CS319: Scientific Computing (with C++)

Extra: Reading CSV Files

Week 12: Extra (These notes were not covered in class)

1. Files (recap)

In Week 9 we learned how to read and write files. IN this "extra" example, we'll read and write files so-called "CSV files" that store data for matrices.

First, we'll recap over some basics...

- To work with files, we need to include the `fstream` header at the start of our C++ programe.

```
#include <cstdlib>
```

- To **read** data from a file, we declare an object of type `ifstream`.

```
ifstream InputFile;
```

- To **write** to a file, declare an object of type `ofstream`.

```
ofstream OutputFile;
```

1. Files (recap)

- The `InputFile` and `OutputFile` objects do not get have any *actual* files associated with them. To link them to files, use the `open method`

```
InputFile.open("Source.txt");
OutFile.open("Output.txt");
```

- **reading** from a file can be tricky, since we may not know how data is stored in it. Often, it is easiest to read a single character at a time.

```
char c;
InFile.get( c );
```

- **writing** to a file is easier: we just use the << operator, like we do with `std::cout`
- When finished with a file, we should call the `close()`method.

1. Files (recap)

- When reading, we also have to check when we get to the `end` of a file. If there are no more `char`acters left in the input stream, then the `eof()` method evaluates as *`true`*.
- If you have read the contents of a file, and want to go back to the start, do this:

```
InFile.clear(); // Clear the eof flag
InFile.seekg(ios::beg); // rewind to begining.
```

2. CSV files

As an example of working with files, we'll write a program that reads data from a "*C*omma *S*eparated *V*alues" (*CSV*) file.

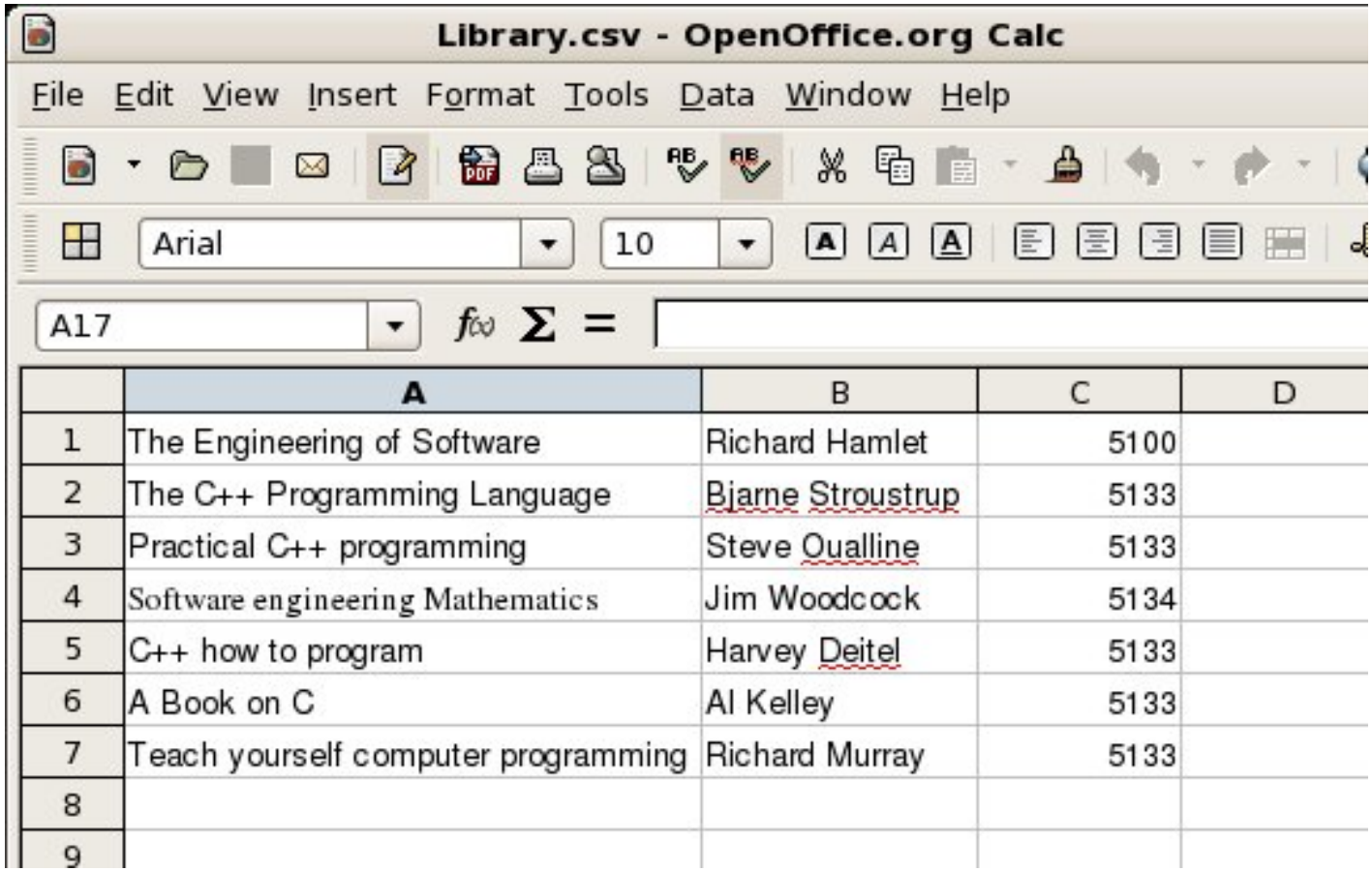
There are many accountancy and spread-sheet packages available. It is necessary for them to be able to share data. Therefore, even though they all have their own file format, they must be able to read and write a neutral data type. This is often *CSV*.

