fit@hcmus

PROGRAMMING TECHNIQUES

Week 10: Binary Files

Nguyễn Hải Minh - 04/2024

WORKING WITH BINARY FILES

Introduction

- Computer programs are associated to work with files as it helps in storing data & information permanently.
- File – itself a bunch of bytes stored on some storage devices.
- In C++ this is achieved through a component header file called `fstream`
- The I/O library manages two aspects: as interface and for transfer of data.
- The library predefine a set of operations for all file related handling through certain classes.

The fstream header file

- A stream is a general term used to name flow of data.

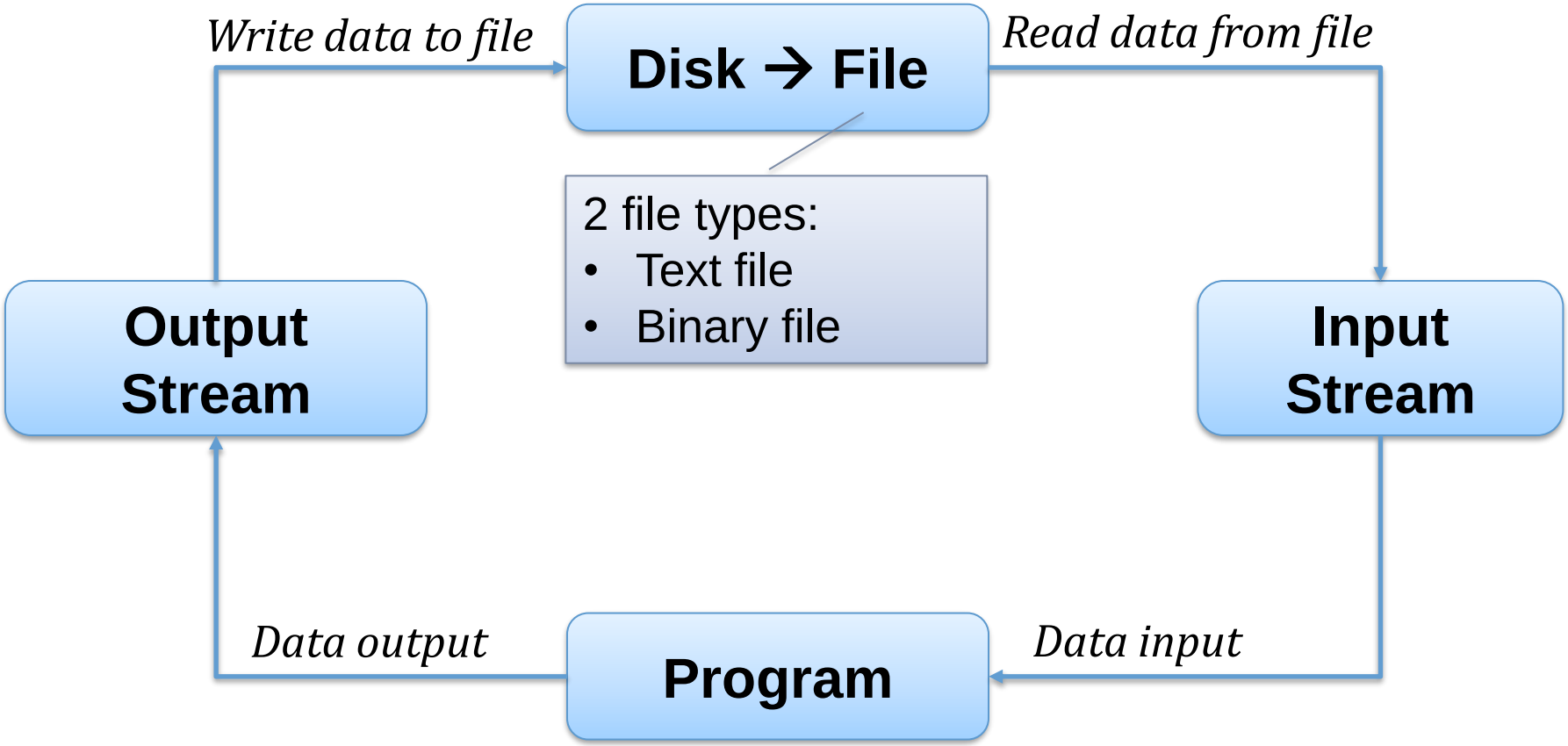
  → *A stream is sequence of bytes.*

- Streams act as an interface between files and programs.

  → *They represent as a sequence of bytes and deals with the flow of data.*

- Every stream is associated with a class having member functions and operations for a particular kind of data flow.
  - File → Program (Input stream) – read
  - Program → File (Output stream) – write
- All designed into `fstream` and hence needs to be included in all file handling programs.
- Diagrammatically as shown in next slide

Introduction



File Types

- Text Files: store information in ASCII characters. In text files, each line of text is terminated by with special character known as EOL (End of Line)

→ some translations takes place when this EOL character is read or written.

- Binary Files: contain the information in the same format as it is held in the memory. In binary files there is no delimiter for a line (No EOL)

→ NO translation occurs in binary files. As a result binary files are faster and easier for program to read and write.

