LAPORAN AKHIR PRAKTIKUM
ALGORITMA PEMROGRAMAN DAN STRUKTUR DATA

Oleh:

Muhammad Ikram Mughni / 124240191

PROGRAM STUDI SISTEM INFORMASI

JURUSAN INFORMATIKA

FAKULTAS TEKNIK INDUSTRI

UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN"

YOGYAKARTA

2025

HALAMAN PENGESAHAN

LAPORAN AKHIR

Disusun oleh:

Muhammad Ikram Mughni 124240191

Telah Diperiksa dan Disetujui oleh Asisten Praktikum
Algoritma Pemrograman dan Struktur Data
Pada Tanggal: 31 Mei 2025

Menyetujui,

Asisten Praktikum	**Asisten Praktikum**
Gradiva Arya Wicaksana	**Athaya Rizqia Fitriani**
123230089	**124210071**

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa yang senantiasa mencurahkan rahmat dan hidayah-Nya sehingga kami dapat menyelesaikan praktikum Dasar-Dasar Pemrograman serta laporan akhir praktikum Algoritma Pemrograman dan Struktur Data. Adapun laporan ini berisi tentang kumpulan tugas dan evaluasi dari hasil pembelajaran selama praktikum berlangsung.

Tidak lupa ucapan terima kasih kepada asisten dosen yang selalu membimbing dan mengajari kami dalam melaksanakan praktikum dan dalam menyusun laporan ini. Laporan ini masih sangat jauh dari kesempurnaan, oleh karena itu kritik serta saran yang membangun kami harapkan untuk menyempurnakan laporan akhir ini.

Atas perhatian dari semua pihak yang membantu penulisan ini, kami ucapkan terima kasih. Semoga laporan ini dapat dipergunakan seperlunya.

Yogyakarta, 29 Mei 2025

Penyusun

DAFTAR ISI

BAB I
SISTEM MANAJEMEN DATA BIOSKOP DOUBLE A

Bioskop Double A ingin memiliki sistem untuk menyimpan data film yang terdiri dari Judul, Kode, dan Rating. Sistem ini digunakan oleh bagian *ticketing* untuk melakukan pencarian dan pengurutan data film berdasarkan kriteria tertentu. Untuk mendukung operasional, sistem ticketing Bioskop Double A membutuhkan fitur utama seperti penampilan data film menggunakan pointer, pencarian berdasarkan kode dengan *Sequential Search*, dan pencarian judul menggunakan *Binary Search* setelah data diurutkan. Sistem juga harus mampu mengurutkan data berdasarkan rating secara ascending dengan *Quick Sort* dan descending dengan *Bubble Sort*. Fitur-fitur ini dirancang untuk mempercepat proses pencarian dan pengelolaan data film secara efisien dan akurat.

1.1 *Source Code* Program

```
#include <iostream>
#include <iomanip>
using namespace std;

struct film{
    string judul;
    string kode;
    float harga;
};

film daftarFilm[5] = {
    {"Avengers", "F001", 8.50},
    {"Titanic", "F002", 9.00},
    {"Inception", "F003", 8.80},
    {"Interstellar", "F004", 9.20},
    {"Joker", "F005", 8.70}
};

void menampilkanFilm(){
    film *pointer = daftarFilm;
    int pArray = sizeof(daftarFilm)/sizeof(*daftarFilm);

    cout << setfill('=') << setw(55) << "" << endl;
    cout << setfill(' ') << left;
```

```
    cout << setw(25) << "Judul";
    cout << setw(15) << "Kode";
    cout << setw(15) << "Rating" << endl;
    cout << setfill('-') << setw(55) << "" << endl;
    cout << setfill(' ');

    for (int i = 0; i < pArray; i++)
    {
        cout << left << setw(25) << (*pointer).judul;
        cout << setw(15) << (*pointer).kode;
        cout  <<  setw(15)  <<  fixed  <<  setprecision(2)  <<
(*pointer).harga << endl;
        pointer++;
    }

    cout << setfill('=') << setw(55) << "" << endl;
}

void mencariBerdasarkanKode(string kode){
    film *pointer = daftarFilm;
    int pArray = sizeof(daftarFilm)/sizeof(*daftarFilm);
    bool ketemu = false;
    int i = 0;

    while (i < pArray)
    {
        if ((*pointer).kode == kode )
        {
            cout << "Kode film   : " << (*pointer).kode << endl;
            cout << "Judul film  : " << (*pointer).judul << endl;
            cout << "Rating film : " << fixed << setprecision(2)
<< (*pointer).harga << endl;
            ketemu = true;
            break;
        }
        i++;
        pointer++;
    }
```

```
    if (!ketemu)
    {
        cout << "Film dengan kode " << kode << " tidak ditemukan
dalam daftar" << endl;
    }
}

void mengurutkanJudulAscendingUntukSearchingBinary() {
    int pArray = sizeof(daftarFilm) / sizeof(*daftarFilm);

    for (int i = 0; i < pArray - 1; i++) {
        for (int j = 0; j < pArray - 1 - i; j++) {
            if (daftarFilm[j].judul > daftarFilm[j + 1].judul) {
                swap(daftarFilm[j], daftarFilm[j + 1]);
            }
        }
    }
}

void mencariDariJudulBinarySearch(string judul) {
    int pArray = sizeof(daftarFilm) / sizeof(*daftarFilm);
    int i = 0, j = pArray - 1, k;
    bool found = false;

    while (!found && i <= j) {
        k = (i + j) / 2;

        if (judul == daftarFilm[k].judul) {
            found = true;
        } else {
            if (judul < daftarFilm[k].judul)
                j = k - 1;
            else
                i = k + 1;
        }
    }
```

```
    if (found) {
        film *pointer = &daftarFilm[k];
        cout << "Judul film  : " << (*pointer).judul << endl;
        cout << "Kode film   : " << (*pointer).kode << endl;
        cout << "Rating film : " << fixed << setprecision(2) <<
(*pointer).harga << endl;
    } else {
        cout << "Film dengan judul " << judul << " tidak ditemukan
dalam daftar" << endl;
    }
}

void mengurutkanQuickSortAscending(film *array, int awal, int
akhir){
    int low = awal, high = akhir;
    float pivot = array[(awal + akhir) / 2].harga;

    do {
        while (array[low].harga < pivot) low++;
        while (array[high].harga > pivot) high--;

        if (low <= high) {
            swap(array[low], array[high]);
            low++;
            high--;
        }
    } while (low <= high);

    if (awal < high) mengurutkanQuickSortAscending(array, awal,
high);
    if (low < akhir) mengurutkanQuickSortAscending(array, low,
akhir);
}

void mengurutkanJudulDescending() {
    int pArray = sizeof(daftarFilm) / sizeof(*daftarFilm);

    for (int i = 0; i < pArray - 1; i++) {
```

```
        for (int j = 0; j < pArray - 1 - i; j++) {
            if (daftarFilm[j].harga < daftarFilm[j + 1].harga) {
                swap(daftarFilm[j], daftarFilm[j + 1]);
            }
        }
    }
}

void garis(){
    cout << "--------------------------------------------------
------" << endl;
}

int main(){
    system("cls");

    int pilihMenu;

    do
    {
        cout << "====== Bioskop Double A ======" << endl;
        cout << endl;
        cout << "1. Tampilkan Film\n";
        cout << "2. Cari Berdasarkan Kode\n";
        cout << "3. Cari Berdasarkan Judul\n";
        cout << "4. Sort Rating Film (asc)\n";
        cout << "5. Sort Rating Film (desc)\n";
        cout << "6. Exit\n";

        cout << "Pilih menu: ";
        cin >> pilihMenu;
        cout << endl;

        switch (pilihMenu)
        {
        case 1:
            menampilkanFilm();
            system("pause");
```

```
            system("cls");
            break;

        case 2:
        {
            string kode;
            cout << "Masukkan kode yang ingin anda cari: ";
            cin >> kode;
            cout << endl;

            garis();
            mencariBerdasarkanKode(kode);
            garis();
            system("pause");
            system("cls");
            break;
        }

        case 3:
        {
            mengurutkanJudulAscendingUntukSearchingBinary();

            string judul;
            cout << "Masukkan judul yang ingin anda cari: ";
            cin >> judul;
            cout << endl;

            garis();
            mencariDariJudulBinarySearch(judul);
            garis();
            system("pause");
            system("cls");
            break;
        }

        case 4:
        {
```

```
            int     pArray     =     sizeof(daftarFilm)     /
sizeof(*daftarFilm);
            mengurutkanQuickSortAscending(daftarFilm, 0, pArray
- 1);

            cout << endl;
            cout << "Mengurutkan data film secara ascending
dengan quick sort berdasarkan rating" << endl;
            menampilkanFilm();
            system("pause");
            system("cls");
            break;
        }

        case 5:
        {
            mengurutkanJudulDescending();

            cout << endl;
            cout << "Mengurutkan data film secara Descending
dengan bubble sort berdasarkan rating" << endl;
            menampilkanFilm();
            system("pause");
            system("cls");
        }
        default:
            break;
        }

    }while (pilihMenu != 6);
    cout << "terimakasih telah menggunakan program ini...";

    return 0;
}

//note di program yang saya bikin, di deklarasi struct saya
menulis harga, tapi yang sebenarnya itu rating
```

1.2 Implementasi Materi

1.2.1 *Struct* Film

```
struct film{
    string judul;
    string kode;
    float harga;
};

film daftarFilm[5] = {
    {"Avengers", "F001", 8.50},
    {"Titanic", "F002", 9.00},
    {"Inception", "F003", 8.80},
    {"Interstellar", "F004", 9.20},
    {"Joker", "F005", 8.70}
};
```

Struktur data *struct* digunakan untuk mengelompokkan atribut terkait seperti judul, kode, dan *rating* film dalam satu kesatuan. Ini memudahkan pengelolaan data secara terstruktur dan akses yang efisien. Dengan *struct* film, setiap data film dapat diorganisasi rapi sehingga program lebih modular dan mudah dikembangkan.

1.2.2 Menampilkan Data Film dengan *Pointer*

```
void menampilkanFilm(){
    film *pointer = daftarFilm;
    int pArray = sizeof(daftarFilm)/sizeof(*daftarFilm);

    cout << setfill('=') << setw(55) << "" << endl;
    cout << setfill(' ') << left;
    cout << setw(25) << "Judul";
    cout << setw(15) << "Kode";
    cout << setw(15) << "Rating" << endl;
    cout << setfill('-') << setw(55) << "" << endl;
    cout << setfill(' ');

    for (int i = 0; i < pArray; i++)
    {
        cout << left << setw(25) << (*pointer).judul;
```

```
        cout << setw(15) << (*pointer).kode;
        cout  <<  setw(15)  <<  fixed  <<  setprecision(2)  <<
(*pointer).harga << endl;
        pointer++;
    }

    cout << setfill('=') << setw(55) << "" << endl;
}
```

Fungsi ”menampilkanFilm()” menggunakan *pointer* untuk mengakses dan menampilkan elemen-elemen dari *array* struktur film. Implementasi *pointer* ini memungkinkan manipulasi data secara langsung di memori, meningkatkan efisiensi dan fleksibilitas dalam pengelolaan data. Selain itu, penggunaan *pointer* memperkuat pemahaman konsep pengaksesan data dinamis pada *array* struktur, yang sangat penting dalam pemrograman C++. Fungsi ini juga menggunakan teknik *formatting* agar *output* tampil rapi dan mudah dibaca.

1.2.3 Pencarian Film Berdasarkan Kode dengan *Sequential Search*

```
void mencariBerdasarkanKode(string kode){
    film *pointer = daftarFilm;
    int pArray = sizeof(daftarFilm)/sizeof(*daftarFilm);
    bool ketemu = false;
    int i = 0;

    while (i < pArray)
    {
        if ((*pointer).kode == kode )
        {
            cout << "Kode film   : " << (*pointer).kode << endl;
            cout << "Judul film  : " << (*pointer).judul << endl;
            cout << "Rating film : " << fixed << setprecision(2)
<< (*pointer).harga << endl;
            ketemu = true;
            break;
        }
        i++;
        pointer++;
    }
```

```
    if (!ketemu)
    {
        cout << "Film dengan kode " << kode << " tidak ditemukan
dalam daftar" << endl;
    }
}
```

Fungsi “mencariBerdasarkanKode()” mengimplementasikan metode *Sequential Search* untuk menemukan film berdasarkan kode secara linear dari awal hingga akhir daftar. Penggunaan *pointer* memudahkan iterasi langsung pada *array* struktur, meningkatkan efisiensi akses data. Metode ini sederhana namun efektif untuk jumlah data yang relatif kecil, sehingga cocok untuk sistem ”ticketing Bioskop Double A”. Jika film ditemukan, data lengkap akan ditampilkan, jika tidak, pesan pemberitahuan dicetak, memberikan pengalaman pengguna yang jelas.

1.2.4 Pengurutan Judul Film Secara *Ascending* untuk *Binary Search*

```
void mengurutkanJudulAscendingUntukSearchingBinary() {
    int pArray = sizeof(daftarFilm) / sizeof(*daftarFilm);

    for (int i = 0; i < pArray - 1; i++) {
        for (int j = 0; j < pArray - 1 - i; j++) {
            if (daftarFilm[j].judul > daftarFilm[j + 1].judul) {
                swap(daftarFilm[j], daftarFilm[j + 1]);
            }
        }
    }
}
```

Fungsi “mengurutkanJudulAscendingUntukSearchingBinary()” menggunakan algoritma *Bubble Sort* untuk mengurutkan *array* film berdasarkan judul secara alfabetis dari A ke Z. Pengurutan ini merupakan tahap penting agar pencarian judul menggunakan Binary Search dapat dilakukan dengan benar dan efisien. Dengan data yang sudah terurut, proses pencarian menjadi lebih cepat dibandingkan pencarian linear, sehingga meningkatkan performa sistem *ticketing* dalam menemukan film berdasarkan judul. Meskipun *Bubble Sort* kurang efisien untuk data besar, algoritma ini cukup sederhana dan efektif untuk jumlah data film yang terbatas pada kasus ini.

1.2.5 Pencarian Judul Film dengan *Binary Search*

```
void mencariDariJudulBinarySearch(string judul) {
    int pArray = sizeof(daftarFilm) / sizeof(*daftarFilm);
    int i = 0, j = pArray - 1, k;
```

```
    bool found = false;

    while (!found && i <= j) {
        k = (i + j) / 2;

        if (judul == daftarFilm[k].judul) {
            found = true;
        } else {
            if (judul < daftarFilm[k].judul)
                j = k - 1;
            else
                i = k + 1;
        }
    }

    if (found) {
        film *pointer = &daftarFilm[k];
        cout << "Judul film  : " << (*pointer).judul << endl;
        cout << "Kode film   : " << (*pointer).kode << endl;
        cout << "Rating film : " << fixed << setprecision(2) <<
(*pointer).harga << endl;
    } else {
        cout << "Film dengan judul " << judul << " tidak ditemukan
dalam daftar" << endl;
    }
}
```

Fungsi “mencariDariJudulBinarySearch” mengimplementasikan metode Bi*nary Search* untuk mencari film berdasarkan judul secara efisien pada *array* yang sudah terurut secara alfabetis. *Binary Search* bekerja dengan membagi ruang pencarian menjadi dua bagian dan membandingkan elemen tengah dengan target pencarian, sehingga mempercepat proses pencarian dibandingkan pencarian linear. Jika film ditemukan, data film tersebut ditampilkan melalui *pointer*, jika tidak, sistem memberikan informasi bahwa film tidak ditemukan. Metode ini membantu bagian *ticketing* dalam mengakses data film dengan cepat dan akurat.

1.2.6 Pengurutan Judul Film Secara *Ascending* dengan *Quick Sort*

```
void mengurutkanQuickSortAscending(film *array, int awal, int
akhir){
```

```
    int low = awal, high = akhir;
    float pivot = array[(awal + akhir) / 2].harga;

    do {
        while (array[low].harga < pivot) low++;
        while (array[high].harga > pivot) high--;

        if (low <= high) {
            swap(array[low], array[high]);
            low++;
            high--;
        }
    } while (low <= high);

    if (awal < high) mengurutkanQuickSortAscending(array, awal,
high);
    if (low < akhir) mengurutkanQuickSortAscending(array, low,
akhir);
}
```

Fungsi ”mengurutkanQuickSortAscending()” menggunakan algoritma *Quick Sort* untuk mengurutkan data film berdasarkan rating secara *ascending*. Metode ini memilih *pivot* sebagai elemen tengah dan membagi *array* menjadi bagian yang lebih kecil dan lebih besar dari *pivot*, kemudian mengurutkan secara *rekursif*. *Quick Sort* efektif untuk pengurutan cepat dan efisien, membantu bagian *ticketing* dalam mengelola data film dengan mudah dan terstruktur.

1.2.7 Pengurutan Judul Film Secara *Descending* dengan *Bubble Sort*

```
void mengurutkanJudulDescending() {
    int pArray = sizeof(daftarFilm) / sizeof(*daftarFilm);

    for (int i = 0; i < pArray - 1; i++) {
        for (int j = 0; j < pArray - 1 - i; j++) {
            if (daftarFilm[j].harga < daftarFilm[j + 1].harga) {
                swap(daftarFilm[j], daftarFilm[j + 1]);
            }
        }
    }
}
```

```
void garis(){
    cout << "------------------------------------------------
------" << endl;
}
```

Fungsi ”mengurutkanJudulDescending()” menggunakan algoritma *Bubble Sort* untuk mengurutkan data film berdasarkan rating secara menurun (*descending*). Algoritma ini bekerja dengan membandingkan pasangan elemen bersebelahan dan menukar posisi jika elemen sebelumnya lebih kecil dari yang berikutnya. Meskipun sederhana, *Bubble Sort* mudah dipahami dan efektif untuk data berukuran kecil, membantu sistem ticketing menampilkan film dengan rating tertinggi terlebih dahulu.

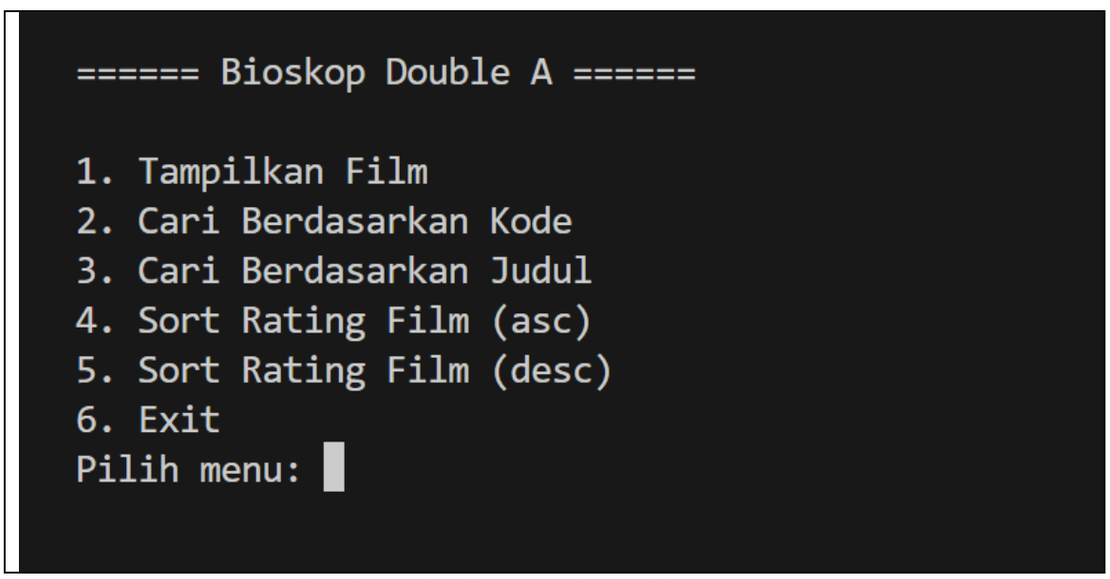
1.3 Catatan Revisi

Tidak ada revisi.