Your favourite data-handling system (e.g., Excel, LibreOffice,...) can read and write these files.

In a CSV file, the contents of cells from the same row are simply separated by commas. (Unlike, say, excel files, it does not contain addition information, such as font type, text alignment, formulae, etc.

3. Example 1: Library

The first example we'll look at is based on some data stored in a file called `Library.csv`:



	A	B	C	D
1	The Engineering of Software	Richard Hamlet	5100	
2	The C++ Programming Language	Bjarne Stroustrup	5133	
3	Practical C++ programming	Steve Qualline	5133	
4	Software engineering Mathematics	Jim Woodcock	5134	
5	C++ how to program	Harvey Deitel	5133	
6	A Book on C	Al Kelley	5133	
7	Teach yourself computer programming	Richard Murray	5133	
8				
9				

3. Example 1: Library

When this is saved to a CSV file we get

```
The Engineering of Software,Richard Hamlet,5100
The C++ Programming Language,Bjarne Stroustrup,5133
Practical C++ programming,Steve Oualline,5133
Software engineering Mathematics,Jim Woodcock,5134
C++ how to program,Harvey Deitel,5133
A Book on C,Al Kelley,5133
Teach yourself computer programming,Richard Murray,5133
```

We'll look at how to write a C++ program that can open this file and read data from it.

We shall assume that we know the structure of the file. In particular, we'll assume we know how many columns there are, and what they contain.

3. Example 1: Library

We'll start by including the necessary headers, including `fstream`

00Library-CSV.cpp

```
#include <iostream>
#include <string>
#include <iomanip>
#include <fstream>
#include <cstdlib> // For EXIT-FAILURE and atoi
```

3. Example 1: Library

In the start of the `main()` function, we'll define the input stream, which we'll call `InFile`. We then open the file, and verify that no error occurred.

00Library-CSV.cpp (continued)

```
12 int main(void) {
     std::ifstream InFile;
14   std::string InFileName="Library.csv";

16   InFile.open(InFileName.c_str());

18   if (InFile.fail())  {
       std::cerr << "Error - can't open " << InFileName << std::endl;
20     exit(EXIT_FAILURE);
     }
```

3. Example 1: Library

Then we count the number of lines in the CSV file. The result will be used for some dynamic memory allocation.
Once we've read the file, we clear the `eof` (end-of-file) flag, and set the file pointer back to the `beg`inning of the file.

00Library-CSV.cpp (continued)

```
   // Count the number of entries
24 char c;
   int Lines=0;
26 InFile.get(c);
   while(!InFile.eof())
28 {
     if (c=='\n')
30     Lines++;
     InFile.get(c);
32 }
   std::cout << "There are " << Lines << " in " <<
34   InFileName << std::endl;
   InFile.clear(); // Clear the eof flag
36 InFile.seekg(std::ios::beg); // rewind to beginning.
```

3. Example 1: Library

Now that we know the number of lines, we'll declare some arrays for storing the data in the file.

