File Input/Output (IO)

Topics

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

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 about standard io streams in earlier chapters
- iostream is used to read the data from standard input (keyboard)
  - data is then stored in computer memory to be manipulated to solve problems
  - result is written to the standard output (monitor) from computer memory
- C++ uses various streams to read 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 (e.g., hard disk and flash drive) and write result and data back to it for permanent storage

File stream

- we use `<fstream>` header to create input and output file streams
- see all the methods and data available in fstream objects: https://en.cppreference.com/w/cpp/io/basic_fstream

File input

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

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

Open file

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

```
//1. create stream object without opening the file
ifstream objectName;
//2. open a file with the objectName
objectName.open("fileName");

// OR 1. 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 demos/file_io/inputfile.txt

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

using namespace std;
```

In [2]:
```
string file_name = "./demos/file_io/inputfile.txt";
```

In [3]:
```
// declare ifstream object
ifstream fin;
// I prefer fin as ifstream object name; rhymes with cin
```

In [4]:
```
// open the file using open method
fin.open(file_name.c_str());
```

In [5]:
```
// declare stream object and open the given file
ifstream fin1("./demos/file_io/inputfile.txt");
```

Read data

- once the ifstream object is created and file opened, reading data is similar to reading from iostream
- we use `>>` input extraction operator and getline functions to read the data
  - similar to standard io
- syntax:

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

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

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

- recall `getline()` reads a single line as string into strVariable

In [6]:
```
// let's read couple of words from inputfile.txt
string word1, word2;
```

In [7]:
```
fin >> word1 >> word2;
```

In [8]:
```
cout << word1 << " " << word2;
```

```
this is
```

In [9]:
```
// let's read the rest of the line
string line
```

In [10]:
```
getline(fin, line);
```

In [11]:
```
cout << line;
```

```
 first sentence.
```

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

```
this is 2nd sentence
```

In [13]:
```
// let's read the next line
getline(fin, line);
cout << line;
```

```
some numbers are below
```

In [14]:
```
// let's read the 3 numbers
int nums[3];
```

In [15]:
```
fin >> nums[0] >> nums[1] >> nums[2];
```

In [17]:
```
cout << nums[0] << " " << nums[1] << " " << nums[2] << endl;
// done reading all the contents of the file
```

```
10 20 30
```

Out[17]:
```
@0x113a03558
```

close file

- use `close()` method on `ifstream` objects

In [18]:
```
fin.close();
```

In [19]:
```
// can check if file is open
fin.is_open();
```

In [20]:
```
fin1.close();
```

Read the whole file into memory

- file can be read in different mode
  - `input, output, binary, append` , etc.
  - see open method - http://www.cplusplus.com/reference/fstream/fstream/open/
- it may be required to read the whole file for some applications
- the following code snippet shows how to read the complete file content as a buffer

In [21]:
```
string file_path = "./demos/file_io/inputfile.txt";
fstream file; // generic filestream object; not input or output
```

In [22]:
```
// open file in binary and put output position at the end of the file
file.open(file_path, file.in | file.binary | file.ate);
```

In [23]:
```
if (!file.is_open())
    cout << "failed to open " << file_path << '\n';
else {
    // findout the size of the the file; get position in input sequence
    size_t size = file.tellg();
    // Set position in input sequence
    file.seekg(0, file.beg );

    // allocate memory to store file contents
    char * buffer = new char[size];
    if (file.read(buffer, size))
    {
        cout << "File contents...\n";
        cout << buffer << endl;
        // parse buffer in memory...
    }
    delete[] buffer;
    file.close();
}
```

```
File contents...
this is first sentence.
this is 2nd sentence
some numbers are below
10
```

```
20
30
```

ifstream member functions

- there are a 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

File output

- steps required to write output data to a file is similar to reading data from a file
- 3 steps:
  1. Create a new file or open an existing file into append mode
  2. Write data to the file
  3. Close the file

create a file

- to write data to a file, first create ofstream object
- create a new file to write data to
  - NOTE: if the file exists, it'll truncate/delete contents of the existing file
- syntax:

```
// 1. create ofstream object without creating a file
ofstream fout;
// 2. create/open file with the object
fout.open("output-filename");

// OR create ofstream object and create a given file
ofstream fout("output-filename");
```

In [24]:
```
#include <fstream> // ifstream and ofstream
#include <iostream>
#include <string>
#include <iomanip>
#include <vector>
#include <algorithm>

using namespace std;
```

In [25]:
```
// create output file stream object
ofstream fout;
```

In [26]:
```
// create/open file
fout.open("./demos/file_io/outputfile.txt");
// you should see a new text file created in the same folder where this notebook
```

In [27]:
```
ofstream fout1("./demos/file_io/outputfile1.txt");
// you should see a new text file created in the same folder where this notebook
```

write data

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

In [28]:
```
// write data to output file stream
fout << "Hello World!" << endl;
fout1 << 2 << " + " << 2 << " = " << (2+2) << endl;
```

close file

- closing file is important especially that was opened for writing
- file remains locked if it's not explictly closed or until the program ends

In [29]:
```
fout.close();
fout1.close();
```

Copy a file

- write a function that copies source file into destination file

In [30]:
```
// returns true when success, false otherwise
bool copyFile(string source_file, string dest_file) {
    // read the data
    ifstream fin;
    fin.open(source_file.c_str(), fin.binary);
    if (not fin.is_open()) return false;
    fin.seekg(0, fin.end);
    size_t size = fin.tellg();
    char *buffer = new char[size];
    fin.seekg(0, fin.beg);
    fin.read(buffer, size);

    // write the data
    ofstream fout;
    fout.open(dest_file.c_str(), fout.binary);
    if (not fout.is_open()) return false;
    fout.write(buffer, size);
    delete[] buffer;
    fin.close();
    fout.close();
    return true;
}
```

In [33]:
```
string source, dest;
```

In [35]:
```
source = "resources/record.png"
```

Out[35]:
```
"resources/record.png"
```

In [36]:
```
cout << boolalpha << copyFile(source, "record_copy.png");
// check the repo folder where record_copy.png should be created if returned tru
```

```
true
```

In [37]:
```
cout << boolalpha << copyFile("./demos/file_io/inputfile.txt", "inputfile_copy.t
```

```
true
```

Formatting file output

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

In [25]:
```
fout.open("./demos/file_io/formatted_output.txt");
```

In [26]:
```
fout << setw(50) << setfill('=') << " " << setfill(' ') << endl;
```

In [27]:
```
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;
```

In [28]:
```
fout.close();
// see the contents of formatted_output.txt file
```

Labs

1. The following lab demonstrates the usage of file input and output.
   - use the partial solution `fileio.cpp` in labs/fileio folder
   - use Makefile to compile and debug the file
   - fix all FIXMEs and write #FIXED# next to each fixme once fixed

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

1. 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

1. 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

1. 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 vector to store three sides of a triangle and vector of vector to store all the triangles info
   - use as many function(s) as possible
   - write at least 3 test cases for each computing functions

see a sample solution for exercise 4 at demos/file_io/triangle/triangle.cpp

1. 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 provieds menu option to display all the avaialable seats
   - program provides menu option to let user pick any available seat
   - program provides menu option to creates total sales report
   - program provides menu option to update price of any seat
   - program saves the data into a file

Kattis problems

- typically Kattis problems don't require File IO
- almost all Kattis problems require standard IO for data input and printing answers

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)

In [ ]: