

EN 001203 Computer Programming

L008: File I/O

Faculty of Engineering, Khon Kaen University

Submission: <https://autolab.en.kku.ac.th>

-
- * Submit an answer to a question with a file with `txt` extension. E.g., an answer for Q1 should be submitted in a text file “Q1.txt”
 - * Submit a program to a (programming) problem with a file with `cpp` extension. E.g., a program for P2 should be named “P2.cpp”
 - * All answers and programs must be packaged together in a tar file.
 - * Each question or problem is worth 60 points. The 240 points are counted as full score. Students are encouraged to work on as many problems as they can, but 240 points should be adequate.
-

“I don't think of all the misery,
but of the beauty that still remains.”

--- Anne Frank

P1. Simple file read. Write a program to ask a user for a file name, read the contents of the file, and print it out.

[Hint: given `fin` is an input file handler and `s` is a string, `fin >> s` get us one word at a time; `getline(fin, s)` get us one line at a time.]

Have the program interact exactly like what shown in the examples. A file name is supplied by a user.

Interacting example: when inputting `transcendence.txt` as a file name, when contents of the file are shown in Figure 1.

File: **transcendence.txt**
I don't think of all the misery,
but of the beauty that still remains.
Anne Frank

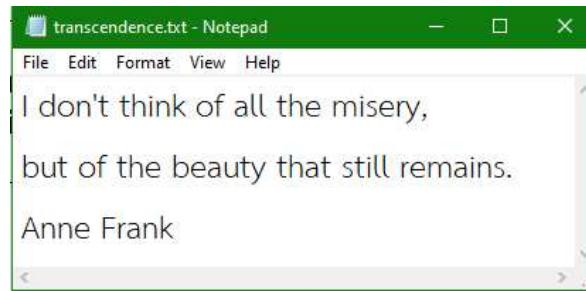


Figure 1. Contents of transcendence.txt

Interacting example: when inputting `forge.txt` as a file name, when contents of the file are shown in Figure 2.

File: **forge.txt**
You cannot dream yourself into a character;
you must hammer and forge yourself one.
James A. Froude

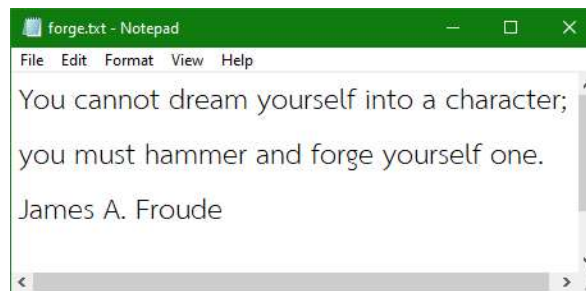


Figure 2. Contents of forge.txt

P2. Simple file write. Write a program to keep asking a user for a number until the user enter any letter. While doing that, the program writes what the user enters into a file, named "`usenum.dat`". Let each number has its own line. (See Figure 3 along.)

[Hint: the `while(cin >> n){ ... }` trick could be handy.]

Have the program interact exactly like what shown in the examples. Numbers are supplied by a user.

Interacting example: when inputting 1, 3, 8, 4, 9, a.

n: 1
n: 3
n: 8

```
n: 4
n: 9
n: a
```

The result can be seen in `usenum.dat`. Notice that a letter 'a' does not get in.

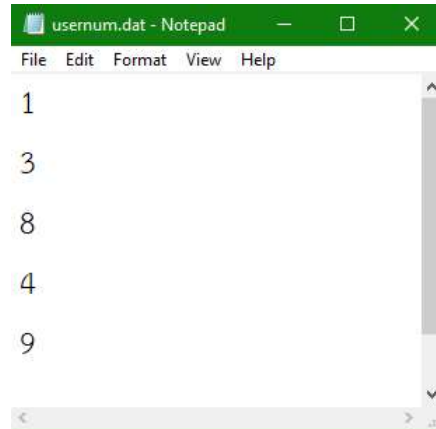


Figure 3. Contents of `usenum.txt` after running P3

P3. Sum numbers in a file. Write a program to ask a user for a file name, read the contents of the file as numbers, sum up all the numbers, and report it out. Note that contents of a file are formatted as shown in Figure 3.

[Hint: we can add up the numbers, while navigating the contents of the file.]

Have the program interact exactly like what shown in the examples. A file name is supplied by a user.

Interacting example: when inputting `usenum.txt`, when contents of the file are shown in Figure 3.

```
File: usenum.dat
sum = 25
```

P4. Count words in a file. Write a program to ask for a file name, read the contents of the file, count a number of words in the contents, and report it out.

[Hint: given `fin` is an input file handler and `s` is a string, `cin >> s` get us one word at a time; `getline(fin, s)` get us one line at a time. Recall that a number of words in a sentence is a number of spaces plus one: $\text{\#words} = \text{\#spaces} + 1$. Function `trim` is handy for cleaning up extra spaces around a line.]

Have the program interact exactly like what shown in the examples. A file name is supplied by a user.

Output example: when inputting `transcendence.txt` as a file name, when contents of `transcendence.txt` is shown in Figure 1.

File: `transcendence.txt`
words = 16

Use a starter code below.

Starter code:

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

using namespace std;

string trim(string w){
    int i, j;
    string r;

    for(i=0; i < w.length(); i++){
        if(w[i] != ' '){
            break;
        }
    }

    for(j=w.length()-1; j > 0; j--){
        if(w[j] != ' '){
            break;
        }
    }

    r = w.substr(i, j-i+1);

    return r;
} // It is good to learn how this trim works.

int main(){

    string fname;

    cout << "File: ";
    cin >> fname;

    ifstream fin(fname);
    string s;
    int wc, count;

    count = 0;
    while(getline(fin, s)){
        s = trim(s); // Clean up the line (Make things easier).
```

```

    // Write your code here!!!
    cout << s << endl; // Learn the mechanism first! Then remove it!
    // count words: count #words in each line and add them up.
}

cout << "# words = " << count << endl;

return 0;
}

```

P5. Extract EEG sequence out of a file. Electroencephalography (EEG) is a neuro-observing method to monitor electrical activities of the brain. Using multiple electrodes placed on the scalp, voltages resulting from neural activities of the brain can be sensed. EEG signals are multiple time-series signals. EEG is used in various applications including medical diagnosis, neuroscience research, brain-computer interface, and scientific studies of meditation.

The EEG data used here is taken from downloaded from:

<https://archive.ics.uci.edu/ml/datasets/eeg+database>.

The data content looks like

```

# co2a0000364.rd
# 120 trials, 64 chans, 416 samples 368 post_stim samples
# 3.906000 msecs uV
# S1 obj , trial 0
# FP1 chan 0
0 FP1 0 -8.921
0 FP1 1 -8.433
0 FP1 2 -2.574

```

Any line starts with ‘#’ is a metadata (information describing data) and we have to extract it to our variable. The example above has data from line 6 on:

```

0 FP1 0 -8.921
0 FP1 1 -8.433
0 FP1 2 -2.574

```

Details are described in the source website. In short, it is data with multiple time-series signals. Each time-series signal is a sequence of numbers, e.g., [-8.921, -8.433, -2.574]. The last column has the time-series values. The third column has their time index. The first and second columns indicate a trial number and a sensor position of the time-series.

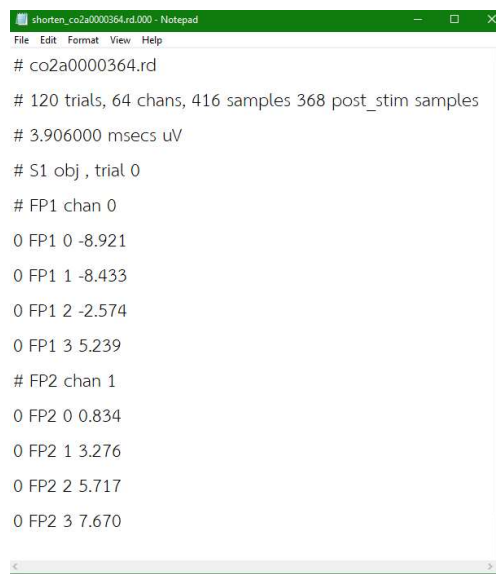
Given this EEG format, write a program to (1) ask a user for EEG file name, trial number, and sensor position, then (2) take a series of the EEG values associating to the specified trial number and sensor position and print it out.

[Hint: it is like navigating the file line by line and for each line take the last part, when the first two parts are matched.]

Have the program interact exactly like what shown in the examples. A file name, trial number, and sensor position are supplied by a user.

Interacting example: when inputting shorten_co2a0000364.rd.000, 1, FP2. , when the contents of shorten_co2a0000364.rd.000 is shown in Figure 4.

File: **shorten_co2a0000364.rd.000**
 Series: 0 FP2
 0.834; 3.276; 5.717; 7.670;



```

shorten_co2a0000364.rd.000 - Notepad
File Edit Format View Help
# co2a0000364.rd
# 120 trials, 64 chans, 416 samples 368 post_stim samples
# 3.906000 msecs uV
# S1 obj , trial 0
# FP1 chan 0
0 FP1 0 -8.921
0 FP1 1 -8.433
0 FP1 2 -2.574
0 FP1 3 5.239
# FP2 chan 1
0 FP2 0 0.834
0 FP2 1 3.276
0 FP2 2 5.717
0 FP2 3 7.670
  
```

Figure 4. Contents of shorten_co2a0000364.rd.000

Use a starter code below.

Starter code:

```

#include <iostream>
#include <fstream>

using namespace std;

string get_last(string txt){
    int i, L;
    L = txt.length();
    // Find the last space
  
```

```

        i = 0;
        for(int j=0; j<L; j++){
            if(txt[j] == ' '){
                i = j;
            }
        }

        return txt.substr(i+1, L - i);
    }

int main(){

    string fname, series, sline;

    cout << "File: ";
    getline(cin, fname);

    cout << "Series: ";
    getline(cin, series); // This helps taking 0 FP1 in one shot!

    ifstream fin(fname);

    while(getline(fin, sline)){
        // Write your code here!!!
        // Check if the series match. // Note: sline.compare may help!!!