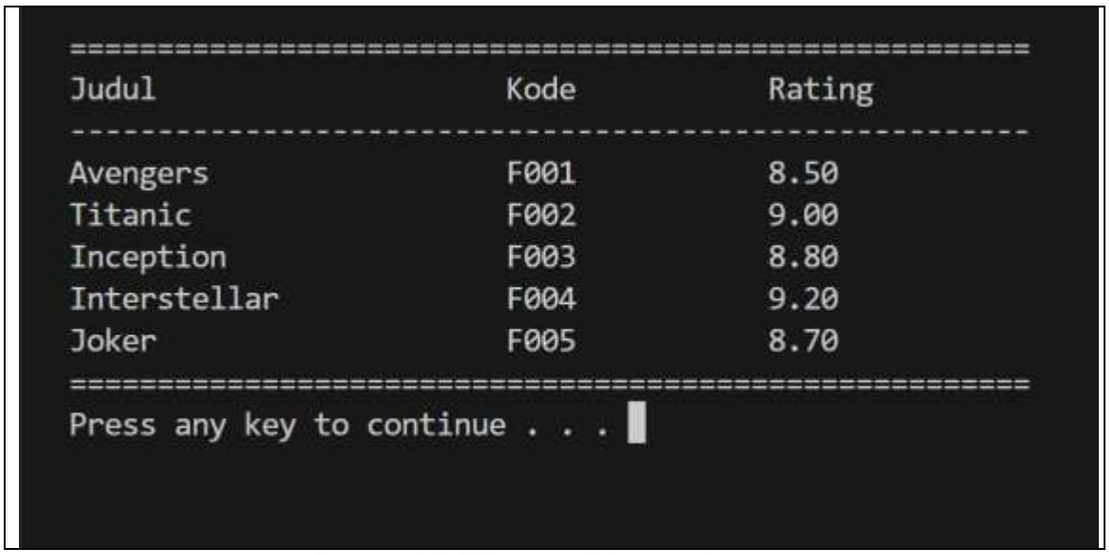
1.4 Revisi Program

```
-
```

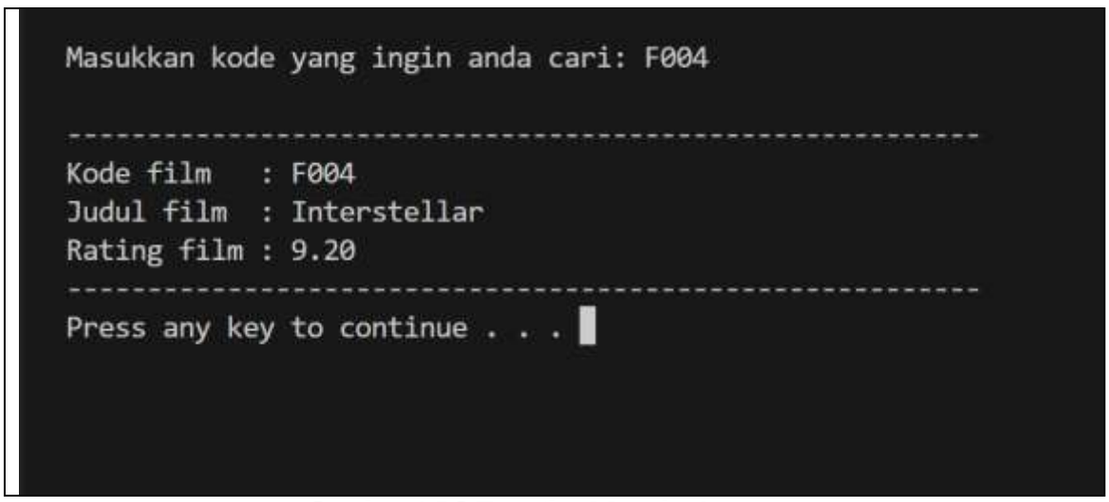
1.5 *Screenshot* Program



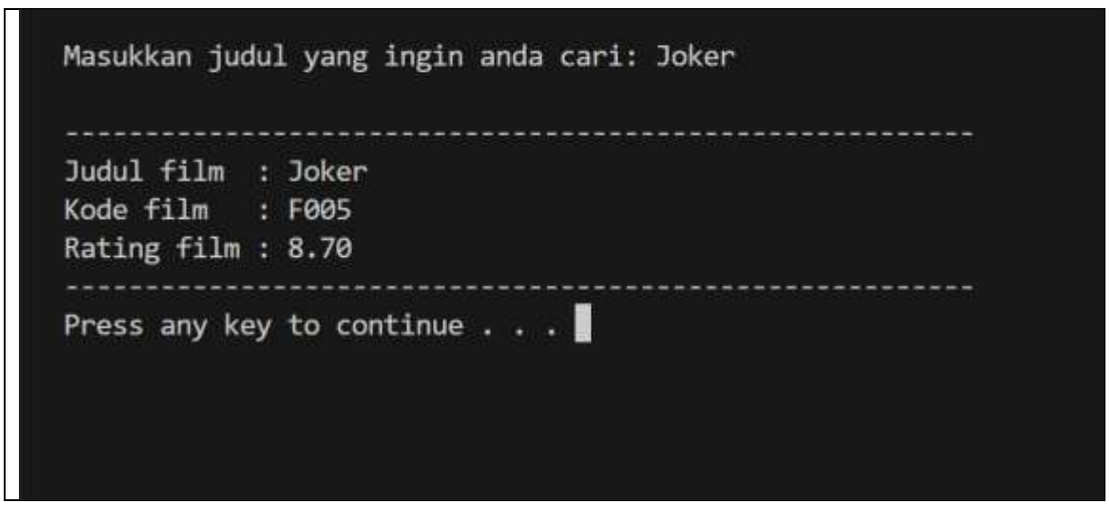
Gambar 1.5.1 Tampilan Menu Utama



Gambar 1.5.2 Tampilan Menu ke-1



Gambar 1.5.3 Tampilan Menu ke-2



Gambar 1.5.4 Tampilan Menu ke-3

```
Mengurutkan data film secara ascending dengan quick sort berdasarkan rating
=====================================================
Judul                   Kode          Rating
-----------------------------------------------------
Avengers                F001          8.50
Joker                   F005          8.70
Inception               F003          8.80
Titanic                 F002          9.00
Interstellar            F004          9.20
=====================================================
Press any key to continue . . .
```

Gambar 1.5.5 Tampilan Menu ke-4

```
Mengurutkan data film secara Descending dengan bubble sort berdasarkan rating
=====================================================
Judul                   Kode          Rating
-----------------------------------------------------
Interstellar            F004          9.20
Titanic                 F002          9.00
Inception               F003          8.80
Joker                   F005          8.70
Avengers                F001          8.50
=====================================================
Press any key to continue . . .
```

Gambar 1.5.6 Tampilan Menu ke-5

```
====== Bioskop Double A ======

1. Tampilkan Film
2. Cari Berdasarkan Kode
3. Cari Berdasarkan Judul
4. Sort Rating Film (asc)
5. Sort Rating Film (desc)
6. Exit
Pilih menu: 6

terimakasih telah menggunakan program ini...
```

Gambar 1.5.7 Menu ke-6 (keluar dari program)

BAB II
TOKO ROTI "MANIS LEZAT"

Toko Roti "Manis Lezat" yang berlokasi di Depok ingin mengembangkan sebuah sistem digital untuk mengelola antrean pesanan yang semakin meningkat. Sistem ini digunakan untuk mencatat dan mengatur pesanan roti secara efisien oleh bagian kasir dan produksi. Fitur utama dari sistem ini meliputi pengambilan antrean dengan memasukkan data pesanan seperti nama pembeli, jenis roti, dan total harga ke dalam struktur data antrean (*queue*), serta pelayanan pembeli dengan memproses pesanan terdepan dan memindahkannya ke riwayat pesanan yang tersimpan dalam file. Selain itu, sistem juga menyediakan kemampuan untuk menampilkan daftar pesanan yang sedang antre, membatalkan pesanan terakhir dalam antrean, dan menampilkan riwayat pesanan yang telah dilayani. Dengan fitur-fitur ini, Toko Roti "Manis Lezat" dapat mempercepat proses pengelolaan pesanan dan meningkatkan kepuasan pelanggan secara efektif dan akurat.

2.1 *Source Code* Program

```
#include <iostream>
#include <cstring>
using namespace std;

void garis()
{
    cout << "==================================\n";
}

void garis2()
{
    cout << "----------------------------------\n";
}

struct Queue //queue dengan linked list
{
    char nama[30];
    int jenisRoti[10];
    int jumlahRoti[10];
    int banyakPesanan;
    float totalHarga;
    Queue *next;
```

```
};

struct stackHistory // stack dengan linked list
{
    char nama[30];
    int jenisRoti[10];
    int jumlahRoti[10];
    int banyakPesanan;
    float totalHarga;
    stackHistory *next;
};

Queue *depan, *belakang;

stackHistory *topHistory = nullptr;
stackHistory *awalHistory = nullptr;

void buatQueue()
{
    depan = belakang = nullptr;
}

bool queueKosong()
{
    return depan == nullptr;
}

float hargaPerKode(int kode) // penentuan harga tergantung jenis
roti
{
    if (kode == 110)
        return 8000;
    else if (kode == 111)
        return 9000;
    else if (kode == 112)
        return 10000;
    else if (kode == 113)
        return 9500;
```

```
    else if (kode == 114)
        return 7000;
    else if (kode == 115)
        return 8000;
    else if (kode == 116)
        return 7500;
    else if (kode == 117)
        return 7500;
    else if (kode == 118)
        return 8000;
    else if (kode == 119)
        return 11000;
    else if (kode == 120)
        return 14000;
    else
        return 0;
}

void enqueueAmbilAntreanTokoRoti()
{
    Queue *baru = new Queue; // membuat antrean baru
    baru->banyakPesanan = 0;
    baru->totalHarga = 0;

    cout << "==== Ambil Antrean ====\n\n";
    cin.ignore();
    cout << "Masukkan nama pelanggan: ";
    cin.getline(baru->nama, 30);

    char lagi;
    do
    {
        if (baru->banyakPesanan >= 10)
        {
            cout << "Maksimal 10 jenis roti per pelanggan.\n";
            break;
        }
```

```
        int pilihJenisRoti, jumlah;
        cout << "\n=== Pilih Jenis Roti ===\n";
        cout << "110. Cokelat (Rp8000)\n";
        cout << "111. Keju (Rp9000)\n";
        cout << "112. Sosis (Rp10000)\n";
        cout << "113. Pisang (Rp9500)\n";
        cout << "114. Tawar (Rp7000)\n";
        cout << "115. Taro (Rp8000)\n";
        cout << "116. Strawbery (Rp7500)\n";
        cout << "117. Bluberry (Rp7500)\n";
        cout << "118. Srikaya (Rp8000)\n";
        cout << "119. Abon (Rp11000)\n";
        cout << "120. Croissant (Rp14000)\n";
        garis();
        cout << "Masukkan kode roti      : ";
        cin >> pilihJenisRoti;

        float hargaSatuan = hargaPerKode(pilihJenisRoti);
        if (hargaSatuan == 0)
        {
            cout << "Kode roti tidak valid!\n";
            continue;
        }

        cout << "Masukkan jumlah roti    : ";
        cin >> jumlah;
        if (jumlah <= 0)
        {
            cout << "Jumlah harus positif!\n";
            continue;
        }

        baru->jenisRoti[baru->banyakPesanan] = pilihJenisRoti;
        baru->jumlahRoti[baru->banyakPesanan] = jumlah;
        baru->totalHarga += hargaSatuan * jumlah;
        baru->banyakPesanan++;

        garis();
```

```
        cout << "Pesanan roti berhasil ditambahkan.\n";

        cout << "Mau pesan roti lain? (y/n): ";
        cin >> lagi;

    } while (lagi == 'y' || lagi == 'Y');

    baru->next = nullptr;
    if (depan == nullptr)
    {
        depan = belakang = baru;
    }
    else
    {
        belakang->next = baru;
        belakang = baru;
    }

    garis();
    cout << "\nTotal harga semua roti: Rp" << baru->totalHarga <<
endl;
    cout << "Pesanan sudah masuk antrean.\n";
}

void pushHistory(const Queue *data)
{
    stackHistory *baru = new stackHistory;
    strcpy(baru->nama, data->nama);
    baru->banyakPesanan = data->banyakPesanan;
    baru->totalHarga = data->totalHarga;

    for (int i = 0; i < data->banyakPesanan; i++)
    {
        baru->jenisRoti[i] = data->jenisRoti[i];
        baru->jumlahRoti[i] = data->jumlahRoti[i];
    }

    baru->next = nullptr;
```

```
    if (awalHistory == nullptr)
    {
        awalHistory = topHistory = baru;
    }
    else
    {
        topHistory->next = baru;
        topHistory = baru;
    }
}

void dequeueLayaniPembeliTokoRoti()
{
    if (queueKosong())
    {
        cout << "Tidak ada antrean yang harus dilayani.\n";
    }
    else
    {
        Queue *hapus = depan;
        cout << "\nMelayani pelanggan: " << hapus->nama << endl;
        garis2();
        cout << "Pesanan:\n";
        for (int i = 0; i < hapus->banyakPesanan; i++)
        {
            int kode = hapus->jenisRoti[i];
            int jumlah = hapus->jumlahRoti[i];
            float harga = hargaPerKode(kode);
            float subtotal = harga * jumlah;

            cout << "- Kode Roti: " << kode;
            cout << "| Jumlah: " << jumlah;
            cout << "| Subtotal: Rp" << subtotal << endl;
        }
        garis2();
        cout  <<  "Total  yang  harus  dibayar:  Rp"  <<  hapus-
>totalHarga << endl;
```

```
        garis();

        pushHistory(hapus);

        depan = depan->next;
        delete hapus;

        if (depan == nullptr)
        {
            belakang = nullptr;
        }

        cout << "\nPelanggan selesai dilayani.\n";
    }
}

void tampilkanPesananTokoRoti()
{
    if (queueKosong())
    {
        cout << "Antrean kosong.\n";
        return;
    }

    Queue *baca = depan;
    int antreanKe = 1;

    while (baca != nullptr)
    {
        cout << "Antrean ke-" << antreanKe << ":\n";
        cout << "Nama Pelanggan: " << baca->nama << endl;
        garis2();
        cout << "Daftar Roti:\n";
        for (int i = 0; i < baca->banyakPesanan; i++)
        {
            int kode = baca->jenisRoti[i];
            int jumlah = baca->jumlahRoti[i];
            float subtotal = hargaPerKode(kode) * jumlah;
```

```
            cout << "- Kode: " << kode;
            cout << "| Jumlah: " << jumlah;
            cout << "| Subtotal: Rp" << subtotal << endl;
        }
        cout << "Total Harga: Rp" << baca->totalHarga << endl;
        garis();
        cout << endl;

        baca = baca->next;
        antreanKe++;
    }
}

void popBatalkanPesananPalingAkhirAntrean()
{
    if (queueKosong())
    {
        cout << "Tidak ada pesanan yang bisa dibatalkan.\n";
        return;
    }

    Queue *bantu, *hapus;

    if (depan == belakang)
    {
        hapus = depan;
        cout << "\nPesanan atas nama " << hapus->nama << " telah
dibatalkan.\n";
        delete hapus;
        depan = belakang = nullptr;
    }
    else
    {
        bantu = depan;
        while (bantu->next != belakang)
        {
            bantu = bantu->next;
```

```
        }
        hapus = belakang;
        cout << "\nPesanan atas nama " << hapus->nama << " telah
dibatalkan.\n";
        delete hapus;
        belakang = bantu;
        belakang->next = nullptr;
    }
}

void tampilkanHistoryPesananTokoRoti()
{
    if (awalHistory == nullptr)
    {
        cout << "History pesanan kosong.\n";
        return;
    }

    stackHistory *baca = awalHistory;
    int nomor = 1;
    while (baca != nullptr)
    {
        cout << "\nHistory ke-" << nomor << ":\n";
        cout << "Nama Pelanggan: " << baca->nama << endl;
        garis2();
        cout << "Daftar Roti:\n";
        for (int i = 0; i < baca->banyakPesanan; i++)
        {
            int kode = baca->jenisRoti[i];
            int jumlah = baca->jumlahRoti[i];
            float subtotal = hargaPerKode(kode) * jumlah;

            cout << "- Kode: " << kode;
            cout << "| Jumlah: " << jumlah;
            cout << "| Subtotal: Rp" << subtotal << endl;
        }
        cout << "Total Harga: Rp" << baca->totalHarga << endl;
        garis();
```

```
        baca = baca->next;
        nomor++;
    }
}

int main()
{
    buatQueue();
    int menu;

    do
    {
        system("cls");
        cout << "\n===== Toko Roti Manis Lezat =====\n";
        cout << "1. Ambil Antrean\n";
        cout << "2. Layani Pembeli\n";
        cout << "3. Tampilkan Pesanan\n";
        cout << "4. Batalkan Pesanan\n";
        cout << "5. Tampilkan History Pesanan\n";
        cout << "0. Keluar Program\n\n";
        cout << "Pilih: ";
        cin >> menu;

        switch (menu)
        {
        case 1:
            system("cls");
            enqueueAmbilAntreanTokoRoti();
            system("pause");
            break;

        case 2:
            system("cls");
            dequeueLayaniPembeliTokoRoti();
            system("pause");
            break;

        case 3:
```

```
            system("cls");
            tampilkanPesananTokoRoti();
            system("pause");
            break;

        case 4:
            system("cls");
            popBatalkanPesananPalingAkhirAntrean();
            system("pause");
            break;

        case 5:
            system("cls");
            tampilkanHistoryPesananTokoRoti();
            system("pause");
            break;

        case 0:
            cout << "Keluar program Toko Roti Manis Lezat,
terimakasih :)\n";
            break;

        default:
            system("cls");
            cout << "Menu tidak valid.\n";
            system("pause");
            break;
        }
    } while (menu != 0);

    return 0;
}
```

2.2 Implementasi Materi

2.2.1 Struktur Data *Linked List* untuk Antrean dan Riwayat Pesanan

```
struct Queue
{
```

```
    char nama[30];
    int jenisRoti[10];
    int jumlahRoti[10];
    int banyakPesanan;
    float totalHarga;
    Queue *next;
};

struct stackHistory
{
    char nama[30];
    int jenisRoti[10];
    int jumlahRoti[10];
    int banyakPesanan;
    float totalHarga;
    stackHistory *next;
};
```

Struktur data *linked list* digunakan untuk menyimpan antrean pesanan dan riwayat pesanan yang selesai. *Queue* menyimpan data pesanan dalam antrean dengan *pointer next* untuk elemen berikutnya, sedangkan ”*stackHistory*” menyimpan riwayat pesanan sebagai *stack* dengan *pointer next*. Pendekatan ini memudahkan penambahan dan penghapusan data secara dinamis tanpa batas ukuran tetap.

2.2.2 Inisialisasi *Pointer* untuk *Linked List* dan *Stack*

```
Queue *depan, *belakang;

stackHistory *topHistory = nullptr;
stackHistory *awalHistory = nullptr;
```

Pointer depan dan belakang digunakan untuk mengelola antrean (*queue*) sebagai penunjuk elemen pertama dan terakhir dalam *linked list*. Sedangkan *pointer* ”*topHistory*” dan ”*awalHistory*” digunakan untuk mengelola riwayat pesanan dengan struktur *stack*, di mana *topHistory* menunjuk ke elemen paling atas *stack*. Inisialisasi *pointer* ini penting untuk memulai struktur data dinamis yang akan digunakan dalam operasi antrean dan riwayat.

2.2.3 Inisialisasi dan Pemeriksaan Kondisi Kosong pada *Queue*

```
void buatQueue()
{
    depan = belakang = nullptr;
```

```
}

bool queueKosong()
{
    return depan == nullptr;
}
```

Fungsi ”buatQueue()” digunakan untuk menginisialisasi *queue* dengan menetapkan *pointer* depan dan belakang ke *nullptr*, yang menandakan antrean masih kosong. Fungsi ”queueKosong()” mengembalikan nilai *boolean* untuk mengecek apakah antrean kosong, yaitu dengan memeriksa apakah *pointer* depan bernilai *nullptr*. Kedua fungsi ini penting untuk memastikan kondisi awal dan memudahkan pengelolaan antrean dalam program.

2.2.4 Fungsi Penentuan Harga Berdasarkan Kode Produk

```
float hargaPerKode(int kode)
{
    if (kode == 110)
        return 8000;
    else if (kode == 111)
        return 9000;
    else if (kode == 112)
        return 10000;
    else if (kode == 113)
        return 9500;
    else if (kode == 114)
        return 7000;
    else if (kode == 115)
        return 8000;
    else if (kode == 116)
        return 7500;
    else if (kode == 117)
        return 7500;
    else if (kode == 118)
        return 8000;
    else if (kode == 119)
        return 11000;
    else if (kode == 120)
        return 14000;
    else
```

```
        return 0;
}
```

Fungsi ”hargaPerKode(int kode)” berfungsi untuk mengembalikan nilai harga sesuai dengan kode produk roti yang diterima sebagai *parameter*. Fungsi ini menggunakan struktur percabangan *if-else* untuk mencocokkan kode dengan harga yang telah ditentukan. Bila kode tidak ditemukan dalam daftar, fungsi akan mengembalikan harga 0 sebagai nilai *default*. Fungsi ini memudahkan proses penghitungan total harga pesanan berdasarkan jenis roti yang dipesan.