File Types

	Text File	Binary File
Handling of new lines	Various character translations are performed (e.g., \r \n) while reading from a file and vice versa while writing	No such translations
Portability	Portable: one can easily transfer text file from one computer to the other	Not Portable: Binary files are dependent. If the new computer uses a different internal representation for values they cannot be transferred
Storage of numbers	As characters: 429876 coccupies 7 bytes	429876 is stored in 4 byes

File Types

	Text File	Binary File
Readability	Easy to read and edit using any word editor.	Not readable
Storage	Occupy more space due to character conversions	Occupy less space
Accuracy	While reading/writing numbers, some conversion errors may occur.	Highly accurate for numbers because it stores the exact internal representation of values.

Working with files

- In C++, we use the following streams
  - fstream: to open for reading and writing
  - ifstream: default is for reading (ios::in)
  - ofstream: default is for writing (ios::out)
- Syntax:

```
fstream f;
f.open(filename, mode);
ifstream fin(filename, mode);
ofstream fout(filename, mode);
```

- What are the differences between these calls?

The modes for file

- Modes:
  - `ios::in` open to read
  - `ios::out` open to write
  - `ios::binary` open a binary file (default: text)
  - `ios::ate` starting from the end of file
  - `ios::app` open to append (write mode only)
  - `ios::trunc` delete the old file and overwrite
- Close file when finishes:
  - `f.close();` disconnect the file with the stream it is associated with.

Status of a file

- To check the status of a file
  - **good()**: check if the stream is ready for reading or writing?
  - **bad()**: if the reading or writing was fail, this flag is turned on
  - **fail()**: similar to bad(), but it also checks the format, i.e. we need to read an integer number but facing a character..
  - **eof()**: the file cursor is at the end of the file

Text File Functions

- Reading/writing a single character from/to a file:
  - `get()`
  - `put()`
- Reading/writing a line from/to a file:
  - `getline()`
  - `<<`
- Reading/writing a word from/to a file:
  - `char ch[20];`
  - `fin.getline(ch, 20, ' ');`
  - We can use `fin >> ch` for reading and `fout <<` for writing a word in text file

Text file - Example

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

int main () {
    ofstream fout;
    fout.open (“FileToWrite.txt”);
    //do something on the file
    fout.close();
    return 0;
}
```

Exercise

1. Write a function in C++ to count the number of uppercase alphabets present in a text file “BOOK.txt”
2. Write a function in C++ to count the number of alphabets present in a text file “BOOK.txt”
3. Write a function in C++ to count the number of digits present in a text file “BOOK.txt”
4. Write a function in C++ to count the number of white spaces present in a text file “BOOK.txt”
5. Write a function in C++ to count the number of vowels present in a text file “BOOK.txt”
6. Assume a text file “Test.txt” is already created. Using this file, write a function to create three files “LOWER.TXT” which contains all the lowercase vowels and “UPPER.TXT” which contains all the uppercase vowels and “DIGIT.TXT” which contains all digits.

Binary File Functions

- read(): read a block of binary data or read a fixed number of bytes from the specified stream and store in a buffer.
  - `fin.read((char *)& s, sizeof(s));`
- write(): write a block of binary data or write fixed number of bytes from a specific memory location to the specified stream.
  - `fout.write((char *)& s, sizeof(s));`
- The first argument is the address of variable s, which is type cast to type char* (pointer to char)

Binary File - Example

```
#include <fstream>

const int LIM = 20;

struct planet {
   char name[LIM]; //name of the planet
   double population; //its population
   double g;       //its acceleration of gravity
};
```

Binary File - Example

```
int main(){
   planet pl;
   InputPlanet(pl);
   ofstream fout("planets.dat", ios::out | ios::app |
                                        ios::binary);
   if (!fout.good())
      return 0;
   fout.write((char*)&pl, sizeof(pl));
   fout.close();
   ifstream fin("planets.dat", ios::in | ios::binary);
   if (!fin.is_open())
      return 0;
   while(fin.read((char*)&pl, sizeof(pl)))
      OutputPlanet(pl);
   fin.close();
   return 1;
}
```

Exercise

- Read and understand the format of a bitmap file
- Implement a program to read and display a BMP file

The get and put cursor of a stream

- In the **ifstream**, the **get** cursor is at the position of the next element to be read
- In the **ofstream**, the **put** cursor is at the position to write the next element
- Some operations on these 2 cursors:
  - **tellg()**: tell the current position of the **get** cursor
  - **tellp()**: tell the current position of the **put** cursor
  - **seekg(position)**: move the **get** cursor to the **position**
  - **seekp(position)**: move the **put** cursor to the **position**

The get and put cursor of a stream

- **seekg(offset, direction)**
- **seekp(offset, direction)**

Move the cursor offset steps based on the direction.

- Direction:
  - **ios::beg**: from the beginning of the stream
  - **ios::cur**: from the current position
  - **ios::end**: from the end of the stream

APPLICATIONS OF BINARY FILES


Q&A