

EEE102

C++ Programming and Software Engineering II

Lecture 10 File Operation

Dr. Rui Lin/Dr. Xue Fei

Rui.Lin/Fei.Xue@xjtlu.edu.cn

Room EE512/EE222



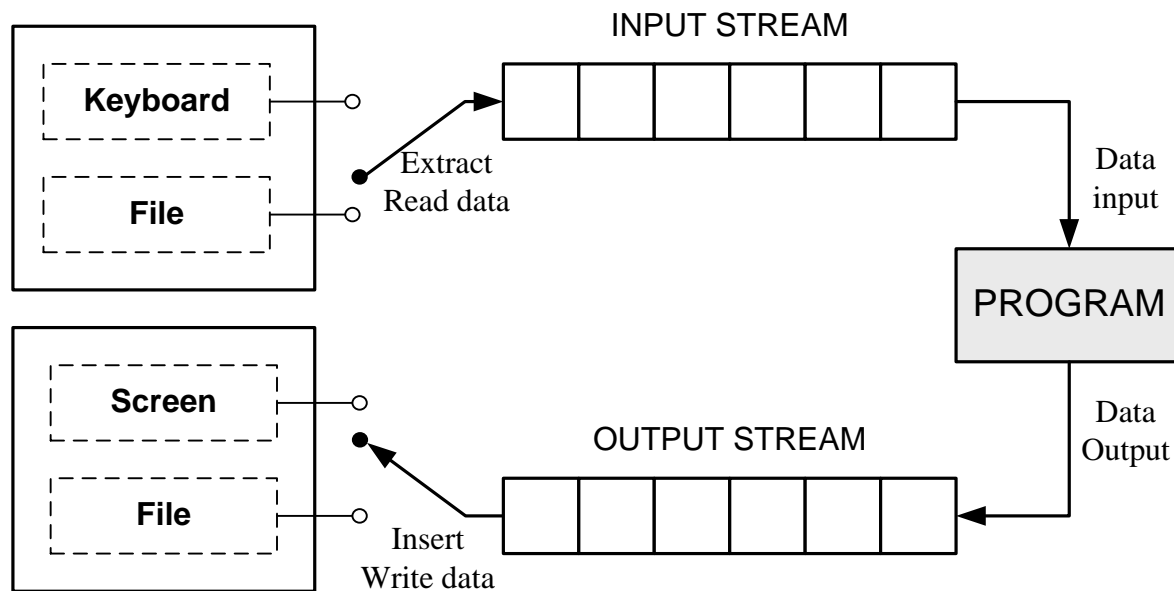
Outline

- File Stream
 - Classes for stream operations
 - Opening and Closing a file
 - Detecting the end-of-file
 - File modes
- Sequential input and output operations
- File Pointers
 - Pointer manipulation functions



1.1 Stream

- The I/O system handles file operations using file streams as the interface between the programs and the files.
 - Input stream: supply data to the program;
 - Output stream: receive data from the program.