2.2.5 *Enqueue* pada Struktur Data *Queue* (Antrean)

```
void enqueueAmbilAntreanTokoRoti()
{
    Queue *baru = new Queue;
    baru->banyakPesanan = 0;
    baru->totalHarga = 0;

    cout << "==== Ambil Antrean ====\n\n";
    cin.ignore();
    cout << "Masukkan nama pelanggan: ";
    cin.getline(baru->nama, 30);

    char lagi;
    do
    {
        if (baru->banyakPesanan >= 10)
        {
            cout << "Maksimal 10 jenis roti per pelanggan.\n";
            break;
        }

        int pilihJenisRoti, jumlah;
        cout << "\n=== Pilih Jenis Roti ===\n";
        cout << "110. Cokelat (Rp8000)\n";
        cout << "111. Keju (Rp9000)\n";
        cout << "112. Sosis (Rp10000)\n";
        cout << "113. Pisang (Rp9500)\n";
        cout << "114. Tawar (Rp7000)\n";
        cout << "115. Taro (Rp8000)\n";
```

```
        cout << "116. Strawbery (Rp7500)\n";
        cout << "117. Bluberry (Rp7500)\n";
        cout << "118. Srikaya (Rp8000)\n";
        cout << "119. Abon (Rp11000)\n";
        cout << "120. Croissant (Rp14000)\n";
        garis();
        cout << "Masukkan kode roti      : ";
        cin >> pilihJenisRoti;

        float hargaSatuan = hargaPerKode(pilihJenisRoti);
        if (hargaSatuan == 0)
        {
            cout << "Kode roti tidak valid!\n";
            continue;
        }

        cout << "Masukkan jumlah roti    : ";
        cin >> jumlah;
        if (jumlah <= 0)
        {
            cout << "Jumlah harus positif!\n";
            continue;
        }

        baru->jenisRoti[baru->banyakPesanan] = pilihJenisRoti;
        baru->jumlahRoti[baru->banyakPesanan] = jumlah;
        baru->totalHarga += hargaSatuan * jumlah;
        baru->banyakPesanan++;

        garis();
        cout << "Pesanan roti berhasil ditambahkan.\n";

        cout << "Mau pesan roti lain? (y/n): ";
        cin >> lagi;

    } while (lagi == 'y' || lagi == 'Y');

    baru->next = nullptr;
```

```
    if (depan == nullptr)
    {
        depan = belakang = baru;
    }
    else
    {
        belakang->next = baru;
        belakang = baru;
    }

    garis();
    cout << "\nTotal harga semua roti: Rp" << baru->totalHarga <<
endl;
    cout << "Pesanan sudah masuk antrean.\n";
}
```

Fungsi ”enqueueAmbilAntreanTokoRoti()” digunakan untuk menambahkan pesanan pelanggan ke antrean toko roti. Data pesanan dimasukkan ke *struct Queue*, termasuk nama pelanggan, kode dan jumlah roti, serta total harga. Fungsi ini menggunakan perulangan untuk memungkinkan pelanggan memesan hingga 10 jenis roti berbeda. Pesanan dimasukkan ke antrean dengan menambahkan *node* baru ke belakang antrean (*linked list*), menjadikan sistem antrean berjalan secara *FIFO (First In First Out).*

2.2.6 *Push* ke *Stack* untuk Menyimpan Riwayat

```
void pushHistory(const Queue *data)
{
    stackHistory *baru = new stackHistory;
    strcpy(baru->nama, data->nama);
    baru->banyakPesanan = data->banyakPesanan;
    baru->totalHarga = data->totalHarga;

    for (int i = 0; i < data->banyakPesanan; i++)
    {
        baru->jenisRoti[i] = data->jenisRoti[i];
        baru->jumlahRoti[i] = data->jumlahRoti[i];
    }

    baru->next = nullptr;
```

```
    if (awalHistory == nullptr)
    {
        awalHistory = topHistory = baru;
    }
    else
    {
        topHistory->next = baru;
        topHistory = baru;
    }
}
```

Fungsi “pushHistory()” digunakan untuk menyimpan pesanan yang sudah dilayani ke dalam *stack* riwayat. Data dari antrean (*Queue*) disalin ke *node* baru bertipe “*stackHistory”*. Stack ini diimplementasikan sebagai *linked list* dengan penambahan di akhir (*top*). Jika *stack* kosong, maka *node* baru menjadi awal dan top. Jika tidak kosong, *node* baru ditambahkan setelah *topHistory*, lalu *topHistory* diperbarui. Stack ini membantu menyimpan jejak transaksi sebelumnya sesuai urutan layanan.

2.2.7 *Queue* dan *Stack* dalam Proses Layanan

```
void dequeueLayaniPembeliTokoRoti()
{
    if (queueKosong())
    {
        cout << "Tidak ada antrean yang harus dilayani.\n";
    }
    else
    {
        Queue *hapus = depan;
        cout << "\nMelayani pelanggan: " << hapus->nama << endl;
        garis2();
        cout << "Pesanan:\n";
        for (int i = 0; i < hapus->banyakPesanan; i++)
        {
            int kode = hapus->jenisRoti[i];
            int jumlah = hapus->jumlahRoti[i];
            float harga = hargaPerKode(kode);
            float subtotal = harga * jumlah;

            cout << "- Kode Roti: " << kode;
```

```
            cout << "| Jumlah: " << jumlah;
            cout << "| Subtotal: Rp" << subtotal << endl;
        }
        garis2();
        cout << "Total yang harus dibayar: Rp" << hapus->totalHarga << endl;
        garis();

        pushHistory(hapus);

        depan = depan->next;
        delete hapus;

        if (depan == nullptr)
        {
            belakang = nullptr;
        }

        cout << "\nPelanggan selesai dilayani.\n";
    }
}
```

Fungsi “dequeueLayaniPembeliTokoRoti()” digunakan untuk memproses antrean pelanggan dari struktur *Queue*. Pelanggan yang berada di depan antrean ditampilkan, dihitung total pembayarannya, lalu dipindahkan ke riwayat pesanan (*stack***)** menggunakan “pushHistory()”. *Node* antrean yang sudah diproses dihapus dari memori, dan jika antrean kosong setelahnya, *pointer* belakang diset *nullptr*. Materi ini menerapkan *queue (FIFO)* untuk antrean masuk dan *stack* (dengan *linked list*) untuk mencatat riwayat transaksi.

2.2.8 Penelusuran Data pada Struktur *Queue*

```
void tampilkanPesananTokoRoti()
{
    if (queueKosong())
    {
        cout << "Antrean kosong.\n";
        return;
    }

    Queue *baca = depan;
```

```
    int antreanKe = 1;

    while (baca != nullptr)
    {
        cout << "Antrean ke-" << antreanKe << ":\n";
        cout << "Nama Pelanggan: " << baca->nama << endl;
        garis2();
        cout << "Daftar Roti:\n";
        for (int i = 0; i < baca->banyakPesanan; i++)
        {
            int kode = baca->jenisRoti[i];
            int jumlah = baca->jumlahRoti[i];
            float subtotal = hargaPerKode(kode) * jumlah;

            cout << "- Kode: " << kode;
            cout << "| Jumlah: " << jumlah;
            cout << "| Subtotal: Rp" << subtotal << endl;
        }
        cout << "Total Harga: Rp" << baca->totalHarga << endl;
        garis();
        cout << endl;

        baca = baca->next;
        antreanKe++;
    }
}
```

Fungsi ”tampilkanPesananTokoRoti()” digunakan untuk menampilkan seluruh antrean pesanan dari struktur *Queue*. Proses dilakukan dengan menelusuri *pointer* dari depan ke belakang, mencetak semua detail pesanan (kode roti, jumlah, subtotal, total harga) untuk setiap pelanggan dalam antrean. Materi ini menekankan penggunaan *pointer* untuk penelusuran *linked list* dan implementasi *queue (FIFO)* untuk pengelolaan antrean data.

2.2.9 Penghapusan *Node* Terakhir dalam *Single Linked List (Queue)*

```
void popBatalkanPesananPalingAkhirAntrean()
{
    if (queueKosong())
    {
        cout << "Tidak ada pesanan yang bisa dibatalkan.\n";
```

```
        return;
    }

    Queue *bantu, *hapus;

    if (depan == belakang)
    {
        hapus = depan;
        cout << "\nPesanan atas nama " << hapus->nama << " telah
dibatalkan.\n";
        delete hapus;
        depan = belakang = nullptr;
    }
    else
    {
        bantu = depan;
        while (bantu->next != belakang)
        {
            bantu = bantu->next;
        }
        hapus = belakang;
        cout << "\nPesanan atas nama " << hapus->nama << " telah
dibatalkan.\n";
        delete hapus;
        belakang = bantu;
        belakang->next = nullptr;
    }
}
```

Fungsi ”popBatalkanPesananPalingAkhirAntrean()” digunakan untuk menghapus pesanan terakhir dalam antrean (node terakhir pada *single linked list*). Jika antrean hanya berisi satu *node*, maka *node* tersebut langsung dihapus. Jika lebih dari satu, *pointer* bantu digunakan untuk menelusuri hingga *node* sebelum terakhir, kemudian *node* terakhir dihapus dan *pointer* belakang diperbarui. Materi ini menerapkan konsep penghapusan *node* di akhir *linked list* serta penggunaan *pointer* untuk *traversal*.

2.2.10 *Traversal Linked List* pada *Stack* untuk Menampilkan Riwayat

```
void tampilkanHistoryPesananTokoRoti()
{
```

```
    if (awalHistory == nullptr)
    {
        cout << "History pesanan kosong.\n";
        return;
    }

    stackHistory *baca = awalHistory;
    int nomor = 1;
    while (baca != nullptr)
    {
        cout << "\nHistory ke-" << nomor << ":\n";
        cout << "Nama Pelanggan: " << baca->nama << endl;
        garis2();
        cout << "Daftar Roti:\n";
        for (int i = 0; i < baca->banyakPesanan; i++)
        {
            int kode = baca->jenisRoti[i];
            int jumlah = baca->jumlahRoti[i];
            float subtotal = hargaPerKode(kode) * jumlah;

            cout << "- Kode: " << kode;
            cout << "| Jumlah: " << jumlah;
            cout << "| Subtotal: Rp" << subtotal << endl;
        }
        cout << "Total Harga: Rp" << baca->totalHarga << endl;
        garis();
        baca = baca->next;
        nomor++;
    }
}
```

Fungsi ”tampilkanHistoryPesananTokoRoti()” digunakan untuk menelusuri dan menampilkan isi *stack* yang berisi riwayat pesanan pelanggan. *Stack* diimplementasikan sebagai *single linked list* dengan *pointer awalHistory* sebagai awal struktur. Fungsi ini menelusuri *node* satu per satu menggunakan *pointer* bantu, lalu mencetak detail pesanan pada setiap *node*. Materi ini menerapkan *traversal pointer* pada *linked list* untuk menampilkan data riwayat secara terurut.

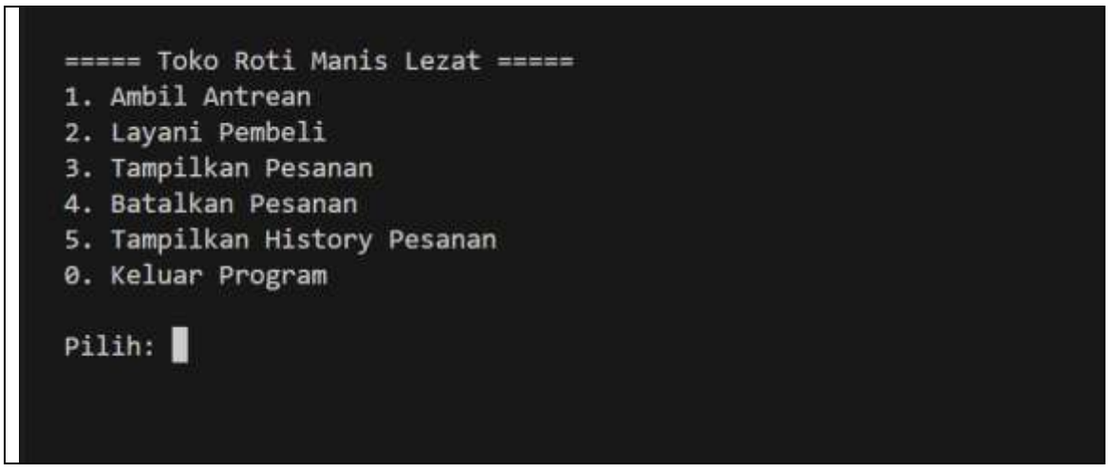
2.3 Catatan Revisi

Tidak ada revisi.

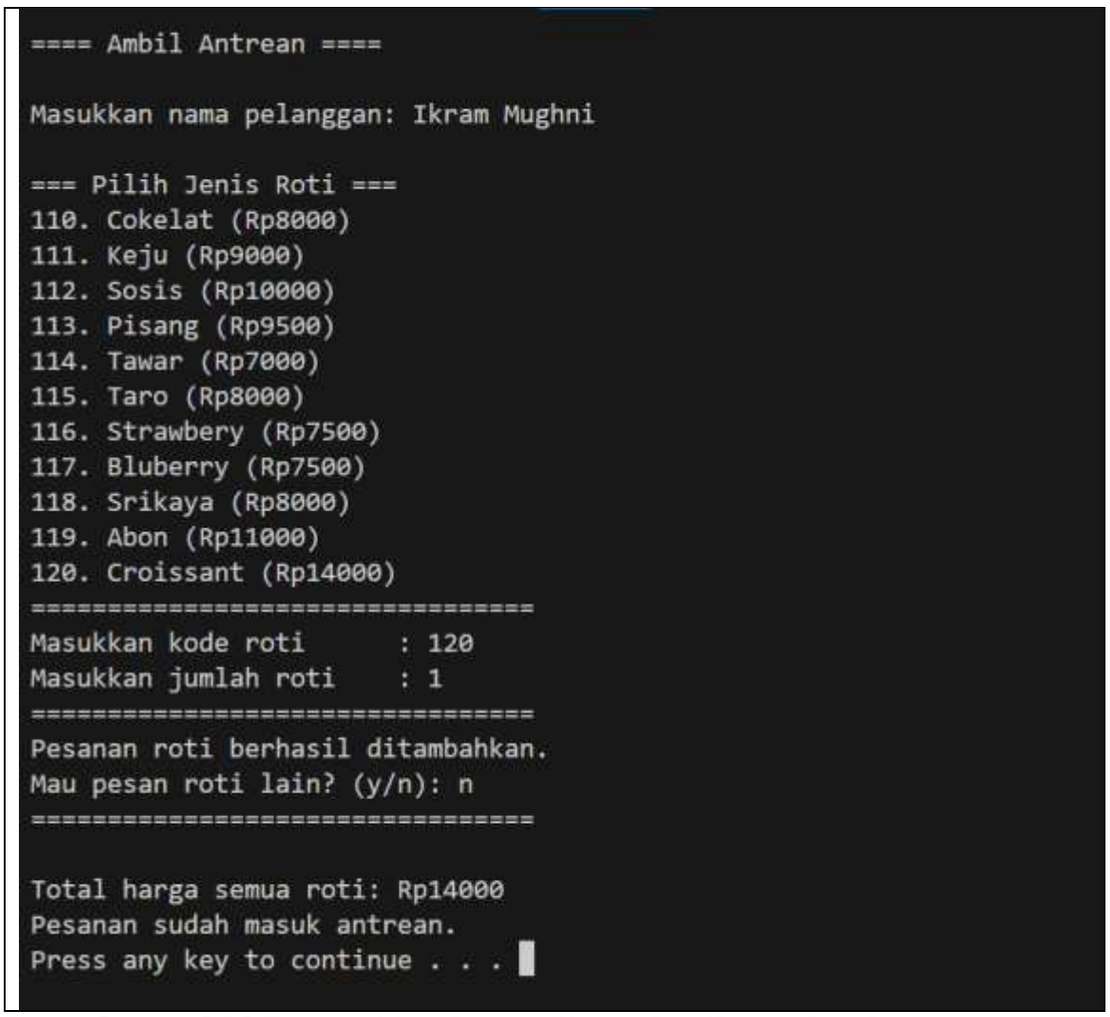
2.4 Revisi Program

-

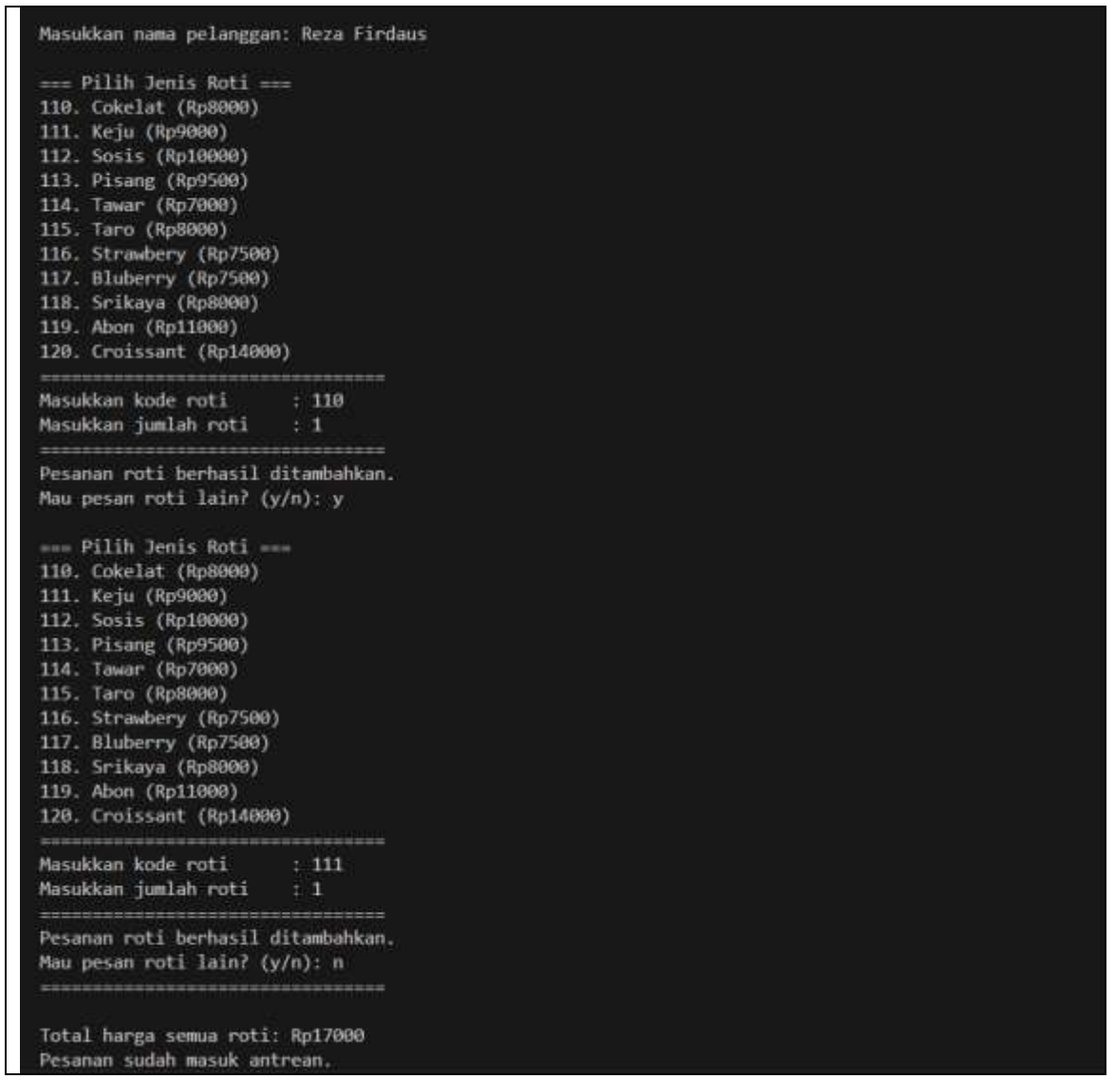
2.5 *Screenshot* Program



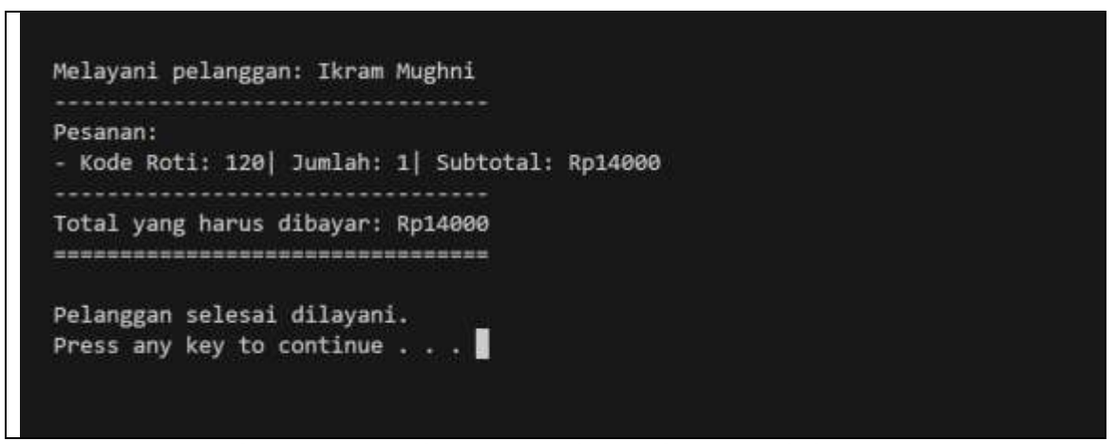
Gambar 2.5.1 Tampilan Menu Utama



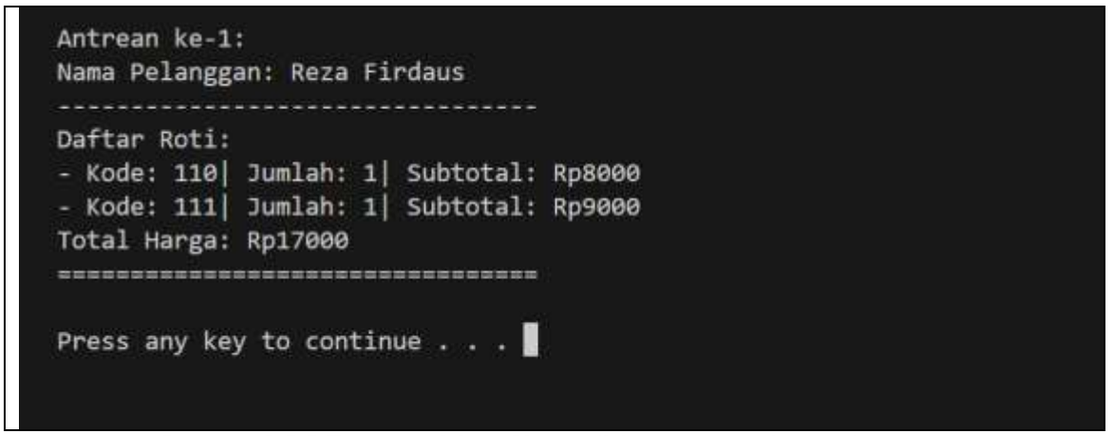
Gambar 2.5.2 Tampilan Menu Pertama (Kondisi Beli Hanya 1 Jenis Roti)



