LAPORAN PRAKTIKUM

Source Code : https://github.com/shuretokki/steam-cli-lib-fetcher

---

Struktur Kode

Kode ini terdiri dari beberapa file utama:

- `slf.hpp`: File header. Di sini aku deklarasiin struktur data (seperti `Game` dan `User`), variabel global, sama prototype fungsi-fungsi yang akan dipakai.
- `slf.cpp`: File implementasi, ttempat aku tulis logika lengkap dari fungsi-fungsi yang udah dideklarasiin di slf.hpp. Jadi, semua logic aplikasi ada di sini.
- `main.cpp`: File utama yang jadi entry point aplikasi. Di sini kita mulai aplikasi, terima input user, dan jalankan perintah.
- `vcpkg.json`: File konfigurasi untuk dependensi library pakai vcpkg.
- `CMakeLists.txt`: File untuk ngatur proses build pakai CMake.

Kode ini pakai library **nlohmann::json** untuk handle JSON, **httplib** untuk akses API Steam, sama **fmt** untuk bikin output colored text.

Isi Kode dan Komponen Utama

1. Struktur Data Utama

- **Struct** `Game`
  Ini untuk simpan info satu game:
  - `std::string name`: Nama game (contoh: "Portal 2").
  - `int appid`: APPID game di Steam (contoh: 620).
  - `int playtime`: Waktu main dalam menit (contoh: 1200).
  - Contoh: `{name: "Portal 2", appid: 620, playtime: 1200}`.
- **Struct** `User`
  Ini untuk simpan info pengguna:
  - `std::string username`: Nama pengguna Steam.
  - `std::string location`: Lokasi pengguna (contoh: "ID").
  - `std::string steamid`: ID Steam 17 digit (contoh: "76561197960287930").
  - Contoh: `{username: "gooner", location: "ID", steamid: "76561197960287930"}`.
- **Global Variable**
  - `std::string data_filename = "data/games.json"`: File tempat simpan data game.
  - `bool has_fetched = false`: Cek apakah data udah diambil dari Steam.
  - `std::string api_key`: API Key Steam untuk akses data.
  - `std::vector<Game> games`: Daftar semua game yang di-fetch.
  - `User user`: Info pengguna yang lagi aktif.
  - `Prefix games_prefix`: Struktur trie untuk nyari game berdasarkan prefix.
  - `std::unordered_map<std::string, size_t> games_map`: Hash map untuk nyari game berdasarkan nama lengkap.

---

2. Fungsi-Fungsi Utama (dari slf.cpp)

Fungsi-fungsi disini terdiri dari:

- **`to_lower(const std::string& s)`**
  Fungsi ini ubah string jadi huruf kecil biar case-insensitive. Contoh: `to_lower("Portal")` jadi `"portal"`. Dipakai untuk nyari game tanpa peduli huruf besar-kecil.
- **`load_api_key()`**
  Fungsi ini ambil API Key dari file `data/config.json`. Kalau file-nya gak ada, minta input dari user, trus disimpen. Hasilnya disimpan di global variable `api_key`. Kalau gagal (misalnya kunci kosong), balik `false`.
- **`save_games()`**
  Simpan data game dan user ke file `data/games.json` dalam format JSON. Contohnya:

```
{
    "user": {
        "username": "Gooner",
        "location": "ID",
        "steamid": "76561197960287930"
    },
    "games": [
        {"name": "Portal 2", "appid": 620, "playtime": 1200}
    ]
}
```