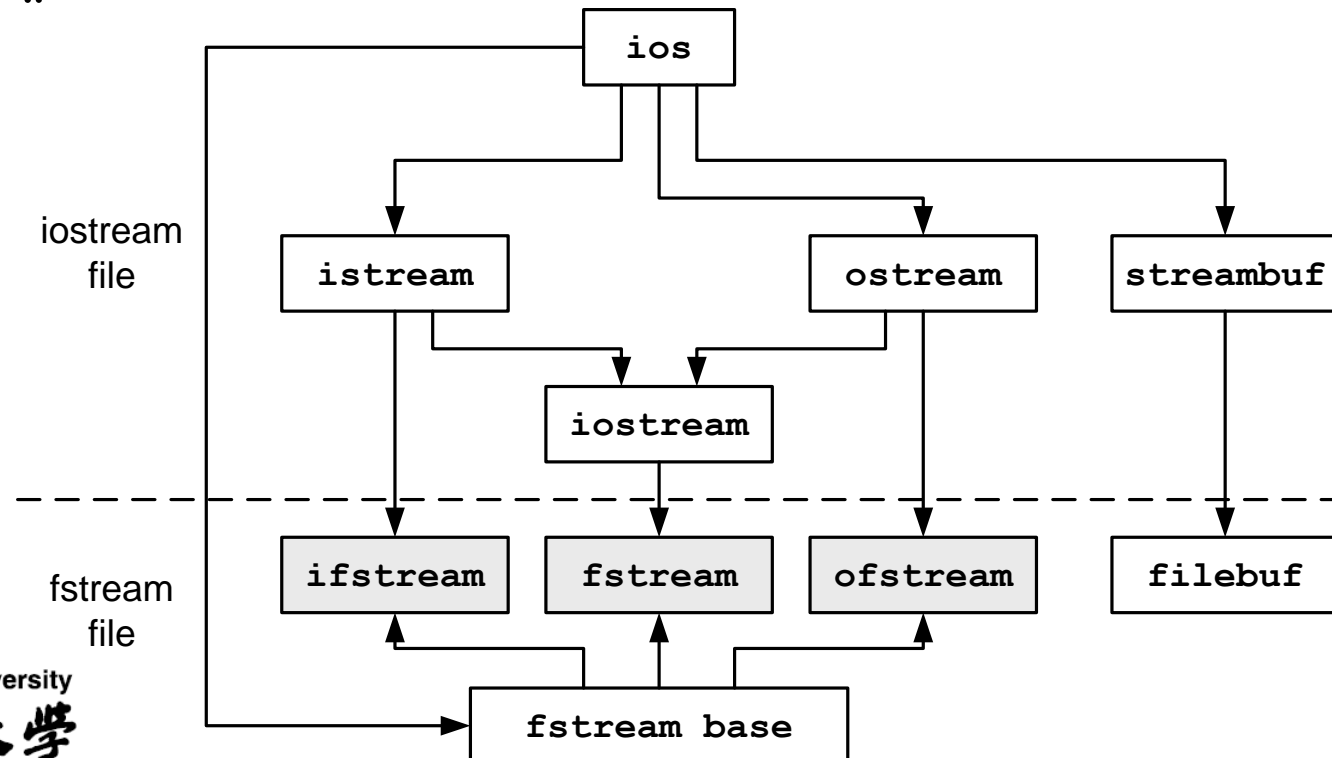


Input stream extracts (read) data from the file and the output stream insert (write) data to the file.



1.2 Classes for stream operations

- The I/O system contains a set of classes that define the file handling methods are declared in “**fstream**”.
 - Therefore, it is always needed to include this file in any program that uses file operation.
 - Syntax: **#include <fstream>**



1.3 Opening and Closing a file

- To open a file
 - First create a file stream => declaration
 - Then link it to the file name => assignment } Initialisation
- A file stream can be defined using the classes **ifstream**, **ofstream** and **fstream**.
- A file can be opened in two ways:
 - 1. Using the constructor function:

```
ifstream infile("data.txt");  
ofstream outfile("newdata.txt");
```
 - 2. Using the method **open()** :

```
ofstream outfile;  
outfile.open("newdata1.txt");  
outfile.open("newdata2.txt");
```



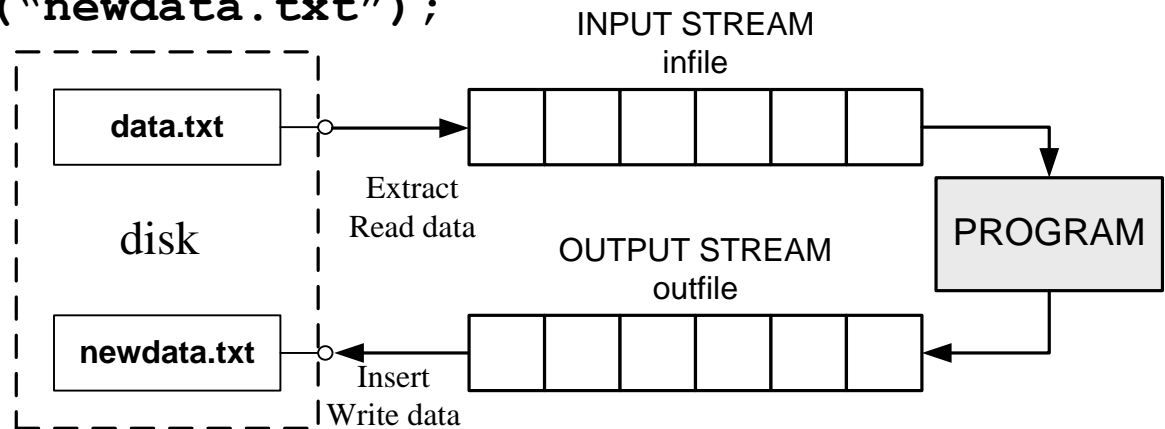
1.3 Opening and Closing a file

- 1. Using the constructor function:

```
ifstream infile("data.txt");  
ofstream outfile("newdata.txt");
```