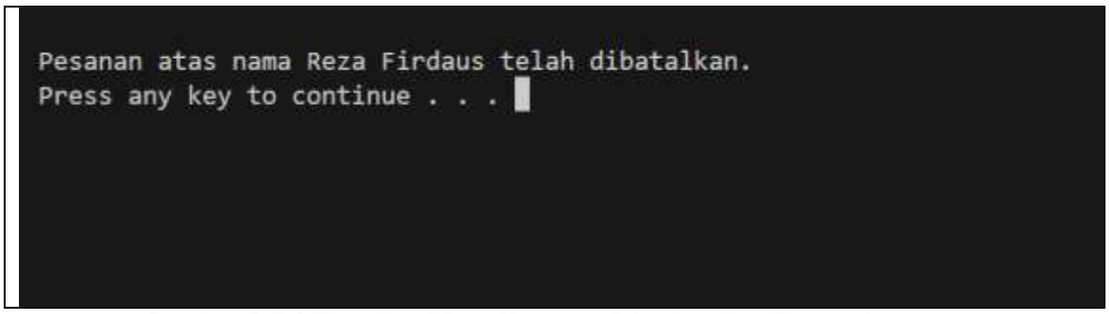
Gambar 2.5.3 Tampilan Menu Pertama (Kondisi Beli Lebih dari 1 Jenis Roti)



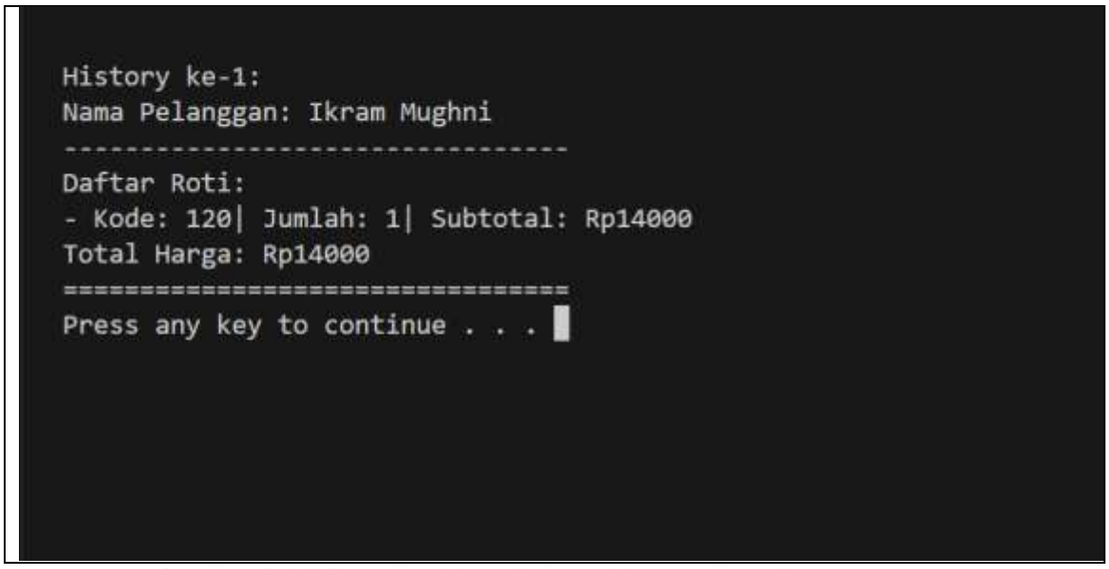
Gambar 2.5.4 Tampilan Menu Kedua (Melayani Pembeli)



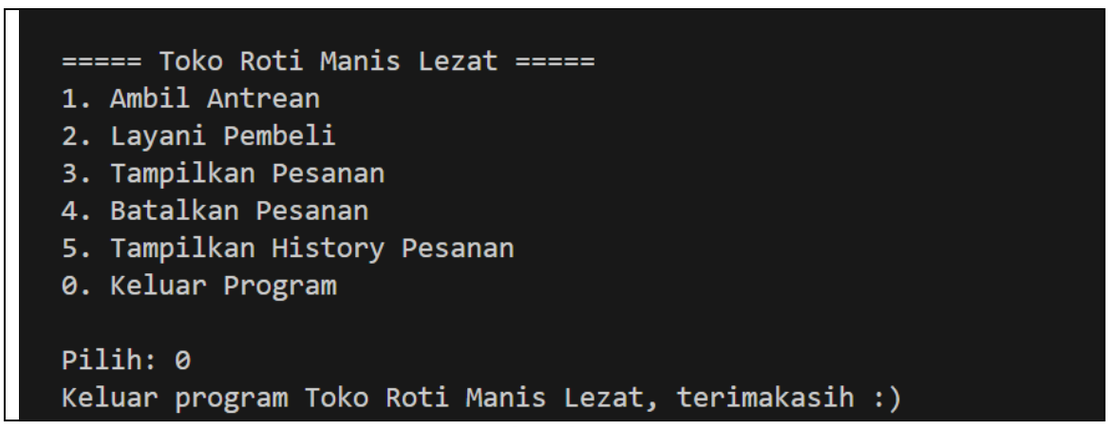
Gambar 2.5.5 Tampilan Menu Ketiga (Menampilkan Antrean)



Gambar 2.5.6 Tampilan Menu Keempat (Membatalkan Pesanan)



Gambar 2.5.7 Tampilan Menu Kelima (Menampilkan *History*)



Gambar 2.5.8 Keluar dari Program

BAB III
PROYEK AKHIR

Proyek akhir ini merupakan sebuah sistem informasi berbasis aplikasi *desktop* yang dirancang untuk mengelola operasional layanan laundry secara efisien dan terstruktur. Sistem ini mengintegrasikan dua jenis layanan utama yaitu *Drop-Off Laundry* dan *Self-Service Laundry*. Dengan menggunakan bahasa pemrograman C++, aplikasi ini memanfaatkan konsep *linked list* untuk pengelolaan data dinamis dan penyimpanan data menggunakan file *biner* dengan metode *C-style file handling*.

Pada layanan *Drop-Off Laundry*, pengguna dapat melakukan proses pencatatan laundry dengan fitur lengkap mulai dari penambahan data, pencarian berdasarkan ID, pengambilan laundry yang sekaligus memindahkan data ke riwayat, hingga menampilkan riwayat transaksi beserta total pendapatan. Sedangkan pada layanan *Self-Service Laundry*, sistem memfasilitasi pencatatan laundry berdasarkan berat cucian, menghitung jumlah mesin yang dibutuhkan serta harga yang sesuai, kemudian mengelola transaksi secara *real-time* menggunakan *linked list*. Data riwayat transaksi *Self-Service* juga disimpan ke file untuk kebutuhan pelaporan.

Sistem ini dirancang dengan antarmuka berbasis teks yang mudah digunakan dan dioperasikan, serta didukung dengan tampilan tabel untuk memudahkan pemantauan data. Proyek ini bertujuan untuk memberikan solusi praktis dalam pengelolaan laundry dengan efisiensi waktu dan akurasi data yang tinggi, serta sebagai media pembelajaran penerapan struktur data dan manajemen file dalam pengembangan perangkat lunak.

3.1 Dasar Teori

Dasar teori pada proyek ini mencakup konsep-konsep dasar pemrograman yang digunakan dalam membangun sistem layanan laundry berbasis C++. Beberapa teori yang menjadi landasan antara lain *pointer*, struktur data *linked list*, pengolahan *file biner*, *searching*, dan *sorting*. *Pointer* digunakan untuk mengelola memori secara dinamis dan memungkinkan pembentukan struktur data seperti *linked list*. *Linked list* merupakan struktur data dinamis yang terdiri dari *node-node* yang saling terhubung, cocok digunakan untuk data yang berubah-ubah seperti antrian laundry. Pengolahan file *biner* digunakan untuk menyimpan dan membaca data riwayat layanan secara efisien. *Searching* digunakan untuk menemukan data tertentu seperti ID pelanggan, sedangkan *sorting* digunakan untuk mengurutkan data berdasarkan kriteria tertentu, seperti harga layanan. Penggunaan teori-teori ini memungkinkan sistem bekerja secara efisien, fleksibel, dan mendukung pengelolaan data yang berkelanjutan sesuai kebutuhan operasional layanan laundry.

3.2 *Source Code* Program

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <iomanip>
using namespace std;

struct dropOfLaundry
{
    int id;
    char nama[50];
    char noTelp[15];
    char tanggal[20];
    char layanan[10];
    float berat;
    int harga;
    dropOfLaundry *next;
};

dropOfLaundry *head = NULL; // Pointer head untuk linked list
Drop-Off

int hitungIDTerbesar()
{
    int lastID = 0;
    dropOfLaundry temp;

    FILE *file = fopen("dropOfDataLaundry.dat", "rb");
    if (file)
    {
        while (fread(&temp, sizeof(dropOfLaundry), 1, file)) //
Baca semua data dan ambil ID terakhir (paling besar dalam file)
            lastID = temp.id;
        fclose(file);
    }

    FILE *fileRiw = fopen("dropOfDataLaundryRiwayat.dat", "rb");
    if (fileRiw)
```

```
    {
        while (fread(&temp, sizeof(dropOfLaundry), 1, fileRiw))
// Baca semua data riwayat dan cari ID terbesar
            if (temp.id > lastID)
                lastID = temp.id;
        fclose(fileRiw);
    }

    return lastID;
}

int hitungHarga(const char *layanan, float berat)
{
    if (strcmp(layanan, "reguler") == 0) // Jika layanan reguler,
harga Rp7.000 per kg
        return (int)(7000 * berat);
    else if (strcmp(layanan, "express") == 0) // Jika layanan
express, harga Rp10.000 per kg
        return (int)(10000 * berat);
    return 0;
}

void loadDataFromFile() // Fungsi untuk memuat data dari file ke
dalam linked list
{
    FILE *file = fopen("dropOfDataLaundry.dat", "rb"); // Buka
file biner berisi data laundry Drop-Off
    if (!file)                                          // Jika
file tidak ditemukan, set head ke NULL dan keluar
    {
        head = NULL;
        return;
    }

    dropOfLaundry temp;          // Menyimpan data sementara dari
file
    dropOfLaundry *last = NULL; // Penunjuk ke node terakhir dalam
list
```

```
    head = NULL;                // Awalnya list kosong

    while (fread(&temp, sizeof(dropOfLaundry), 1, file)) // Baca file dan masukkan setiap data ke dalam linked list
    {
        dropOfLaundry *baru = new dropOfLaundry; // Alokasikan memori untuk node baru dan salin datanya
        *baru = temp;
        baru->next = NULL;

        if (head == NULL) // Jika list masih kosong, inisialisasi head dan last
        {
            head = baru;
            last = baru;
        }
        else
        {
            last->next = baru; // Tambahkan node baru di akhir list
            last = baru;
        }
    }

    fclose(file); // Tutup file setelah selesai dibaca
}

void dropOfTambahLaundryMasuk() // Fungsi untuk menambahkan data laundry Drop-Off baru ke linked list dan menyimpannya ke file
{
    loadDataFromFile(); // Memuat data lama dari file ke linked list (jika ada)

    dropOfLaundry *baru = new dropOfLaundry; // Alokasi memori untuk node baru
    baru->next = NULL;
```

```
    int lastID = hitungIDTerbesar(); // Menentukan ID baru
berdasarkan ID terbesar yang pernah ada
    baru->id = lastID + 1;

    cin.ignore();
    cout << "Masukkan Nama: "; // Input data pelanggan dari
pengguna
    cin.getline(baru->nama, 50);

    cout << "Masukkan No Telp: ";
    cin.getline(baru->noTelp, 15);

    cout << "Masukkan Tanggal (dd-mm-yyyy): ";
    cin.getline(baru->tanggal, 20);

    cout << "Masukkan Layanan (reguler/express): ";
    cin.getline(baru->layanan, 10);

    cout << "Masukkan Berat (kg): ";
    cin >> baru->berat;

    baru->harga = hitungHarga(baru->layanan, baru->berat); //
Hitung harga berdasarkan layanan dan berat

    if (head == NULL)
    {
        head = baru;
    }
    else
    {
        dropOfLaundry *temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = baru;
    }

    cout << "\nLaundry berhasil ditambahkan.\n";
```

```
    FILE *file = fopen("dropOfDataLaundry.dat", "wb"); // Simpan
seluruh linked list ke file (overwrite)
    if (file == NULL)
    {
        cout << "Gagal membuka file untuk menulis.\n";
        return;
    }

    dropOfLaundry *curr = head;
    while (curr != NULL)
    {
        fwrite(curr, sizeof(dropOfLaundry), 1, file);
        curr = curr->next;
    }

    fclose(file);

    cout << "Data laundry berhasil disimpan ke file.\n";
}

void dropOfTampilLaundryMasuk()
{
    FILE *file = fopen("dropOfDataLaundry.dat", "rb");
    if (file == NULL)
    {
        cout << "File dropOfDataLaundry.dat tidak ditemukan atau
kosong.\n";
        return;
    }

    dropOfLaundry temp;

    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    cout << "| ID  | Nama               | No Telp       | Tanggal
| Layanan  | Berat  | Harga    |\n";
    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
```

```
    while (fread(&temp, sizeof(dropOfLaundry), 1, file)) // Baca
dan tampilkan semua data laundry dari file
    {
        cout << "| " << setw(3) << right << temp.id << " "
             << "| " << setw(19) << left << temp.nama
             << "| " << setw(14) << left << temp.noTelp
             << "| " << setw(11) << left << temp.tanggal
             << "| " << setw(9) << left << temp.layanan
             <<  "|  "  <<  setw(6)  <<  right  <<  fixed  <<
setprecision(2) << temp.berat
             << " | " << setw(8) << right << temp.harga << " |\n";
    }

    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    fclose(file);
}

void dropOfAmbilLaundry()
{
    loadDataFromFile(); // Muat data laundry dari file ke linked
list
    int idCari;
    cout << "Masukkan ID laundry yang ingin diambil: ";
    cin >> idCari;

    dropOfLaundry *curr = head, *prev = NULL;
    bool found = false;

    while (curr != NULL) // Cari laundry berdasarkan ID yang
dimasukkan
    {
        if (curr->id == idCari)
        {
            found = true;
            break;
        }
```

```
        prev = curr;
        curr = curr->next;
    }

    if (!found)
    {
        cout << "Data dengan ID " << idCari << " tidak
ditemukan.\n";
        return;
    }

    if (prev == NULL) // Hapus node laundry dari linked list
        head = curr->next;
    else
        prev->next = curr->next;

    FILE *file = fopen("dropOfDataLaundry.dat", "wb"); // Simpan
linked list yang sudah dihapus node tadi kembali ke file
(overwrite)
    dropOfLaundry *temp = head;
    while (temp != NULL)
    {
        fwrite(temp, sizeof(dropOfLaundry), 1, file);
        temp = temp->next;
    }
    fclose(file);

    FILE *riwayat = fopen("dropOfDataLaundryRiwayat.dat", "ab");
// Tambahkan data laundry yang diambil ke file riwayat (append)
    fwrite(curr, sizeof(dropOfLaundry), 1, riwayat);
    fclose(riwayat);

    cout << "Laundry ID " << idCari << " telah diambil dan
dipindahkan ke riwayat.\n";
    delete curr; // Hapus node dari memori
}

void dropOfTampilRiwayat(bool urutHarga)
```

```
{
    FILE *file = fopen("dropOfDataLaundryRiwayat.dat", "rb");
    if (!file)
    {
        cout << "Riwayat masih kosong.\n";
        return;
    }

    dropOfLaundry *riwayat = NULL;

    dropOfLaundry temp;
    while (fread(&temp, sizeof(dropOfLaundry), 1, file)) // Muat
semua data riwayat ke linked list sementara
    {
        dropOfLaundry *baru = new dropOfLaundry;
        *baru = temp;
        baru->next = NULL;

        if (riwayat == NULL)
        {
            riwayat = baru;
        }
        else
        {
            dropOfLaundry *last = riwayat;
            while (last->next != NULL)
                last = last->next;
            last->next = baru;
        }
    }
    fclose(file);

    if  (urutHarga)  //  Jika  diminta,  urutkan  linked  list
berdasarkan harga secara ascending (bubble sort)
    {
        for (dropOfLaundry *i = riwayat; i != NULL; i = i->next)
        {
```

```
            for (dropOfLaundry *j = i->next; j != NULL; j = j-
>next)
            {
                if (i->harga > j->harga)
                {
                    swap(i->id, j->id);
                    swap(i->nama, j->nama);
                    swap(i->noTelp, j->noTelp);
                    swap(i->tanggal, j->tanggal);
                    swap(i->layanan, j->layanan);
                    swap(i->berat, j->berat);
                    swap(i->harga, j->harga);
                }
            }
        }
    }

    int totalPendapatan = 0;
    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    cout << "| ID  | Nama               | No Telp       | Tanggal
| Layanan  | Berat  | Harga    |\n";
    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";

    dropOfLaundry *curr = riwayat;
    while (curr)
    {
        cout << "| " << setw(3) << right << curr->id << " "
             << "| " << setw(19) << left << curr->nama
             << "| " << setw(14) << left << curr->noTelp
             << "| " << setw(11) << left << curr->tanggal
             << "| " << setw(9) << left << curr->layanan
             <<  "|  "  <<  setw(6)  <<  right  <<  fixed  <<
setprecision(2) << curr->berat
             << " | " << setw(8) << right << curr->harga << "
|\n";
        totalPendapatan += curr->harga;
```

```
        curr = curr->next;
    }

    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    cout << "Total Pendapatan: Rp" << totalPendapatan << endl;

    while (riwayat) // Bebaskan memori linked list sementara
    {
        dropOfLaundry *hapus = riwayat;
        riwayat = riwayat->next;
        delete hapus;
    }
}

void dropOfCariNotaLaundryByID()
{
    int idCari;
    cout << "Masukkan ID laundry yang ingin dicari: ";
    cin >> idCari;

    FILE *file = fopen("dropOfDataLaundry.dat", "rb");
    if (!file)
    {
        cout << "File dropOfDataLaundry.dat tidak ditemukan atau
kosong.\n";
        return;
    }

    dropOfLaundry temp;
    bool found = false;

    while (fread(&temp, sizeof(dropOfLaundry), 1, file)) // Baca
file record per record dan cari yang ID-nya sama dengan input
    {
        if (temp.id == idCari)
        {
            found = true;
```

```
            cout                                                    <<
"=========================================\n";
            cout << "            Nota Laundry ID: " << temp.id <<
endl;
            cout << "-----------------------------------------
-\n";
            cout << "Nama    : " << temp.nama << endl;
            cout << "No Telp : " << temp.noTelp << endl;
            cout << "Tanggal : " << temp.tanggal << endl;
            cout << "Layanan : " << temp.layanan << endl;
            cout << "Berat   : " << fixed << setprecision(2) <<
temp.berat << " kg\n";
            cout << "Harga   : Rp" << temp.harga << endl;
            cout                                                    <<
"=========================================\n";

            break;
        }
    }

    if (!found)
    {
        cout  <<  "Data  dengan  ID  "  <<  idCari  <<  "  tidak
ditemukan.\n";
    }

    fclose(file);
}

void dropOfHapusRiwayatByID()
{
    int idHapus;
    cout << "Masukkan ID laundry riwayat yang ingin dihapus: ";
    cin >> idHapus;

