LAPORAN PRAKTIKUM

POSTTEST 6

ALGORITMA PEMROGRAMAN LANJUT

Disusun oleh:

Langgeng Dimas

Saputra

(2409106121)

Kelas (C2 ‘24)

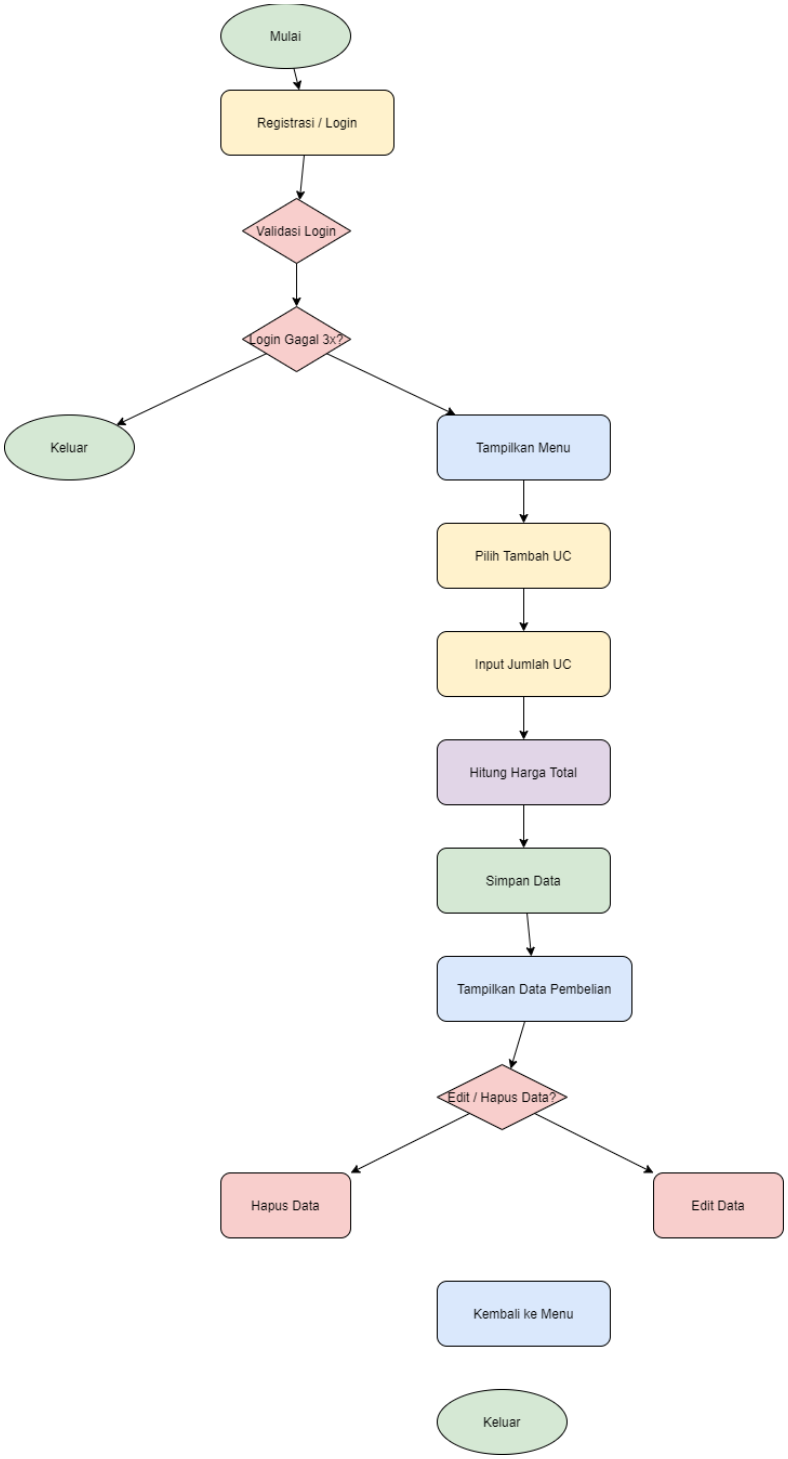
PROGRAM STUDI INFORMATIKA

UNIVERSITAS MULAWARMAN

SAMARINDA

2025

1. Flowchart


Mulai
Registrasi / Login
Validasi Login
Login Gagal 3x?
Keluar
Tampilkan Menu
Pilih Tambah UC
Input Jumlah UC
Hitung Harga Total
Simpan Data
Tampilkan Data Pembelian
Edit / Hapus Data?
Hapus Data
Edit Data
Kembali ke Menu
Keluar

2. Analisis Program

2.1 Deskripsi Singkat Program

Tujuan:

Program ini dibuat untuk **mensimulasikan transaksi pembelian Unknown Cash (UC) dalam game PUBG**. Program ini memiliki dua jenis pengguna:

1. **Admin** → Dapat melihat daftar harga dan pesanan pembeli.
2. **Pembeli** → Dapat melihat harga UC, melakukan pembelian, dan melihat total pembayaran.

Program ini juga menerapkan sistem login untuk **admin** (username = "dimas", password = "121"), sedangkan **pembeli tidak perlu login**.

2.2 Penjelasan Alur & Algoritma

A. Penjelasan Alur Program

Program ini adalah sistem pembelian UC (Unknown Cash) yang berbasis multiuser. Pengguna bisa mendaftar (registrasi), login, dan melakukan transaksi pembelian UC. Program memiliki fitur CRUD (Create, Read, Update, Delete) untuk data pembelian UC. Harga 1 UC adalah 3000.
Fitur utama:

1. Multiuser → Bisa registrasi dan login dengan nama serta NIM.
2. Pembelian UC → Pengguna memasukkan jumlah UC, dan harga langsung dihitung.
3. CRUD Pembelian → Bisa menambah, melihat, mengedit, dan menghapus transaksi pembelian.
4. Batas Login → Jika gagal login 3 kali berturut-turut, program akan berhenti.
5. Looping Menu → Program berjalan terus hingga pengguna memilih logout atau keluar.

B. Algoritma

Tampilkan menu utama:

- Pilihan: Registrasi, Login, atau Keluar.

Jika pengguna memilih Registrasi:

- Input nama dan NIM.
- Simpan ke dalam daftar user.
- Kembali ke menu utama.

Jika pengguna memilih Login:

- Input nama dan NIM.
- Periksa apakah user ada dalam daftar.
- Jika salah, beri kesempatan login ulang hingga maksimal 3 kali.
- Jika berhasil, masuk ke menu pembelian UC.

Di menu pembelian UC, pengguna bisa memilih:

- Tambah Pembelian UC:
    - Masukkan jumlah UC.
    - Harga dihitung otomatis (jumlah UC × 3000).
    - Simpan ke dalam daftar transaksi user.
- Lihat Data Pembelian:
    - Tampilkan daftar UC yang sudah dibeli beserta total harganya.
- Edit Data Pembelian:
    - Pilih transaksi yang ingin diubah.
    - Masukkan jumlah UC baru, lalu hitung ulang harga.
- Hapus Data Pembelian:
    - Pilih transaksi yang ingin dihapus dari daftar.
- Logout:

- Kembali ke menu utama.

Jika pengguna memilih Keluar, program akan berhenti.

