LAPORAN PRAKTIKUM

POSTTEST 3

ALGORITMA PEMROGRAMAN LANJUT

Disusun oleh:

Langgeng Dimas

Saputra

(2409106121)

Kelas (C2 ‘24)

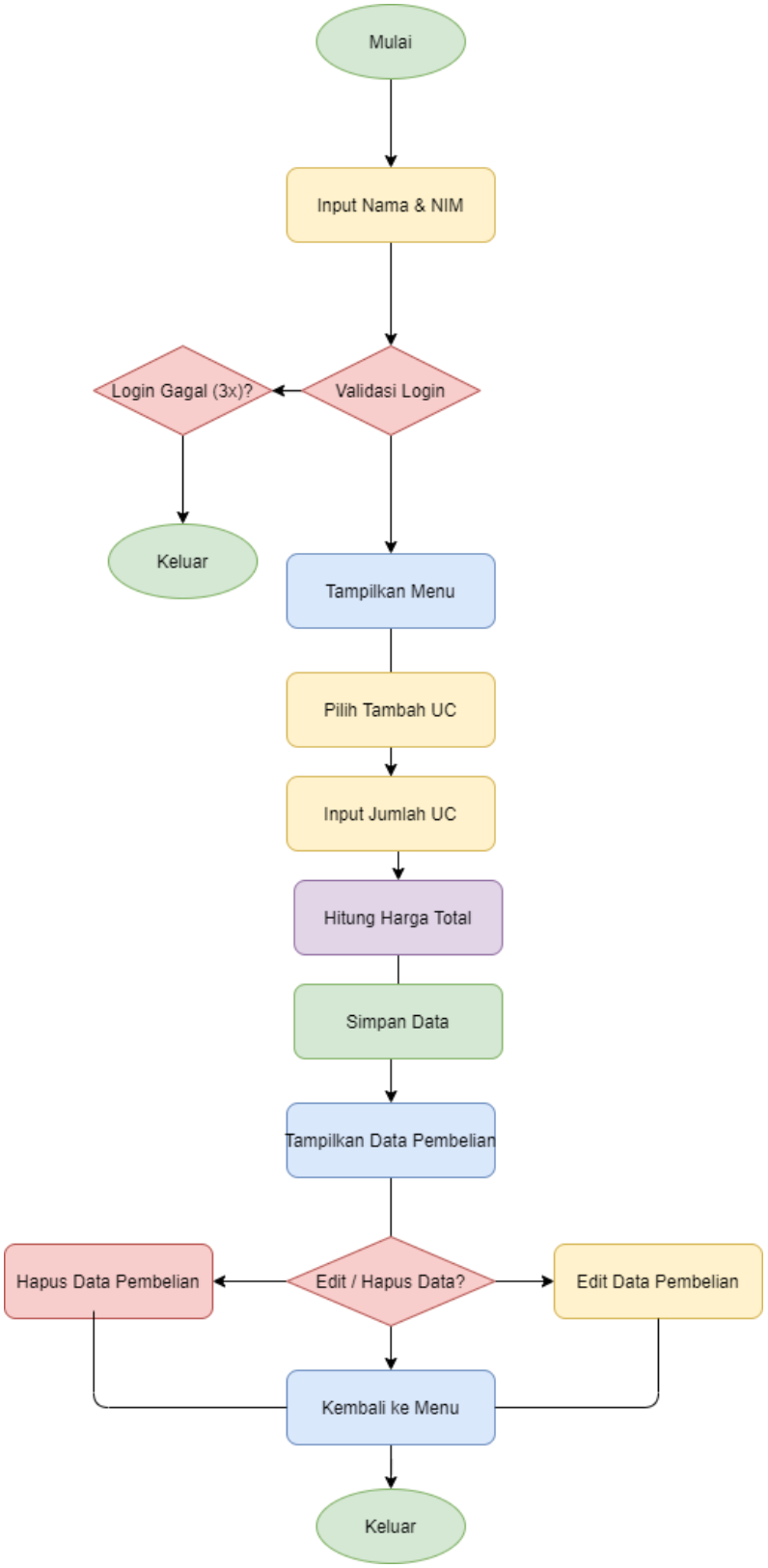
PROGRAM STUDI INFORMATIKA

UNIVERSITAS MULAWARMAN

SAMARINDA

2025

1. Flowchart


Mulai
Input Nama & NIM
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 Pembelian
Edit Data Pembelian
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 untuk menyimpan data user
struct User {
    string nama;
    string nim;
};