    FILE *file = fopen("dropOfDataLaundryRiwayat.dat", "rb");
    if (!file)
```

```
    {
        cout << "File riwayat kosong atau tidak ditemukan.\n";
        return;
    }

    dropOfLaundry *headRiw = NULL, *last = NULL;
    dropOfLaundry temp;

    while (fread(&temp, sizeof(dropOfLaundry), 1, file)) // Muat
semua data riwayat ke linked list
    {
        dropOfLaundry *baru = new dropOfLaundry;
        *baru = temp;
        baru->next = NULL;

        if (headRiw == NULL)
        {
            headRiw = baru;
            last = baru;
        }
        else
        {
            last->next = baru;
            last = baru;
        }
    }
    fclose(file);

    dropOfLaundry *curr = headRiw; // Cari node dengan ID yang
akan dihapus
    dropOfLaundry *prev = NULL;
    bool found = false;

    while (curr != NULL)
    {
        if (curr->id == idHapus)
        {
            found = true;
```

```
            if (prev == NULL) // Hapus node dari linked list
                headRiw = curr->next;
            else
                prev->next = curr->next;

            delete curr;
            break;
        }
        prev = curr;
        curr = curr->next;
    }

    if (!found)
    {
        cout << "Data dengan ID " << idHapus << " tidak ditemukan
di riwayat.\n";

        while (headRiw != NULL) // Bersihkan memori linked list
        {
            dropOfLaundry *hapus = headRiw;
            headRiw = headRiw->next;
            delete hapus;
        }
        return;
    }

    file = fopen("dropOfDataLaundryRiwayat.dat", "wb"); // Tulis
ulang file riwayat dengan data linked list yang sudah dihapus
node tertentu
    if (!file)
    {
        cout << "Gagal membuka file riwayat untuk menulis
ulang.\n";

        while (headRiw != NULL) // Bersihkan memori linked list
        {
            dropOfLaundry *hapus = headRiw;
            headRiw = headRiw->next;
```

```
            delete hapus;
        }
        return;
    }

    curr = headRiw;
    while (curr != NULL)
    {
        fwrite(curr, sizeof(dropOfLaundry), 1, file);
        curr = curr->next;
    }
    fclose(file);

    while (headRiw != NULL)
    {
        dropOfLaundry *hapus = headRiw;
        headRiw = headRiw->next;
        delete hapus;
    }

    cout << "Data riwayat dengan ID " << idHapus << " berhasil
dihapus.\n";
}

//================================================================
=========================================//

struct selfServiceLaundry
{
    int id;
    char nama[50];
    char noTelp[15];
    char tanggal[20];
    float berat;
    int jumlahMesin;
    int harga;
    selfServiceLaundry *next;
};
```

```
selfServiceLaundry *selfHead = NULL;

// Hitung jumlah mesin yang dibutuhkan berdasarkan berat laundry,
// setiap mesin maksimal 7 kg, jika berat > 7 maka mesin bertambah.
int hitungJumlahMesin(float berat)
{
    return (int)(berat / 7.0 + 0.999);
}

//  Hitung  harga  Self-Service  berdasarkan  jumlah  mesin  yang
digunakan.
// Tarif Rp 20.000 per mesin.
int hitungHargaSelf(int jumlahMesin)
{
    return jumlahMesin * 20000;
}

// Cari ID terbesar dari file riwayat Self-Service Laundry,
// digunakan untuk generate ID baru secara incremental.
int hitungIDTerbesarSelfService()
{
    int lastID = 0;
    selfServiceLaundry temp;

    FILE *fileRiw = fopen("selfServiceLaundryRiwayat.dat", "rb");
    if (fileRiw)
    {
        while   (fread(&temp,   sizeof(selfServiceLaundry),   1,
fileRiw))
            if (temp.id > lastID)
                lastID = temp.id;
        fclose(fileRiw);
    }

    return lastID;
}
```

```
void selfServiceTambahLaundry()
{
    selfServiceLaundry *baru = new selfServiceLaundry; // buat
node baru
    baru->next = NULL;

    static int idSelf = hitungIDTerbesarSelfService() + 1; //
inisialisasi ID unik
    baru->id = idSelf++;

    cin.ignore();
    cout << "Masukkan Nama: ";
    cin.getline(baru->nama, 50);
    cout << "Masukkan No Telp: ";
    cin.getline(baru->noTelp, 15);
    cout << "Masukkan Tanggal (dd-mm-yyyy): ";
    cin.getline(baru->tanggal, 20);
    cout << "Masukkan Berat Cucian (kg): ";
    cin >> baru->berat;

    baru->jumlahMesin = hitungJumlahMesin(baru->berat);
    baru->harga = hitungHargaSelf(baru->jumlahMesin);

    if (selfHead == NULL) // tambahkan node ke akhir linked list
        selfHead = baru;
    else
    {
        selfServiceLaundry *temp = selfHead;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = baru;
    }

    cout    <<    "\nData    self-service    laundry    berhasil
ditambahkan.\n";
}

void selfServiceCetakNota()
```

```
{
    if (selfHead == NULL)
    {
        cout << "Tidak ada data laundry untuk dicetak.\n";
        return;
    }

    selfServiceLaundry *hapus = selfHead; // ambil node depan
    selfHead = selfHead->next;              // hapus dari linked
list

    cout                                                    <<
"\n=====================================================\n";
    cout << "            Nota Self-Service Laundry\n";
    cout << "-----------------------------------------------------
\n";
    cout << "ID        : " << hapus->id << endl;
    cout << "Nama      : " << hapus->nama << endl;
    cout << "No Telp   : " << hapus->noTelp << endl;
    cout << "Tanggal   : " << hapus->tanggal << endl;
    cout << "Berat     : " << fixed << setprecision(2) << hapus-
>berat << " kg\n";
    cout << "Mesin     : " << hapus->jumlahMesin << " mesin\n";
    cout << "Harga     : Rp" << hapus->harga << endl;
    cout                                                    <<
"=====================================================\n";

    FILE *file = fopen("selfServiceLaundryRiwayat.dat", "ab"); //
simpan data ke file riwayat (append binary)
    if (file)
    {
        fwrite(hapus, sizeof(selfServiceLaundry), 1, file);
        fclose(file);
    }

    delete hapus; // hapus node dari memory
}
```

```
void selfServiceTampilRiwayat(bool urutHarga)
{
    FILE *file = fopen("selfServiceLaundryRiwayat.dat", "rb"); //
buka file riwayat baca
    if (!file)
    {
        cout << "Riwayat self-service masih kosong.\n";
        return;
    }

    selfServiceLaundry *riwayat = NULL; // linked list untuk
simpan data riwayat sementara

    selfServiceLaundry temp;
    while (fread(&temp, sizeof(selfServiceLaundry), 1, file)) //
baca data file per record
    {
        selfServiceLaundry *baru = new selfServiceLaundry; //
buat node baru
        *baru = temp;                                        //
salin data
        baru->next = NULL;

        if (riwayat == NULL) // tambahkan node ke akhir linked
list
            riwayat = baru;
        else
        {
            selfServiceLaundry *last = riwayat;
            while (last->next != NULL)
                last = last->next;
            last->next = baru;
        }
    }
    fclose(file);

    if (urutHarga) // jika pilih urut berdasarkan harga
    {
```

```
        for (selfServiceLaundry *i = riwayat; i != NULL; i = i->next)
        {
            for (selfServiceLaundry *j = i->next; j != NULL; j = j->next)
            {
                if (i->harga > j->harga)
                {
                    swap(i->id, j->id);
                    swap(i->nama, j->nama);
                    swap(i->noTelp, j->noTelp);
                    swap(i->tanggal, j->tanggal);
                    swap(i->berat, j->berat);
                    swap(i->jumlahMesin, j->jumlahMesin);
                    swap(i->harga, j->harga);
                }
            }
        }
    }

    int totalPendapatan = 0;
    cout << "+-----+--------------------+--------------+-------------+--------+-------------+----------+\n";
    cout << "| ID  | Nama               | No Telp      | Tanggal     | Berat  | Jumlah Mesin | Harga    |\n";
    cout << "+-----+--------------------+--------------+-------------+--------+-------------+----------+\n";

    selfServiceLaundry *curr = riwayat;
    while (curr)
    {
        cout << "| " << setw(3) << right << curr->id << " "
             << "| " << setw(19) << left << curr->nama
             << "| " << setw(14) << left << curr->noTelp
             << "| " << setw(11) << left << curr->tanggal
             <<  "|  "  <<  setw(6)  <<  right  <<  fixed  << setprecision(2) << curr->berat
             << " | " << setw(12) << right << curr->jumlahMesin
```

```
            << " | " << setw(8) << right << curr->harga << "
|\n";

        totalPendapatan   +=   curr->harga;   //   hitung   total
pendapatan
        curr = curr->next;
    }

    cout << "+-----+--------------------+--------------+-------
-----+--------+-------------+----------+\n";
    cout << "Total Pendapatan: Rp" << totalPendapatan << endl;

    while (riwayat) // hapus linked list agar tidak memory leak
    {
        selfServiceLaundry *hapus = riwayat;
        riwayat = riwayat->next;
        delete hapus;
    }
}

void selfServiceHapusRiwayatByID()
{
    int idHapus;
    cout << "Masukkan ID riwayat yang ingin dihapus: ";
    cin >> idHapus;

    FILE *file = fopen("selfServiceLaundryRiwayat.dat", "rb");
    if (!file)
    {
        cout << "Riwayat self-service masih kosong atau file tidak
ditemukan.\n";
        return;
    }

    selfServiceLaundry *head = NULL, *tail = NULL; // baca semua
data dari file ke linked list
    selfServiceLaundry temp;
    while (fread(&temp, sizeof(selfServiceLaundry), 1, file))
```

```
    {
        selfServiceLaundry *baru = new selfServiceLaundry;
        *baru = temp;
        baru->next = NULL;

        if (head == NULL)
            head = tail = baru;
        else
        {
            tail->next = baru;
            tail = baru;
        }
    }
    fclose(file);

    selfServiceLaundry *curr = head; // cari node dengan ID yang
ingin dihapus
    selfServiceLaundry *prev = NULL;
    bool found = false;

    while (curr != NULL)
    {
        if (curr->id == idHapus)
        {
            found = true;
            if (prev == NULL) // hapus head
                head = curr->next;
            else // hapus node selain head
                prev->next = curr->next;

            delete curr; // free memory node yang dihapus
            break;
        }
        prev = curr;
        curr = curr->next;
    }

    if (!found)
```

```
    {
        cout << "Data dengan ID " << idHapus << " tidak ditemukan
di riwayat.\n";

        while (head) // bersihkan linked list sebelum return
        {
            selfServiceLaundry *hapus = head;
            head = head->next;
            delete hapus;
        }
        return;
    }

    file = fopen("selfServiceLaundryRiwayat.dat", "wb"); // tulis
ulang file dengan data yang sudah dihapus node tertentu
    if (!file)
    {
        cout << "Gagal membuka file untuk penulisan ulang.\n";

        while (head) // bersihkan linked list sebelum return
        {
            selfServiceLaundry *hapus = head;
            head = head->next;
            delete hapus;
        }
        return;
    }

    curr = head;
    while (curr)
    {
        fwrite(curr, sizeof(selfServiceLaundry), 1, file);
        curr = curr->next;
    }
    fclose(file);

    while (head) // hapus linked list dari memory
    {
```

```
        selfServiceLaundry *hapus = head;
        head = head->next;
        delete hapus;
    }

    cout << "Data riwayat dengan ID " << idHapus << " berhasil
dihapus.\n";
}

int main()
{
    int pilih, subPilih;
    do
    {
        cout << "==== Menu Laundry Sistem ====\n\n";
        cout << "1. Drop-Off Laundry\n";
        cout << "2. Self-Service Laundry\n";
        cout << "0. Keluar\n";
        cout << "Pilih: ";
        cin >> pilih;

        system("cls");
        switch (pilih)
        {
        case 1:
            do
            {
                system("cls");
                cout << "\n==== Drop-Off Laundry ====\n";
                cout << "1. Tambah Laundry Masuk\n";
                cout << "2. Tampilkan Laundry [Masuk]\n";
                cout << "3. Ambil Laundry (Selesai)\n";
                cout << "4. Tampilkan Laundry [Riwayat]\n";
                cout << "5. Nota Laundry [Masuk] Berdasarkan
ID\n";
                cout << "6. Hapus Laundry [Riwayat] Berdasarkan
ID\n";
                cout << "0. Kembali\n";
```

```
                cout << "Pilih: ";
                cin >> subPilih;

                switch (subPilih)
                {
                case 1:
                    system("cls");
                    dropOfTambahLaundryMasuk();
                    system("pause");
                    break;
                case 2:
                    system("cls");
                    dropOfTampilLaundryMasuk();
                    system("pause");
                    break;
                case 3:
                    system("cls");
                    dropOfAmbilLaundry();
                    system("pause");
                    break;
                case 4:
                {
                    int subSubPilih;
                    do
                    {
                        system("cls");
                        cout << "==== Drop-Off Laundry - Tampilkan Laundry [Riwayat] ====\n";
                        cout << "1. Tampilkan Riwayat Laundry Natural\n";
                        cout << "2. Tampilkan Riwayat Laundry Terurut Berdasarkan Harga\n";
                        cout << "0. Kembali\n";
                        cout << "Pilih: ";
                        cin >> subSubPilih;

                        switch (subSubPilih)
                        {
```

```
                        case 1:
                            system("cls");
                            cout << "Menampilkan data tanpa
urut...\n";
                            dropOfTampilRiwayat(false);
                            system("pause");
                            break;
                        case 2:
                            system("cls");
                            cout << "Menampilkan data
terurut...\n";
                            dropOfTampilRiwayat(true);
                            system("pause");
                            break;
                        case 0:
                            cout << "Kembali ke menu
sebelumnya...\n";
                            system("pause");
                            break;
                        default:
                            cout << "Pilihan tidak valid!\n";
                            system("pause");
                            break;
                        }

                    } while (subSubPilih != 0);
                }
                break;
                case 5:
                    system("cls");
                    dropOfCariNotaLaundryByID();
                    system("pause");
                    break;
                case 6:
                    system("cls");
                    dropOfHapusRiwayatByID();
                    system("pause");
                case 0:
```

```
                    cout << "Kembali ke menu utama...\n";
                    system("pause");
                    system("cls");
                    break;
                default:
                    cout << "Pilihan tidak valid!\n";
                    system("pause");
                    break;
                }
            } while (subPilih != 0);
            break;

        case 2:
            do
            {
                system("cls");
                cout << "\n==== Self-Service Laundry ====\n";
                cout << "1. Tambah Laundry [Masuk]\n";
                cout << "2. Cetak Nota Laundry\n";
                cout << "3. Tampilkan Laundry [Riwayat]\n";
                cout << "4. Hapus Laundry [Riwayat] Berdasarkan
ID\n";
                cout << "0. Kembali\n";
                cout << "Pilih: ";
                cin >> subPilih;

                switch (subPilih)
                {
                case 1:
                    system("cls");
                    selfServiceTambahLaundry();
                    system("pause");
                    break;
                case 2:
                    system("cls");
                    selfServiceCetakNota();
                    system("pause");
                    break;
```

```
                case 3:
                {
                    int subSubPilih;
                    do
                    {
                        system("cls");
                        cout << "==== Self-Service Laundry - 
Tampilkan Laundry [Riwayat] ====\n";
                        cout << "1. Tampilkan Riwayat Laundry 
Natural\n";
                        cout << "2. Tampilkan Riwayat Laundry 
Terurut Berdasarkan Harga\n";
                        cout << "0. Kembali\n";
                        cout << "Pilih: ";
                        cin >> subSubPilih;

                        switch (subSubPilih)
                        {
                        case 1:
                            system("cls");
                            cout << "Menampilkan data tanpa 
urut...\n";
                            selfServiceTampilRiwayat(false);
                            system("pause");
                            break;
                        case 2:
                            system("cls");
                            cout << "Menampilkan data 
terurut...\n";
                            selfServiceTampilRiwayat(true);
                            system("pause");
                            break;
                        case 0:
                            cout << "Kembali ke menu 
sebelumnya...\n";
                            system("pause");
                            break;
                        default:
```

```
                            cout << "Pilihan tidak valid!\n";
                            system("pause");
                            break;
                        }

                    } while (subSubPilih != 0);
                }
                break;
                case 4:
                    system("cls");
                    selfServiceHapusRiwayatByID();
                    system("pause");
                    break;
                case 0:
                    cout << "Kembali ke menu utama...\n";
                    system("cls");
                    break;
                default:
                    cout << "Pilihan tidak valid!\n";
                    system("pause");
                    break;
                }

            } while (subPilih != 0);
            break;

        case 0:
            cout << "Terima kasih telah menggunakan sistem
laundry. Semoga sehat selalu :)\n";
            break;

        default:
            cout << "Pilihan tidak valid!\n";
            system("pause");
            break;
        }

    } while (pilih != 0);
```

```
    return 0;
}
```

3.3 Implementasi Materi

3.3.1 *Struct Linked List*

```
struct dropOfLaundry
{
    int id;
    char nama[50];
    char noTelp[15];
    char tanggal[20];
    char layanan[10];
    float berat;
    int harga;
    dropOfLaundry *next;
};
```

Struktur dropOfLaundry mengimplementasikan konsep *linked list* untuk menyimpan data transaksi laundry secara dinamis. Dengan menggunakan *pointer next*, setiap data terhubung satu sama lain, memungkinkan penambahan dan penghapusan elemen tanpa batasan jumlah seperti pada *array*. Hal ini sangat bermanfaat dalam sistem laundry karena jumlah transaksi tidak tetap, sehingga penyimpanan data menjadi lebih fleksibel dan efisien.

3.3.2 *Pointer*

```
dropOfLaundry *head = NULL;
```

Deklarasi *dropOfLaundry *head = NULL*, mengimplementasikan materi *pointer* untuk menunjuk ke *node* pertama dalam struktur *linked list*. Variabel head berfungsi sebagai titik awal dari data laundry yang tersimpan secara dinamis. Inisialisasi dengan *NULL* menandakan bahwa saat program dimulai, belum ada data yang dimasukkan. Penggunaan *pointer* ini penting untuk mengelola alokasi memori secara efisien dan memudahkan manipulasi data seperti penambahan dan penghapusan *node* dalam *linked list*.

3.3.3 *File Handling (Biner)* dan *Searching*

```
int hitungIDTerbesar()
{
    int lastID = 0;
    dropOfLaundry temp;
```

```
    FILE *file = fopen("dropOfDataLaundry.dat", "rb");
    if (file)
    {
        while (fread(&temp, sizeof(dropOfLaundry), 1, file))
            lastID = temp.id;
        fclose(file);
    }

    FILE *fileRiw = fopen("dropOfDataLaundryRiwayat.dat", "rb");
    if (fileRiw)
    {
        while (fread(&temp, sizeof(dropOfLaundry), 1, fileRiw))
            if (temp.id > lastID)
                lastID = temp.id;
        fclose(fileRiw);
    }

    return lastID;
}
```

Fungsi “hitungIDTerbesar()” mengimplementasikan konsep *file handling* biner dan *searching* untuk mencari ID terbesar dari dua file data (dropOfDataLaundry.dat dan dropOfDataLaundryRiwayat.dat). Fungsi ini membuka kedua file dalam mode baca *biner* (*rb*), membaca setiap data menggunakan *“fread”*, lalu menyimpan nilai ID terbesar yang ditemukan. Hal ini berguna untuk memastikan bahwa ID transaksi berikutnya akan selalu unik dan berurutan, meskipun data sudah berpindah ke file riwayat.

3.3.4 Fungsi dan *Pointer*

```
int hitungHarga(const char *layanan, float berat)
{
    if (strcmp(layanan, "reguler") == 0)
        return (int)(7000 * berat);
    else if (strcmp(layanan, "express") == 0)
        return (int)(10000 * berat);
    return 0;
}
```

Fungsi hitungHarga menggunakan *parameter pointer* layanan dan variabel berat untuk menghitung total harga *laundry* berdasarkan jenis layanan dan berat pakaian. Fungsi

ini memanfaatkan perbandingan *string* menggunakan *"strcmp"* untuk menentukan tarif per kilogram yang berbeda antara layanan *reguler* dan *express*, sehingga memungkinkan perhitungan harga yang fleksibel dan akurat sesuai layanan yang dipilih pelanggan.

3.3.5 *Linked List* dan *File Handling (Biner)*