3 Source Code

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

struct User {
    string nama;
    string nim;
};

struct PembelianUC {
    int jumlah;
    double harga;
};

struct DataUser {
    User user;
    vector<PembelianUC> pembelian;
};

vector<DataUser> daftarUser;

// --- Fungsi Registrasi dan Login ---
void registrasiUser(vector<DataUser>* daftar) {
    DataUser newUser;
    cout << "\n--- Registrasi User ---\n";
    cout << "Masukkan Nama: "; cin.ignore(); getline(cin, newUser.user.nama);
    cout << "Masukkan NIM: "; cin >> newUser.user.nim;
    daftar->push_back(newUser);
    cout << "Registrasi berhasil! Silakan login.\n";
}

int loginUser(vector<DataUser>* daftar) {
    string nama, nim;
    int attempts = 3;
    while (attempts--) {
        cout << "\n--- Login ---\n";
        cout << "Masukkan Nama: "; cin.ignore(); getline(cin, nama);
        cout << "Masukkan NIM: "; cin >> nim;

        for (int i = 0; i < daftar->size(); i++) {
            if ((*daftar)[i].user.nama == nama && (*daftar)[i].user.nim == nim) {
                cout << "Login berhasil!\n";
                return i;
            }
        }
        cout << "Login gagal! Sisa percobaan: " << attempts << "\n";
```

```
    }
    return -1;
}

// --- Fungsi CRUD ---
void tambahPembelian(DataUser* user) {
    PembelianUC pembelian;
    cout << "\nMasukkan jumlah UC yang ingin dibeli: ";
    cin >> pembelian.jumlah;
    pembelian.harga = pembelian.jumlah * 3000;
    user->pembelian.push_back(pembelian);
    cout << "Total harga: " << pembelian.harga << "\n";
}

void tampilkanData(const DataUser& user) {
    cout << "\n--- Data Pembelian ---\n";
    for (int i = 0; i < user.pembelian.size(); i++) {
        cout << i + 1 << ". Jumlah UC: " << user.pembelian[i].jumlah
             << ", Total Harga: " << user.pembelian[i].harga << "\n";
    }
}

void editPembelian(DataUser* user) {
    int index;
    cout << "\n--- Edit Pembelian ---\n";
    cout << "Masukkan nomor pembelian yang ingin diedit: ";
    cin >> index;
    if (index > 0 && index <= user->pembelian.size()) {
        cout << "Masukkan jumlah UC baru: ";
        cin >> (*user).pembelian[index - 1].jumlah;
        (*user).pembelian[index - 1].harga = (*user).pembelian[index - 1].jumlah * 3000;
        cout << "Data berhasil diperbarui!\n";
    } else {
        cout << "Nomor tidak valid!\n";
    }
}

void hapusPembelian(DataUser* user) {
    int index;
    cout << "\n--- Hapus Pembelian ---\n";
    cout << "Masukkan nomor pembelian yang ingin dihapus: ";
    cin >> index;
    if (index > 0 && index <= user->pembelian.size()) {
        user->pembelian.erase(user->pembelian.begin() + index - 1);
        cout << "Data berhasil dihapus!\n";
    } else {
        cout << "Nomor tidak valid!\n";
    }
}

// --- Sorting Methods ---
// 1. Selection Sort (Sorting Nama Ascending)
void selectionSortNama(vector<DataUser>& daftar) {
    for (int i = 0; i < daftar.size() - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < daftar.size(); j++) {
```

```
            if (daftar[j].user.nama < daftar[minIndex].user.nama) {
                minIndex = j;
            }
        }
        swap(daftar[i], daftar[minIndex]);
    }
}

// 2. Merge Sort (Sorting Jumlah UC Descending)
void merge(vector<PembelianUC>& arr, int left, int mid, int right) {
    vector<PembelianUC> leftArr(arr.begin() + left, arr.begin() + mid + 1);
    vector<PembelianUC> rightArr(arr.begin() + mid + 1, arr.begin() + right + 1);

    int i = 0, j = 0, k = left;
    while (i < leftArr.size() && j < rightArr.size()) {
        if (leftArr[i].jumlah > rightArr[j].jumlah) { // Descending
            arr[k++] = leftArr[i++];
        } else {
            arr[k++] = rightArr[j++];
        }
    }
    while (i < leftArr.size()) arr[k++] = leftArr[i++];
    while (j < rightArr.size()) arr[k++] = rightArr[j++];
}

void mergeSort(vector<PembelianUC>& arr, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

