

CS2311 Computer Programming

LT08: File I/O

Computer Science, City University of Hong Kong

Semester B 2022-23

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

`int variableA;`

ifstream

- To **declare** an ifstream 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 ofstream 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.,
`ifstream fin("filename", mode);`
 - Method II: use the member function open with the stream object, e.g.,
`ifstream fin;`
`fin.open("filename", mode);`

File I/O Modes

<code>ios::in</code>	open for input operations
<code>ios::out</code>	open for output operations
<code>ios::binary</code>	open in binary mode
<code>ios::ate</code>	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

<code>ios::app</code>	all output operations are performed at the end of the file, appending the content to the current content of the file
-----------------------	---

<code>ios::trunc</code>	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::app);
```


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 test.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

```
// dump the content from input.txt to output.txt
// assuming input.txt contains only integers

ifstream fin;
ofstream fout;
fin.open("input.txt");
fout.open("output.txt");
int x;
while (!fin.eof()) {
    fin >> x;
    fout << x << " ";
}
```

eof(): Example II

```
// 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";
```

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";
    if(i==3) break; // used to end the program
} while (!fin.eof());
```

0: 123
1:
2:

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";
    if(i==3) break; // used to end the program
} while (!fin.eof());
```

0: 123
1: 456
2: 789

Exercise

- What output will be produced when the following lines are executed, assuming the file list.txt contains the data shown (and assuming the lines are embedded in a complete and correct program with the proper include and using directives)?

1
2
3
3

1 2
3

list.txt

```
ifstream ins;  
ins.open("list.txt");  
int count = 0, next;  
while (ins >> next)  
{  
    count++;  
    cout << next << endl;  
}  
ins.close( );  
cout << count;
```