CS2310 Computer Programming

LT08: File I/O

Computer Science, City University of Hong Kong

Semester A 2023-24

File I/O vs. Console I/O

- "Console I/O" refers to "keyboard input/screen output"
  - console I/O is volatile

- Files I/O is non-volatile
  - input file can be used again and again
  - output file retains results
- Allow off-line processing
- Useful for debugging especially when volume of data is huge

File Streams

- File stream class in C++
  - #include <fstream> // similar with "#include <iostream>"
  - ifstream: stream class for file input, similar with cin
  - ofstream: stream class for file output, similar with cout

- To declare an objects of class ifstream or ofstream, use
  - ifstream fin; // *fin* is a variable name
  - ofstream fout; // *fout* is a variable name

ifstream

- To declare an ifsteam type/object
  - ifstream fin;
- To open a file for reading
  - fin.open("infile.dat"); // infile.dat is the filename
- To read the file content
  - fin >> x; // x is a variable
- To close the file
  - fin.close();

ofstream

- To declare an ofsteam type/object
  - ofstream fout;
- To open a file for writing
  - fout.open("myfile.dat"); // myfile.dat is the filename
- To write something to the file
  - fout << x; // x is a variable
- To close the file
  - fout.close();
- *ps: fin.open() and fout.open() refer to different functions*

Examples

```
#include <fstream>
using namespace std;
int main() {
      ifstream fin;
      ofstream fout;
      int x, y, z;
      fin.open("input.txt");
      fout.open("output.txt");
      fin >> x >> y >> z;
      fout << "The sum is " << x+y+z;
      fin.close();
      fout.close();
      return 0;
}
```

Open a File

- An open file is represented by a stream object of ifstream or ofstream
  - ifstream fin; // fin is a stream object of ifstream
  - ofstream fout: // fout is a stream object of ofstream
  - Any I/O performed on this stream object will be applied to the file
- To open a file, specify the filename and open mode
  - Method I: directly open when declare the stream object, e.g.,

    ```
    iostream fin("filename", mode);
    ```

  - Method II: use the member function open with the stream object, e.g.,

    ```
    iostream fin;
    fin.open("filename", mode);
    ```

File I/O Modes

ios::in	open for input operations
ios::out	open for output operations
ios::binary	open in binary mode
ios::ate	set the initial position at the end of the file if this flag is not set, the initial position is the beginning of the file
ios::app	all output operations are performed at the end of the file, appending the content to the current content of the file
ios::trunc	if the function is opened for output operations and it's already existed, its previous content is deleted and replaced by the new one

Example:

```
ofstream fout;
fout.open("filename", ios::binary);
```

Text Files

- when ios::binary is NOT set, the file is treated as a text file.
  - All input/output is assumed to be text and may suffer formatting transformations.
- I/O for text files is similar to I/O for console, i.e., through the input/output operators >> and <<
- other reading methods:
  - fin.get(); // get a single character
  - fin.getline(char str[], size); // read a line from the file

Internal State Flags

- goodbit: No errors
- eofbit: End-of-file reached, can be queried using eof()
- failbit: Logical error on i/o operation, can be queried using fail()
- badbit: read/write error on i/o operation, can be queried using bad()

fail(): Example I

```
fstream fin("test.txt");
if (fin.fail()) {
      cout << "fail to open "test.txt\n";
      exit(1);
}
```

// when an I/O operation fails, one may call exit() to abort the program execution

// the argument in exit() is returned to the calling party -- usually the OS

// typically, exit(1) is used to abort program when there's an error

fail(): Example II

```
fstream fin("test.txt"); // Assume text.txt contains a line "12345"
if (fin.fail()) {
      cout << "fail to open test.txt\n";
      exit(1);
}
char buf[4];
fin.getline(buf, 4);
if (fin.fail()) {
      cout << "getline failed when reading from test.txt\n";
      exit(1);
}
```

eof(): Example I

```
// count lines in input.txt
// assume no line in input.txt is longer than 3

fstream fin("input.txt");
char buf[4];
int num_of_lines = 0;
while (true) {
      fin.getline(buf, 4);
      if (!fin.eof())
            num_of_lines++;
      else
            break;
}
cout << "input.txt has " << num_of_lines << " lines\n";
```

eof(): Example II

```
// dump the content from input.txt to output.txt
// assuming input.txt contains only integers, e.g., 12345
ifstream fin;
ofstream fout;
fin.open("input.txt");
fout.open("output.txt");
int x;
while (!fin.eof()) {
    fin >> x;
    fout << x << " ";
}
```

✗

```
// dump the content from input.txt to output.txt
// assuming input.txt contains only integers, e.g., 12345
ifstream fin;
ofstream fout;
fin.open("input.txt");
fout.open("output.txt");
int x;
while (fin >> x) {
    fout << x << " ";
}
```

clear()

- Used to reset internal state flags, so that further operations on file stream object can continue
- Run the following program, compare the results with and without fin.clear(), and explain the output

```
fstream fin("input.txt"); // assume input.txt contains 2 lines
                          // line 1: 123456; line 2: 789
char buf[4];
int i = 0;
do {
       fin.getline(buf, 4);
       fin.clear();
       cout << i++ << ": " << buf << "\n";
       getchar(); // used to pause the program
} while (!fin.eof());
```

I/O Re-directions

- A facility offered by many OS
- Allows the program input and output to be redirected from/to specified files
- E.g. suppose you have an executable file hello.exe. If you type:
    - hello > outfile1.dat
- The output is written to the file outfile1.dat instead of the screen
- Similarly, hello < infile1.dat specifies that the input is from infile1.dat instead keyboard