// 3. Quick Sort (Sorting Harga Ascending)
int partition(vector<PembelianUC>& arr, int low, int high) {
    double pivot = arr[high].harga;
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j].harga < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(vector<PembelianUC>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```
int main() {
    int pilihan;
    while (true) {
        cout << "\n--- Sistem Multiuser ---\n";
        cout << "1. Registrasi\n2. Login\n3. Keluar\nPilih: ";
        cin >> pilihan;

        if (pilihan == 1) {
            registrasiUser(&daftarUser);
        }
        else if (pilihan == 2) {
            int index = loginUser(&daftarUser);
            if (index == -1) {
                cout << "Anda telah gagal login 3 kali. Program berhenti.\n";
                break;
            }

            int menuPilihan;
            do {
                cout << "\n--- Menu ---\n";
                cout << "1. Tambah Pembelian UC\n2. Tampilkan Data\n3. Edit Pembelian\n4. Hapus
Pembelian\n";
                cout << "5. Sorting Nama (Asc)\n6. Sorting Jumlah UC (Desc)\n7. Sorting Harga
(Asc)\n8. Logout\nPilih: ";
                cin >> menuPilihan;

                switch (menuPilihan) {
                    case 1: tambahPembelian(&daftarUser[index]); break;
                    case 2: tampilkanData(daftarUser[index]); break;
                    case 3: editPembelian(&daftarUser[index]); break;
                    case 4: hapusPembelian(&daftarUser[index]); break;
                    case 5: selectionSortNama(daftarUser); cout << "Data user telah diurutkan berdasarkan
Nama (Ascending).\n"; break;
                    case 6: mergeSort(daftarUser[index].pembelian, 0, daftarUser[index].pembelian.size() -
1); cout << "Data pembelian diurutkan berdasarkan Jumlah UC (Descending).\n"; break;
                    case 7: quickSort(daftarUser[index].pembelian, 0, daftarUser[index].pembelian.size() -
1); cout << "Data pembelian diurutkan berdasarkan Harga (Ascending).\n"; break;
                    case 8: cout << "Logout...\n"; break;
                    default: cout << "Pilihan tidak valid!\n";
                }
            } while (menuPilihan != 8);
        }
        else if (pilihan == 3) {
            cout << "Keluar...\n";
            break;
        }
        else {
            cout << "Pilihan tidak valid!\n";
        }
    }