```
void loadDataFromFile()
{
    FILE *file = fopen("dropOfDataLaundry.dat", "rb");
    if (!file)
    {
        head = NULL;
        return;
    }

    dropOfLaundry temp;
    dropOfLaundry *last = NULL;
    head = NULL;

    while (fread(&temp, sizeof(dropOfLaundry), 1, file))
    {
        dropOfLaundry *baru = new dropOfLaundry;
        *baru = temp;
        baru->next = NULL;

        if (head == NULL)
        {
            head = baru;
            last = baru;
        }
        else
        {
            last->next = baru;
            last = baru;
        }
    }

    fclose(file);
}
```

Fungsi “loadDataFromFile” mengimplementasikan konsep *linked list* dan *file handling* biner untuk memuat data laundry Drop-Off dari file ke dalam memori secara dinamis. Data dibaca satu per satu dari file biner dan setiap entri disimpan ke node baru dalam *linked list*. *Pointer head* menyimpan alamat *node* pertama, sementara *pointer last* membantu menambahkan *node* baru di akhir *list*, sehingga struktur data tetap terhubung secara berurutan dan efisien.

3.3.6 *Linked List, Pointer*, dan *File Handling (Biner)*

```
void dropOfTambahLaundryMasuk()
{
    loadDataFromFile();

    dropOfLaundry *baru = new dropOfLaundry;
    baru->next = NULL;

    int lastID = hitungIDTerbesar();
    baru->id = lastID + 1;

    cin.ignore();
    cout << "Masukkan Nama: ";
    cin.getline(baru->nama, 50);

    cout << "Masukkan No Telp: ";
    cin.getline(baru->noTelp, 15);

    cout << "Masukkan Tanggal (dd-mm-yyyy): ";
    cin.getline(baru->tanggal, 20);

    cout << "Masukkan Layanan (reguler/express): ";
    cin.getline(baru->layanan, 10);

    cout << "Masukkan Berat (kg): ";
    cin >> baru->berat;

    baru->harga = hitungHarga(baru->layanan, baru->berat);

    if (head == NULL)
    {
```

```
        head = baru;
    }
    else
    {
        dropOfLaundry *temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = baru;
    }

    cout << "\nLaundry berhasil ditambahkan.\n";

    FILE *file = fopen("dropOfDataLaundry.dat", "wb");
    if (file == NULL)
    {
        cout << "Gagal membuka file untuk menulis.\n";
        return;
    }

    dropOfLaundry *curr = head;
    while (curr != NULL)
    {
        fwrite(curr, sizeof(dropOfLaundry), 1, file);
        curr = curr->next;
    }

    fclose(file);

    cout << "Data laundry berhasil disimpan ke file.\n";
}
```

Fungsi ”dropOfTambahLaundryMasuk” menggabungkan konsep *linked list*, *pointer*, dan *file handling biner* untuk menambahkan data *laundry Drop-Off* baru. Data lama dimuat dari file ke *linked list*, kemudian data baru dimasukkan sebagai *node* baru di akhir *list*. ID otomatis ditentukan berdasarkan ID terbesar saat ini. Setelah *input* data, seluruh *linked list* ditulis ulang ke file, menjaga data tetap konsisten. Pendekatan ini memastikan data dapat bertambah secara dinamis dan tersimpan permanen.

3.3.7 *File Handling (Biner)* dan *Output* Format

```
void dropOfTampilLaundryMasuk()
{
    FILE *file = fopen("dropOfDataLaundry.dat", "rb");
    if (file == NULL)
    {
        cout << "File dropOfDataLaundry.dat tidak ditemukan atau
kosong.\n";
        return;
    }

    dropOfLaundry temp;

    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    cout << "| ID  | Nama               | No Telp       | Tanggal
| Layanan  | Berat  | Harga    |\n";
    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";

    while (fread(&temp, sizeof(dropOfLaundry), 1, file))
    {
        cout << "| " << setw(3) << right << temp.id << " "
             << "| " << setw(19) << left << temp.nama
             << "| " << setw(14) << left << temp.noTelp
             << "| " << setw(11) << left << temp.tanggal
             << "| " << setw(9) << left << temp.layanan
             <<  "|  "  <<  setw(6)  <<  right  <<  fixed  <<
setprecision(2) << temp.berat
             << " | " << setw(8) << right << temp.harga << " |\n";
    }

    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    fclose(file);
}
```

Fungsi “dropOfTampilLaundryMasuk” mengimplementasikan *file handling* biner untuk membaca data *laundry Drop-Off* dari file dan menampilkan informasi secara

terformat menggunakan manipulasi *output (setw, setprecision, dll).* Fungsi ini memastikan bahwa data dari file dapat ditampilkan dengan rapi dalam bentuk tabel yang mudah dibaca, sehingga memudahkan pengguna untuk melihat daftar laundry yang sedang masuk.

3.3.8 *Linked List, Pointer,* dan *File Handling (Biner)*

```
void dropOfAmbilLaundry()
{
    loadDataFromFile();
    int idCari;
    cout << "Masukkan ID laundry yang ingin diambil: ";
    cin >> idCari;

    dropOfLaundry *curr = head, *prev = NULL;
    bool found = false;

    while (curr != NULL)
    {
        if (curr->id == idCari)
        {
            found = true;
            break;
        }
        prev = curr;
        curr = curr->next;
    }

    if (!found)
    {
        cout << "Data dengan ID " << idCari << " tidak
ditemukan.\n";
        return;
    }

    if (prev == NULL)
        head = curr->next;
    else
        prev->next = curr->next;
```

```
    FILE *file = fopen("dropOfDataLaundry.dat", "wb");
    dropOfLaundry *temp = head;
    while (temp != NULL)
    {
        fwrite(temp, sizeof(dropOfLaundry), 1, file);
        temp = temp->next;
    }
    fclose(file);

    FILE *riwayat = fopen("dropOfDataLaundryRiwayat.dat", "ab");
    fwrite(curr, sizeof(dropOfLaundry), 1, riwayat);
    fclose(riwayat);

    cout << "Laundry ID " << idCari << " telah diambil dan
dipindahkan ke riwayat.\n";
    delete curr;
}
```

Fungsi ”dropOfAmbilLaundry” menggabungkan penggunaan *linked list* dan file *handling biner* untuk menghapus data laundry berdasarkan ID dari *linked list* dan memperbarui file data utama. Data yang dihapus kemudian disalin ke file riwayat untuk pencatatan. Proses ini melibatkan pencarian *node* menggunakan *pointer*, penghapusan *node* dari *linked list*, dan penulisan ulang data ke file, sehingga menjamin konsistensi data dan pencatatan riwayat transaksi.

3.3.9 *Linked List, Sorting (Bubble Sort),* dan *File Handling (Biner)*

```
void dropOfTampilRiwayat(bool urutHarga)
{
    FILE *file = fopen("dropOfDataLaundryRiwayat.dat", "rb");
    if (!file)
    {
        cout << "Riwayat masih kosong.\n";
        return;
    }

    dropOfLaundry *riwayat = NULL;

    dropOfLaundry temp;
    while (fread(&temp, sizeof(dropOfLaundry), 1, file))
```

```
    {
        dropOfLaundry *baru = new dropOfLaundry;
        *baru = temp;
        baru->next = NULL;

        if (riwayat == NULL)
        {
            riwayat = baru;
        }
        else
        {
            dropOfLaundry *last = riwayat;
            while (last->next != NULL)
                last = last->next;
            last->next = baru;
        }
    }
    fclose(file);

    if (urutHarga)
    {
        for (dropOfLaundry *i = riwayat; i != NULL; i = i->next)
        {
            for (dropOfLaundry *j = i->next; j != NULL; j = j-
>next)
            {
                if (i->harga > j->harga)
                {
                    swap(i->id, j->id);
                    swap(i->nama, j->nama);
                    swap(i->noTelp, j->noTelp);
                    swap(i->tanggal, j->tanggal);
                    swap(i->layanan, j->layanan);
                    swap(i->berat, j->berat);
                    swap(i->harga, j->harga);
                }
            }
        }
```

```
    }

    int totalPendapatan = 0;
    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    cout << "| ID  | Nama               | No Telp       | Tanggal
| Layanan  | Berat  | Harga    |\n";
    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";

    dropOfLaundry *curr = riwayat;
    while (curr)
    {
        cout << "| " << setw(3) << right << curr->id << " "
             << "| " << setw(19) << left << curr->nama
             << "| " << setw(14) << left << curr->noTelp
             << "| " << setw(11) << left << curr->tanggal
             << "| " << setw(9) << left << curr->layanan
             <<  "|  "  <<  setw(6)  <<  right  <<  fixed  <<
setprecision(2) << curr->berat
             << " | " << setw(8) << right << curr->harga << "
|\n";
        totalPendapatan += curr->harga;
        curr = curr->next;
    }

    cout << "+-----+--------------------+--------------+-------
-----+----------+--------+----------+\n";
    cout << "Total Pendapatan: Rp" << totalPendapatan << endl;

    while (riwayat)
    {
        dropOfLaundry *hapus = riwayat;
        riwayat = riwayat->next;
        delete hapus;
    }
}
```

Fungsi “dropOfTampilRiwayat” mengimplementasikan penggunaan *linked list* untuk memuat data riwayat laundry dari file *biner* ke memori, kemudian menampilkan data tersebut. Jika *parameter* urutHarga diaktifkan, data diurutkan menggunakan metode b*ubble sort* berdasarkan harga secara *ascending*. Penggunaan *linked list* memudahkan pengelolaan data dinamis, sedangkan *sorting* dan *file handling* memastikan data tersimpan dan dapat ditampilkan dengan urutan yang diinginkan, sekaligus menghitung total pendapatan secara langsung.

3.3.10 Pencarian Data (*Searching*) pada *File Biner*

```
void dropOfCariNotaLaundryByID()
{
    int idCari;
    cout << "Masukkan ID laundry yang ingin dicari: ";
    cin >> idCari;

    FILE *file = fopen("dropOfDataLaundry.dat", "rb");
    if (!file)
    {
        cout << "File dropOfDataLaundry.dat tidak ditemukan atau
kosong.\n";
        return;
    }

    dropOfLaundry temp;
    bool found = false;

    while (fread(&temp, sizeof(dropOfLaundry), 1, file))
    {
        if (temp.id == idCari)
        {
            found = true;

            cout                                                 <<
"=========================================\n";
            cout << "           Nota Laundry ID: " << temp.id <<
endl;
            cout << "-----------------------------------------
-\n";
```

```
            cout << "Nama    : " << temp.nama << endl;
            cout << "No Telp : " << temp.noTelp << endl;
            cout << "Tanggal : " << temp.tanggal << endl;
            cout << "Layanan : " << temp.layanan << endl;
            cout << "Berat   : " << fixed << setprecision(2) <<
temp.berat << " kg\n";
            cout << "Harga   : Rp" << temp.harga << endl;
            cout                                                <<
"=========================================\n";

            break;
        }
    }

    if (!found)
    {
        cout  <<  "Data  dengan  ID  "  <<  idCari  <<  "  tidak
ditemukan.\n";
    }

    fclose(file);
}
```

Fungsi ini menggunakan teknik pencarian *linear* untuk mencari data laundry berdasarkan ID pada file *biner* (dropOfDataLaundry.dat). Setiap *record* dibaca satu per satu menggunakan "*fread"* hingga ditemukan ID yang sesuai atau file habis dibaca. Jika data ditemukan, nota lengkap ditampilkan dengan format rapi menggunakan *iomanip*. Jika tidak ditemukan, pesan pemberitahuan ditampilkan. Pendekatan ini sederhana dan efektif untuk file dengan ukuran kecil hingga menengah tanpa memerlukan *indeks* khusus.

3.3.11 Penghapusan Data dari *File Biner* dengan *Linked List*

```
void dropOfHapusRiwayatByID()
{
    int idHapus;
    cout << "Masukkan ID laundry riwayat yang ingin dihapus: ";
    cin >> idHapus;

    FILE *file = fopen("dropOfDataLaundryRiwayat.dat", "rb");
    if (!file)
```

```
    {
        cout << "File riwayat kosong atau tidak ditemukan.\n";
        return;
    }

    dropOfLaundry *headRiw = NULL, *last = NULL;
    dropOfLaundry temp;

    // Memuat seluruh data riwayat dari file ke linked list
    while (fread(&temp, sizeof(dropOfLaundry), 1, file))
    {
        dropOfLaundry *baru = new dropOfLaundry;
        *baru = temp;
        baru->next = NULL;

        if (headRiw == NULL)
        {
            headRiw = baru;
            last = baru;
        }
        else
        {
            last->next = baru;
            last = baru;
        }
    }
    fclose(file);

    // Mencari dan menghapus node linked list dengan ID yang
sesuai
    dropOfLaundry *curr = headRiw;
    dropOfLaundry *prev = NULL;
    bool found = false;

    while (curr != NULL)
    {
        if (curr->id == idHapus)
        {
```

```
            found = true;
            if (prev == NULL)
                headRiw = curr->next;
            else
                prev->next = curr->next;

            delete curr;
            break;
        }
        prev = curr;
        curr = curr->next;
    }

    if (!found)
    {
        cout << "Data dengan ID " << idHapus << " tidak ditemukan
di riwayat.\n";
        // Bersihkan memori linked list sebelum keluar
        while (headRiw != NULL)
        {
            dropOfLaundry *hapus = headRiw;
            headRiw = headRiw->next;
            delete hapus;
        }
        return;
    }

    // Menulis ulang file riwayat dengan data linked list yang
sudah dihapus node tertentu
    file = fopen("dropOfDataLaundryRiwayat.dat", "wb");
    if (!file)
    {
        cout << "Gagal membuka file riwayat untuk menulis
ulang.\n";
        // Bersihkan memori linked list sebelum keluar
        while (headRiw != NULL)
        {
            dropOfLaundry *hapus = headRiw;
```

```
            headRiw = headRiw->next;
            delete hapus;
        }
        return;
    }

    curr = headRiw;
    while (curr != NULL)
    {
        fwrite(curr, sizeof(dropOfLaundry), 1, file);
        curr = curr->next;
    }
    fclose(file);

    // Bersihkan memori linked list setelah selesai
    while (headRiw != NULL)
    {
        dropOfLaundry *hapus = headRiw;
        headRiw = headRiw->next;
        delete hapus;
    }

    cout << "Data riwayat dengan ID " << idHapus << " berhasil
dihapus.\n";
}
```

Fungsi ini mengilustrasikan penghapusan data spesifik dari file *biner* dengan memanfaatkan *linked list* sebagai struktur data sementara. Proses dimulai dengan membaca seluruh isi file riwayat (dropOfDataLaundryRiwayat.dat) ke dalam *linked list* dinamis, memungkinkan manipulasi data secara fleksibel. Setelah data dimuat, pencarian dilakukan untuk menemukan *node* dengan ID yang sesuai. Jika ditemukan, *node* tersebut dihapus dari *linked list* dan memori dibebaskan. Selanjutnya, file riwayat ditulis ulang secara penuh berdasarkan isi *linked list* yang sudah *terupdate*, sehingga menghilangkan data yang dihapus. Pendekatan ini efektif untuk file yang tidak besar, karena melibatkan pembacaan dan penulisan ulang seluruh file. Pengelolaan memori yang baik dilakukan dengan menghapus semua *node linked list* saat fungsi selesai untuk mencegah *memory leak*.

5.3.12 Struct Linked List

```
struct selfServiceLaundry
```

```
{
    int id;
    char nama[50];
    char noTelp[15];
    char tanggal[20];
    float berat;
    int jumlahMesin;
    int harga;
    selfServiceLaundry *next;
};
```

Struktur “selfServiceLaundry” adalah tipe data node dalam *linked list* tunggal yang menyimpan data pelanggan *laundry Self-Service*. *Node* ini berisi beberapa atribut penting seperti id untuk identifikasi unik, nama pelanggan, noTelp sebagai kontak, tanggal transaksi, berat cucian dalam kilogram, jumlahMesin yang diperlukan (dihitung berdasarkan berat cucian), dan harga yang merepresentasikan total biaya layanan. Atribut *next* adalah *pointer* yang menghubungkan *node* ini dengan *node* berikutnya dalam *linked list*, memungkinkan pengelolaan data secara dinamis dan efisien di memori.

3.3.13 Deklarasi *Pointer Head* untuk *Linked List*

```
selfServiceLaundry *selfHead = NULL;
```

Variabel *selfHead* adalah pointer bertipe “selfServiceLaundry” yang berfungsi sebagai penunjuk (*head*) awal dari linked list untuk data *laundry Self-Service*. Inisialisasi dengan *NULL* menunjukkan bahwa *linked list* pada awalnya kosong, belum ada data pelanggan yang tercatat. Pointer ini menjadi titik awal untuk *traversing* dan manipulasi seluruh data laundry dalam *list* tersebut.

3.3.14 Fungsi Perhitungan Mesin, Harga, dan ID di *Self-Service* Laundry

```
int hitungJumlahMesin(float berat)
{
    return (int)(berat / 7.0 + 0.999);
}

int hitungHargaSelf(int jumlahMesin)
{
    return jumlahMesin * 20000;
}

int hitungIDTerbesarSelfService()
{
```

```
    int lastID = 0;
    selfServiceLaundry temp;

    FILE *fileRiw = fopen("selfServiceLaundryRiwayat.dat", "rb");
    if (fileRiw)
    {
        while   (fread(&temp,   sizeof(selfServiceLaundry),   1,
fileRiw))
            if (temp.id > lastID)
                lastID = temp.id;
        fclose(fileRiw);
    }

    return lastID;
}
```

Fungsi “hitungJumlahMesin” menghitung jumlah mesin yang diperlukan untuk mencuci dengan berat tertentu dimana setiap mesin dapat menangani maksimal 7 kg Penambahan 0.999 sebelum konversi ke *integer* memastikan pembulatan ke atas. Fungsi ”hitungHargaSelf” menghitung total biaya berdasarkan jumlah mesin yang digunakan dengan tarif tetap Rp 20.000 per mesin. Fungsi ”hitungIDTerbesarSelfService” membuka file riwayat *Self-Service Laundry* membaca semua data untuk mencari ID terbesar yang ada yang kemudian digunakan untuk menentukan ID baru secara berurutan agar tidak ada duplikasi Jika file tidak ditemukan fungsi mengembalikan nilai 0 sebagai ID terakhir.

3.3.15 *Linked List*

```
void selfServiceTambahLaundry()
{
    selfServiceLaundry *baru = new selfServiceLaundry;
    baru->next = NULL;

    static int idSelf = hitungIDTerbesarSelfService() + 1;
    baru->id = idSelf++;

    cin.ignore();
    cout << "Masukkan Nama: ";
    cin.getline(baru->nama, 50);
    cout << "Masukkan No Telp: ";
    cin.getline(baru->noTelp, 15);
```

```
    cout << "Masukkan Tanggal (dd-mm-yyyy): ";
    cin.getline(baru->tanggal, 20);
    cout << "Masukkan Berat Cucian (kg): ";
    cin >> baru->berat;

    baru->jumlahMesin = hitungJumlahMesin(baru->berat);
    baru->harga = hitungHargaSelf(baru->jumlahMesin);

    if (selfHead == NULL)
        selfHead = baru;
    else
    {
        selfServiceLaundry *temp = selfHead;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = baru;
    }

    cout    <<    "\nData    self-service    laundry    berhasil
ditambahkan.\n";
}
```

Fungsi ini menambah data *laundry self-service* baru ke dalam *linked list* secara dinamis ID unik dihasilkan berdasarkan nilai terbesar ID yang sudah ada di file riwayat lalu *increment* setiap penambahan data baru *Input user* berupa nama, nomor telepon, tanggal, dan berat cucian Berat cucian digunakan untuk menghitung jumlah mesin yang diperlukan dan harga total berdasarkan fungsi “hitungJumlahMesin” dan “hitungHargaSelf” *Node* baru kemudian ditambahkan ke akhir “*linked list selfHead”* Jika *linked list* kosong *node* baru menjadi *head.*

3.3.16 *Linked List* dan File

```
void selfServiceCetakNota()
{
    if (selfHead == NULL)
    {
        cout << "Tidak ada data laundry untuk dicetak.\n";
        return;
    }
```

```
    selfServiceLaundry *hapus = selfHead;
    selfHead = selfHead->next;

    cout                                                    <<
"\n=====================================================\n";
    cout << "            Nota Self-Service Laundry\n";
    cout << "-----------------------------------------------------
\n";
    cout << "ID        : " << hapus->id << endl;
    cout << "Nama      : " << hapus->nama << endl;
    cout << "No Telp   : " << hapus->noTelp << endl;
    cout << "Tanggal   : " << hapus->tanggal << endl;
    cout << "Berat     : " << fixed << setprecision(2) << hapus-
>berat << " kg\n";
    cout << "Mesin     : " << hapus->jumlahMesin << " mesin\n";
    cout << "Harga     : Rp" << hapus->harga << endl;
    cout                                                    <<
"=====================================================\n";

    FILE *file = fopen("selfServiceLaundryRiwayat.dat", "ab");
    if (file)
    {
        fwrite(hapus, sizeof(selfServiceLaundry), 1, file);
        fclose(file);
    }

    delete hapus;
}
```

Fungsi ini mengambil data laundry paling depan dari *linked list self-service*, menampilkan nota laundry lengkap, lalu menyimpan data tersebut ke file riwayat secara *binary* dan menghapus *node* tersebut dari memori, sehingga data yang sudah diambil tidak lagi tersimpan di antrian *linked list*.

3.3.17 *Linked List*, File, dan *Sorting*