- **`load_games()`**
  Baca data dari `data/games.json`, isi `games`, `user`, `games_prefix`, dan `games_map`. Kalau filenya gak ada, akan di skip. Ini fungsi pertama yang jalan kalau aplikasi dibuka.
- **`fetch_games(const std::string& id)`**
  Ambil data game dari Steam API berdasarkan `id` (bisa SteamID64 atau custom ID). Langkahnya:
  1. Kalau `id` bukan 17 digit (input bisa custom ID), akan dikonversi pakai endpoint `ResolveVanityURL`.
  2. Ambil info user (nama, lokasi) pakai `GetPlayerSummaries`.
  3. Ambil daftar game pakai `GetOwnedGames`.
  4. Isi `games`, `games_prefix`, dan `games_map`, trus simpan ke JSON.
  - Output: `Fetched <total> games.` + link profil Steam.
- **`search_prefix(const std::string& prefix)`**
  Mencari game yang namanya mulai dengan `prefix` pakai trie. Hasilnya ditampikan dalam tabel:

```
Matching games:
| AppID | Name     | Playtime   |
| ----- | -------- | ---------- |
| 620   | Portal 2 | 1200 min   |
| 70    | Portal   | Not Played |
```

- **`count_played()`**
  Hitung game yang pernah dimainkan (playtime > 0). Contoh output: `Number of games played: <total>`.
- **`export_csv(const std::string& filename)`**
  Export daftar game ke file CSV di `data/exported/`. Contoh output:

```
Exported <totalgame> games to data/exported/mygames.csv.
```

  Isinya:

```
AppID,Name,Playtime
620,Portal 2,1200 min
70,Half-Life,Not played
```

- **Fungsi List (list_games, list_table, list_by_letter, list_by_playtime)**
  Nampilin game dengan cara berbeda:
- `list_games`: Daftar nama game urut dari awalan nama (0-9, aA-zZ).
- `list_table`: Tabel dengan AppID, nama, playtime.
- `list_by_letter`: Dikelompokkan berdasarkan awalan nama.
- `list_by_playtime`: Diurut berdasarkan playtime (terbanyak paling atas).
- Contoh (list_table):

```
| AppID | Name     | Playtime   |
| ----- | -------- | ---------- |
| 620   | Portal 2 | 1200 min   |
| 70    | Portal   | Not Played |
Total games: 2
```

- **`show_help()`**
  Menampilkan info user (kalau sudah di-fetch) dengan daftar command:

```
Current Account:
Username: gooner
Location: ID
SteamID: 76561197960287930
Link to Profile: https://steamcommunity.com/profiles/76561197960287930/

Commands:
fetch <steamid>           - Fetch games by SteamID64 or Unique ID
search "<name>"           - Search for games starting with name
count_played              - Count games with playtime
list                      - Show names in alphabetical order
list -l                   - Show AppID, name, playtime
list -n                   - Show name and AppID by letter
list -p                   - Show sorted by playtime
export <filename>         - Export games to CSV
help                      - Show this help
exit                      - Exit the program</filename></name></steamid>
```

- **`parse_command(const std::string& line)`**
  Parse/pecah input user jadi array argumen, handle quote agar bisa mencari multi-word. Contoh: `search "Portal"` jadi `["search", "Portal"]`.
- **`process_command(const std::vector<std::string>& args)`**
  Run command berdasarkan argumen. Kalau tdk valid, akan error.

---

3. Fungsi Utama di `main.cpp`

- **Remove File Data (Opsional)**
  Kalau file `data/games.json` sudah ada, akan dihapus dulu agar mulai dari nol. Ini agar tidak ada data lama yang mengganggu.
- **Call `load_games()`**
  Langsung call fungsi `load_games()` untuk coba baca data dari `data/games.json`. Kalau ada data sebelumnya, akan di-load ke `games` dan `user`.
- **Print Header Aplikasi**

```
Steam Game Fetcher v1.0 - Type 'help' for commands
------
```

- **Loop Utama**

  Core dari aplikasi: loop yang akan terus jalan sampai user input `exit`. Langkahnya:
  1. Print prompt: `>` .
  2. Baca input user pakai `std::getline(std::cin, line)`.
  3. Kalau input kosong, print error: `Error: Empty input. Type 'help' for commands.`.
  4. Kalau ada input, parse dgn `parse_command(line)`.
  5. Run command dgn `process_command(args)`.
  6. Kalau ada error (misalnya user ketik `exit`), catch dgn `try-catch`. Kalau error-nya "exit", hapus file `data/games.json`, print pesan `Goodbye!`, trus keluar.

