Processing Disk Files

CS 150 – C++ Programming I
Lecture 12

Stream Review

- `<iostream>`: `istream`, `ostream`
  - Global standard stream objects: `cin`, `cout`, `cerr`
- Data loop; stops when source "out of data"
  - Three ways to read data from a stream
  - `while (cin.get(ch)) cout.put(ch);`
  - `while (getline(in, line)) cout << line << endl;`
  - `while (in >> n) cout << n << endl;`
- Exercise: using a data loop with *`sumEvens`*

More Stream Review

- Filter programs: read from *stdin*, write to *stdout*
  - State filters look for changes in input
    - cecho only printed when certain conditions were met
    - The printing Boolean flag used to monitor stream state
  - Process filters modify stream contents
    - Example: the *toupper* filter
    - ```
      char ch;
      while (cin.get(ch))
        cout.put(toupper(ch));
      ```
- Exercise: The CS 150 Encryption Library
      ```

Using Disk Files for I/O

- Classes to explicitly specify source/sink in code
  - Header `<fstream>` for disk file I/O
  - `ifstream`: connects to a file and reads data
  - `ofstream`: connect to a file and writes data
  - Also need `cin` / `cout`? include `<iostream>`
- Using explicit files involves three steps:
  - 1. Create an instance of the stream class
  - 2. Associate stream with a source or sink (open)
  - 3. Read from or write to the stream
  - Automatically closed by destructor so don't need close

Steps 1 & 2 for Files

- Separate or combine these two steps
  - `ifstream fin;` `// variable`
    `fin.open("junk.txt");` `// open`
  - `ifstream fin("junk.txt");` `// combined`
    - Use the same pattern for `ofstream` (output)
- Check if file opened OK
  - `if (! fin)` ...*or*... `if (fin.fail())`
    - What to do when you can't open?
    - Use a loop and re-prompt for the correct file
    - Print error message (on `cerr`) and return or exit

Line-Oriented Data Loops

- Read one line of text at a time
  - `while (getline(in, line)) { . . . }`
  - Does NOT return the `string` (passed by reference)
  - Returns the stream (use as the loop condition)
  - Must remember to `#include <string>`
- Exercise: the *FlipLines* Problem
  - Open the file or print an error message
  - Reverse first pair of lines, second pair, etc.

Token-Oriented Data Loops

- Read one token at a time
  - `while (in >> var) { . . . }`
- Fails at end-of-file and when it cannot convert token
  - Clear error flags with: `in.clear()`
  - Still need to remove offending input
  -
```
string errdata;
if (in.fail()) {
    in.clear();
    in >> errdata;
}
```
- Exercise: *Expenses* - process file with characters & tokens