

Ch12-FileIO

August 16, 2020

1 File Input/Output (IO)

1.1 Topics

- input/output streams
- file input stream
- file output stream
- reading unstructured and structured text files
- formatting file output

1.2 Streams

- a **stream** is an abstract object that represents the flow of data from a source like keyboard or a file to a destination like the screen or a file
- we've learned earlier about standard io streams
- `iostream` is used to read the data from standard input (keyboard)
 - data is then stored in computer memory to be manipulated to solve problems
 - data result is written to the standard output (monitor) from computer memory
- C++ uses various streams to reading data from and write data to
 - `stringstream` is another stream that creates stream of strings
- often programs need to read data, process it and write the result back to secondary devices for permanent storage
- file stream is used to read data from secondary storage like disk and write result and data back to it for permanent

1.3 File stream

- we use `<fstream>` header to create input and output file streams

1.4 File input

- **ifstream** object is created to read data from file
- it creates a stream that flows from the file into the program (memory)

1.4.1 steps for file input

1. open file to read data from
 - file must exist; run-time error otherwise
2. read file contents
3. close the file

1.4.2 open file

- to open the file you need to create ifstream object
- syntax to create ifstream object:

```
// create stream without opening the file
ifstream objectName;
// create object and open the given file
ifstream objectName("file_name");
```

- objectName is any identifier you want to use it for this particular ifstream
- file name is passed as an argument; we'll learn how to read text files
- file name must be present to read data from
- let's open and read this sample text file called `inputfile.txt`

```
[1]: #include <fstream> // ifstream
#include <iostream>
#include <string>

using namespace std;
```

```
[2]: // declare ifstream object
ifstream fin;
// i prefer fin as stream object name; rhymes with cin
```

```
[3]: // open the file using open method
fin.open("inputfile.txt");
```

```
[4]: // declare stream object and open the given file
ifstream fin1("inputfile.txt");
```

1.4.3 read data

- once the ifstream object is created, reading data is similar to reading from istream
- we use >> (input extraction) and getline functions to read the data
- syntax:

```
ifstreamObject >> variable1 >> variable2 >> ...;
```

- >> - extracts one value of variable type and stops at a whitespace or mismatch type

```
getline(ifstreamObject, strVariable);
```

- getline() - reads the whole line as string into strVariable

```
[5]: // let's read couple of words from inputfile.txt
string word1, word2;
```

```
[6]: fin >> word1 >> word2;
```

```
[7]: cout << word1 << " " << word2;
```

this is

```
[8]: // let's read the rest of the line  
string line
```

```
[9]: getline(fin, line);
```

```
[10]: cout << line;
```

first sentence.

```
[11]: // let's read the next line  
getline(fin, line);  
cout << line;
```

this is 2nd sentence

```
[12]: // let's read the next line  
getline(fin, line);  
cout << line;
```

some numbers are below

```
[13]: // let's read the 3 numbers  
int nums[3];
```

```
[14]: fin >> nums[0] >> nums[1] >> nums[2];
```

```
[15]: cout << nums[0] << " " << nums[1] << " " << nums[2];  
// done reading all the contents of the file
```

10 20 30

```
[15]: @0x10d2b5ec0
```

1.4.4 close file

- use close() method on ifstream objects

```
[16]: fin.close();
```

```
[17]: // can check if file is open  
fin.is_open()
```

```
[17]: false
```

```
[18]: fin1.close();
```

1.4.5 ifstream member functions

- there are a whole bunch of methods available in ifstream objects
- all the methods can be found here with examples:
https://en.cppreference.com/w/cpp/io/basic_ifstream

1.5 File output

- sending output data to a file is similar to reading data
- 3 steps:
 1. Create a new file
 2. Write data to the file
 3. Close the file

1.5.1 create file

- to write data to a file, first create ofstream object
- syntax:

```
// create ofstream object without creating a file
ofstream fout;
// create ofstream object with a given file
ofstream fout("output-filename");
```

- if the file exists, the file will be truncated
 - existing data will be lost

```
[1]: #include <fstream> // ifstream and ofstream
#include <iostream>
#include <string>
#include <iomanip>
#include <vector>
#include <algorithm>

using namespace std;
```

```
[2]: // create output file stream object
ofstream fout;
```

```
[3]: // create file
fout.open("outputfile.txt");
// you should see a new text file created in the same folder as this notebook
```

```
[4]: ofstream fout1("outputfile1.txt");
// you should see a new text file created in the same folder as this notebook
```