Case 1:

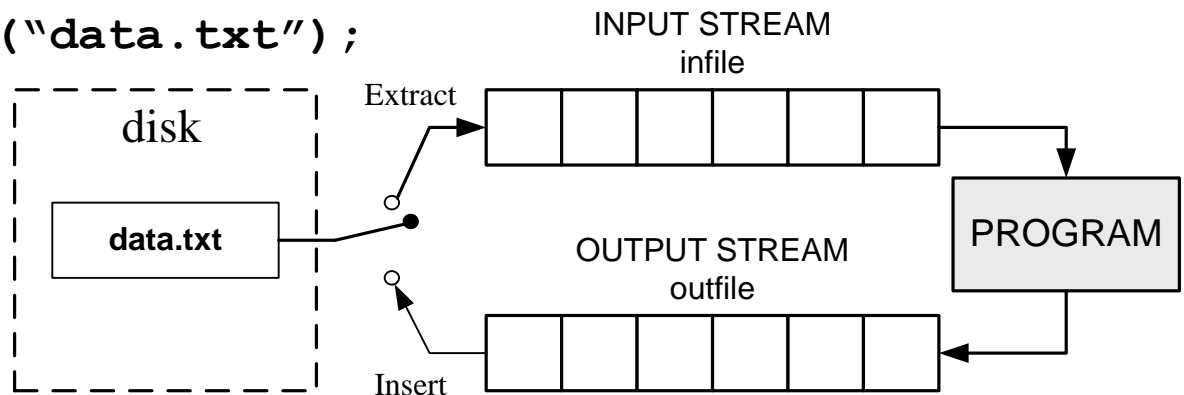
Link to different file



```
ifstream infile("data.txt");  
ofstream outfile("data.txt");
```

Case 2:

Link to the same file



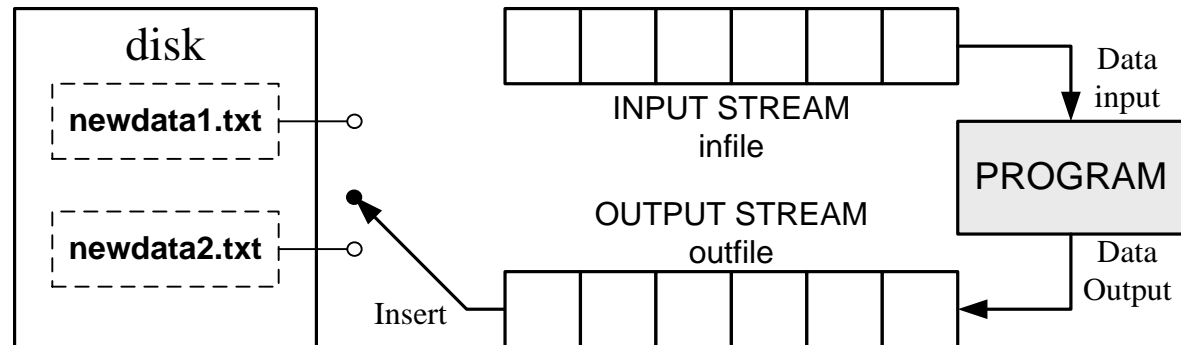
1.3 Opening and Closing a file

- 2. Using the method **open ()** :

```
ofstream fout;  
fout.open("newdata1.txt");  
fout.open("newdata2.txt");
```

Case 3:

Link to same stream



- When a file is opened for writing, a new file is created if there is no file of that name;
- If a file by that name exists already, then its contents are deleted and the file is presented as a clean file.



