LAPORAN PRAKTIKUM

PEMROGRAMAN DASAR

Vector

NAMA: Farid Gantari Siwanda

NIM: 25104410016

PERIODE: SEMESTER GANJIL 2024/2025

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ISLAM BALITAR

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam pengembangan perangkat lunak menggunakan bahasa C++, efisiensi dalam pengelolaan data dinamis merupakan aspek yang sangat krusial. Seringkali, jumlah data yang perlu diproses tidak diketahui secara pasti saat program dikompilasi, sehingga penggunaan *array* statis menjadi tidak fleksibel. Untuk mengatasi hal ini, C++ menyediakan pustaka Standard Template Library (STL) berupa **std::vector**, yang memungkinkan penyimpanan data secara dinamis dengan kemampuan alokasi memori otomatis sesuai kebutuhan pengguna.

Selain manajemen memori, integrasi antara program dengan sumber data eksternal juga menjadi kebutuhan dasar dalam pemrograman. Data sering kali disimpan dalam format teks (seperti file **.txt**) dengan struktur tertentu, misalnya menggunakan pemisah titik koma (**;**). Proses membaca data dari file, melakukan parsing atau pemecahan string menggunakan **stringstream**, dan menyimpannya ke dalam struktur data yang tepat adalah teknik mendasar yang harus dikuasai untuk membangun aplikasi yang mampu berinteraksi dengan basis data sederhana atau file konfigurasi.

1.2 Tujuan Praktikum

Adapun tujuan dari praktikum ini adalah:

1. Memahami implementasi **std::vector** untuk menyimpan objek dari sebuah **struct**.
2. Mempelajari mekanisme pembacaan data dari file teks menggunakan **ifstream**.
3. Menguasai teknik manipulasi string dan parsing data menggunakan **stringstream** serta fungsi **getline**.
4. Mampu menampilkan data yang telah diolah dalam format tabel yang rapi menggunakan **iomanip**.

BAB II

TEORI DASAR

Pemrograman C++ mendukung pengelompokan variabel dengan tipe data berbeda ke dalam satu kesatuan melalui fitur **struct**. Dalam program ini, **struct siswa** digunakan sebagai cetak biru untuk menyimpan informasi nomor, nama, dan nilai. Untuk menampung banyak data siswa secara dinamis, digunakan **std::vector**. Berbeda dengan *array* konvensional, **vector** dapat bertambah ukurannya secara otomatis saat fungsi **push_back()** dipanggil, sehingga pengembang tidak perlu khawatir akan terjadinya *buffer overflow* atau pemborosan memori yang signifikan pada data yang bersifat fluktuatif.

Proses input/output file dalam C++ ditangani oleh pustaka **fstream**. Objek **ifstream** digunakan secara khusus untuk membuka dan membaca konten dari file eksternal. Dalam konteks pemrosesan data teks, teknik *parsing* sangat penting untuk memisahkan informasi yang digabung dalam satu baris. Penggunaan **stringstream** memungkinkan sebuah string diperlakukan seperti aliran data (*stream*), sehingga fungsi **getline** dapat mengekstrak bagian-bagian teks (token) berdasarkan karakter pembatas tertentu, seperti tanda titik koma (**;**) atau titik dua (**:**).

Selain itu, konversi tipe data dari string ke numerik sering kali diperlukan saat membaca data dari file teks, yang dalam C++ dapat dilakukan menggunakan fungsi **stoi** (string to integer). Untuk menghasilkan luaran yang mudah dibaca oleh pengguna, pengaturan format pada konsol dilakukan dengan bantuan pustaka **iomanip**. Fungsi seperti **setw()** dan **left** sangat berguna untuk mengatur lebar kolom dan perataan teks, sehingga data yang ditampilkan di layar tersusun dalam bentuk tabel yang konsisten dan profesional.

BAB III

PENJELASAN ISI PROGRAM

3.1 Penjelasan Baris 4 Sampai 6

Baris 4 sampai 6 memasukkan pustaka standar yang dipakai program: **#include <sstream>** menyediakan kelas **std::stringstream** yang memungkinkan program memperlakukan satu baris teks sebagai aliran sehingga bisa dipecah menjadi potongan-potongan dengan **getline(ss, token, ';')** untuk parsing data (nomor, nama, nilai); **#include <vector>** menyediakan kontainer dinamis **std::vector** yang dipakai untuk menyimpan kumpulan struktur **siswa** (**vector<siswa> daftarSiswa;** dan **daftarSiswa.push_back(s)**), sehingga jumlah siswa bisa bertambah sesuai isi file; sedangkan **#include <iomanip>** menyediakan manipulator format I/O seperti **std::setw** dan **std::left** yang dipakai saat menampilkan tabel hasil (mis. **cout << left << setw(6) << student.nomor << setw(15) << student.nama << setw(8) << student.nilai**) untuk merapikan lebar kolom dan perataan output.