1.5.2 write data

- writing data to file is similar to writing data to std output stream
- use << (output insertion operator) with the stream object

```
[5]: // write data to output file stream
fout << "Hello World!" << endl;
fout1 << 2 << " + " << 2 << " = " << (2+2) << endl;
```

1.5.3 close file

- closing file is important specially that was opened to write
- file will be locked if it's not explicitly closed until the program ends

```
[6]: fout.close();
      fout1.close();
```

1.6 Formatting file output

- iomanip manipulators work exactly the same way for file output
- fixed, setw(), setprecision(), left, right, ws, setfill(), etc. all can be used to format the file output

```
[7]: fout.open("outputfile.txt");
```

```
[10]: fout << setw(50) << setfill('=') << " " << setfill(' ') << endl;
```

```
[11]: fout << fixed << setprecision(2);
      fout << setw(25) << left << "Item" << setw(25) << right << "Price" << endl;
      fout << setw(50) << setfill('=') << " " << setfill(' ') << endl;
      fout << setw(25) << left << "Apple" << setw(25) << right << 5.99 << endl;
      fout << setw(25) << left << "Carrots" << setw(25) << right << 2.55 << endl;
      fout << setw(50) << setfill('*') << " " << setfill(' ') << endl;
```

```
[12]: fout.close();
      // see the contents of outputfile.txt
```

1.7 Exercises

1. Write a program that computes distance between two points in Cartesian coordinates.
 - prompt user to enter name of the input file that contains a bunch of points
 - using a text editor manually create a file with two coordinate points (x, y) per line
 - use vector to store points
 - use as many function(s) as possible
 - write at least 3 test cases for each computing functions
 - program continues to run until user wants to quit
 - most of the part is done in Jupyter Notebook demo
2. Write a program to compute area and circumference of a circle.

- prompt user to enter name of the input text file that contains a bunch of radii of several circles
 - using a text editor manually create a file that contains an arbitrary number of radii
 - use vector to store data from the input file
 - use as many function(s) as possible
 - write at least 3 test cases for each computing functions
 - program continues to run until user wants to quit
3. Write a program to compute area and perimeter of a rectangle.
- prompt user to enter name of the input text file that contains lengths and widths of several rectangles
 - using a text editor manually create a file with length and width of a rectangle per line
 - use as many function(s) as possible
 - write at least 3 test cases for each computing functions
 - program continues to run until user wants to quit
4. Write a program to compute area and perimeter of a triangle given 3 sides.
- prompt user to enter name of the file that contains 3 sides of several triangles
 - using a text editor manually create a file that contains 3 sides of a triangle per line
 - use as many function(s) as possible
 - write at least 3 test cases for each computing functions

1.7.1 see a sample solution for exercise 4 at [demo_programs/Ch12/triangle.cpp](#)

5. A Grade Book:
- Write a C++ menu-driven program that let's professors keep track of students grades with the following requirements:
 - program must use struct to keep track of students grades
 - program prompts user to enter name of the input text file that contains students information in the following format
 - first name, last name, test1, test2, test3, test4, test5
 - program calculates average grade and the letter grade (A-F) based on the average grade
 - program sorts the student records based on grade in non-increasing order (highest to lowest)
 - program lets user add a new student
 - program lets user update existing student's information
 - program lets user delete existing student
 - program saves the data back into the same input file as a database
 - program creates a cleanly formatted report of students' grades
6. Airline Reservation System:
- Write a C++ menu-driven CLI-based program that let's an airline company manage airline reservation on a single aircraft they own with the following requirements:
 - aircraft has 10 rows with 2 seat on each row
 - program provides menu option to display all the available seats
 - program provides menu option to let user pick any available seat
 - program provides menu option to create total sales report
 - program provides menu option to update price of any seat
 - program saves the data into a file

1.8 Kattis problems

- typically Kattis problems don't require File IO

1.9 Summary

- the notebook covered file streams (input and output)
- learned how to read structured and unstructured data
- write and format output to a output file
- exercises and sample solution(s)

[]: