", Harga Satuan: " + current.data.harga_satuan);
            current = current.next;
        }
    }
}
```

Script "`tampilkan()`" menampilkan semua barang dalam *stack* dari atas ke bawah. Jika *stack* kosong "`(top == null)`", fungsi mencetak "`Kosong`" dan keluar. Jika tidak, fungsi mencetak informasi setiap barang "`(ID, nama, kategori, jumlah stok, dan harga satuan)`" dengan berjalan melalui setiap *node* mulai dari top hingga akhir stack.

1.5.6 *Class Tree*

```
public class Tree {
    private class Node {
        String kategori;
        LinkedList barangList;
        Node left, right;

        Node(String kategori) {
            this.kategori = kategori;
            this.barangList = new LinkedList();
            this.left = this.right = null;
        }
    }

    private Node root;

    public void insertKategori(String kategori) {
        root = insertKategoriRecursive(root, kategori);
    }

    private Node insertKategoriRecursive(Node root, String
kategori) {
        if (root == null) {
            root = new Node(kategori);
            return root;
        }

        if (kategori.compareTo(root.kategori) < 0) {
            root.left = insertKategoriRecursive(root.left,
kategori);
        } else if (kategori.compareTo(root.kategori) > 0) {
            root.right = insertKategoriRecursive(root.right,
kategori);
        }

        return root;
    }
```

```
    public void tambahBarang(Barang barang) {
        Node kategoriNode = searchKategori(barang.kategori);
        if (kategoriNode != null) {
            kategoriNode.barangList.tambah(barang);
        } else {
            System.out.println("Kategori tidak ditemukan!
Silakan tambahkan kategori terlebih dahulu.");
        }
    }

    private Node searchKategori(String kategori) {
        return searchKategoriRecursive(root, kategori);
    }

    private Node searchKategoriRecursive(Node root, String
kategori) {
        if (root == null || root.kategori.equals(kategori)) {
            return root;
        }
        if (kategori.compareTo(root.kategori) < 0) {
            return searchKategoriRecursive(root.left,
kategori);
```

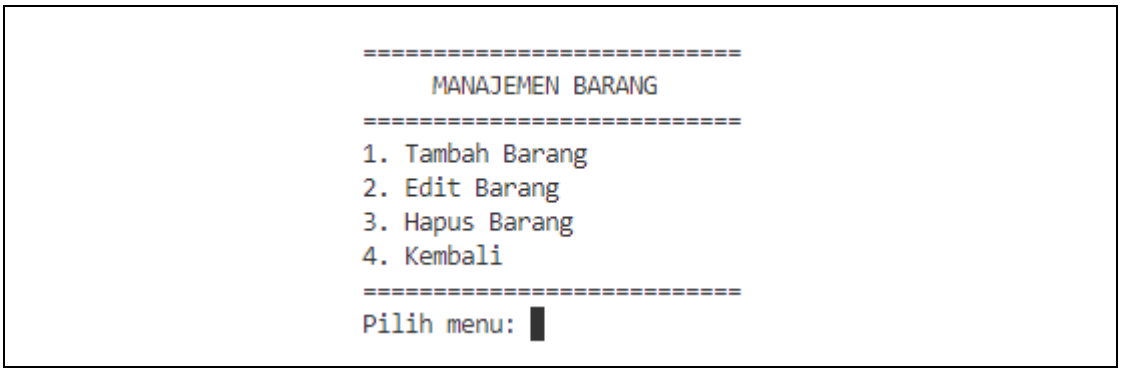
1.6 Output Program

Berikut merupakan *output* dari program sistem pengelolaan barang pada gudang yang telah kami buat:

```
============================
        STOCK TRACK
============================
1. MANAJEMEN BARANG
2. LIHAT DAFTAR BARANG
3. PENGIRIMAN BARANG
4. EXIT
============================
Pilih menu:
```

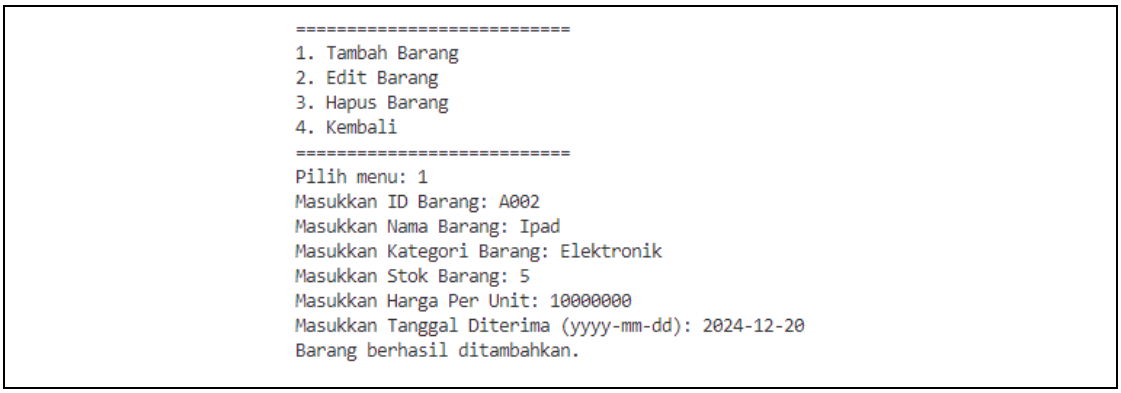
Gambar 1.1 Tampilan Menu Utama

Pada gambar di atas menampilkan menu utama, menu *stock track* pada sistem ini memudahkan pengguna untuk mengelola dan memantau stok barang. Melalui submenu lihat barang, pengguna dapat melihat daftar barang beserta jumlah stok yang ada. Menu pengiriman barang akan mengurangi stok sesuai dengan jumlah barang yang dikirim. *Exit* pada sistem ini digunakan untuk keluar dari aplikasi. Ketika pengguna memilih opsi ini, aplikasi akan menutup dan mengakhiri sesi yang sedang berjalan.



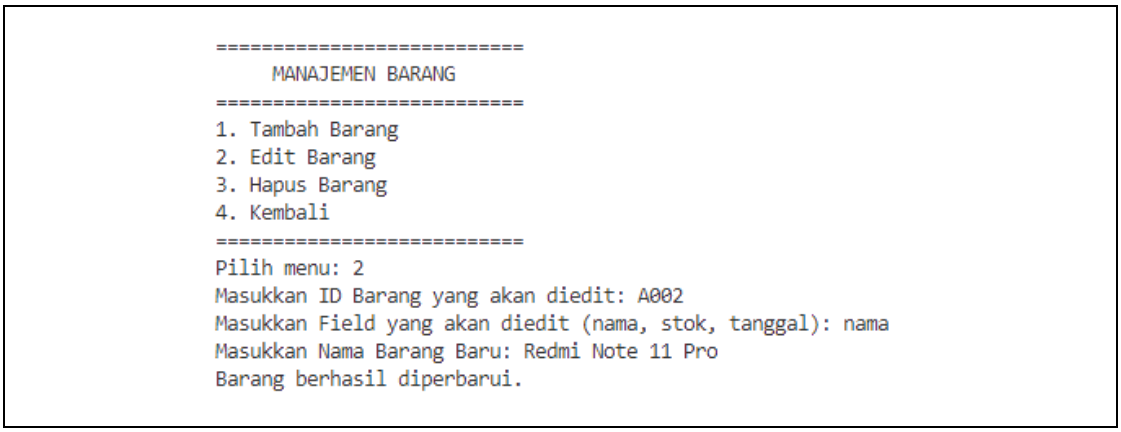
Gambar 1.2 Tampilan Menu Manajemen Barang

Pada gambar di atas menampilkan menu manajemen barang pengguna untuk mengelola barang dalam sistem secara efektif. Di dalam menu ini, pengguna dapat menambahkan barang baru dengan memasukkan informasi seperti ID, nama, kategori, stok, harga, dan tanggal diterima. Selain itu, pengguna juga dapat mengedit informasi barang yang sudah ada, menghapus barang berdasarkan ID, serta melihat daftar barang yang ada. Fitur pencarian dan sortir memungkinkan pengguna untuk menemukan barang tertentu atau mengurutkan barang berdasarkan kategori atau atribut lainnya. Menu ini memberikan kemudahan untuk mengatur dan memantau stok barang secara terorganisir.



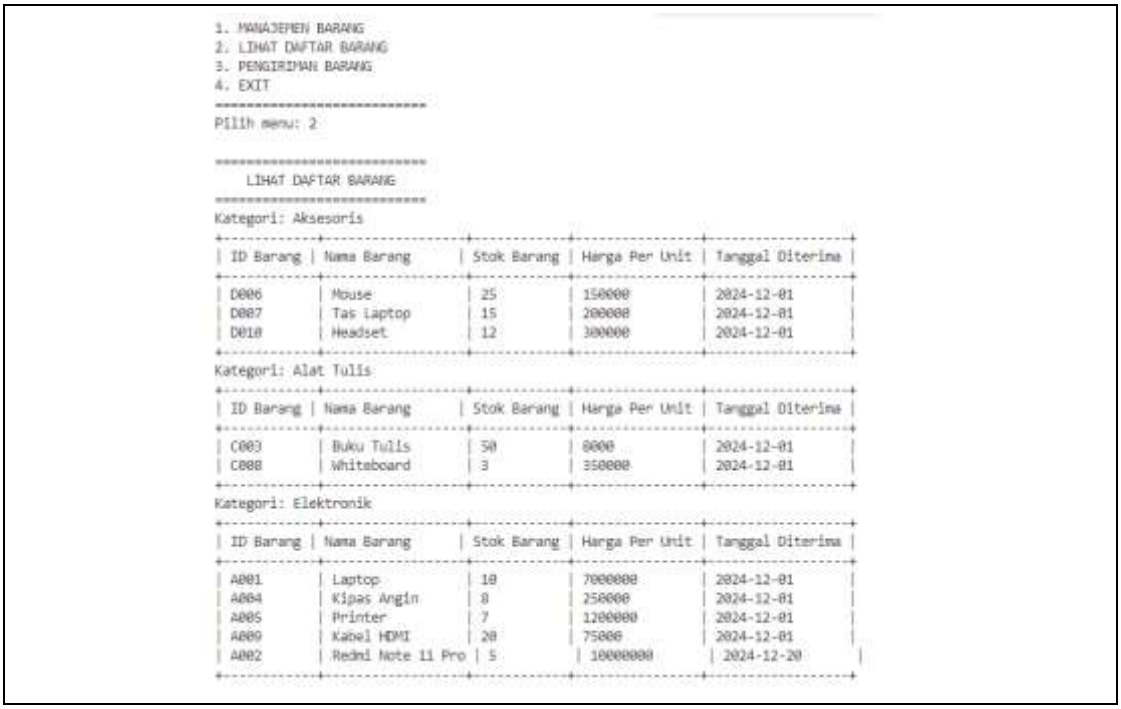
Gambar 1.3 Tampilan Menu

Pada gambar di atas menampilkan menu tambah barang menyediakan pengguna untuk menambahkan barang baru ke sistem dengan detail lengkap seperti ID, nama, kategori, jumlah stok, harga, dan tanggal diterima. Menu edit barang memberi pengguna untuk memperbarui informasi barang yang sudah ada, seperti nama, stok, atau tanggal barang, berdasarkan ID yang diberikan. Menu hapus barang menyediakan pengguna untuk menghapus barang yang ada di sistem dengan memasukkan ID barang yang ingin dihapus. Terakhir, menu kembali memberikan opsi untuk kembali ke menu sebelumnya tanpa melakukan perubahan apapun, sehingga pengguna bisa kembali ke layar utama atau menu sebelumnya.



