

CS319: Scientific Computing

**Projects; Strings, and Files and Streams;
Vectors**

Dr Niall Madden

Week 8: 6 + 8 March, 2025



Slides and examples: <https://www.niallmadden.ie/2324-CS319>

0. Outline

~~1 Projects!~~

~~2 Recall: objects~~

~~3 Strings~~

~~■ Operator overloading~~

~~4 I/O streams as objects~~

~~5 Files~~

~~■ ifstream and ofstream~~

~~■ open a file~~

~~■ Reading from the file~~

■ Tip: working with files

6 Portable Bitmap Format (pbm)

7 Review of classes

8 Vectors and Matrices

~~9 A vector class~~

~~■ Vectors~~

~~■ C++ "Project"~~

~~■ Adding two vectors~~

~~10 Solving Linear Systems~~

Slides and examples:

<https://www.niallmadden.ie/2425-CS319>

5. Files

All of the C++ programs we have looked at so far take their input from the *standard input stream*, which is usually the keyboard. Example:

```
std::cout << "Enter an integer: ";  
std::cin >> i;
```

Although the *standard input stream* can be redirected to be, for example, a file (easily done on a Mac and on Linux), it is usually necessary to open a file **from within the program** and take the data from there. The data is then processed and written to a new file.

5. Files

To achieve either of these tasks in **C++**, we create a **file stream** and use it just as we would **cin** or **cout**. We'll with a simple example.

02CountChars.cpp

- (i) This program opens an input file called **CPlusPlusTerms.txt**
- (ii) It opens an output file called **Output.txt**
- (iii) It counts the number of characters in the input file.
- (iv) It writes that result to the output file.

Download the input file from

<https://www.niallmadden.ie/2425-CS319>. Save it to the folder containing the executable that you compile.

Once we have the basic idea, we'll take a closer look at each operation (opening, reading, writing).

When working with files, we need to include the *fstream* header file.

To **read** from a file, declare an object of type *ifstream*.

Open the file by calling the *open()* method on that object.

02CountChars.cpp

```
8  #include <iostream>
   #include <fstream>
10 #include <cstdlib>

12 int main(void )
   {
14     char c;
       std::ifstream InFile;
16     InFile.open("CPlusPlusTerms.txt");
       std::cout << "Processing the contents of CPlusPlusTerms.txt"
18         << std::endl;
```

To **write** to a file, declare an object of type `ofstream`. Then (again) open the file by calling the `open()` method on that object.

To read a single character, can use `InFile.get()`

02CountChars.cpp

```
20  std::ofstream OutFile;  
    OutFile.open("Output.txt");  
22  std::cout << " See file Output.txt for more information."  
        << std::endl;  
  
    int i=0;  
26  InFile.get( c );
```

So "OutFile" is an object of type "ofstream"
and "ofstream" means "output file stream"

eof = "end of file"

If there are no more characters left in the input stream, then `InFile.eof()` evaluates as *true*.

Use the stream objects just as you would use `cin` or `cout`:

`InFile >> data` or
`OutFile << data.`

Close the files:

`InFile.close()`,
`OutFile.close()`

01CountChars.cpp

```
28 while( ! InFile.eof() ) {  
    i++;  
    InFile.get( c );  
30 }  
32 OutFile << "CplusplusTerms.txt contains "  
34     << i << " characters \n";  
36 InFile.close();  
    OutFile.close();  
    return(0);  
40 }
```

= "Stream insertion operator"

The method `open` works differently for `ifstream` and `ofstream`:

- ▶ `InFile.open()` Opens an existing file for reading,
- ▶ `OutFile.open()` Opens a file for writing. If it already exists, its contents are overwritten.

The first argument to `open()` contains the file name, and is an array of `char`acters. More precisely, it is of type `const char*`.

For example, we could have opened the input file in the last example with:

```
2  char InFileName[20]="CPlusPlusTerms.txt";  
3  ...  
4  std::cout << "Processing the contents of "  
5      << InFileName << std::endl;  
6  ...  
7  InFile.open(InFileName);
```

Note that file name is stored as a **"C string"**.

If we want to use C++ style strings, use the `c_str()` method. In this example we'll prompt the user to enter the file name.

```
2  std::ifstream InFile;  
   std::string InFileName;  
  
4  std::cout << "Input the name of a file: " << std::endl;  
   std::cin >> InFileName;  
  
   InFile.open(InFileName.c_str())
```

If you are typing the file name, there is a chance you will mis-type it, or have it placed in the wrong folder: so **always** check that the file was opened successfully. To do this, use the `fail()` function, which evaluates as `true` if the file was not opened correctly:

```
if (InFile.fail())
{
    std::cerr << "Error - cannot open " <<
        InFileName << std::endl;
    exit(1);
}
```

A better approach in this case might be to use a `while` loop, so the user can re-enter the filename. See [02CountCharsV02.cpp](#)

Recall that if you open an existing file for **output**, its contents are lost. If you wish to **append** data to the end of an existing file, use

To open an existing file and **append** to its contents, use

```
OutFile.open("Output.txt", std::ios::app);
```

.....

Other related functions include `is_open()` and, of course, `close()`

Above we also saw that `InFile.eof()` evaluates as *true* if we have reached the end of the (read) file.

