Потоковые хитрости. Конспект темы

Зачем сбрасывать буфер

Выведенный в поток текст вместо немедленной записи сохраняется в некоторое промежуточное хранилище и сбрасывается на диск только по мере наполнения этого хранилища либо в случае явной команды `flush`. Такая оптимизация называется **буферизацией**, а промежуточное хранилище — **буфером**.

`endl` — это вывод конца строки с одновременным сбросом буфера. Сброс буфера снижает эффективность программы. Лучше избегать его, когда интерактивность вывода не требуется, но важна производительность.

Используйте `endl` при отладке или профилировке, если есть риск, что программа упадёт и не успеет выдать нужную информацию. В остальном `endl` и `\n` работают одинаково — ваша задача найти баланс между эффективностью и удобством.

Сбрасывать буфер можно не в начале новой строки, а просто по желанию. Для этого используют манипулятор `std::flush` и функцию потока `flush`:

```
#include <fstream>
#include <string>

using namespace std;

int main() {
    ofstream out_file("ballad.txt"s);
    for (int i = 0; i < 10; ++i) {
        // такой же эффект, как если бы мы написали endl
        out_file << "С любимыми не расставайтесь\n"s << flush;
    }

    throw;
}
```

Связь потоков: в поисках оптимального ввода и вывода

Чтение из `cin` заставляет буфер `cout` и `cerr` опустошаться, как если бы вы добавили `cout.flush()` перед ним. Чтобы исправить ситуацию, отвяжите `cin` от `cout`: `cin.tie(nullptr);`.

```
#include <iostream>
#include <string>

#include "log_duration.h"

using namespace std;

int main(int argc, const char** argv) {
    // не забываем, что один аргумент — это название программы,
    // поэтому argc должно быть как минимум 2
    if (argc < 2) {
        cerr << "Пожалуйста, задайте как минимум 1 аргумент"s << endl;
        return 1;
    }

    int arg = std::stoi(argv[1]);

    if (arg == 1) {
        LOG_DURATION("endl"s);
        int i;
        while (cin >> i) {
            cout << i * i << endl;
        }
    }

    if (arg == 2) {
        LOG_DURATION("\\n"s);
        int i;
        while (cin >> i) {
            cout << i * i << "\n"s;
        }
    }

    if (arg == 3) {
        LOG_DURATION("\\n with tie"s);
        cin.tie(nullptr);
        int i;
        while (cin >> i) {
            cout << i * i << "\n"s;
        }
    }
}
```

Отвязывание `cin` от `cout` повлияет на всё последующее выполнение. Если разрабатываете основную функцию программы, это неважно. Но если такое происходит в функции библиотеки, которая будет использована в разных программах, лучше в конце работы привяжите `cin` обратно:

```
...
if (arg == 3) {
    LOG_DURATION("\\n with tie"s);
    auto tied_before = cin.tie(nullptr);

    int i;
    while (cin >> i) {
        cout << i * i << "\n"s;
    }

    cin.tie(tied_before);
}
...
```

Ещё сильнее ускорить ввод и вывод в `cin` и `cout` может магическая к

Команда `ios_base::sync_with_stdio(false)` ускоряет ввод и вывод в `cin` и `cout`, но может помешать в следующих случаях:

- при подключении сторонних библиотек, выводящих информацию в стандартный вывод,
- при многопоточной работе.

Команда `sync_with_stdio` отключает синхронизацию между стандартными потоками C и C++, сохраняя ресурсы, а также между разными потоками выполнения.

Потоки ввода и вывода `stream` передают данные между произвольным источником `source` и потребителем `sink`.

Потоки выполнения `thread` служат для одновременного выполнения нескольких операций в программе или для ускорения задачи за счёт выполнения на разных ядрах процессора. Термины «многопоточный», «потокобезопасный» относятся всегда к потокам выполнения.