COMP 2012H Honors Object-Oriented Programming and Data Structures

Self-study: File I/O

Dr. Desmond Tsoi

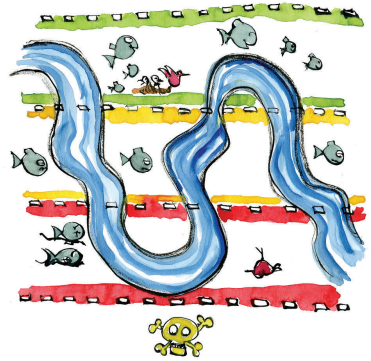
Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China

C++ Stream

Up to now, you only know how to interactively

- input data from the keyboard using cin $\gg$
- output data to the screen using cout $\ll$

- In general, C++ allows you to input/output data to/from files and devices (e.g. printer, hard disk, USB memory stick) using an abstraction called stream.



C++ Stream

C++ Stream

A stream is simply a sequence of characters.

- The data transferred between a C++ program and a file/device are modeled as a stream of characters, *regardless* of the data type (basic types: int, float, etc.; user-defined types: btree, linked_list, etc.).
- A device can also be treated like a file. In the following, when we say file, we mean both file and device.

Stream I/O Operations $\gg$, $\ll$

- To perform I/O, create a stream object (from various stream classes) for each file.
- These stream objects all support the 2 basic input/output operators: $\gg$, $\ll$.
  - Both $\gg$ and $\ll$ are implemented so that they convert input/output data of the required type from/to a sequence of characters.
  - The input operator $\gg$ always skip whitespaces — spaces, tabs, newlines, formfeeds, carriage returns — before reading the next datum.

Common Stream Member Functions

The stream objects of various stream classes also support the following class member functions:

- open(const char* filename): create a stream and associate it with a file with the given filename.
- close( ): close a stream created by open( ).
- eof( ): check if the end of a file is reached.
- get(char& c): get the next character into the variable c from an input stream.
- getline(char* s, int max-num-char, char terminator='\n'): get a stream of characters and put it into the char array pointed by the variable s. getline( ) stops when either
  - (max-num-char - 1) characters are read; or,
  - the *stopping* character terminator ('\n' by default) is seen.

  Notice that the stopping character is not read into the array, and the null character is automatically inserted at the end of s.
- put(char c): put the character represented by the variable c onto an output stream.

Interactive Stream: iostream

Stream Type	Class Name	Header Filename
input stream	istream	istream
output stream	ostream	ostream

- The header file "iostream" combines the 2 header files "istream" and "ostream".
- C++ already creates the following istream/ostream objects for you:

  istream cin : standard (or console) input, by default, is the keyboard.

  ostream cout : standard (or console) output, by default, is the screen.

  ostream cerr : standard (or console) error output, by default, is the screen. From now on, you should send your error messages to cerr instead of cout.

File Stream: fstream

Stream Type	Class Name
input file stream	ifstream
output file stream	ofstream

- The header file "fstream" contains the definitions of 2 classes: "ifstream" and "ofstream".
- The input file must exist before you create an input file stream for it.
- If the output file doesn't exist when you create its output file stream, it will be created for you. If it already exists, its content will be erased, and overwritten by the new output data.

Open and Close a File Stream

- Create an input file stream from a file called "input.txt" and an output file stream associated with a file called "output.txt" by one of the following 2 ways:

Example: Open a File Stream

```
#include <fstream>
// [1] Use a special form of ifstream/ofstream constructor
ifstream ifs("input.txt");
ofstream ofs("output.txt");

// [2] Use the default form of ifstream/ofstream constructor,
//     and then their open() member function
ifstream ifs; ifs.open("input.txt");
ofstream ofs; ofs.open("output.txt");
```

- Close a file stream by

Example: Close a File Stream

```
ifs.close();
ofs.close();
```

Example: File Copy

```cpp
#include <iostream>     /* File: filecopy.cpp */
#include <fstream>
using namespace std;
int main()
{
    char ip_file[32], op_file[32]; // Input and output filename

    cout << "Enter the input filename: "; cin >> ip_file;
    ifstream ifs(ip_file); // One way to create a fstream object
    if (!ifs)
    { cerr << "Error: Can't open \"" << ip_file << "\"\n"; return -1; }

    cout << "Enter the output filename: "; cin >> op_file;
    ofstream ofs; ofs.open(op_file); // Another way to create a fstream object
    if (!ofs)
    { cerr << "Error: Can't open \"" << op_file << "\"\n"; return -1; }

    char c; ifs.get(c); // Try to get the first char
    while (!ifs.eof())  // Check if EOF is reached
    {
        ofs.put(c);     // Copy one char at a time
        ifs.get(c);     // Try to get the next char
    }
    ifs.close(); ofs.close(); return 0; // Close input/output file streams
}
```