    return 0;
}
```

Penjelasan :

1. Penambahan Fitur Sorting
Program dasar sebelumnya hanya menangani fitur registrasi, login, pembelian, pengeditan, dan penghapusan data pembelian. Pada pengembangan ini, ditambahkan fitur **sorting data**, yang dijalankan melalui tiga metode berbeda:

- **Selection Sort** digunakan untuk **mengurutkan nama pengguna** dalam daftar user secara **ascending** (A–Z). Sorting ini dilakukan berdasarkan field nama dalam struct User.
- **Merge Sort** digunakan untuk **mengurutkan jumlah UC yang dibeli** secara **descending** (dari jumlah terbesar ke terkecil). Sorting ini diterapkan pada vektor pembelian di setiap DataUser.
- **Quick Sort** digunakan untuk **mengurutkan harga total pembelian**. Sorting ini dibebaskan, namun dalam implementasi dipilih sorting secara **ascending** (dari harga termurah ke termahal) pada field harga dalam struct PembelianUC.

2. Penambahan Fungsi Sorting
Untuk mengakomodasi ketiga metode sorting tersebut, dibuat tiga fungsi baru:

- void selectionSortNama(vector<DataUser>& daftar) untuk mengurutkan nama menggunakan algoritma Selection Sort.
- void mergeSortJumlah(vector<PembelianUC>& pembelian, int left, int right) untuk mengurutkan jumlah UC dengan algoritma Merge Sort.
- void quickSortHarga(vector<PembelianUC>& pembelian, int low, int high) untuk mengurutkan harga menggunakan Quick Sort.

Masing-masing fungsi memiliki logika algoritma yang khas sesuai metode sorting-nya, seperti pemilihan elemen minimum pada Selection Sort, pembagian dan penggabungan data pada Merge Sort, serta penggunaan metode partisi pada Quick Sort.

3. Modifikasi Menu Program
Pada menu utama setelah login, dilakukan modifikasi khusus pada pilihan **Tampilkan Data**. Saat pengguna memilih menu tersebut, program akan menampilkan submenu baru yang berisi pilihan:

1. Menampilkan data dengan sorting berdasarkan **Nama (Selection Sort - Ascending)**.
2. Menampilkan data dengan sorting berdasarkan **Jumlah UC (Merge Sort - Descending)**.
3. Menampilkan data dengan sorting berdasarkan **Harga Total (Quick Sort - Ascending)**.

Dengan submenu ini, pengguna dapat memilih metode pengurutan sesuai kebutuhan saat ingin melihat data pembeliannya.

4. Integrasi Dengan Sistem CRUD
Penambahan fitur sorting tidak mengganggu fitur CRUD (Create, Read, Update, Delete) yang sudah ada sebelumnya. Sorting hanya diterapkan pada saat menampilkan data, sehingga data asli tetap utuh kecuali terjadi perubahan atau penghapusan pembelian oleh pengguna.

5. Kesimpulan
Dengan adanya penambahan fitur sorting menggunakan tiga algoritma berbeda ini, program menjadi lebih kompleks dan memenuhi ketentuan tugas. Pengguna kini dapat mengakses data pembelian dengan tampilan yang lebih terstruktur berdasarkan nama, jumlah pembelian UC, maupun total harga, sesuai kebutuhan. Pengembangan ini meningkatkan fleksibilitas, keterbacaan data, serta memperkaya pemahaman terhadap berbagai metode sorting klasik dalam dunia pemrograman.

4. Uji Coba dan Hasil Output

4.1 Uji Coba

- Pengguna melakukan login dengan nama dan NIM yang benar.
- Pengguna memilih menu "Tambah Pembelian UC".
- Pengguna memasukkan jumlah UC yang ingin dibeli.
- Sistem menghitung total harga berdasarkan jumlah UC.
- Data pembelian berhasil disimpan.

4.2 Hasil Output

```
--- Sistem Multiuser ---
1. Registrasi
2. Login
3. Keluar
Pilih: 1

--- Registrasi User ---
Masukkan Nama: dimas
Masukkan NIM: 121
Registrasi berhasil! Silakan login.

--- Sistem Multiuser ---
1. Registrasi
2. Login
3. Keluar
Pilih: 2

--- Login ---
Masukkan Nama: dimas
Masukkan NIM: 121
Login berhasil!

--- Menu ---
1. Tambah Pembelian UC
2. Tampilkan Data
3. Edit Pembelian
4. Hapus Pembelian
5. Logout
Pilih: 1

Masukkan jumlah UC yang ingin dibeli: 80
Total harga: 240000

--- Menu ---
1. Tambah Pembelian UC
2. Tampilkan Data
3. Edit Pembelian
4. Hapus Pembelian
5. Logout
Pilih: 5
Logout...
```

5. Git

1. Konfigurasi User Git

```
ASUS@LAPTOP-7JK1V59G MINGW64 ~/Documents/praktikum-apl (master)
$ git config --global user.email "langgengdimas121@gmail.com"
```

- Perintah ini digunakan untuk mengatur alamat email yang akan digunakan dalam commit Git.
- Opsi --global berarti pengaturan ini berlaku untuk semua repository di komputer.

2. Inisialisasi Repository Git

```
ASUS@LAPTOP-7JK1V59G MINGW64 ~/Documents/praktikum-apl (master)
$ git init
Reinitialized existing Git repository in C:/Users/ASUS/Documents/praktikum-apl/.git/
```

- Perintah ini digunakan untuk menginisialisasi repository Git di dalam folder saat ini.
- Jika repository Git sudah ada, Git akan menampilkan pesan **"Reinitialized existing Git repository"**.

3. Menambahkan Remote Repository

