12 – Exercises on File Streams

Université Jean Monnet – Saint-Étienne 2026/2027

Quick reference

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

// --- Writing ---
std::ofstream out("notes.txt");   // open for writing (overwrites)
if (!out) { /* file could not be opened */ }
out << "hello " << 42 << "\n";   // same << syntax as std::cout
out.close();                     // optional but good practice

// --- Reading word by word (operator >>) ---
std::ifstream in("notes.txt");
if (!in) { /* file could not be opened */ }
std::string word;
int number;
in >> word >> number;            // reads "hello" then 42
                                 // >> skips whitespace and newlines

// --- Reading line by line (std::getline) ---
std::string line;
while (std::getline(in, line)) {  // returns false at end of file
    // process line
}
```

Key points:

- Always check that a file opened successfully with `if (!stream)`.
- `>>` skips leading whitespace and stops at the next whitespace character — it reads one *token* at a time.
- `std::getline` reads everything up to (and discarding) the next `'\n'`, including spaces.
- After `>>` reads the last token on a line, a stray `'\n'` may remain in the buffer. Call `in.ignore()` before `std::getline` to discard it.

Part A — Writing to a file

Problem 1 — What ends up in the file?

For each program, write the exact content that would appear in the output file.

Writing numbers

```cpp
#include <fstream>

int main() {
    std::ofstream out("track.txt");
    out << "title: " << "Echoes" << "\n";
    out << "bpm: "  << 120     << "\n";
    out << "bars: " << 4 * 16 << "\n";
    out.close();
    return 0;
}
```

Content of `track.txt`:

Writing in a loop

```cpp
#include <fstream>

int main() {
    std::ofstream out("pixels.txt");
    int values[4] = {10, 50, 200, 80};
    for (int i = 0; i < 4; i = i + 1)
        out << i << " " << values[i] << "\n";
    return 0;
}
```

Content of `pixels.txt`:

Problem 2 — Spot the bug

This program should write five scores to `scores.txt`, one per line. It has two bugs.

Buggy writer

```cpp
#include <fstream>

```

```cpp
int main() {
    std::ofstream out;

    int scores[5] = {72, 88, 45, 95, 60};
    for (int i = 0; i < 5; i = i + 1)
        out >> scores[i] >> "\n";

    return 0;
}
```

Bug 1:

Fix 1:

Bug 2:

Fix 2:

Problem 3 — Write the code

Write a complete program that takes a filename and a list of integers from the command line and writes each integer on its own line to that file.

Invocation:

```
./write notes.txt 60 62 64 65 67
```

Expected content of `notes.txt`:

```
60
62
64
65
67
```

If the file cannot be opened, print `"error: could not open file"` and return `1`.

Part B — Reading with `>>`

Problem 1 — Trace the reads

`track.txt` contains:

```
Echoes 120 64
```

What does this program print?

Reading tokens

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::ifstream in("track.txt");
    std::string title;
    int bpm, bars;

    in >> title >> bpm >> bars;

    std::cout << title << std::endl;   // line A
    std::cout << bpm << std::endl;     // line B
    std::cout << bars << std::endl;    // line C
    std::cout << bpm * bars << std::endl; // line D

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Problem 2 — Reading multiple lines with >>

`pixels.txt` contains (as written by exercise 1.1):

```
0 10
1 50
2 200
3 80
```

What does this program print?

Summing pixel values

```cpp
#include <iostream>
#include <fstream>

int main() {
    std::ifstream in("pixels.txt");
    int idx, value, total = 0, count = 0;

    while (in >> idx >> value) {
```

```
        total = total + value;
        count = count + 1;
    }

    std::cout << count << std::endl;      // line A
    std::cout << total << std::endl;      // line B
    std::cout << total / count << std::endl; // line C

    return 0;
}
```

Line A:

Line B:

Line C:

What does `while (in >> idx >> value)` actually test? When does it become false?

Problem 3 — Spot the bug

This program reads a file of the form `<name> <score>` per line and should print the name and score of each entry. It has one bug.

`results.txt`:

```
Alice 88
Bob 74
Clara 95
```

Buggy reader

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

int main() {
    std::ifstream in("results.txt");
    if (!in) {
        std::cout << "error" << std::endl;
        return 1;
    }

    std::string name;
    int score;
```

```
14
15      while (in >> score >> name) {
16          std::cout << name << ": " << score << std::endl;
17      }
18
19      return 0;
20  }
```

Bug:

Fix:

What does the buggy version actually read into each variable on the first iteration?

Part C — Reading with `std::getline`

Problem 1 — `>>` vs `getline`: what do they read?

`song.txt` contains:

```
Miles Davis
Kind of Blue
1959
```

For each program, write what is stored in each variable after the reads.

Version A (using `>>`):

```
std::ifstream in("song.txt");
std::string a, b;
int c;
in >> a >> b >> c;
```

`a =` `b =` `c =`

Why does `a` not contain the full first line?

Version B (using `getline`):

```
std::ifstream in("song.txt");
std::string a, b, c;
std::getline(in, a);
std::getline(in, b);
std::getline(in, c);
```

a = b = c =

Problem 2 — Trace: reading a whole file line by line

`setlist.txt` contains:

```
So What
Freddie Freeloader
Blue in Green
All Blues
Flamenco Sketches
```

What does this program print?

Line counter

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::ifstream in("setlist.txt");
    std::string line;
    int count = 0;

    while (std::getline(in, line)) {
        count = count + 1;
        std::cout << count << ": " << line << std::endl;
    }

    std::cout << "total: " << count << std::endl;
    return 0;
}
```

Output:

Problem 3 — Mixing >> and `getline`

`tracks.txt` contains:

```
3
So What
Freddie Freeloader
Blue in Green
```

The first line is the number of tracks; the rest are track names (which may contain spaces). What does this program print?

Mixed read

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main() {
    std::ifstream in("tracks.txt");
    int n;
    in >> n;
    in.ignore();                // discard the '\n' after the number

    std::cout << n << std::endl; // line A

    std::string name;
    for (int i = 0; i < n; i = i + 1) {
        std::getline(in, name);
        std::cout << i + 1 << ": " << name << std::endl;
    }

    return 0;
}
```

Output:

Try removing `in.ignore()`. What happens then?

Part D — Putting it all together

Problem 1 — Write the code: pixel file statistics

`image.txt` contains one integer per line (pixel brightness values, 0–255). The number of lines is not known in advance.

Write a complete program that:

1. Takes the filename as `argv[1]`; prints an error and returns `1` if the argument is missing or the file cannot be opened.
2. Reads all pixel values from the file.
3. Prints the count, sum, minimum, and maximum.

Given `image.txt`:

```
34
210
88
5
175
120
```

Expected output:

```
count: 6
sum: 632
min: 5
max: 210
```

Problem 2 — Write the code: setlist filter

Write a program that:

1. Takes two filenames from `argv`: an input file and an output file.
2. Reads the input line by line. Each line has the format `<title> <duration>` where `<title>` is a single word and `<duration>` is an integer (seconds).
3. Writes to the output file only the lines whose duration is strictly above 200 seconds, preserving the same format.
4. After writing, prints the number of songs kept.

Given `setlist.txt`**:**

```
Echoes 340
Shine 195
Breathe 163
Time 252
Money 382
```

Expected content of output file:

```
Echoes 340
Time 252
Money 382
```

Expected terminal output:

```
3 songs kept
```