Introduction to Std C++ File I/O

C++ Object Oriented Programming

Pei-yih Ting

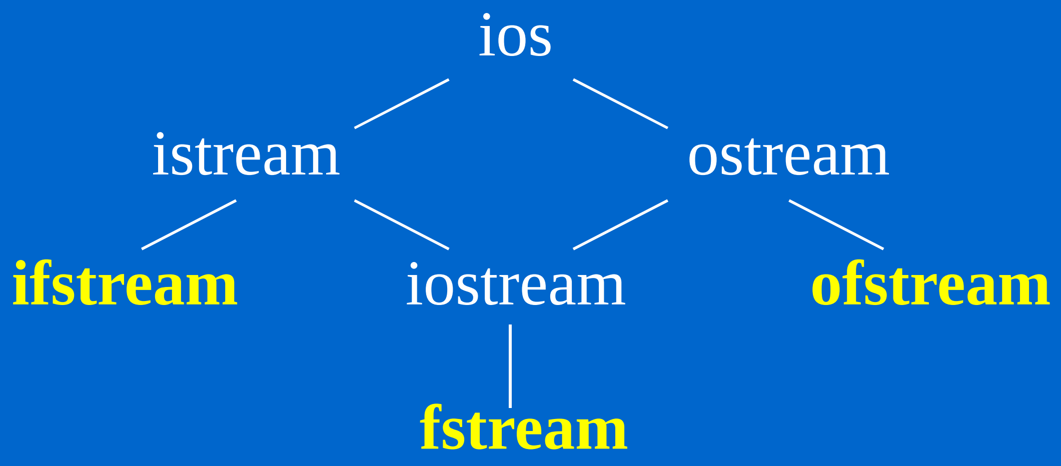
NTOU CS

Contents

Class Hierarchy

✧ File classes are *inherited* from console classes

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



✧ Why inheritance?

- ✶ All operations for the console classes are available in exactly the same form for file processing
- ✶ More device-independent than their counterparts in C

✧ Formatted and unformatted I/O

- ✶ Console data is always in formatted form, i.e. ASCII printable integers, strings, floats…
- ✶ File I/O can be formatted or unformatted (raw bytes)

Basic File I/O Operations

- Reading characters from a file and printing to the screen

```
char cBuf;
ifstream myFile("testFile");  // open the file implicitly
if (!myFile) {  // check for correct opening or !myFile.is_open()
    cerr << "File can't be opened";
    return;
}
while (myFile.get(cBuf)) cout << cBuf;
```

& operator not required

  - operator ! is overloaded in class **ios** to return false if the failbit or badbit has been set after attempting to open the file
  - get() will return false when EOF is reached, otherwise it will return the file stream object

- Explicitly open or close of a file if you plan to reuse the ifstream obj

```
ifstream myFile;
myfile.open("testFile");
…
myFile.close();  // this will also be invoked in inherited destructor
```

Basic File I/O Operations (cont’d)

- Writing characters to a file

```
ofstream myFile("testFile");  // creates the file with this name
char *string = "test output string";
if (!myFile) {
    cerr << "File can’t be created\n";
    return;
}
for (i=0; i<strlen(string); i++)
    myFile.put(string[i]);
```

  - You could also put a letter to the console window: cout.put('A');

- File modes:

```
ios::out            // open the file and erase the contents, default
   ofstream myFile("testFile", ios::out);
ios::app            // append data to the end of the file
ios::nocreate       // open fails if the file doesn’t exist
ios::noreplace      // open fails if the file exists
```

Insertion and Extraction operators

- File objects have the same interface as console objects: >>, <<

```
int number1 = 10;
int number2 = 20;
int number3 = 30;
ofstream myFile("numberData.txt");
if (!myFile) {
    cerr << "File can’t be created\n";
    return;
}
myFile << number1 << '  ' << number2 << '  ' << number3 << endl;
```

Output is a text file:

```
10 20 30
```

  - << and >> are for formatted I/O, the codes converts the internal formats of the built-in types to printed characters

```
int number,
ifstream myFile("numberData.txt");
while (myFile >> number)
    cout << number;
```