```
ASUS@LAPTOP-7JK1V59G MINGW64 ~/Documents/praktikum-apl (master)
$ git remote add origin https://github.com/2409106121langgengdimassaputra/praktikum-apl.git
error: remote origin already exists.
```

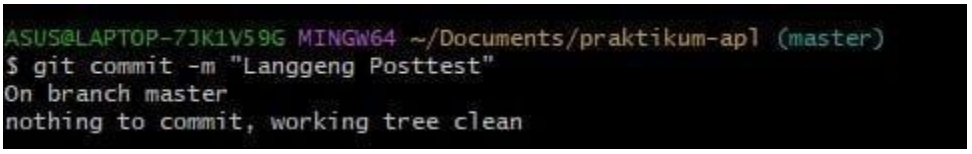
- Perintah ini menghubungkan repository lokal dengan repository online di GitHub.
- **origin** adalah nama default untuk remote repository.
- Jika sebelumnya sudah ada remote dengan nama **origin**, maka akan muncul pesan **"remote origin already exists"**.

4. Menambahkan File ke Staging Area

```
ASUS@LAPTOP-7JK1V59G MINGW64 ~/Documents/praktikum-apl (master)
$ git add .
```

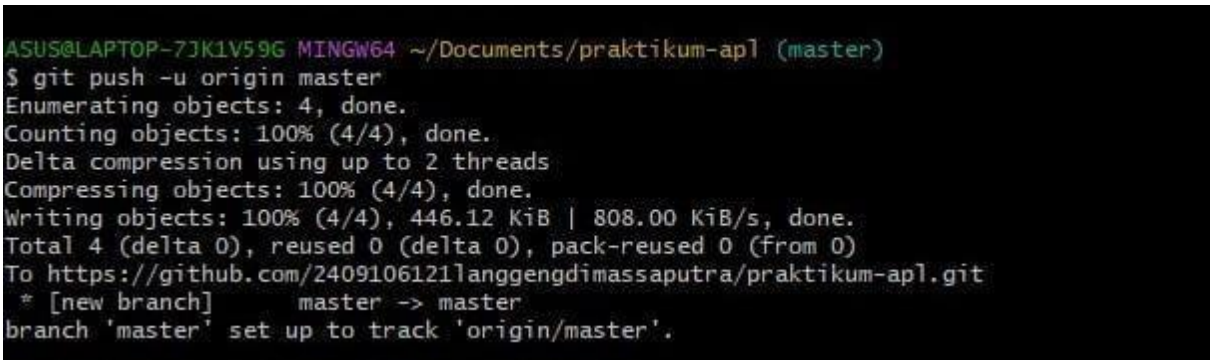
- Perintah ini digunakan untuk menambahkan semua file yang telah diubah atau ditambahkan ke dalam **staging area** sebelum dilakukan commit.

5. Membuat Commit



- Commit digunakan untuk menyimpan perubahan ke dalam repository lokal.
- Opsi -m "Langgeng Posttest" menambahkan pesan deskriptif tentang perubahan yang dilakukan.
- Jika tidak ada perubahan, Git akan menampilkan **"nothing to commit, working tree clean"**.

6. Mengunggah ke Repository GitHub (Push)



- Perintah ini mengunggah perubahan dari branch **master** di lokal ke branch **master** di remote **origin** (GitHub).
- Opsi -u (--set-upstream) menghubungkan branch lokal dengan branch di remote, sehingga ke depannya cukup menggunakan git push tanpa parameter tambahan.
- Pesan **"Please complete authentication in your browser..."** muncul karena GitHub membutuhkan autentikasi sebelum bisa mengunggah perubahan.
- Setelah autentikasi berhasil, proses *push* berjalan hingga selesai.
- Menunjukkan bahwa file berhasil dikompresi dan dikirim ke GitHub.