CS2311 Computer Programming

LT11: File I/O

Outline

- Stream
- Open File
- File I/O
- Error Handling

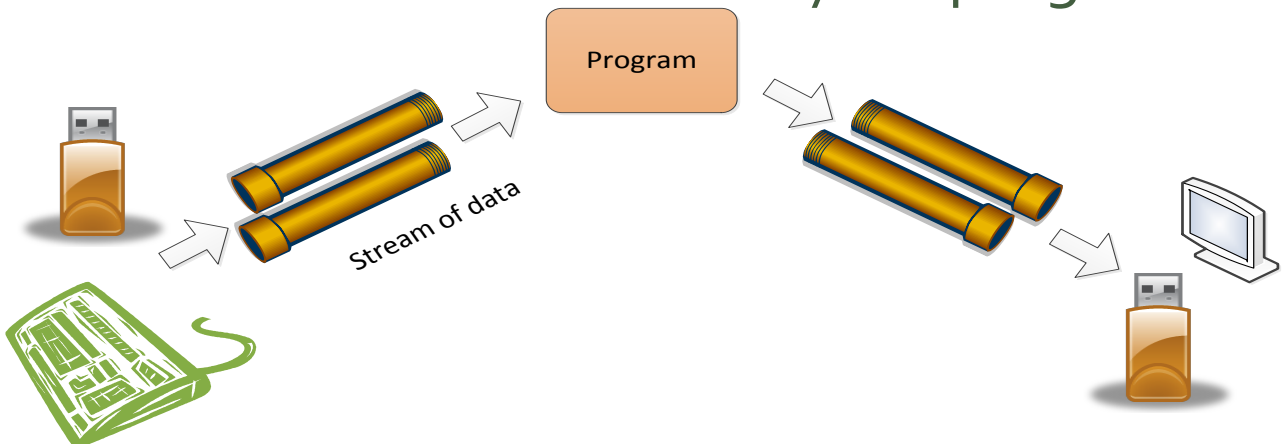
File I/O vs. Console I/O

- "**Console**" refers to "**keyboard** + **screen**"
- Keyboard input and screen output are volatile
- Input file can be used again and again

- Useful for debugging especially when volume of data is huge
- Allow off-line processing
- **Output file** retains the results after execution

Basic I/O – Keyboard and Screen

- Program read input from keyboard (**console**) or disk storage (**file**) and write data to screen (console) or disk storage(file)
- Sequence of inputs is conceptually treated as an object called "**`Stream`**"
- Stream – a flow (sequence) of data
- Input stream – a flow of data into your program
- Output stream – a flow of data out of your program



Streams

- Predefined console streams in C++

```
#include <iostream>
```

`cin` : input stream physically linked to the keyboard

`cout` : output stream physically linked to the screen

- File streams class in C++

```
#include <fstream>
```

`ifstream` : stream **class** for file input

`ofstream` : stream **class** for file output

- To declare an objects of class `ifstream` or `ofstream`, use

```
ifstream fin;    // fin is the variable name
ofstream fout;   // fout is the variable name
```

ifstream

- To declare an **ifsteam** object

```
ifstream fin;
```

- To open a file for reading

```
fin.open("infile.dat");
```

- To read the file content

```
fin >> x; //x is a variable
```

- To close the file

```
fin.close();
```

ofstream

- To declare an **ofsteam** object

```
ofstream fout;
```

- To open a file for writing

```
fout.open("myfile.dat") ;
```

- 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 within a program by a stream
  - (i.e. an object of **`ifstream`** or **`ofstream`**), and
  - any I/O performed on this stream object will be applied to the file associated with it.
- To open a file, use the member function **open** with the stream object: **`open(filename, mode);`**

  `filename` : **a string**

  `mode` : **optional**

Modes for File I/O

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	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.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

Example:

```
ofstream    fout;
fout.open(image_filename, ios::binary);
```

Text File

- 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 **`<<`**
- Read:
  - **`fin.get()`**: get a single character
  - **`fin.getline(char str[],int size)`**: read the file line by line

Detecting I/O Failures

- Member function **`fail()`** returns **`true`** if and only if the previous I/O operation on that stream fails
  - E.g. file not exists when opening an input stream
  - PS: one may call function **`exit()`** when an I/O operation fails to abort the program execution.
    - the argument in **`exit()`** is returned to the calling party – usually the OS
- Important to check I/O failures in I/O operations

Example

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

int main() {
   ifstream in1, in2;
   in1.open("infile1.dat");
   in2.open("infile2.dat");
   if (in1.fail()) {
     cout << "Input file 1 opening failed." << endl;
     exit(1);     // 1 stands for error
   }
  ...
  return 0;
}
```

Detecting End-of-File (`eof`)

- Member function **`eof`** returns **`true`** if and only if we reach the end of the input file (no more data)
  - Only for objects of class **`ifstream`**

    e.g.

```
fin >> x;
if (!fin.eof()) ...
```

- The expression **`fin >> x`** has value **`0`** if **`fin`** has no more data

  e.g.

```
while (fin >> x){...}
```

Example: (eof)

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   ifstream fin;
   ofstream fout;
   int x;
   fin.open("input.txt");
   fout.open("output.txt");
   fin >> x;
   while (!fin.eof()) {
      fout << x << " ";
      fin >> x;
   }
   fin.close();
   fout.close();
   return 0;
}
```

Examples: File Dump (integer only)

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

int main() {
   ifstream fin;
   ofstream fout;
   int x;
   fin.open("input.txt");
   fout.open("output.txt");
   if (!fin.fail() && !fout.fail())
      while (fin >> x)
         fout << x << " ";
   fin.close();
   fout.close();
   return 0;
}
```

return 0 if fin has no more data

Binary Files

- When **`ios::binary`** is set, all I/O is performed on a **binary** basis.
  - The byte values are directly used independent of formatting considerations.
- Input/output operators may not be efficient for binary files, unless you want to read/write in text format.
  - Specific functions can be used.

```
fin.read(char* target, int num)
fout.write(const char* source, int num)
```

Binary Files

```
fin.read(char* target, int num)
```

- **target**
  - a pointer to **char** (1 byte). It represents the address of an array of bytes (a memory block) where the data can be stored.
- **num**
  - the integer size of the memory block (how many bytes)

An Example of Binary File I/O

```
// Create a binary file then read from it
#include <iostream>
#include <fstream>
using namespace std;
const int LineLen = 128;
int main() {
    char ibuffer[LineLen];
    cin.getline(ibuffer, LineLen);
    ofstream outfile("mydata.bin", ios::out | ios::binary);
    outfile.write(ibuffer, LineLen);    // write to "data.bin"
    outfile.close();

    ifstream infile("mydata.bin", ios::in | ios::binary);
    infile.read(ibuffer, LineLen);  // read from "data.bin"
    if (!infile) {
        cerr << "Error reading from ""mydata.bin""; only " << infile.gcount()
            << " bytes read. " << endl;
        infile.clear();
    }
    else
        cout << ibuffer << endl;
    infile.close();
    return 0;
}
```

Error reading from **mydata.bin**, only **0** bytes read.

Reference Only: I/O Re-directions

- A facility offered by many OS's
- Allows the program input and output to be redirected from/to specified files
  - e.g. suppose you have an executable file **`hello.exe`**. If you type:

    ```
    hello > outfile1.dat
    ```

  - in the **MSDOS** prompt, the output is written to the file **`outfile1.dat`** instead of the screen
- Similarly, **`hello < infile1.dat`** specifies that the input is from **`infile1.dat`** instead keyboard