  - The operator << of ifstream class will return false when EOF is reached

Unformatted File I/O

- Unformatted files store data as raw bytes
- Using member functions read() and write()

```
int array[SIZE], newArray[SIZE];
ofstream outputFile("binaryData.dat", "std::ofstream::binary");
if (!outputFile) {
    cerr << "File can’t be created\n";
    return;
}
for (i=0; i<SIZE; i++) array[i] = i;
outputFile.write((char *)array, sizeof(int)*SIZE);
outputFile.close();
ifstream inputFile("binaryData.dat", "std::ifstream::binary");
if (!inputFile) {
    cerr << "File can’t be opened\n";
    return;
}
inputFile.read((char *)newArray, sizeof(int)*SIZE);
for (i=0; i<SIZE; i++) cout << newArray[i];
```

Random Access Files

- Simultaneous input and output   ios::in | ios::out
- Absolute file positioning

```
seekg(offset)   // seek get, used with input streams, relative to file beginning
seekp(offset)   // seek put, used with output streams
```

- Relative file positioning functions

```
seekg(offset, ios::beg)
seekg(offset, ios::cur)
seekg(offset, ios::end)  // offset must be negative
seekp(offset, ios::beg)
seekp(offset, ios::cur)
seekp(offset, ios::end)
```

- tellg() returns the current file position as a long integer

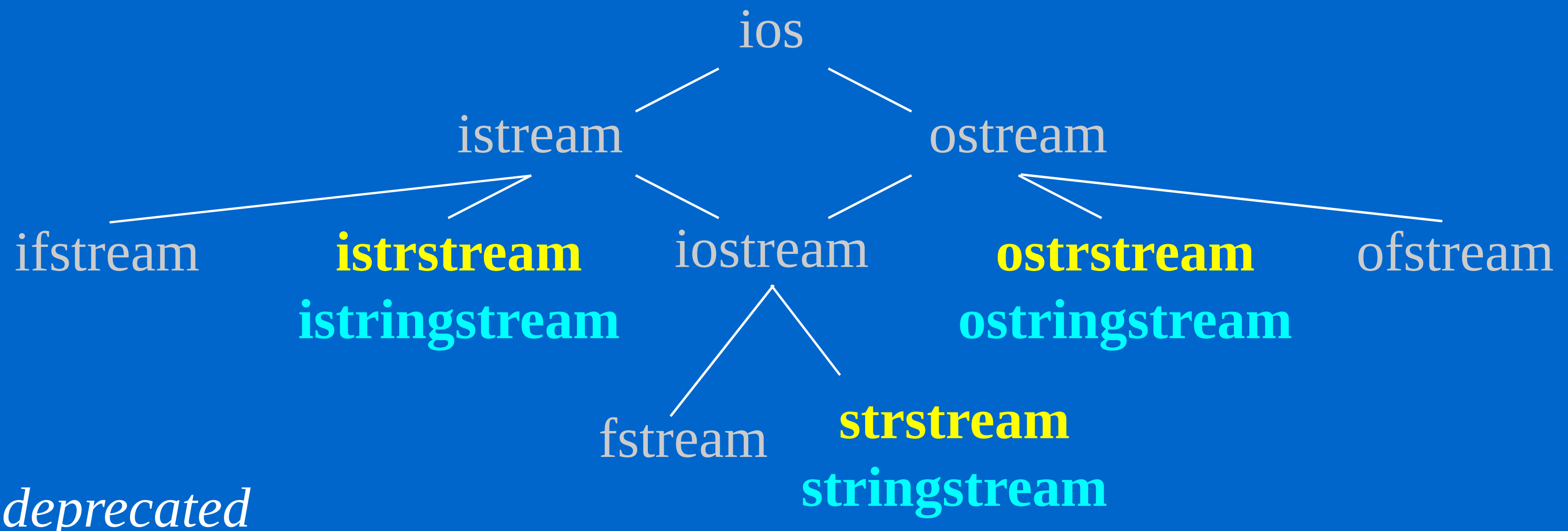
Using Random Access File

- Ex.

```
int data[SIZE];
fstream fileSteam("data.dat", ios::in | ios::out);
if (!fileStream) {
    cerr << "File can’t be opened\n";
    return;
}
for (i=0; i<100; i++)
   fileStream.write((char *)data, sizeof(data));
…
index = 70;
fileStream.seekp(sizeof(data)*index);
fileStream.write((char *)data, sizeof(data));
…
index = 20;
fileStream.seekg(sizeof(data)*index);
inputFile.read((char *)newArray, sizeof(int)*SIZE);
```