- book title (we'll store as a `std::string`)
- Author (also a `std::string`)
- Call number (which we'll treat as an int)

We reserve memory for each array using the `new` operator.

00Library-CSV.cpp (continued)

```
38    std::string *Author = new std::string [Lines],
        *Title = new std::string [Lines];
40    int *CallNumber = new int [Lines];
```

3. Example 1: Library

We'll now read the data from the file. First we declare a `char` array of length 100, where we'll store each cell, temporarily. We'll read each cell using the `get()` method. Note that the third argument, which tells `get()` where to stop reading. For the first two lines that a comma; for the last it is a new-line.
Then we `ignore()` that character.

```
42    char str_tmp[100];
      for (int i=0; i< Lines; i++) {
44       InFile.get(str_tmp, 99, ',');
         Title[i] = str_tmp;
46       InFile.ignore();

48       InFile.get(str_tmp, 99, ',');
         Author[i] = str_tmp;
50       InFile.ignore();

52       InFile.get(str_tmp, 99, '\n');
         CallNumber[i] = atoi(str_tmp);
54       InFile.ignore();
      }
```

3. Example 1: Library

We'll check if it worked by outputting a subset of the data. Specifically, we'll output the author and title for any book that has a 5133 call number.

```
   std::cout << "Here are the 5133 books: " << std::endl;
58 for (int i=0; i< Lines; i++)
   {
60    if (CallNumber[i] == 5133)
        std::cout << std::setw(20) << Author[i] << ": "
62            << Title[i] << std::endl;
   }
64 return(0);
}
```

4. Example 2: CSV to Matrix

In this example, we'll write a program which can read and write a `Matrix` to/from a file.

The matrix we'll work with is

$$\begin{pmatrix} 11.0 & 2.2 & 3.123 \\ -4.2 & 15.6 & 6.0 \\ -7.3 & -8.0 & 19.0 \end{pmatrix}$$

The data are stored in the `matrix1.cpp` file. Here is its contents:

```
11.0,  2.2,  3.123
-4.2, 15.6,  6.0
-7.3, -8.0, 19.0
```

Once loaded from the file, we'll compute the transpose of this matrix, and save that to another file.

4. Example 2: CSV to Matrix

To run the following example, you'll need the following files

- `01Matrix-CSV.cpp` (contains the `main()` function)
- `Matrix11.h` and `Matrix11.cpp` (the latter has a minor bug fix).
- `Vector10.h` and `Vector10.cpp`
- `matrix1.csv`

Your project must include all three `.cpp` files; the `.h` and `.csv` files must be in the appropriate folder. You can get the code from https://www.niallmadden.ie/2324-CS319/Week12/extras

You can also run this code online at https://www.online-cpp.com/lRbNcHXsEM

4. Example 2: CSV to Matrix

The main program, 01Matrix-CSV.cpp starts with comments and the usual include lines (not show here).

It then includes the headers for the Matrix and Vector classes. After that, we give the headers for functions that read and write the matrices to/from a CSV file.
ReadMatrixCSV() takes a file name as input and returns a Matrix.
WriteMatrixCSV() takes as inputs a Matrix, and string containing a file name, and the precision (i.e., number of digits) for doubles in the file.

01Matrix-CSV.cpp

```
16 #include "Matrix11.h"
   #include "Vector10.h"

   // Headers for functions for reading and writing matrices
20 Matrix ReadMatrixCSV(std::string FileName);
   void WriteMatrixCSV(Matrix M, std::string FileName, int p=5);
```

At the start of the `main()` we define a `Matrix` object, `M`, and read its values from a file. The code for the `ReadMatrixCSV()` function is described further on.