Example: Read an Array of Integers I

```cpp
#include <iostream>     /* File: read-int-array.cpp */
#include <fstream>
using namespace std;

/* Expected input file format:
 * array size on the first line, followed by the array elements.
 */
int main()
{
    const int MAX_SIZE = 128;
    int x[MAX_SIZE];    // An integer array
    char ip_file[32];   // Input filename

    // Open the file to read
    cout << "Enter the input filename: "; cin >> ip_file;
    ifstream ifs(ip_file); // One way to create a fstream object

    if (!ifs)
    { cerr << "Error: Can't open \"" << ip_file << "\"\n"; return -1; }
```


Example: Read an Array of Integers II

```cpp
    int array_size; ifs >> array_size;
    if (array_size > MAX_SIZE)
    {
        cerr << "Error: array size (" << array_size
             << ") > max size of array (" << MAX_SIZE << ")\n";
        return -1;
    }

    // Read in the array
    for (int j = 0; j < array_size; j++)
        ifs >> x[j];

    // Print the array to screen
    for (int j = 0; j < array_size; j++)
        cout << x[j] << endl;

    ifs.close( ); // Close input file stream
    return 0;
}
```


Further Reading: Binary File I/O

Binary File

- A binary file can be conceptually thought of as an array of bytes.
- If there are k bytes in the binary file, the bytes are indexed from 0 to k - 1.
- As for text file, binary file can be opened with either constructor or open member function, except you need to specify the open mode.

```
ifstream inFile("data.bin", ios::in | ios::binary);
ofstream outFile("dataout.bin", ios::out | ios::binary);
fstream inOutFile("data.bin", ios::in | ios::out | ios::binary);
```

- Some common open modes:
  - ios::app - open and write appends to file
  - ios::ate - at the end
  - ios::binary - I/O in binary mode instead of text
  - ios::in - open for reading
  - ios::out - open for writing
  - ios::trunc - eliminate contents when open

File Pointers

- File pointers are positions in a file for reading and writing.
- Get pointer is used for reading: points to the next byte to read.
- Put pointer is used for writing: points to the next byte location for writing.
- Both are usable at the same time only if working with fstream open for read/write.
- Get and put pointers are independent in the sense that they don't have to point to the same place in the file.
- However, moving one invalidates the other.

Useful File Pointers Functions

- Get pointer
  - tellg: find position of get pointer.
  - seekg(streampos pos): move get pointer to an absolute location indicated by pos, where pos is a long integer.
  - seekg(streamoff offset, ios::seek_dir loc): move get pointer to a relative location indicated by offset from loc, where offset is a long integer.
- Put pointer
  - tellp: find position of put pointer.
  - seekp(streampos pos): move put pointer to an absolute location indicated by pos, where pos is a long integer.
  - seekp(streamoff offset, ios::seek_dir loc): move put pointer to a relative location indicated by offset from loc, where offset is a long integer.

Reading and Writing Data

- Data can be read from binary file using read member function of ifstream object.
  - read(char* target, int num): Read num bytes from the file stream into the storage pointed to by target.
- Data can be written to binary file using write member function of ostream object.
  - write(char* source, int num): Write num consecutive bytes to the current position in output stream starting with the byte located in memory at source.

Example: Binary File Copy I

```
#include <iostream> /* File: filecopy.cpp */
#include <fstream>
using namespace std;

int main()
{
    char ip_file[32], op_file[32];
    cout << "Enter the input filename: "; cin >> ip_file;
    ifstream ifs(ip_file, ios::in | ios::binary);
    if (!ifs)
    { cerr << "Error: Can't open \"" << ip_file << "\"\n"; return -1; }

    cout << "Enter the output filename: "; cin >> op_file;
    ofstream ofs; ofs.open(op_file, ios::out | ios::binary);
    if (!ofs)
    { cerr << "Error: Can't open \"" << op_file << "\"\n"; return -1; }

    // Set the get pointer to 0 offset relative to the end, i.e.,
    // set the get pointer to the end of the binary file
    ifs.seekg(0, ios::end);

    // Find the position of the get pointer, which is equivalent to
    // the size of the file (in byte)
    int size = ifs.tellg();
```


Example: Binary File Copy II

```
    // Set the get pointer to 0 offset relative to the beginning, i.e.,
    // set the get pointer to the beginning of the binary file
    ifs.seekg(0);

    char* buffer = new char[size]; // Allocate memory for the buffer
    ifs.read(buffer, size);        // Read data
    ofs.write(buffer, size);       // Write data
    delete [] buffer;              // De-allocate memory

    ifs.close(); ofs.close(); return 0;
}
```


That's all!

Any questions?