CIS 22A – Lecture 13

Manish Goel

Using Files - Recap

1. Requires `fstream` header file
   - use `ifstream` data type for input files
   - use `ofstream` data type for output files
   - use `fstream` data type for both input, output files
2. Can use >>, << to read from, write to a file
3. Can use `eof` member function to test for end of input file

4. `fstream` object can be used for either input or output
5. Must specify mode on the `open` statement
6. Sample modes:

   `ios::in` – input `ios::out` – output
7. Can be combined on `open` call:

```
dFile.open("class.txt", ios::in | ios::out);
```

File Access Flags

Table 12-2

File Access Flag	Meaning
`ios::app`	Append mode. If the file already exists, its contents are preserved and all output is written to the end of the file. By default, this flag causes the file to be created if it does not exist.
`ios::ate`	If the file already exists, the program goes directly to the end of it. Output may be written anywhere in the file.
`ios::binary`	Binary mode. When a file is opened in binary mode, data is written to or read from it in pure binary format. (The default mode is text.)
`ios::in`	Input mode. Data will be read from the file. If the file does not exist, it will not be created and the open function will fail.
`ios::out`	Output mode. Data will be written to the file. By default, the file's contents will be deleted if it already exists.
`ios::trunc`	If the file already exists, its contents will be deleted (truncated). This is the default mode used by `ios::out`.

Using Files - Example

```
// copy 10 numbers between files
// open the files
fstream infile("input.txt", ios::in);
fstream outfile("output.txt", ios::out);
int num;
for (int i = 1; i <= 10; i++)
{
   infile >> num;     // use the files
   outfile << num;
}
infile.close();       // close the files
outfile.close();
```

Default File Open Modes

- `ifstream`:
  - open for input only
  - file cannot be written to
  - `open` fails if file does not exist
- `ofstream`:
  - open for output only
  - file cannot be read from
  - file created if no file exists
  - file contents erased if file exists
- Can use filename, flags in definition:

```
ifstream gradeList("grades.txt");
```

- File stream object set to `0` (`false`) if open failed:

```
if (!gradeList) ...
```

- Can check `fail` member function to detect file open error:

```
if (gradeList.fail()) ...
```

Error Testing

- Can examine error state bits to determine stream status
- Bits tested/cleared by stream member functions

`ios::eofbit`	set when end of file detected
`ios::failbit`	set when operation failed
`ios::hardfail`	set when error occurred and no recovery
`ios::badbit`	set when invalid operation attempted
`ios::goodbit`	set when no other bits are set

`eof()`	true if `eofbit` set, false otherwise
`fail()`	true if `failbit` or `hardfail` set, false otherwise
`bad()`	true if `badbit` set, false otherwise
`good()`	true if `goodbit` set, false otherwise
`clear()`	clear all flags (no arguments), or clear a specific flag

File I/O Formatting, Reading and Writing

- Same techniques for `fstream` objects as with `iostream`
  `cout`: `showpoint, setw(x), showprecision(x),` etc
- Requires `iomanip` to use manipulators
- Functions - for input with whitespace, to perform single character I/O, or to return to the beginning of an input file
- Member functions:
  `getline`: reads input including whitespace
  `get`: reads a single character
  `put`: writes a single character

The `getline` Function

- Three arguments:
  - Name of a file stream object
  - Name of a `string` object
  - Delimiter character of your choice
  - Examples, using the file stream object `myFile`, and the `string` objects `name` and `address`:

```
getline(myFile, name);
getline(myFile, address, '\t');
```

  - If left out, `'\n'` is default for third argument

Single Character I/O

- `get`: read a single character from a file

```
char letterGrade;
gradeFile.get(letterGrade);
```

Will read any character, including whitespace

- `put`: write a single character to a file

```
reportFile.put(letterGrade);
```

Working with Multiple Files

- Can have more than file open at a time in a program
- Files may be open for input or output
- Need to define file stream object for each file