Gambar 1.3 Tampilan Menu Manajemen Barang

Pada gambar di atas menampilkan menu manajemen barang adalah bagian dari menu yang menyediakan pengguna untuk mengelola data barang dalam inventaris. Menu ini terdiri dari beberapa pilihan, seperti tambah barang, edit barang, hapus barang, dan kembali. Dengan menu ini, pengguna dapat menambahkan barang baru ke dalam sistem, mengubah informasi barang yang sudah ada, atau menghapus barang yang tidak lagi dibutuhkan. Terdapat juga opsi untuk kembali ke menu sebelumnya. Menu manajemen barang ini membantu dalam mengatur dan memperbarui data barang secara efisien, memastikan informasi inventaris tetap terorganisir dan *up-to-date*.



Gambar 1.4 Hasil *Run* Program

Pada gambar di atas menampilkan hasil *output* dari program.

Gambar 1.5

Pada gambar di atas menampilkan hasil *output* dari program.

Gambar 1.6

Pada gambar di atas menampilkan hasil *output* dari program.

Gambar 1.7

Pada gambar di atas menampilkan hasil *output* dari program.

Gambar 1.8

Pada gambar di atas menampilkan hasil *output* dari program.

Gambar 1.9

Pada gambar di atas menampilkan hasil *output* dari program.

1.7 Kesimpulan

Kesimpulan dari hasil proyek praktikum Algoritma dan Struktur Data, dapat disimpulkan bahwa

1.8 Referensi

[1] Hikam, "Manajemen Pengecekan Inventaris Perusahaan Berbasis Program Dinamis,". Departemen Teknik Informatika, Institut Teknologi Bandung, Jl. Ganesha 10, Bandung, 2007.

[2] T. Handayani, A. H. Furqon, and S. Supriyono, "Rancang Bangun Sistem Inventori Pengendalian Stok Barang Berbasis Java Pada PT Kalibesar Artah Perkasa," *Eranda*, vol. 3, no. 1, pp. 1-10, 2020.

[3] J. Junaidi, R. Setianingsih, and K. Khotimah, "Rancang Bangun Sistem Penerimaan dan Pengeluaran Barang Menggunakan Java Aplikasi," *Oct. 2015*.