Лекція 7. Робота з файлами. Потоки

Версія 24 квітня 2023 р.

Анотація

Ця лекція знайомить з форматом роботи з файлами та потоками в більш широкому розумінні в C++.

Зміст

1 Текстові потоки

Для роботи з файлами в сучасному C++ прийнято використовувати так звані файлові потоки. Серед переваг є уніфікований і простий для всіх потоків інтерфейс та вбудована підтримка RAII, що проявляється в тому, що користувачу не обов'язково самостійно закривати файл методом `close()` – він закривається автоматично деструктором потоку.

Але що ж саме таке ці потоки? Якщо простими словами, уявіть сутність, що тримає в собі буфер, який може безперешкодно розширюватись та звужуватись, та дозволяє послідовно зчитувати та записувати дані відповідного типу. Найчастіше ми маємо справу з текстовими потоками. Додатковою зручністю використання потоків STL є автоматичне конвертування типів, за можливістю, при запису в змінну іншого типу.

Файлові потоки задаються в заголовку `fstream`. Наприклад, конструктор std::ifstream, що є простим псевдонімом `std::basic_ifstream<char>`,

тобто поток елементів `char`, іншими словами, звичайний UTF-8 текст, приймає назву файла, та режим, як його відкривати, що за замовченням є для зчитування.[1]

```
#include <fstream>
std::ifstream f_in("my_file.txt");
```

Аналогічно для потоків запису та двусторонніх потоків.

При роботі з `std::ifsteam` найголовнішим є оператор зчитування >>, так само як для `std::ifsteam` є ключовим оператор << для запису. Вони записують або зчитують текст до наступного роздільника слова і видають наступне при повторному виклику. Існують також методи для більш складного маніпулювання буфером про які ви можете дізнатися з довідки.

Ви могли помітити, що робота з файловими потоками повністю аналогічна роботі з використанням звичних для вас `std::cout` та `std::cin`. Це в тему переваг використання зручних і зрозумілих інтерфейсів, бо `std::cout` та `std::cin` це ніщо інше як глобальні екземпляри класів `std::ostream` та `std::istream`, відповідно.

2 Двійкові файли

Звісно, все на комп'ютері є набором бітів, одиниць та нулів. Але тим не менш, є відомі формати, для праці з якими існують бібліотеки, як от текстові файли. Проте ніщо не завадить нам записати вміст пам'яті програми напряму в файл. Це має зрозумілу перевагу в швидкості читання та запису, а також займає мінімальне місце на диску.[2] Те, що цей файл неможливо прочитати як текст, може бути як перевагою (приватність), так і недоліком (важко працювати, складно переглянути). Також, програма, яка буде зчитувати цей файл, має точно знати що і в якому порядку там записано, щоб правильно інтерпретувати байти. Для трансформації об'єктів в та з послідовності байтів використовується вбудований шаблон `reinterpret_cast`. Наприклад, запис:

```
#include <fstream>
#include <vector>
int main () {
    std::string file_name{"image.dat"};
    std::ofstream file(file_name, std::ios_base::out |
        std::ios_base::binary);
    int rows = 2, cols = 3;
```

[1] Більше про доступні режими можна подивитись тут.

[2] Мова не йде про стиснення та архівація файлів, звісно.

```
    std::vector<float> vec(rows * cols);
    file.write(reinterpret_cast <char*>(&rows), sizeof(rows));
    file.write(reinterpret_cast <char*>(&cols), sizeof(cols));
    file.write(reinterpret_cast <char*>(&vec.front()), vec.size()
        * sizeof(float));
    return 0;
    }
```

і зчитування

```
#include <fstream>
#include <iostream>
#include <vector>
int main() {
    std::string file_name{"image.dat"};
    int r = 0, c = 0;
    std::ifstream in(file_name, std::ios_base::in |
        std::ios_base::binary);
    if (!in) { return EXIT_FAILURE; }
    in.read(reinterpret_cast <char*>(&r), sizeof(r));
    in.read(reinterpret_cast <char*>(&c), sizeof(c));
    std::cout << "Dim: " << r << " x " << c << std::endl;
    std::vector <float> data(r * c, 0);
    in.read(reinterpret_cast <char*>(&data.front()), data.size()
        * sizeof(data.front()));
    for(auto d : data) { cout << d << endl; }
    return 0;
}
```

3 Бібліотека `std::filesystem`

В C++ існує бібліотека, яка надає широкий спектр інструментів для роботи з файловою системою, яка називається `std::filesystem` та підключається додаванням заголовку `filesystem`. Ось декілька корисних класів та функцій, на які варто звернути увагу

- path
- directory_iterator
- absolute
- copy
- copy_file

- current_path
- exists
- remove

та багато інших в залежності від дій, які вам треба зробити.

Література

- Stroustrup, Bjarne. The C++ programming language. Pearson Education, 2013 (Глава 38).
- CPPReference, basic_fstream.
- CPPReference, filesystem.