3.2 Penjelasan Baris 10 Sampai 14

Baris 10–14 mendefinisikan sebuah tipe data terstruktur bernama **siswa** yang mengelompokkan tiga field terkait: **int nomor;** untuk menyimpan nomor urut/ID siswa, **string nama;** untuk menyimpan nama siswa, dan **int nilai;** untuk menyimpan nilai/score siswa; setelah **};** definisi **struct** selesai sehingga Anda bisa membuat variabel **siswa s;** atau kontainer seperti **vector<siswa> daftarSiswa;.** Perlu dicatat bahwa member **struct** bersifat public secara default dan karena tidak ada inisialisasi awal pada **int**, nilai numerik bisa tidak terinisialisasi jika tidak diisi—namun pada kode Anda setiap field diisi dari hasil parsing sebelum objek **siswa** dimasukkan ke vektor sehingga aman.

3.3 Penjelasan Isi Program int main() { (baris 16 sampai 73)

Program ini membuka file teks "bacaAku.txt", memeriksa apakah file berhasil dibuka, lalu membaca file baris per baris. Setiap baris di-split menggunakan **stringstream** dengan pemisah **;** untuk mengambil **nomor, nama, dan nilai;** ada penanganan khusus jika **nama** mengandung **:** diikuti nilai (maka nilai dipisahkan dari nama). Data setiap siswa dimasukkan ke **vector<siswa>** menggunakan **push_back**. Setelah selesai membaca, file ditutup dan seluruh daftar siswa dicetak rapi ke layar menggunakan **setw** dan perataan (**left**) untuk membentuk tampilan kolom. Jika file tidak dapat dibuka, program menampilkan pesan error dan keluar. Untuk penjelasan lengkapnya akan di jelaskan sebagai berikut:

a. Baris 17

```
vector<siswa> daftarSiswa;
```

vector<siswa> daftarSiswa; mendeklarasikan sebuah variabel bernama **daftarSiswa** bertipe **std::vector** yang menyimpan elemen-elemen bertipe **siswa**; artinya ini adalah kontainer dinamis (array yang ukurannya bisa berubah) yang awalnya kosong dan akan diisi dengan objek **siswa** hasil parsing menggunakan **daftarSiswa.push_back(s);**. Karena menggunakan **using namespace std;**, penulisan **vector** singkat tanpa awalan **std::** valid. Keuntungan vector di sini: menyimpan elemen secara berurutan, memungkinkan iterasi mudah (mis. **for (const auto& student : daftarSiswa)**), dan otomatis mengatur alokasi memori saat elemen baru ditambahkan

b. Baris 20

```
ifstream fileku("bacaAku.txt");
```

ifstream fileku("bacaAku.txt");: membuat objek input file stream bernama **fileku** dan langsung mencoba membuka file **"bacaAku.txt"** untuk dibaca; keberhasilan pembukaan akan diperiksa pada baris berikutnya dengan **fileku.is_open()**.

c. Baris 23 sampai 26

```
if (!fileku.is_open()) {
    cout << "file gagal dibuka!" << endl;
    return 1;
}
```

memeriksa apakah file berhasil dibuka: jika **fileku.is_open()** bernilai false (gagal buka), program menampilkan pesan **"file gagal dibuka!"** ke layar lalu mengakhiri eksekusi dengan **return 1;** — artinya program keluar lebih awal dan memberi kode keluar non-nol untuk menandakan terjadi kesalahan.

d. Baris 29 sampai 42

```
string baris;
while (getline(fileku, baris)) {
    //melewati baris kosong
    if(baris.empty()) continue;

    siswa s;
    stringstream ss(baris);

    string token;
    getline(ss, token, ';');
    s.nomor = stoi(token);

    getline(ss, token, ';');
    s.nama = token;
```

melakukan pembacaan tiap baris dari file dan mem-parse isi baris itu menjadi objek **siswa**: pertama dibuat variabel **string baris;** lalu **while (getline(fileku, baris))** membaca satu baris per iterasi. Jika baris kosong dilewati **(if(baris.empty()) continue;**). Kemudian dibuat **siswa s;** dan **stringstream ss(baris);** untuk memproses teks baris sebagai aliran. Dengan **getline(ss, token, ';')** diambil token pertama (sampai **;**) lalu **s.nomor = stoi(token);** mengubahnya jadi integer untuk nomor; pemanggilan **getline(ss, token, ';')** berikutnya mengambil token kedua dan disimpan ke **s.nama**.