// Struct untuk data pembelian UC
struct PembelianUC {
    int jumlah;
    double harga;
};

// Struct utama yang menggabungkan semua data
struct DataUser {
    User user;
    vector<PembelianUC> pembelian;
};

vector<DataUser> daftarUser;

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

        if (pilihan == 1) {
            DataUser newUser;
            cout << "\n--- Registrasi User ---\n";
            cout << "Masukkan Nama: "; cin.ignore(); getline(cin,
newUser.user.nama);
            cout << "Masukkan NIM: "; cin >> newUser.user.nim;
            daftarUser.push_back(newUser);
            cout << "Registrasi berhasil! Silakan login.\n";
        }
        else if (pilihan == 2) {
            string nama, nim;
            int attempts = 3, userIndex = -1;
            while (attempts--) {
                cout << "\n--- Login ---\n";
```

```
                cout << "Masukkan Nama: "; cin.ignore(); getline(cin,
nama);
                cout << "Masukkan NIM: "; cin >> nim;

                for (int i = 0; i < daftarUser.size(); i++) {
                    if (daftarUser[i].user.nama == nama &&
daftarUser[i].user.nim == nim) {
                        userIndex = i;
                        cout << "Login berhasil!\n";
                        break;
                    }
                }
                if (userIndex != -1) break;
                cout << "Login gagal! Sisa percobaan: " << attempts <<
"\n";
            }
            if (userIndex == -1) {
                cout << "Anda telah gagal login 3 kali. Program
berhenti.\n";
                return 0;
            }

            int menuPilihan;
            do {
                cout << "\n--- Menu ---\n";
                cout << "1. Tambah Pembelian UC\n2. Tampilkan Data\n3.
Edit Pembelian\n4. Hapus Pembelian\n5. Logout\nPilih: ";
                cin >> menuPilihan;

                if (menuPilihan == 1) {
                    PembelianUC pembelian;
                    cout << "\nMasukkan jumlah UC yang ingin dibeli: ";
                    cin >> pembelian.jumlah;
                    pembelian.harga = pembelian.jumlah * 3000; // Harga
1 UC = 3000
                    cout << "Total harga: " << pembelian.harga << "\n";

daftarUser[userIndex].pembelian.push_back(pembelian);
                }
                else if (menuPilihan == 2) {
                    cout << "\n--- Data Pembelian ---\n";
                    for (int i = 0; i <
daftarUser[userIndex].pembelian.size(); i++) {
                        cout << i + 1 << ". Jumlah UC: " <<
daftarUser[userIndex].pembelian[i].jumlah
                             << ", Total Harga: " <<
daftarUser[userIndex].pembelian[i].harga << "\n";
                    }
                }
                else if (menuPilihan == 3) {
                    int index;
                    cout << "\n--- Edit Pembelian ---\n";
                    cout << "Masukkan nomor pembelian yang ingin diedit:
"; cin >> index;
                    if (index > 0 && index <=
```

```
daftarUser[userIndex].pembelian.size()) {
                        cout << "Masukkan jumlah UC baru: "; cin >>
daftarUser[userIndex].pembelian[index - 1].jumlah;
                        daftarUser[userIndex].pembelian[index - 1].harga
= daftarUser[userIndex].pembelian[index - 1].jumlah * 3000;
                        cout << "Data berhasil diperbarui!\n";
                    } else {
                        cout << "Nomor tidak valid!\n";
                    }
                }
                else if (menuPilihan == 4) {
                    int index;
                    cout << "\n--- Hapus Pembelian ---\n";
                    cout << "Masukkan nomor pembelian yang ingin
dihapus: "; cin >> index;
                    if (index > 0 && index <=
daftarUser[userIndex].pembelian.size()) {