Related to this are

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

In the above example, we read a character from the file using `InFile.get(c)`. This reads the next character from the `InFile` stream and stores it in `c`. It will do this for any character, even non-printable ones (such as the newline char). For example, if we wanted to extend our code above to count the number of lines in the file, as well as the number of characters, we could use:

```
1  std::ifstream InFile;
   int CharCount=0, LineCount=0;
3  ...
   // Open the file, etc.
5  InFile.get( c );
   while( ! InFile.eof() ) {
7      CharCount++;
       if ( c == '\n' )
9          LineCount++;
       InFile.get( c );
11 }
```

— "if c is a newline character"

Alternatively, we could use the **stream extraction operator**:

```
InFile >> c;
```

However, this would ignore non-printable characters.

One can also use get() to read C-style strings. However, to achieve this task, it can be better to use getline(), which allows us to specify a delimiter character.

One of the complications of working with files, is knowing where to store input files so that your code can find them.

For some, IDEs, this is made additionally complicated by the fact that the compiled version of the program may not be in the same folder as the source code. So you have to work out where that is.

One way that can help, is change the `int main(void)` line to

```
1 int main(int argc, char * argv[])  
  {  
3     std::cout << "This program is running as " << argv[0];  
     std::cout << "\nDownload the input file to the same folder";  
5     std::cout << std::endl;
```

Alternatively, you can try opening a `ofstream` file with a very particular name, and then search for it.

If using an online compiler, you'll need one that allows multiple files, such as

<https://www.jdoodle.com/online-compiler-c++-ide>

6. Portable Bitmap Format (pbm)

Some self-study

We won't go through this section in class: please review in your own time.

Image analysis and processing is an important sub-field of scientific computing.

There are many different formats: you are probably familiar with JPEG/JPG, GIF, PNG, BMP, TIFF, and others. One of the simplest formats is the **Netpbm format**, which you can read about at https://en.wikipedia.org/wiki/Netpbm_format

6. Portable Bitmap Format (pbm)

There are three variants:

Portable BitMap files represent black-and-white images, and have file extension *.pbm*

Portable GrayMap files represent gray-scale images, and have file extension *.pgm*

Portable PixMap files represent 8-bit colour (RGB) images, and have file extension *.ppm*

In this example, we'll focus on *.pbm* files.

6. Portable Bitmap Format (pbm)

CS319.pbm

```
1 P1
2 25 9
3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0
5 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
6 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
7 0 1 0 0 0 0 1 1 1 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0
8 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
9 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
10 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

CS319

- ▶ The first line is the “magic number”. Here “P1” means that it is a PBM format ASCII (i.e, plain-text) file.
- ▶ The second line has two integer representing the number of columns and rows of pixels in the image, respectively.
- ▶ The remaining lines store the matrix of pixel values: 0 is “white”, and 1 is “black”.

6. Portable Bitmap Format (pbm)

The file `03FlipPBM.cpp` shows how to read such an image, and output its negative. (See notes from class).

03FlipPBM.cpp

```
16  std::ifstream InFile;
    std::ofstream OutFile;
    std::string InFileName, OutFileName;

20  std::cout << "Input the name of a PBM file: " << std::endl;
    std::cin >> InFileName;
    InFile.open(InFileName.c_str());
```

6. Portable Bitmap Format (pbm)

03FlipPBM.cpp

```
24 while (InFile.fail() )
    {
26     std::cout << "Cannot open " << InFileName << " for reading."
        << std::endl;
28     std::cout << "Enter another file name : ";
        std::cin >> InFileName;
        InFile.open(InFileName.c_str());
30 }
    std::cout << "Successfully opened " << InFileName << std::endl;
```

6. Portable Bitmap Format (pbm)

03FlipPBM.cpp

```
34 // Open the output file
    OutFileName = "Negative_"+InFileName;
    OutFile.open(OutFileName.c_str());

    std::string line;
38 // Read the "P1" at the start of the file
    InFile >> line;
40 OutFile << "P1" << std::endl;

42 // Read the number of columns and rows
    unsigned int rows, cols;
44 InFile >> cols >> rows;
    OutFile << cols << " " << rows << std::endl;

    std::cout << "read: cols=" << cols << ", rows="
48         << rows << std::endl;
```

6. Portable Bitmap Format (pbm)

03FlipPBM.cpp

```
50  for (unsigned int i=0; i<rows; i++)
    {
52      for (unsigned int j=0; j<cols; j++)
        {
54          int pixel;
            InFile >> pixel;
56          OutFile << 1-pixel << " ";
        }
58      OutFile << std::endl;
    }
60  InFile.close();
    OutFile.close();

    std::cout << "Negative of " << InFileName << " written to "
64      << OutFileName << std::endl;
    return(0);
```

7. Review of classes

class

In C++, we defined new class with the `class` keyword.
An instance of the class is called an *“object”*.
A `class` combines by data and functions.

Within a class, code and data may be either

- ▶ **Private**: accessible only to another part of that object, or
- ▶ **Public**: other parts of the program can access it.

Roughly,

- ▶ keep data elements `private`,
- ▶ make function elements `public`.

Finish here
Friday