e. Baris 45 sampai 57

```
size_t colon_pos = s.nama.find(':');
if (colon_pos != string::npos) {
    string nilai_str = s.nama.substr(colon_pos + 1);
    s.nama = s.nama.substr(0, colon_pos);
    s.nilai = stoi(nilai_str);
} else {
    getline(ss, token, ';');
    if (!token.empty()) {
        s.nilai = stoi(token);
    } else {
        s.nilai = 0;
    }
}
```

mengecek dan mengambil nilai siswa setelah nama: kode pertama mencari apakah **s.nama** mengandung karakter **':'** (format gabungan seperti **Nama:Nilai**); jika ditemukan, bagian setelah **':'** diambil sebagai string nilai (**nilai_str**), bagian sebelum **':'** disimpan kembali sebagai **s.nama**, lalu **s.nilai** diisi dengan konversi **stoi(nilai_str)**. Jika tidak ada **':'**, program membaca token berikutnya dari **stringstream** (mengharapkan nilai setelah pemisah **;**); jika token itu tidak kosong maka dikonversi jadi integer untuk **s.nilai**, kalau kosong maka **s.nilai** di-set ke 0 — sehingga kedua format input (nilai terpisah atau nilai menempel pada nama) bisa ditangani dengan aman.

f. Baris 60 dan 63

```
    //5. simpan data ke vector
    daftarSiswa.push_back(s);
}

fileku.close();
```

Baris 60, **daftarSiswa.push_back(s);**, menambahkan objek **s** (hasil parsing satu baris file) ke akhir vektor **daftarSiswa** sehingga semua data siswa tersimpan berurutan untuk diproses atau ditampilkan nanti. Baris 63, **fileku.close();**, menutup file yang sebelumnya dibuka — ini melepaskan sumber daya (file handle) dan menandakan bahwa program telah selesai membaca file; meskipun destructor **ifstream** akan menutup file saat keluar dari scope, pemanggilan **close()** di sini adalah praktek baik untuk segera membebaskan resource.

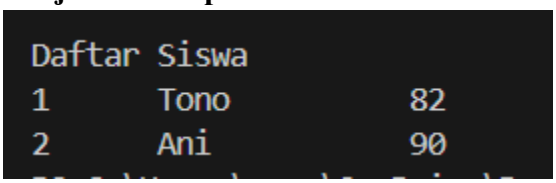
g. Baris 66 sampai 72

```
//6. tampilkan data
cout << "\nDaftar Siswa\n";
for (const auto& student : daftarSiswa) {
    cout << left
         << setw(6) << student.nomor
         << setw(15) << student.nama
         << setw(8) << student.nilai
         << endl;
}
return 0;
```

mencetak judul lalu menampilkan tiap data siswa dari vektor secara rapi: **cout << "\nDaftar Siswa\n";** menulis judul dengan baris kosong di depan, lalu **for (const auto& student : daftarSiswa)** adalah loop range-based yang mengambil tiap elemen **student** dari vektor tanpa menyalinnya (menggunakan **const** dan referensi). Di dalam

loop **cout << left << setw(6) << student.nomor << setw(15) << student.nama << setw(8) << student.nilai << endl;** mencetak nomor, nama, dan nilai dalam kolom berlebar tetap (6, 15, dan 8 karakter) dengan perataan kiri (**left**) agar tampilannya rapi seperti tabel, dan **endl** memindahkan kursor ke baris berikutnya. Baris terakhir **return 0;** mengakhiri program dan memberi kode keluaran 0 yang menandakan eksekusi berhasil.

3.4 Penjelasan output



Output tersebut menampilkan tabel sederhana berjudul "Daftar Siswa" (dengan baris kosong di atas judul), di mana setiap baris mewakili satu siswa: kolom pertama adalah nomor urut (1, 2), kolom kedua adalah nama ("Tono", "Ani"), dan kolom ketiga adalah nilai/skor masing-masing (82, 90). Jarak antar kolom berasal dari manipulasi format I/O (**left** dan **setw**) sehingga setiap field dicetak dalam kolom berlebar tetap sehingga tampak rapi seperti tabel; nilai diambil dari file input yang diparsing sebelumnya (mis. baris seperti **1;Tono;82 dan 2;Ani;90**).

BAB IV

KESIMPULAN

Berdasarkan praktikum yang telah dilakukan, dapat disimpulkan bahwa kombinasi antara **struct** dan **std::vector** merupakan cara yang sangat efektif untuk mengelola data record secara dinamis dalam C++. Program berhasil melakukan tugas integrasi data dengan membaca file teks eksternal, melakukan validasi pembukaan file, serta melakukan parsing string yang kompleks (menangani variasi format pemisah **;** dan **:).** Penggunaan **stringstream** terbukti mempermudah ekstraksi data dari format baris tunggal menjadi atribut objek yang terpisah. Secara keseluruhan, teknik ini memungkinkan pembuatan program yang fleksibel dalam menangani masukan data dari luar sistem dengan hasil keluaran yang terorganisir.