String Stream Processing

- Counterparts of sscanf(), sprintf() in stdio library
  - Take advantage of the console formatting library to construct strings



deprecated

```
#include <strstream>
using namespace std;
```

```
#include <sstream>
using namespace std;
```

ostrstream

- Create a simple formatted string

```
ostrstream outputStringStream;
char        *result;
outputStringStream.precision(18);
outputStringStream << "The value of pi to a precision of 18 is " << pi << ends;
result = outputStringStream.str();
cout << result;
outputStringStream.freeze(false);
```

```
Output on the console is:
The value of pi to a precision of 18 is 3.14159265358979324
```

  - The manipulator **ends** inserts the null terminator
  - The address of the internal buffer is returned by str()
  - Once str() is invoked, no additional data can be added (the buffer is frozen)
  - The client program owns the buffer and is responsible for deleting the buffer
  - The client program should call freeze(false) after str() is called to unfreeze the buffer

ostrstream (cont’d)

- The following usage causes an error

```
result = outputStringStream.str();  // buffer frozen
outputStringStream << "more data";
if (outputStringStream.fail())  // This will be true
    cout << "failure";
```

String not properly terminated

- The data is dynamically allocated within the ostrstream object.
- ostrstream has a second overloaded constructor whereby the client supplies a fixed-size character array to be used as the buffer.

```
const int cSize=12;
char buffer[cSize], *result;
ostrstream outputStringStream(buffer, cSize);
outputStringStream.precision(18);
outputStringStream << "The value of pi to a precision of 18 is " << pi << ends;
result = outputStringStream.str();
cout << result << "\n[" << result[11] << "]\n";
if (outputStringStream.fail()) cout << "failure"; // failbit will be set
```

```
Output:
The value of ¶ú


[f]
failure
```

istrstream

- An istrstream object contains a character array from which formatted data can be extracted
- Ex.

```
const int cBufSize = 100;
const int cStrSize = 50;
void main()
{
    char buffer[cBufSize] = "pi is 3.14159";
    istrstream inputStream(buffer, cBufSize);
    char string1[cStrSize], string2[cStrSize];
    double value;
    inputStream >> string1 >> string2 >> value;
    cout << string1 << ' ' << string2 << ' ' << value;
}
```

  - Note: istrstream’s failbit is NOT turned on till the end of the buffer in VC6.

    The null character in the buffer does not terminate the stream.

ostringstream

- #include <sstream>
  using namespace std;
- The internal implementation is string object instead of C char array

```
#include <string>          // std::string
#include <iostream>      // std::cout
#include <sstream>       // std::stringstream, std::stringbuf
void main () {
    std::stringstream ss;
    ss.str("123456789x123456789y123456789z1234567890");
                                        // initialize the buffer of the stringstream
    ss << "Overwriting the initial string"; // no need for ends
    std::string s = ss.str(); // obtaining a copy of the internal buffer of stringstream
    std::cout << s << '\n';
}
```

```
Output:
Overwriting the initial string1234567890
```

istringstream

- #include <sstream>
  using namespace std;
- Example

```
#include <string>          // std::string
#include <iostream>      // std::cout
#include <sstream>       // std::istringstream
void main () {
    std::istringstream iss;
    std::string strvalues = "32 240 2 1450";
    iss.str(strvalues);
    for (int n=0; n<4; n++) {
        int val;
        iss >> val;
        std::cout << val << '\n';
    }
    std::cout << "Finished writing the numbers in: ";
    std::cout << iss.str() << '\n';
}
```

```
32
240
2
1450
Finished writing the numbers in: 32 240 2 1450
```

User-defined types

- Overload the << and >> operators for a class
- Ex. Overloaded operators for CComplex

```
ostream &operator<<(ostream &os, CComplex number) {
    os << number.m_real << "+" << number.m_imaginary << "i";
    return os;
}
istream &operator>>(istream &is, CComplex &number) {
    char dummy;
    is >> number.m_real >> dummy >> number.m_imaginary >> dummy;
    return is;
}
…
CComplex number(-5, -2);
ofstream outputFile("outputFile.txt");
outputFile << number;
```

Note: An ofstream object is a fstream object.