01Matrix-CSV.cpp: main()

```
int main(void)
{
  Matrix M;
  M = ReadMatrixCSV("matrix1.csv");    // Read a matrix from a file
  std::cout << "Matrix read from matrix1.csv. It is: "
            << std::endl;
  M.print();
```

Next we'll compute the transpose of `M`, and write that to a file called `transpose.m`

01Matrix-CSV.cpp: main()

```
    // Make the transpose of that matrix
32  Matrix T(M.size());
    for (unsigned i=0; i<M.size(); i++)
34    for (unsigned j=0; j<M.size(); j++)
        T.setij(j,i, M.getij(i,j));

    std::cout << "\nTranspose of M is : " << std::endl;
38  T.print();
    // Write that matrix to a file, to 8 digits of precision
40  std::cout << "Writing the tranpose to transpose.csv\n";
    WriteMatrixCSV(T, "transpose.csv", 8);

    return(0);
44  }
```

4. Example 2: CSV to Matrix `ReadMatrixCSV()`

The function `ReadMatrixCSV()` takes a file name, which is stored as a `string`, and tries to open it in a **input stream**.

01Matrix-CSV.cpp: ReadMatrixCSV()

```
46 // Function to read a matrix stored in a CSV file.
   // Input: string containing the file name
48 // Output: Matrix object
   Matrix ReadMatrixCSV(std::string InputFileName)
50 {
     std::ifstream InputFile;

     // open InputFileName for reading
54   InputFile.open(InputFileName.c_str());
     if (InputFile.fail())
56   {
       std::cerr << "Error - can't open " << InputFileName
58               << std::endl;
       exit(1);
60   }
```

4. Example 2: CSV to Matrix `ReadMatrixCSV()`

Next the function needs to determine the size of the matrix stored in the file. We'll be quite lazy about that: we read the first line, and count the number of commas.
We then define a matrix of size N, and set its entries to zero.

01Matrix-CSV.cpp: ReadMatrixCSV()

```
62    // We'll determine the size of the matrix by reading the
      // first line and counting the commas
64    unsigned N=0;
      char c; //
66    InputFile.get(c);
      while((c!='\n') && (!InputFile.eof()) )
68    {
        if (c==',')
70        N++;
        InputFile.get(c);
72    }
      N++;
74    std::cout << InputFileName << " has " << N << " columns.\n";
      Matrix M(N); // make an N-N matrix
76    M.zero();
```

4. Example 2: CSV to Matrix `ReadMatrixCSV()`

Then, after resetting the file pointer to the start of the file, we read the contents, alternating between extracting a `double` (which will be the matrix entries) and a `char` which will be the comma or new-line.

01Matrix-CSV.cpp: ReadMatrixCSV()

```
78    InputFile.clear(); // Clear the eof flag
      InputFile.seekg(std::ios::beg); // reset point to start of file
80    double f;
      for (unsigned i=0; i<N; i++)
82      for (unsigned j=0; j<N; j++)
        {
84        InputFile >> f;
          if (InputFile.eof())
86        {
            std::cout << "WARNING: end of file before matrix read";
88          break;
          }
90        M.setij(i,j,f);
          InputFile >> c;
92      }
```

When we are finished reading the file contents, we close the file, and return the `Matrix` object.

01Matrix-CSV.cpp: ReadMatrixCSV()

```
96    InputFile.close();
      return(M);
}
```

4. Example 2: CSV to Matrix `WriteMatrixCSV()`

Our function to write a matrix to a file takes three arguments:

1. The `Matrix` object that we are going to write to the file;
2. The name of the file, stored as `string` object.
3. An `int` storing the maximum precision we'll use for the data. Its default value, set in the header, is 5.

We use the stream insertion operator (`<<`) for writing the data. When finished, we close the file.

The entire code for the function is shown on the next slide.

01Matrix-CSV.cpp: WriteMatrixCSV()

```
   void WriteMatrixCSV(Matrix M, std::string FileName, int p) {
102   std::ofstream OutputFile;

104   OutputFile.open(FileName.c_str());
      if (OutputFile.fail())  {
106     std::cerr << "Error - can't open " << FileName << std::endl;
        exit(1);
108   }
      OutputFile.precision(p);


      unsigned N=M.size();
112   for (unsigned i=0; i<N; i++)
        for (unsigned j=0; j<N; j++) {
114       OutputFile << M.getij(i,j);
          if (j<(N-1))
116         OutputFile << ",";
          else
118         OutputFile << std::endl;
        }
120   OutputFile.close();
   }
```