1.3 Opening and Closing a file

- The connection with a file is closed automatically when the stream object expires (when the program terminates)
- It is invalid to link one file to different stream, or link two files to one stream simultaneously -> disconnect / close the file before reconnection.
 - Example: Case 2 and 3 in previous slides
- To close a file
 - Syntax:

Case2: `ifstream infile("data.txt");`
 `ofstream outfile("data.txt");`

Case3: `ofstream fout;`
 `fout.open("newdata1.txt");`
 `fout.open("newdata2.txt");`



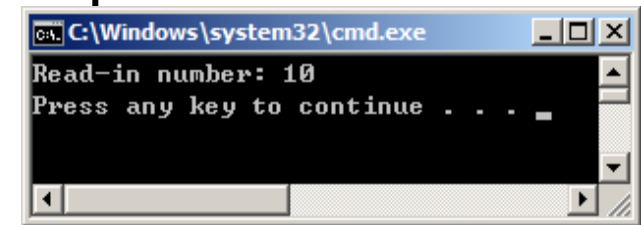
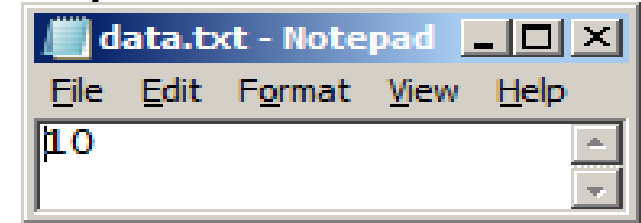
Example of Using Files I/O

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    int num=10,result;
    ofstream outfile("data.txt");
    outfile<<num<<endl;
    outfile.close();

    ifstream infile("data.txt");
    infile>>result;
    cout<<"Read-in number: "<<result<<endl;
    infile.close();

    return 0;
}
```



D: \ \CppCode \ \data.txt

*Use escape sequence
for the complete path
of the file name*



1.4 Detecting the end-of-file

- Detection of the end-of-file condition is necessary for preventing any further attempt to read data from the file.
- **eof()** is a member function of **ios** class, which returns a non-zero value if the end-of-file (EOF) condition is encountered.

```
ifstream fin("data.txt");  
fin.eof()          % !=0, not EOF  
fin.eof()          % ==0, EOF
```

- Use the following statement to terminate the program on reaching the end of the file

```
if(fin.eof() !=0)  
{ exit(1); }
```



1.5 More about `open ()` : file modes

- When opening the file (connecting the file to an stream object), use second argument to specify the file mode:
 - Syntax:
`ifstream fin("data.txt",mode);`
 - The prototype functions contain default values as:
`ios::in` for `ifstream` (open for read-only)
`ios::out` for `ofstream` (open for write-only)
 - More file mode parameters:
`ios::app` Append to end-of-file
`ios::ate` Go to end-of-file on opening
`ios::trunc` Delete the contents of the file if exist
`ios::binary` Binary file
 - The mode can combine, such as:
`fout("data.txt",ios::app|ios::binary);`



2. Sequential input and output operations

- The stream classes support a number of member functions for performing the input and output operations on files.

- Extraction and insertion symbol:

- `<<` and `>>`

- Example:

```
outfile <<num <<endl;  
infile >>result;
```

- Single character operation:

- `put()` and `get()`

- Example:

```
outfile.put('A');  
infile.get(ch);  
infile.getline(cstr,20);
```



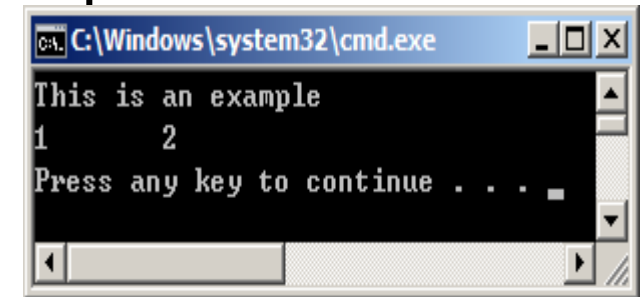
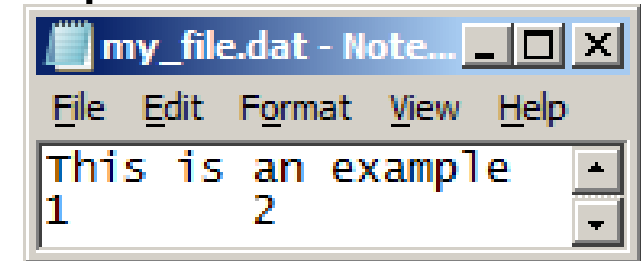
Example 2 of Using Files I/O

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    char line[20];
    int m1, m2;
    ofstream fout("my_file.dat");
    fout<<"This is an example"<<endl;
    fout<<1<< "\t" <<2<<endl;
    fout.close();

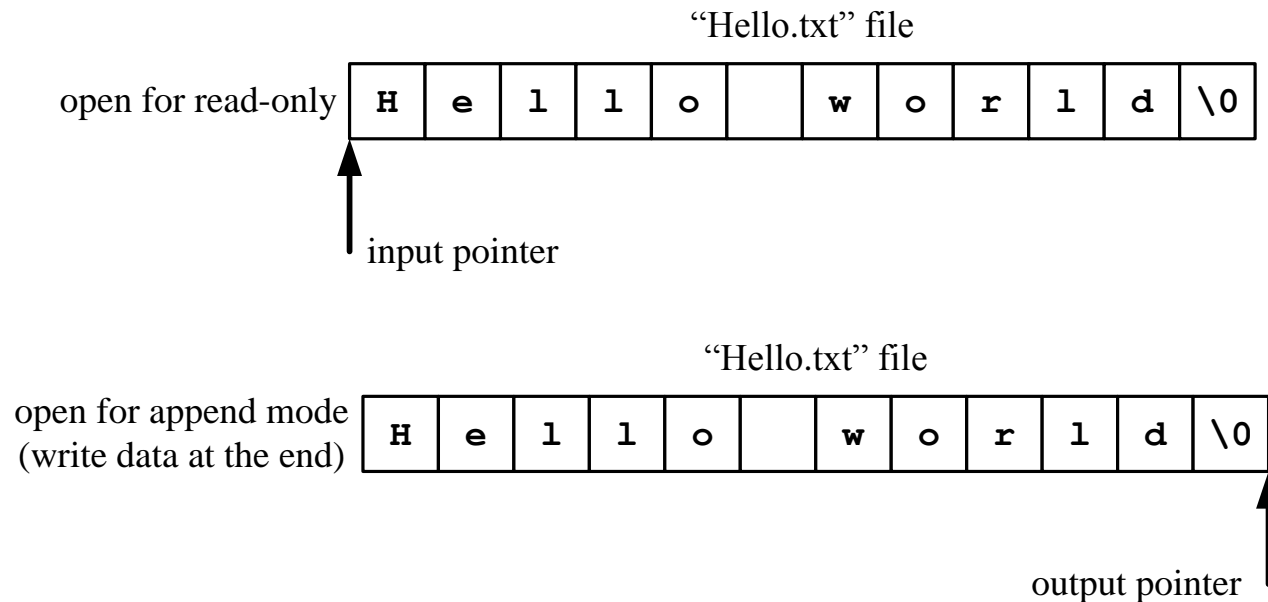
    ifstream fin("my_file.dat");
    fin.getline(line,20);
    fin>>m1>>m2;
    cout<<line<<endl;
    cout<<m1<<"\t" <<m2<<endl;
    fin.close();

    return 0;
}
```



3.1 File Pointers

- Each file has two associated pointers: input pointer (or get pointer), and the output pointer (or put pointer).
 - get pointer: for reading the content of a given file location
 - put pointer: for writing to a given file location



3.2 Pointer manipulation functions

- The file stream classes support the following functions to manage the pointer:
 - **seekg()** : moves get pointer (input) to a specified location
 - **seekp()** : moves put pointer (output) to a specified location
 - **tellg()** : gives the current position of the get pointer (input)
 - **tellp()** : gives the current position of the put pointer (output)
- These functions take two arguments:
 - **offset**: number of bytes
 - **refposition**: reference position
 - **ios::beg**, **ios::cur**, **ios::end**
 - Example:

```
fout.seekg(0,ios::beg); // Go to beginning of the file
```

```
fout.seekg(N,ios::cur); // Go forward by N byte from the current position
```

```
four.seekg(-1,ios::end); // Go backward by 1 byte from the end
```

