File I/O

Due this week

- **HW 5**
  - Write solutions in VSCode and paste in **CodeRunner**.
  - Extra-credit start early
  - Zip your .cpp files and submit on canvas. Check the due date! **No late submissions!!**
- **3-2-1 Reflection on Friday**
- **Mandatory Grading Interview**
- **Quiz 5.** Check the due date! **No late submissions!!**

Warm up activity

- **Write a program to estimate the number of sentences in “The Hobbit”**
- **You can also just write pseudocode**
- **Can work alone or with a partner**
- **Hint: utilize the program we wrote on Monday!**

Recap

Reading and Writing Disk Files

You can also read and write files stored on your hard disk:

- plain text files
- binary information (a binary file)
  - Such as images or audio recording

To read/write files, you use *variables* of the stream types:

`ifstream` for input from plain text files.
`ofstream` for output to plain text files.
`fstream` for input and output from binary files.

You must `#include <fstream>`

Code for opening a stream

```
ifstream in_file;
in_file.open("input.txt"); //filename is input.txt
```

An alternative shorthand syntax combines the 2 statements:

```
ifstream in_file(“input.txt”);
string name;
int number;
in_file >> name >> number;
```

Closing a Stream

- When the program ends, all streams that you have opened will be automatically closed.
- You *can* manually close a stream with the **close** member function:

```
in_file.close();
```

1. Create variable
2. Open file (provide filename)
3. Check if file opened successfully
4. Read from file
5. Close file

Reading from a stream

- The **>>** operator returns a “not failed” condition, allowing you to combine an input statement and a test.
- A “failed” read yields a **`false`** and a “not failed” read yields a **`true`**.

```
if (in_file >> name >> number)
{
   // Process input
}
```

Reading from a stream

- You can even read ALL the data from a file because running out of things to read causes that same “failed state” test to be returned:

```
while (in_file >> name >> number)
{
   // Process input
}
```

Reading A Whole Line: `getline`

- The function **`getline()`** reads a whole line up to the next '\n', into a C++ string.

- The '\n' is then deleted, and NOT saved into the string.

```
string line;
ifstream in_file("myfile.txt");

getline(in_file, line);
```

Reading A Whole Line in a Loop: `getline`

- The **`getline`** function, like the others we've seen, returns the "not failed" condition.
- To process a whole file line by line:

```
string line;
while( getline(in_file, line)) //reads whole file
{
   // Process line
}
```

Functions in <cctype> (Handy for Lookahead)

Function	Accepted Characters
isdigit	0 ... 9
isalpha	a ... z, A ... Z
islower	a ... z
isupper	A ... Z
isalnum	a ... z, A ... Z, 0 ... 9
isspace	White space (space, tab, newline, and the rarely used carriage return, form feed, and vertical tab)

Reading Words and Characters

What really happens when reading a `string`?

```
string word;
in_file >> word;
```

1. Any whitespace is skipped (whitespace is: `'\t' '\n' ' '`).
2. The first character that is not white space is added to the string `word`. More characters are added until either another white space character occurs, or the end of the file has been reached.

Reading Words and Characters

You can read a single character, including whitespace, using `get()`:

```
char ch;
in_file.get(ch);
```

The `get` method returns the "not failed" condition so:

```
//reads entire file, char by char
while (in_file.get(ch))
{
   // Process the character ch
}
```

Reading a Number Only If It Is a Number

- You can look at a character after reading it and then put it back.
- This is called *one-character lookahead*. A typical usage: check for numbers before reading them so that a failed read won't happen:

```
char ch;
int n=0; //for reading an entire int
in_file.get(ch);

if (isdigit(ch)) // Is this a number?
{
   // Put the digit back so that it will be part of the number we read
   in_file.unget();

   in_file >> n; // Read integer starting with ch

}
```

Writing to Files

Writing to a Stream

Here's everything:

1. create output stream variable
2. open the file
3. write to file
4. close file!

```
ofstream out_file;
out_file.open("output.txt");
if (in_file.fail()) { return 0; }
out_file << name << " " << value << endl;
out_file << "CONGRATULATIONS!!!" << endl;
```

Working with File Streams