daftarUser[userIndex].pembelian.erase(daftarUser[userIndex].pembelian.be
gin() + index - 1);
                        cout << "Data berhasil dihapus!\n";
                    } else {
                        cout << "Nomor tidak valid!\n";
                    }
                }
                else if (menuPilihan == 5) {
                    cout << "Logout...\n";
                    break;
                }
                else {
                    cout << "Pilihan tidak valid!\n";
                }
            } while (true);
        }
        else if (pilihan == 3) {
            cout << "Keluar...\n";
            return 0;
        }
        else {
            cout << "Pilihan tidak valid!\n";
        }
    }
}
```

Penjelasan :

Program ini adalah sistem pembelian UC (Unknown Cash) yang menggunakan fitur CRUD (Create, Read, Update, Delete). Pengguna dapat menambahkan, melihat, mengedit, dan menghapus data pembelian UC. Program akan berjalan terus sampai pengguna memilih keluar. Data disimpan dalam array of struct, dan harga dihitung otomatis dengan 1 UC = Rp3000. Pada awal program, pengguna diminta memasukkan nama pengguna sebagai identitas pembeli. Setelah itu, program akan menampilkan menu utama yang terdiri dari pilihan Tambah Pembelian

UC, Tampilkan Data Pembelian, Edit Pembelian, Hapus Pembelian, dan Keluar. Program menggunakan do-while loop agar tetap berjalan sampai pengguna memilih keluar.
Saat pengguna memilih menu Tambah Pembelian UC, mereka akan diminta memasukkan jumlah UC yang ingin dibeli. Harga dihitung otomatis berdasarkan jumlah UC yang dimasukkan. Setelah itu, data pembelian akan disimpan ke dalam array dan jumlah data bertambah.
Untuk menampilkan semua pembelian yang sudah dilakukan, program akan mencetak informasi setiap pembelian dalam format yang rapi. Jika pengguna memilih menu Edit Pembelian, mereka dapat memasukkan nomor data yang ingin diubah. Jika nomor data valid, pengguna bisa memasukkan jumlah UC baru, dan harga akan diperbarui berdasarkan jumlah UC terbaru.
Jika pengguna ingin menghapus data, mereka harus memasukkan nomor data yang ingin dihapus. Program akan menggeser elemen array ke kiri untuk menghapus data yang dipilih dan mengurangi jumlah data yang tersimpan.
Program juga dilengkapi dengan validasi input untuk memastikan bahwa pengguna tidak bisa mengedit atau menghapus data yang tidak ada dalam daftar. Program akan terus berjalan dan menampilkan menu utama hingga pengguna memilih opsi keluar.

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.
```

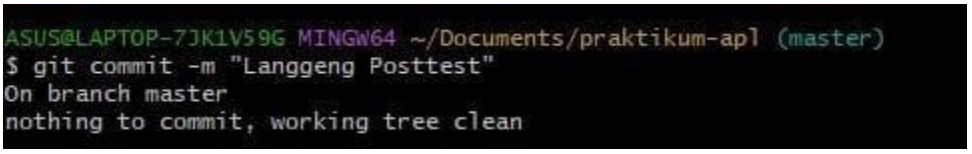
- Perintah ini menghubungkan repository lokal dengan repository online di GitHub.
- **origin** adalah nama default untuk remote repository.
- Jika sebelumnya sudah ada remote dengan nama **origin**, maka akan muncul pesan **"remote origin already exists"**.

4. Menambahkan File ke Staging Area

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

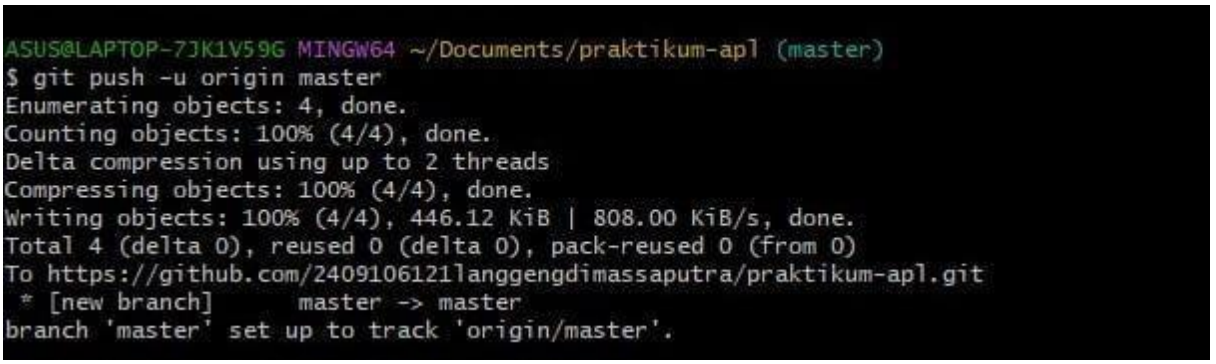
- Perintah ini digunakan untuk menambahkan semua file yang telah diubah atau ditambahkan ke dalam **staging area** sebelum dilakukan commit.

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.