        cout << get_last(sline) << " "; // This get_last is handy!!!
    }

    return 0;
}

```

P6. SIR model. The SIR model is a mathematical model to estimate the spread of an infectious disease. The SIR model categorizes population into 3 groups: susceptible, infectious, and removed groups. People can be moved between groups.

The susceptible group is a subpopulation of susceptible individuals. Variable S represents a number of susceptible individuals. Susceptible individuals do not have the disease, but can be infected. When a susceptible individual come into (infectious) contact with an infectious individual, the susceptible individual is infected and re-assigned to the infectious group.

The infectious group refers to individuals who have been infected and can infect susceptible individuals. Variable I represents a number of infectious individuals. After a period of time, an infectious individual will either recover with immunity or die. Either case, the individual will be moved to the recovered group. Note that, a conventional SIR model assumes that a proportion of having been recovered without immunity and become susceptible again is negligible.

The removed group refers to individuals who cannot be infected anymore, i.e., having recovered from the disease with immunity or died. Variable ***R*** represents a number of immune or deceased individuals. A removed individual will stay in this group: he/she cannot be re-infected nor infect others.

To simplify the matter, an SIR model assumes that: (1) the epidemic is short; the total population remains constant; (2) a rate of increase in ***I*** is constant; and (3) a rate of increase in ***R*** is constant.

That is, the progression of group sizes can be illustrated in Figure 5.

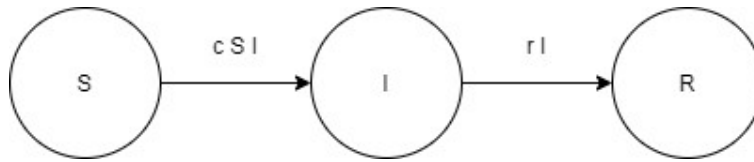


Figure 5. Progression in an SIR model

The progression of the disease is modeled as:

The change in ***S***:

$$\frac{dS}{dt} = -c S I$$

A number of susceptible individuals is decreased by infection, which depends on the infectious contact. The contact rate ***c*** is assumed constant.

The change in ***I***:

$$\frac{dI}{dt} = c S I - r I$$

A number of infectious individuals is changed by newly infected individuals and newly removed individuals. The removal rate ***r*** is assumed constant.

The change in ***R***:

$$\frac{dR}{dt} = r I$$

A number of removed individuals is increased by individuals recovered or died.

To determine whether the decrease will spread or be contained, it can be seen from

$$\frac{dI}{dt} = c S_o I - r I,$$

where ***S_o*** is an initial susceptible number. If ***c S_o*** > ***r*** (making the change in ***I*** positive), the disease will spread. The rates ***c*** and ***r*** depend on social practice,

countermeasure, and healthcare capability for prevention and intervention. They both can be determined from data through model fitting. (Learn more at *Wikipedia: Compartmental models in epidemiology*.)

Write a program to take in initial conditions (S, I, R), model parameters (c and r), and a number of periods, then compute the progression of the disease and write the results into a file named `epidemic.txt`, along with summarize if the disease is spreading.

[Hint: the original SIR model is a differential model; here we simplify it to a discrete model. See a starter code.]

Have the program interact exactly like what shown in the examples. Initial conditions, model parameters, and a number of simulated periods are supplied by a user.

Interacting example 1: when inputting 50000, 32, 0, $4.2\text{e-}6$, 0.04, and 5. Note: $4.2\text{e-}6$ is a notation for 4.2×10^{-6} . The program also writes `epidemic.txt` file, whose contents are shown in Figure 6.

```
Initial S, I, R: 50000 32 0
c, r: 4.2e-6 0.04
periods: 5
--> It is spreading!
```

```
epidemic.txt - Notepad
File Edit Format View Help
0: S = 50000; I = 32; R = 0
1: S = 49993; I = 37; R = 1
2: S = 49985; I = 44; R = 3
3: S = 49976; I = 51; R = 5
4: S = 49965; I = 60; R = 7
5: S = 49953; I = 70; R = 9
```

Figure 6. Contents of `epidemic.txt` after running P6 (example 1)

Interacting example 2: when inputting 8000, 100, 0, $5\text{e-}6$, 0.04, and 3. (Contents of the output file is not shown.)

Initial S, I, R: **8000 100 0**
 c, r: **5e-6 0.04**
 periods: **3**
 --> It is contained!

Use a starter code below.

Starter code:

```
#include <iostream>
#include <iomanip>
#include <fstream>

using namespace std;

int main(){

    float S, I, R, c, r;
    float dS, dI, dR;
    int N;

    cout << "Initial S, I, R: ";
    cin >> S >> I >> R;

    cout << "c, r: ";
    cin >> c >> r;

    cout << "periods: ";
    cin >> N;

    if // Write your code here!!!
        cout << "--> It is spreading!" << endl;
        cout << "--> It is contained!" << endl;

    ofstream fout("epidemic.txt");
    fout << fixed << setprecision(0); // This makes autograder work easier

    fout << "0: S = " << S << "; I = " << I << "; R = " << R << endl;

    for(int n=0; n < N; n++){
        dS = - c*S*I;
        S += dS;

        // Write your code here!
        fout << n+1 << ": S = " << S << "; I = " << I << "; R = " << R << endl;
    }

    return 0;
}
```