```
void selfServiceTampilRiwayat(bool urutHarga)
{
    FILE *file = fopen("selfServiceLaundryRiwayat.dat", "rb");
    if (!file)
```

```
    {
        cout << "Riwayat self-service masih kosong.\n";
        return;
    }

    selfServiceLaundry *riwayat = NULL;

    selfServiceLaundry temp;
    while (fread(&temp, sizeof(selfServiceLaundry), 1, file))
    {
        selfServiceLaundry *baru = new selfServiceLaundry;
        *baru = temp;
        baru->next = NULL;

        if (riwayat == NULL)
            riwayat = baru;
        else
        {
            selfServiceLaundry *last = riwayat;
            while (last->next != NULL)
                last = last->next;
            last->next = baru;
        }
    }
    fclose(file);

    if (urutHarga)
    {
        for (selfServiceLaundry *i = riwayat; i != NULL; i = i-
>next)
        {
            for (selfServiceLaundry *j = i->next; j != NULL; j =
j->next)
            {
                if (i->harga > j->harga)
                {
                    swap(i->id, j->id);
                    swap(i->nama, j->nama);
```

```
                    swap(i->noTelp, j->noTelp);
                    swap(i->tanggal, j->tanggal);
                    swap(i->berat, j->berat);
                    swap(i->jumlahMesin, j->jumlahMesin);
                    swap(i->harga, j->harga);
                }
            }
        }
    }

    int totalPendapatan = 0;
    cout << "+-----+--------------------+--------------+-------
-----+--------+-------------+----------+\n";
    cout << "| ID  | Nama               | No Telp       | Tanggal
| Berat  | Jumlah Mesin | Harga    |\n";
    cout << "+-----+--------------------+--------------+-------
-----+--------+-------------+----------+\n";

    selfServiceLaundry *curr = riwayat;
    while (curr)
    {
        cout << "| " << setw(3) << right << curr->id << " "
             << "| " << setw(19) << left << curr->nama
             << "| " << setw(14) << left << curr->noTelp
             << "| " << setw(11) << left << curr->tanggal
             << "| " << setw(6) << right << fixed << 
setprecision(2) << curr->berat
             << " | " << setw(12) << right << curr->jumlahMesin
             << " | " << setw(8) << right << curr->harga << "
|\n";

        totalPendapatan += curr->harga;
        curr = curr->next;
    }

    cout << "+-----+--------------------+--------------+-------
-----+--------+-------------+----------+\n";
    cout << "Total Pendapatan: Rp" << totalPendapatan << endl;
```

```
    while (riwayat)
    {
        selfServiceLaundry *hapus = riwayat;
        riwayat = riwayat->next;
        delete hapus;
    }
}
```

Fungsi ini membaca data riwayat *laundry self-service* dari *file binary* ke *linked list* sementara, kemudian jika parameter urutHarga bernilai *"true"*, melakukan pengurutan *ascending* berdasarkan harga menggunakan algoritma *bubble sort* dengan menukar seluruh isi *node*. Data riwayat lalu ditampilkan dalam format tabel rapi menggunakan *setw* dan *setprecision* serta menghitung total pendapatan dari seluruh transaksi. Setelah itu *linked list* dibersihkan agar tidak terjadi *memory leak*.

3.3.18 *Linked List* dan File

```
void selfServiceHapusRiwayatByID()
{
    int idHapus;
    cout << "Masukkan ID riwayat yang ingin dihapus: ";
    cin >> idHapus;

    FILE *file = fopen("selfServiceLaundryRiwayat.dat", "rb");
    if (!file)
    {
        cout << "Riwayat self-service masih kosong atau file tidak
ditemukan.\n";
        return;
    }

    selfServiceLaundry *head = NULL, *tail = NULL;
    selfServiceLaundry temp;
    while (fread(&temp, sizeof(selfServiceLaundry), 1, file))
    {
        selfServiceLaundry *baru = new selfServiceLaundry;
        *baru = temp;
        baru->next = NULL;
```

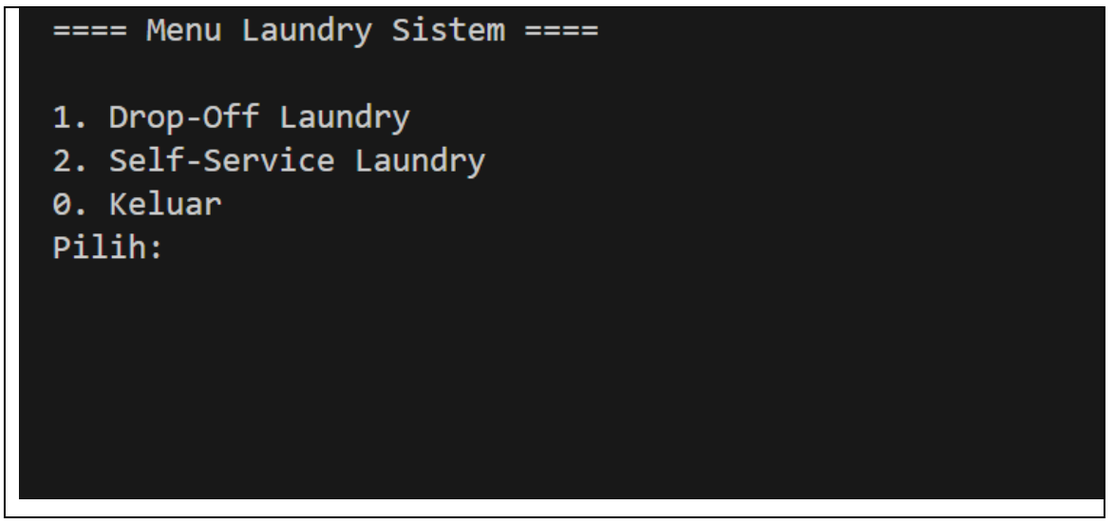
```
        if (head == NULL)
            head = tail = baru;
        else
        {
            tail->next = baru;
            tail = baru;
        }
    }
    fclose(file);

    selfServiceLaundry *curr = head;
    selfServiceLaundry *prev = NULL;
    bool found = false;

    while (curr != NULL)
    {
        if (curr->id == idHapus)
        {
            found = true;
            if (prev == NULL)
                head = curr->next;
            else
                prev->next = curr->next;

            delete curr;
            break;
        }
        prev = curr;
        curr = curr->next;
    }

    if (!found)
    {
        cout << "Data dengan ID " << idHapus << " tidak ditemukan
di riwayat.\n";

        while (head)
        {
```

```
            selfServiceLaundry *hapus = head;
            head = head->next;
            delete hapus;
        }
        return;
    }

    file = fopen("selfServiceLaundryRiwayat.dat", "wb");
    if (!file)
    {
        cout << "Gagal membuka file untuk penulisan ulang.\n";

        while (head)
        {
            selfServiceLaundry *hapus = head;
            head = head->next;
            delete hapus;
        }
        return;
    }

    curr = head;
    while (curr)
    {
        fwrite(curr, sizeof(selfServiceLaundry), 1, file);
        curr = curr->next;
    }
    fclose(file);

    while (head)
    {
        selfServiceLaundry *hapus = head;
        head = head->next;
        delete hapus;
    }

