Робота з файлами 2

Ліна Явдошак

2 грудня 2024 р.

Зміст

1 Тема реферату

Тема: Використання бібліотеки `<filesystem>` для обробки файлів та директорій у мові програмування C++.

У цьому рефераті розглянуто практичні аспекти використання `<filesystem>` для вирішення таких задач, як зміна розширень файлів, редагування коментарів, видалення та перенесення файлів за критеріями, а також аналіз середнього розміру текстових файлів.

2 Вступ

Бібліотека `<filesystem>`, починаючи з C++17, пропонує стандартизовані засоби для роботи з файлами та директоріями. Вона значно спрощує вирішення завдань, пов'язаних з файловою системою, роблячи код зрозумілим та надійним.

3 Детальний опис програми

3.1 Завдання 1: Заміна розширень файлів

```
static void replace_extension(const fs::path& directory) {
    for (const auto& entry : fs::recursive_directory_iterator(directory)) {
        if (entry.is_regular_file() && entry.path().extension() == ".c") {
            fs::path new_path = entry.path();
            new_path.replace_extension(".cpp");
            fs::rename(entry.path(), new_path);
            std::cout << "Renamed: " << entry.path() << " -> " << new_path
                ↪ << "\n";
        }
    }
}
```

Аналіз рядків: - `fs::recursive_directory_iterator` — ітератор, який дозволяє рекурсивно обходити всі файли в директорії. - `entry.is_regular_file()` — перевіряє, чи є об'єкт файлом. - `entry.path().extension() == ".c"` — порівнює розширення файлу. - `new_path.replace_extension(".cpp")` — змінює розширення. - `fs::rename()` — перейменовує файл.

—

3.2 Завдання 2: Заміна коментарів у коді

```
static void replace_comments(const fs::path& directory) {
    for (const auto& entry : fs::recursive_directory_iterator(directory)) {
        if (entry.is_regular_file() && entry.path().extension() == ".cpp") {
            std::ifstream file(entry.path());
            if (!file) {
                std::cerr << "Could not open file: " << entry.path() << "\n"
                    ↪ ;
                continue;
            }
            std::string content((std::istreambuf_iterator<char>(file)), std
                ↪ ::istreambuf_iterator<char>());
            file.close();

            std::regex comment_regex(R"(//(.*))");
            std::string new_content = std::regex_replace(content,
                ↪ comment_regex, "/*$1 */");

            std::ofstream out_file(entry.path());
            if (!out_file) {
                std::cerr << "Could not write to file: " << entry.path() <<
                    ↪ "\n";
                continue;
            }
            out_file << new_content;
            out_file.close();

            std::cout << "Processed: " << entry.path() << "\n";
        }
    }
}
```

Аналіз рядків: - `std::ifstream file(entry.path())` — відкриває файл для читання. - `std::regex comment_regex(R"(//(.*))")` — створює регулярний вираз для по-

шуку однорядкових коментарів. - `std::regex_replace()` — виконує заміну коментарів. - `std::ofstream out_file(entry.path())` — відкриває файл для запису.

—

3.3 Завдання 3: Видалення файлів `.txt` за датою

```
std::chrono::system_clock::time_point parse_date(const std::string& date_str
    ↪ ) {
    std::tm tm = {};
    std::istringstream ss(date_str);
    ss >> std::get_time(&tm, "%Y-%m-%d");
    if (ss.fail()) {
        throw std::runtime_error("Invalid date format. Use YYYY-MM-DD.");
    }
    return std::chrono::system_clock::from_time_t(std::mktime(&tm));
}
```

Аналіз ключових операцій: - `std::istringstream ss(date_str)` — створює потік для аналізу рядка дати. - `std::mktime(&tm)` — перетворює дату в формат `time_t`.

—

3.4 Завдання 4: Переміщення старих файлів

```
static void move_old_txt_files(const fs::path& source_dir, const fs::path&
    ↪ dest_dir) {
    fs::create_directories(dest_dir);
    auto now = std::chrono::system_clock::now();
    auto one_year_ago = now - std::chrono::hours(24 * 365);

    for (const auto& entry : fs::recursive_directory_iterator(source_dir)) {
        if (entry.is_regular_file() && entry.path().extension() == ".txt") {
            auto creation_time = fs::last_write_time(entry);
            if (to_time_t(creation_time) > std::chrono::system_clock::
                ↪ to_time_t(one_year_ago)) {
                fs::path new_path = dest_dir / entry.path().filename();
                fs::rename(entry.path(), new_path);
                std::cout << "Moved: " << entry.path() << " -> " << new_path
                    ↪  << "\n";
            }
        }
    }
}
```

Аналіз ключових операцій: - `auto one_year_ago` — обчислює час, що минув рік тому. - `fs::last_write_time(entry)` — отримує дату останньої модифікації файлу.

—

3.5 Завдання 5: Видалення невеликих Word-файлів

```
static void delete_small_word_files(const fs::path& directory) {
    for (const auto& entry : fs::recursive_directory_iterator(directory)) {
        if (entry.is_regular_file()) {
            auto ext = entry.path().extension().string();
            if ((ext == ".doc" || ext == ".docx") && fs::file_size(entry) <
                ↪ 100 * 1024) {
                fs::remove(entry.path());
```

```
                std::cout << "Deleted: " << entry.path() << "\n";
            }
        }
    }
}
```

Аналіз ключових операцій: - `fs::file_size(entry)` — визначає розмір файлу. - `fs::remove(entry.path())` — видаляє файл.

—

3.6 Завдання 6: Розрахунок середнього розміру текстових файлів

```
static double calculate_average_txt_file_size(const fs::path& directory) {
    std::vector<std::size_t> file_sizes;

    for (const auto& entry : fs::directory_iterator(directory)) {
        if (entry.is_regular_file() && entry.path().extension() == ".txt") {
            file_sizes.push_back(fs::file_size(entry.path()));
        }
    }

    if (file_sizes.empty()) {
        return 0.0;
    }

    std::size_t total_size = 0;
    for (const auto& size : file_sizes) {
        total_size += size;
    }

    return static_cast<double>(total_size) / file_sizes.size();
}
```

Аналіз рядків: - `std::vector<std::size_t> file_sizes` — зберігає розміри текстових файлів. - `fs::file_size(entry.path())` — отримує розмір кожного файлу. - `total_size / file_sizes.size()` — розраховує середній розмір.

—

4 Висновок

Розглянуті задачі демонструють широкі можливості бібліотеки `<filesystem>` у C++. Вона дозволяє ефективно працювати з файлами та директоріями, забезпечуючи компактність та читабельність коду. Розуміння цих принципів дозволяє автоматизувати обробку даних у практичних проєктах.