Alur:

```
[Start]
  |
[Remove data/games.json]
  |
[Panggil load_games()]
  |
[Tampilin header aplikasi]
  |
[Loop: Output "> "]
  |
[Ambil input user]
  | Kosong?         | Tidak
  v                 v
[Error]          [Parse input]
  |
[Running command]
  |
[Exit?] ----->  [Reset file, Goodbye!]
  |
[Kembali ke loop]
```

---

4. Konsep Tree

Struktur `Prefix`:

- `Node`: Punya `children` (map char ke node lain) dan `indices(ix, di dalam code)` (daftar indeks game).
- `root`: Node awal dari tree.
- Fungsi:
  - `insert`: Input nama game ke tree dengan indeksnya.
  - `search`: Search game berdasarkan prefix.
  - `clear`: Remove node untuk clear memory.

Visualisasi Trie: Kalau ada game "Portal" (ix 0) dan "Pong" (ix 1):

```
       [root]
          |
          P
         / \
        o   o
       /     \
      r       n
     /         \
    t           g
   /             \
  a               [1]
 /
 l
 |
[0]
```

Search "Po" akan kasih [0, 1] (Portal, Pong).

---

5. Konsep Search

Pencarian di kode ini ada dua method:

- **Tree (search_prefix)**: Search berdasarkan awalan nama. Contoh: `search wu`, bisa output game dengan nama "Wuthering Waves".
- **Hash Map (games_map)**: Search nama game pakai `unordered_map`. Contoh: kalo nyari "Portal 2" langsung ke indeksnya. Ini O(1) rata-rata, tapi kurang terpakai karena harus input nama game dgn full/lengkap. Contoh: `search "Wuthering Waves"`, akan output game dengan nama tsb.

---

6. Konsep Hash Map

`std::unordered_map<std::string, size_t> games_map` dipakai untuk nyimpan pair nama game (dalam huruf kecil) sama indeksnya di `games`. Contoh:

- Key: "portal 2" → Value: 0
- Key: "half-life" → Value: 1

Contoh di Kode: Di `fetch_games`, tiap game akan dimasukan ke `games_map` agar mudah dicek nantinya.

Cara Pakai Aplikasi

1. **Setup**:
   - Ambil API key dari Steam (https://steamcommunity.com/dev/apikey).
   - Run program, kalo tdk ada API_KEY di `data/config.json`, akan diminta input.
2. **Command**:
   - `fetch <steamid>`: Ambil game (contoh: `fetch 76561197960287930`).
   - `search "<prefix>"`: Cari game (contoh: `search "Port"`).
   - `count_played`: Hitung game yang dimainkan.
   - `list`: Daftar nama game.
   - `list -l`: Tabel lengkap.
   - `list -n`: Kelompokkan berdasarkan huruf.
   - `list -p`: Urut berdasarkan playtime.
   - `export <filename>`: Export ke CSV (contoh: `export gooner`).
   - `help`: Lihat helper.
   - `exit`: Keluar.

Langkah-Langkah Build

1. **Install vcpkg**
   - Clone vcpkg `git clone <https://github.com/microsoft/vcpkg.git>`
   - Masuk ke folder vcpkg, lalu bootstrap: `.\bootstrap-vcpkg.bat` atau `./bootstrap-vcpkg.sh`
2. **Set Environment Variable**
   - Tambah directory vcpkg ke varible 'VCPKG_ROOT'
3. **Clone Repository ini**
   - `git clone <https://github.com/microsoft/vcpkg.git>`
4. **Buat Folder Build**
   - Buat directory baru di root folder setelah di clone
5. **Run CMake**
   - `cmake -B build -S . -DCMAKE_TOOLCHAIN_FILE=$VCPKG_ROOT/scripts/buildsystems/vcpkg.cmake -DVCPKG_TARGET_TRIPLET=x64-mingw-static`
6. **Build Aplikasi**
   - `cmake --build build --config Release`
7. **Jalankan Aplikasi**
   - `cd steamlibfetcher`
   - `steamlibfetcher.exe` atau
   - `./steamlibfetcher`