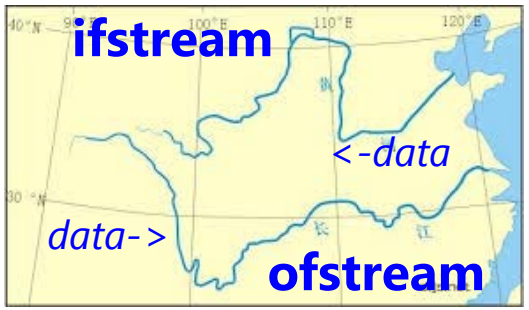
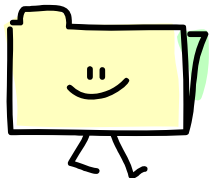
Introduction to Sequential File Processing

Meng-Hsun Tsai
CSIE, NCKU

program





file

write_file.cpp

```
 1 #include <iostream>
 2 #include <string>
 3 #include <fstream>
 4 #include <cstdlib>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    string name;
10    float hw, proj, exam;
11    ofstream outFile("outfile", ios::out);
12    if(!outFile) {
13       cerr << "Failed opening" << endl;
14       exit(1);
15    }
16    cout << "Enter NAME, HW, PROJ,
                 EXAM each line.\n"
17        << "EOF to finish.\n" << "? ";
18    outFile <<
        "Name\tHW\tProj\tExam\tTotal\n";
19
20    while(cin >> name >> hw >> proj
              >> exam) {
21       outFile << name << "\t" << hw
              << "\t" << proj << "\t"
22            << exam << "\t" << hw*0.15 +
              proj*0.5 + exam*0.35 << endl;
23       cout << "? ";
24    }
25    cout << endl;
26    return 0;
27 }
```

```
> ./write_file
Enter NAME, HW,
PROJ, EXAM each line.
EOF to finish.
? Lo 90 57 69
? Lin 82 94 78
? ^D
```

```
> cat outfile
Name    HW    Proj    Exam  Total
Lo      90    57      69    66.15
Lin     82    94      78    86.6
```

Creating a Sequential File

- Two arguments are passed to an ofstream object's constructor—the filename and the file-open mode (line 11).

```
11    ofstream outFile("outfile", ios::out);
```

- Existing files opened with mode `ios::out` are truncated—all data in the file is discarded.
- If the specified file does not yet exist, then the `ofstream` object creates the file, using that filename.

Creating a Sequential File (cont.)

- For an `ofstream` object, the file-open mode can be either `ios::out` to output data to a file or `ios::app` to append data to the end of a file.

Mode	Description
ios::app	Append all output to the end of the file
ios::ate	Open a file for output and move to the end of the file (normally used to append data to a file). Data can be written anywhere in the file.
ios::in	Open a file for input.
ios::out	Open a file for output.
ios::trunc	Discard the file's contents (this also is the default action for ios::out).
ios::binary	Open a file for binary (i.e., nontext) input or output.

Creating a Sequential File (cont.)

- An `ofstream` object can be created without opening a specific file—a file can be attached to the object later.
- For example, the statement
  - `ofstream outFile;`
- creates an `ofstream` object named `outFile`.
- The `ofstream` member function `open` opens a file and attaches it to an existing `ofstream` object as follows:
  - `outFile.open("outfile", ios::out);`

Creating a Sequential File (cont.)

- The `if` statement in lines 12–15 uses the overloaded `ios` member function `operator!` to determine whether the `open` operation succeeded.
- Some possible errors are
  - attempting to open a nonexistent file for reading
  - attempting to open a file for reading or writing without permission
  - opening a file for writing when no disk space is available

```
12    if(!outFile) {
13        cerr << "Failed opening" << endl;
14        exit(1);
15    }
```

Creating a Sequential File (cont.)

- When end-of-file is encountered or bad data is entered, the `while` statement terminates.
- Ctrl-D in Unix and Ctrl-Z in Windows represent end-of-file.

```
20    while(cin >> name >> hw >> proj   >> exam) {
21        outFile << name << "\t" << hw  << "\t" << proj << "\t"
22            << exam << "\t" << hw*0.2 +  proj*0.5 + exam*0.3 << endl;
23        cout << "? ";
24    }
```

```
? ^D
```

Creating a Sequential File (cont.)

- Line 21 writes a set of data to the file `outfile`, using the stream insertion operator `<<` and the `outFile` object associated with the file at the beginning of the program.

```
21      outFile << name << "\t" << hw  << "\t" << proj << "\t"
22           << exam << "\t" << hw*0.2 + proj*0.5 + exam*0.3 << endl;
```

- Once the user enters the end-of-file indicator, `main` terminates.
- This implicitly invokes `outFile`'s destructor, which closes the `outfile` file.
- You also can close the `ofstream` object explicitly, using member function `close` in the statement

read_file.cpp

```
 1 #include <iostream>
 2 #include <string>
 3 #include <fstream>
 4 #include <cstdlib>
 5 using namespace std;
 6
 7 int main()
 8 {
 9     string name, headline;
10     float hw, proj, exam, total;
11     ifstream inFile("infile", ios::in);
12     if(!inFile) {
13         cerr << "Failed opening" << endl;
14         exit(1);
15     }
16
17     getline(inFile, headline);
18     cout << headline << endl;
19     while(inFile >> name >> hw >> proj >> exam
         >> total) {
20       cout << name << "\t" << hw << "\t" << proj
21            << "\t" << exam << "\t" << total << endl;
22     }
23     return 0;
24 }
```

```
> cat infile
Name    HW    Proj    Exam    Total
Lo      90    57      69      67.2
Lin     82    94      78      86.8
```

```
> ./read_file
Name    HW    Proj    Exam    Total
Lo      90    57      69      67.2
Lin     82    94      78      86.8
```

Reading Data from a Sequential File

- Creating an `ifstream` object opens a file for input.
- The `ifstream` constructor can receive the filename and the file open mode as arguments.
- Line 11 creates an `ifstream` object called `inFile` and associates it with the `infile` file.

```
11    ifstream inFile("infile", ios::in);
```

- Objects of class `ifstream` are opened for input by default.
- We could have used the statement

```
ifstream inFile("infile");
```

to open `infile` for input.

Reading Data from a Sequential File (cont.)

- Just as with an `ofstream` object, an `ifstream` object can be created without opening a specific file, because a file can be attached to it later.
- Each time line 19 executes, it reads another record from the file into the variables `name`, `hw` , `proj` , `exam` and `total`.
- When the end of file has been reached, the `ifstream` destructor function closes the file and the program terminates.

```
19   while(inFile >> name >> hw >> proj >> exam >> total) {
20      cout << name << "\t" << hw << "\t" << proj
21           << "\t" << exam << "\t" << total << endl;
22   }
```