    cout << "Data riwayat dengan ID " << idHapus << " berhasil
dihapus.\n";
```

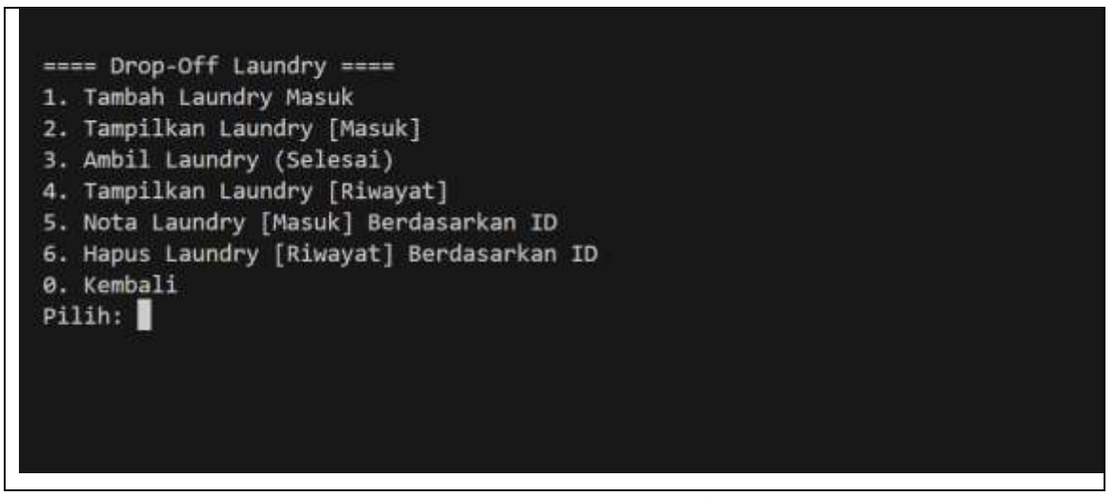
```
}
```

Fungsi ini membaca semua data dari file riwayat *self-service laundry* ke dalam *linked list*, mencari node dengan ID yang ingin dihapus, dan jika ditemukan, menghapus node tersebut dari *linked list*. Setelah itu file ditulis ulang menggunakan isi *linked list* terbaru tanpa *node* yang dihapus. Fungsi ini menggabungkan penggunaan *linked list* sebagai struktur sementara dengan file sebagai penyimpanan permanen, dan memastikan tidak terjadi *memory leak* dengan menghapus seluruh *node* setelah proses selesai.

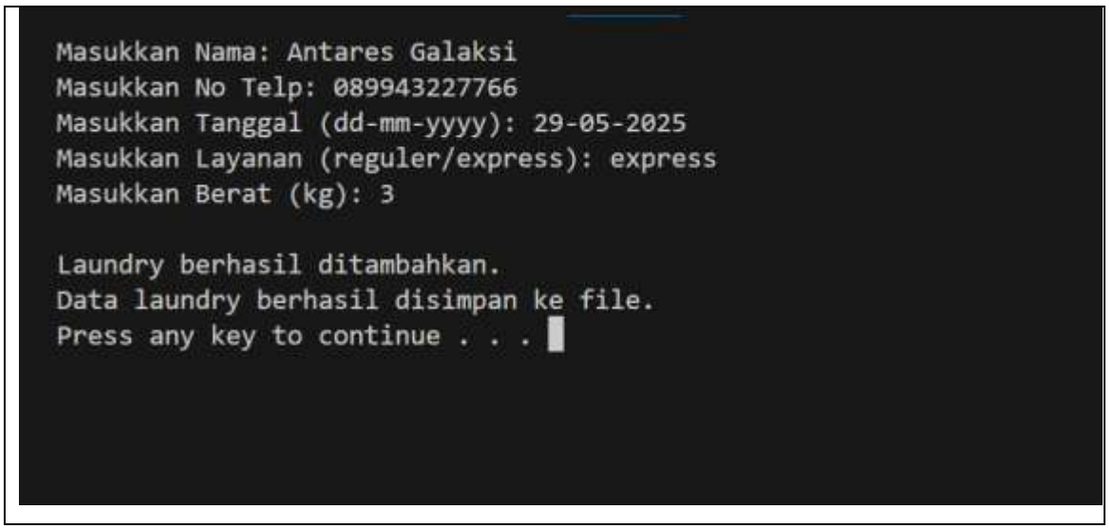
3.4 *Screenshot* Program



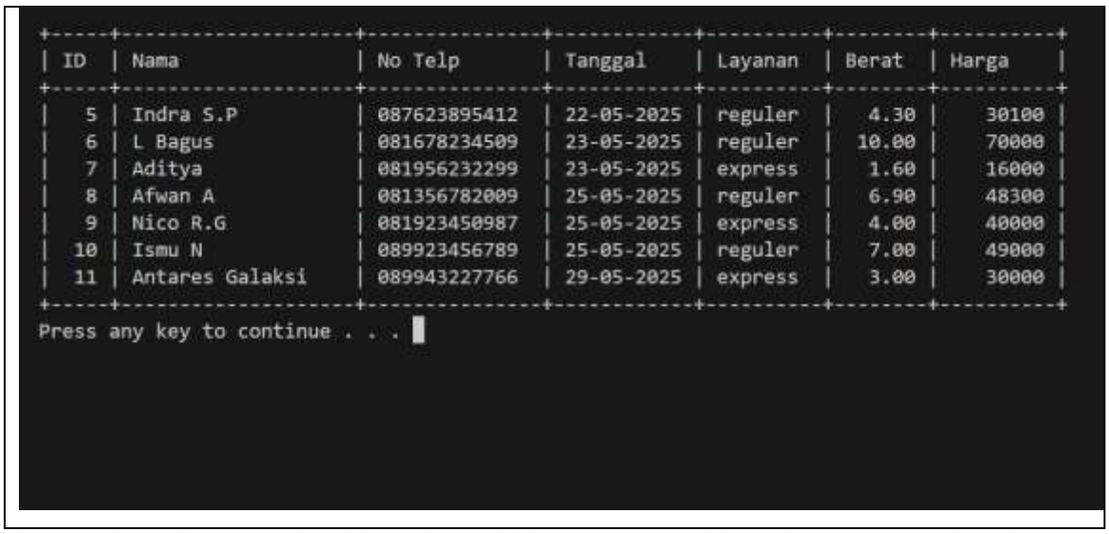
Gambar 3.4.1 Tampilan Menu Utama



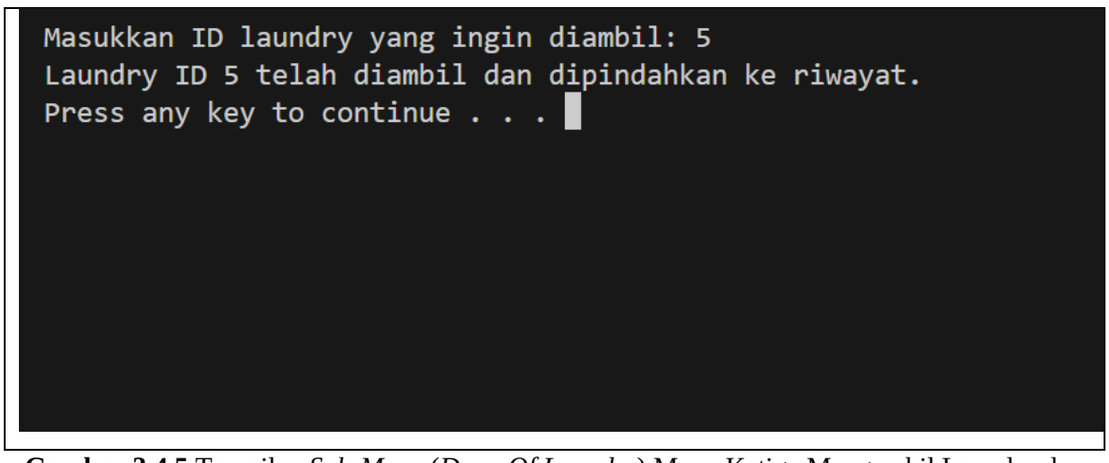
Gambar 3.4.2 Tampilan *Sub-Menu* (*Drop-Of Laundry*)



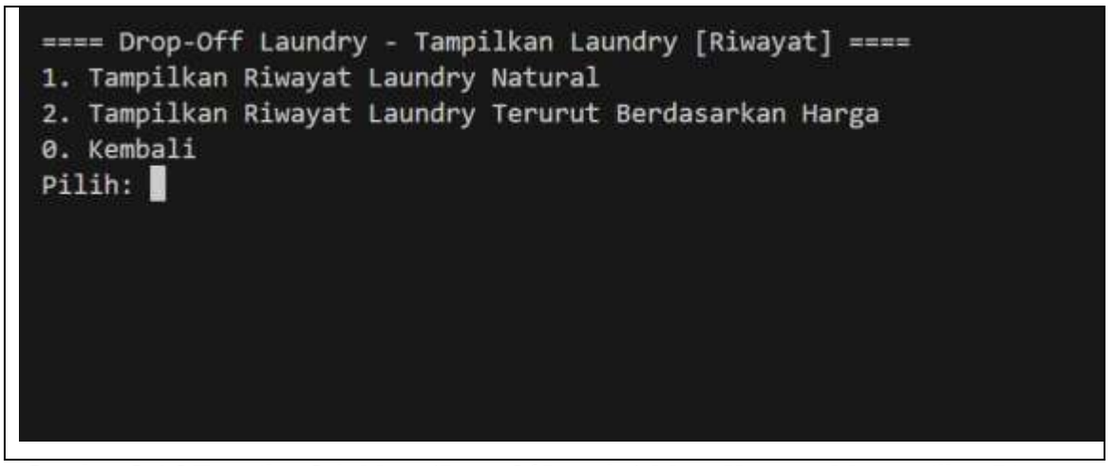
Gambar 3.4.3 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Pertama Menambahkan Data



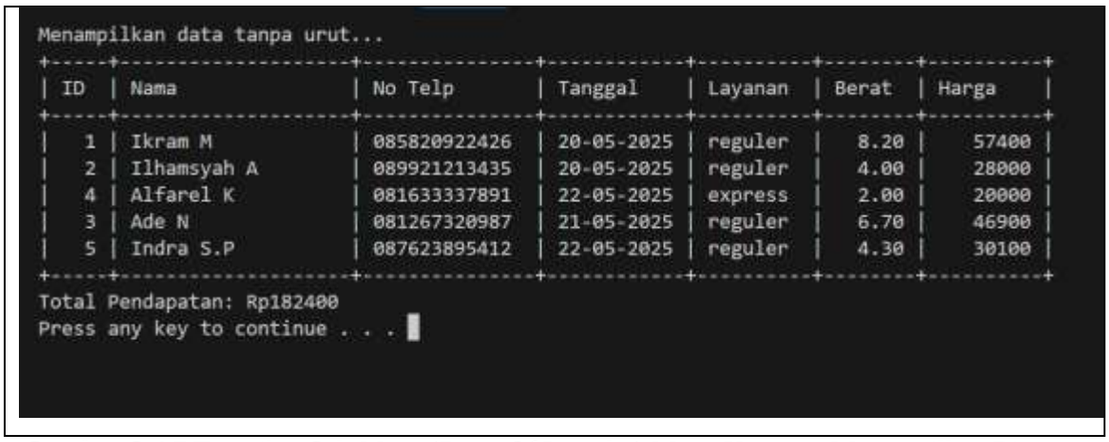
Gambar 3.4.4 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Kedua Menampilkan Data Laundry Masuk



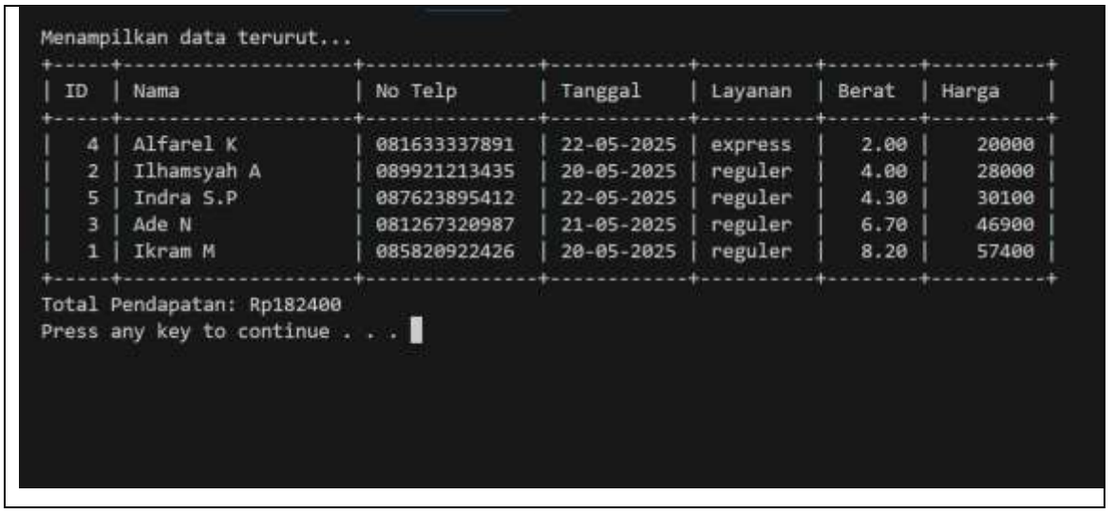
Gambar 3.4.5 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Ketiga Mengambil Laundry dan Memindahkan ke File Riwayat



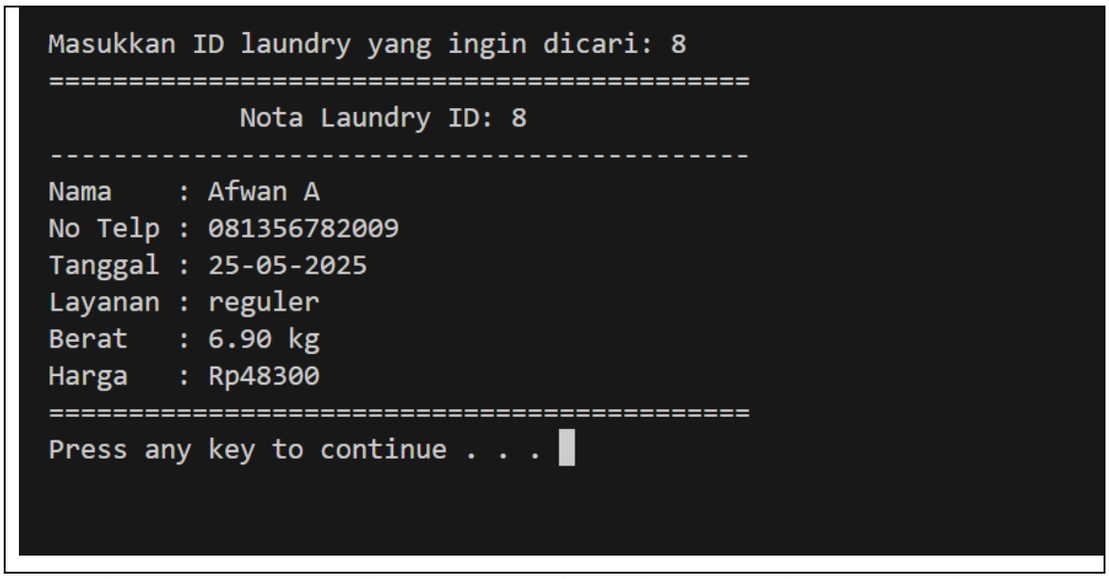
Gambar 3.4.6 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Keempat Menampilkan Riwayat Laundry Terdapat Dua SubSub-Menu



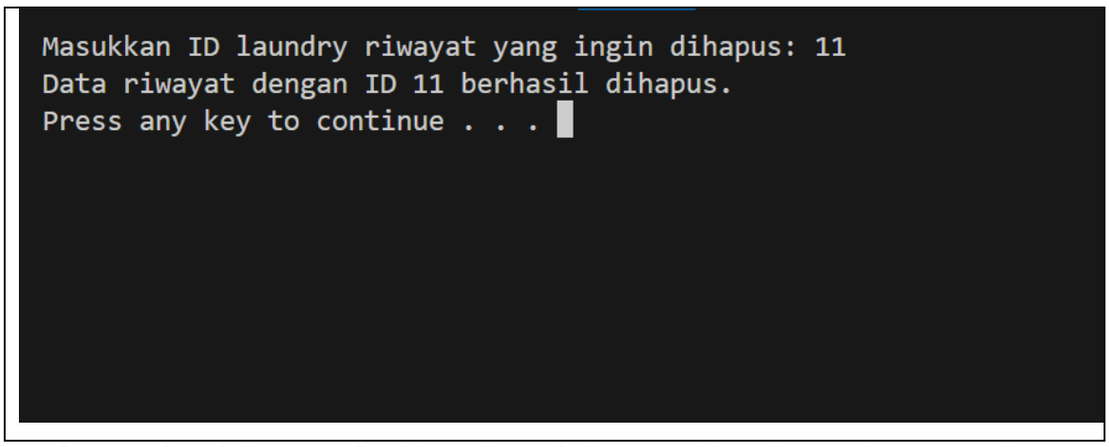
Gambar 3.4.7 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Keempat Menampilkan Riwayat Natural



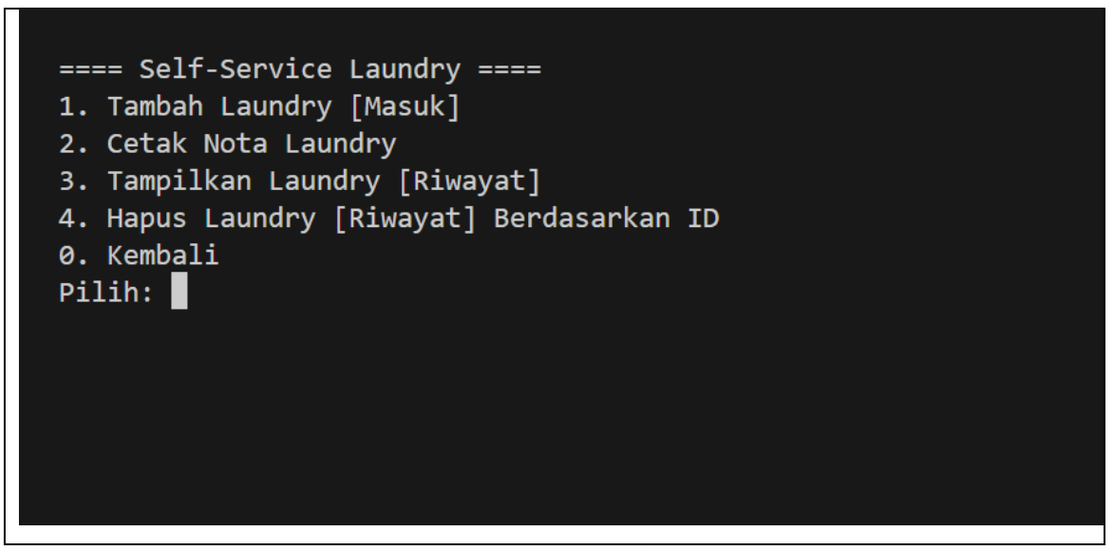
Gambar 3.4.8 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Keempat Menampilkan Riwayat Terurut



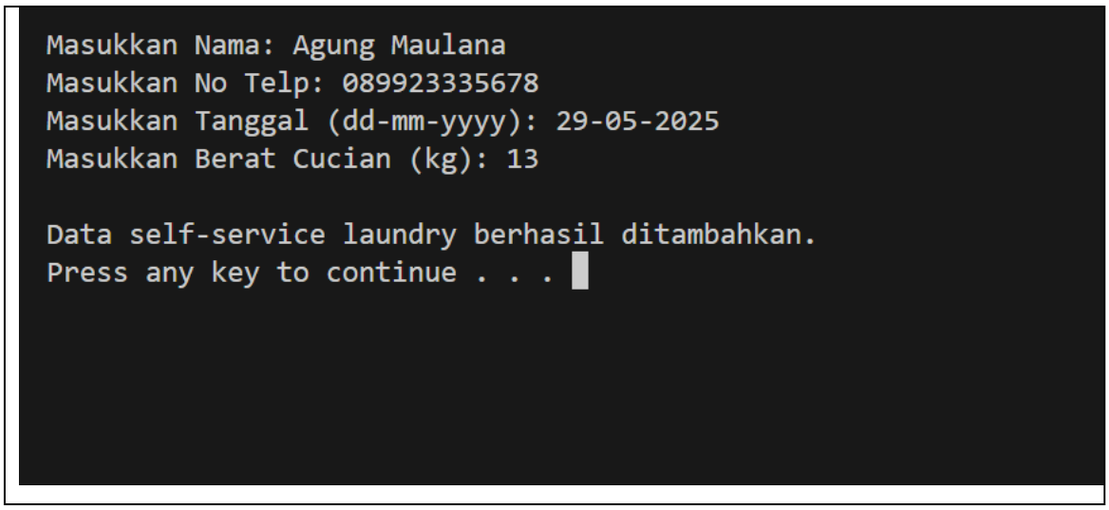
Gambar 3.4.9 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Kelima Menampilkan Nota Laundry Masuk dari ID



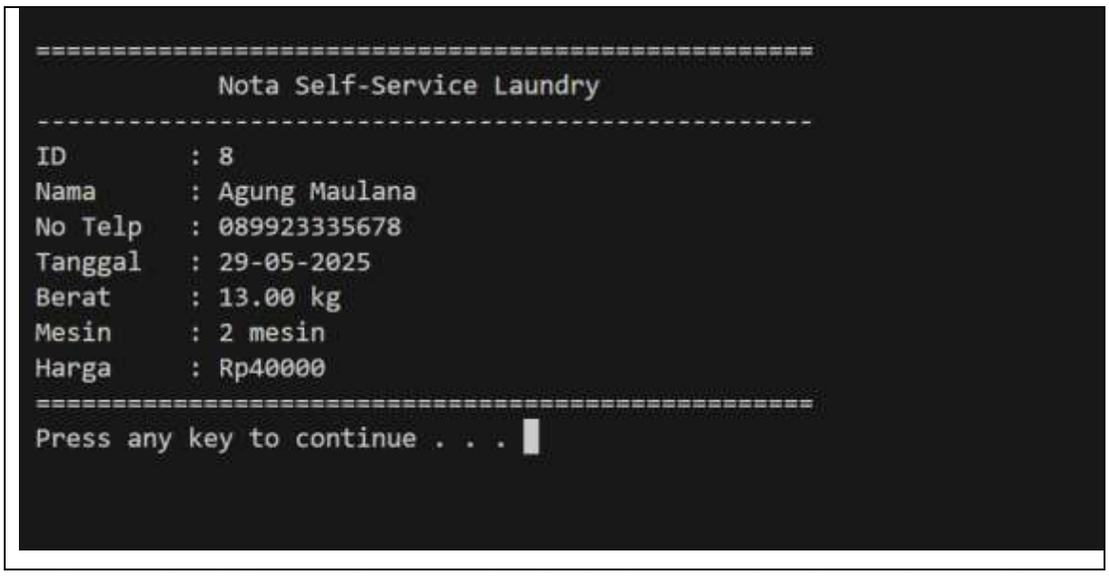
Gambar 3.4.10 Tampilan *Sub-Menu* (*Drop-Of Laundry*) Menu Keenam Menghapus Data Laundry Riwayat



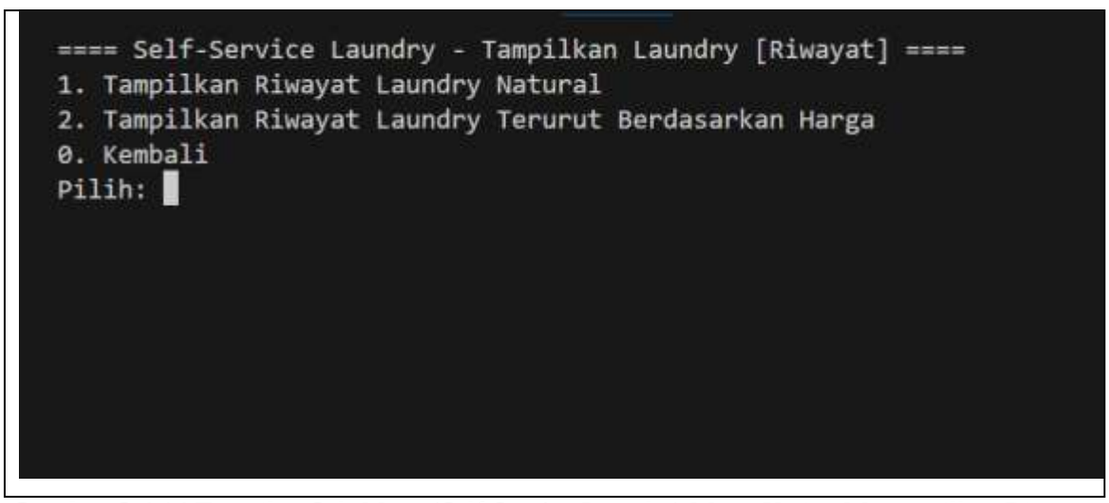
Gambar 3.4.11 Tampilan *Sub-Menu* (*Self-Service Laundry*)



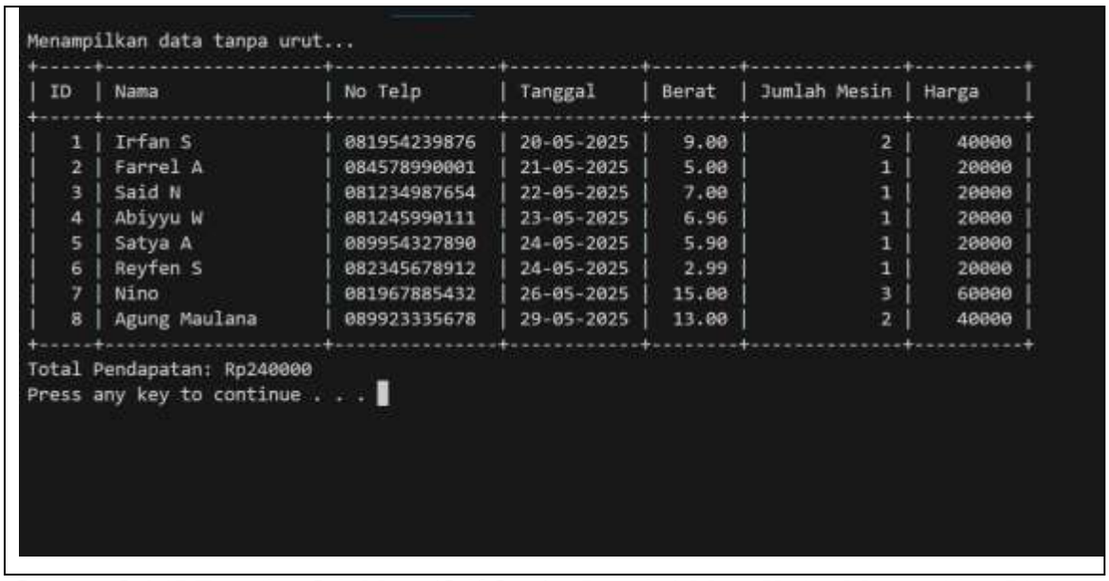
Gambar 3.4.12 Tampilan *Sub-Menu* (*Self-Service Laundry*) Menu Pertama Menambahkan Data Laundry



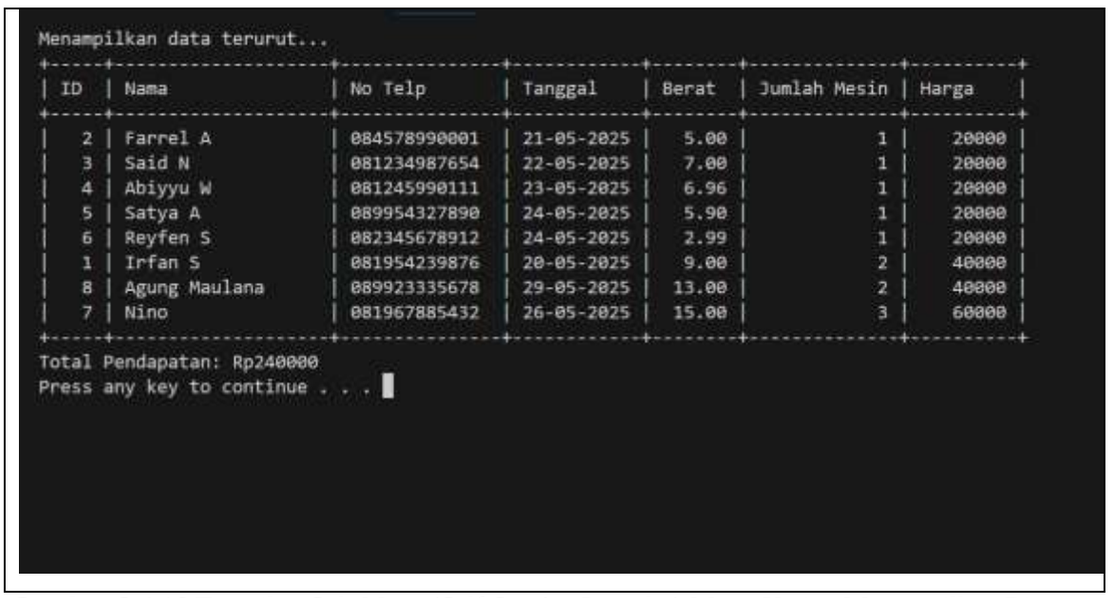
Gambar 3.4.13 Tampilan *Sub-Menu* (*Self-Service Laundry*) Menu Kedua Mencetak Nota Laundry dari Antrian Paling Depan dan Memindahkan ke Riwayat



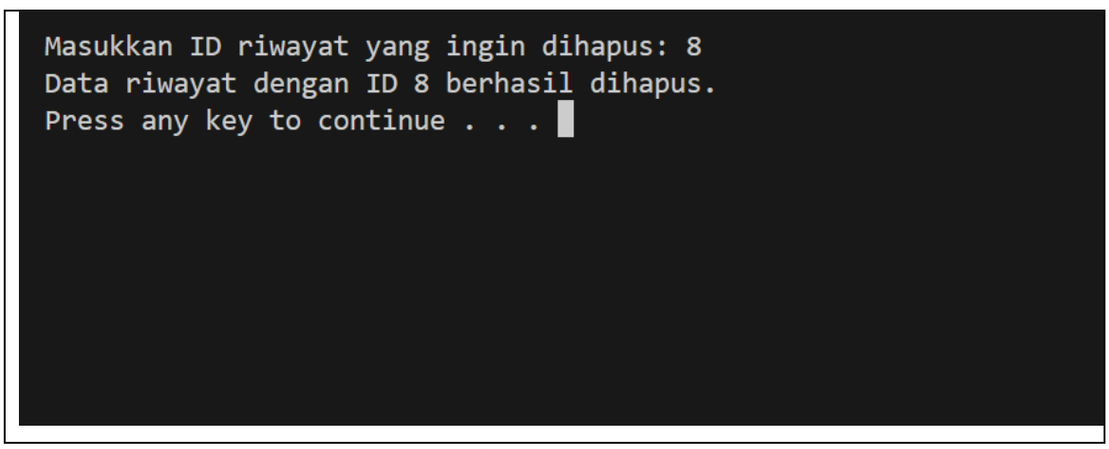
Gambar 3.4.14 Tampilan *Sub-Menu* (*Self-Service Laundry*) Menu Ketiga Menampilkan Riwayat Laundry Terdapat Dua SubSub-Menu



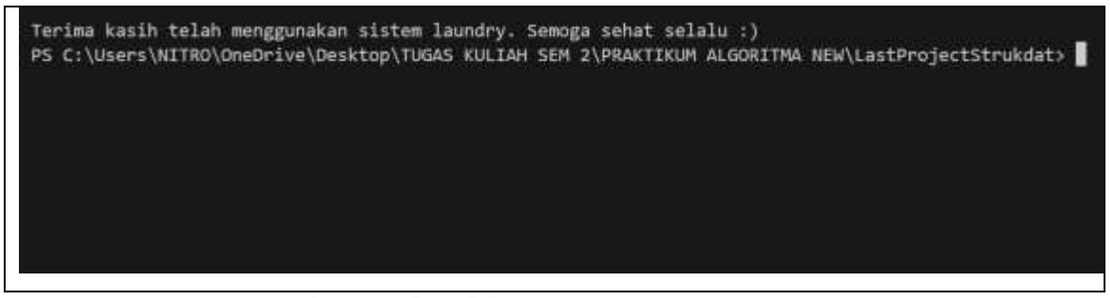
Gambar 3.4.15 Tampilan *Sub-Menu* (*Self-Service Laundry*) Menu Ketiga Menampilkan Riwayat Natural



Gambar 3.4.16 Tampilan *Sub-Menu* (*Self-Service Laundry*) Menu Ketiga Menampilkan Riwayat Terurut



Gambar 3.4.17 Tampilan *Sub-Menu* (*Self-Service Laundry*) Menu Keempat Menghapus Riwayat Laundry dari ID



Gambar 3.4.18 Keluar dari Program

3.5 Jadwal Pengerjaan

Berikut ini merupakan jadwal singkat dalam pengerjaan proyek akhir "Sistem Laundry", yaitu:

Tabel 3.5.1 Jadwal Pengerjaan Proyek Akhir "Struk Transaksi Apotek"

No	Kegiatan	Pengerjaan				
		Mei				
		1	2	3	4	5
1	Mengkaji Masalah					
2	Merancang Alur Program					
3	Integrasi *Code*					
4	Mengimplementasi *Code Drop-Of Laundry*					
5	Mengimplementasi *Code Self-Service Laundry*					
6	Membuat Tampilan Menu					
7	Membuat Tampilan *Output*					

3.6 Pembagian Tugas

Berikut ini merupakan pembagian tugas selama pengerjaan proyek akhir "Sistem Laundry", yaitu:

Tabel 3.6.1 Pembagian Tugas Proyek Akhir "Struk Transaksi Apotek"

No	Kegiatan	Penanggung Jawab	
		Muhammad Ikram Mughni	Ilhamsyah Adi Krisna
1	Mengkaji Masalah		
2	Merancang Alur Program		
3	Integrasi *Code*		
4	Mengimplementasi *Code Drop-Of Laundry*		

5	Mengimplementasi *Code Self-Service Laundry*		
6	Membuat Tampilan Menu		
7	Membuat Tampilan *Output*		

LAMPIRAN