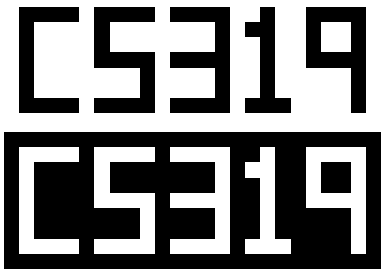
CS319: Scientific Computing

Projects; Strings, and Files and Streams (draft)

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
   - manipulators
5. Files
   - ifstream and ofstream
   - open a file
   - Reading from the file
   - Tip: working with files
6. Portable Bitmap Format (pbm)

Slides and examples:
https://www.niallmadden.ie/2425-CS319

1. Projects!

Notes for this part are at:

https://www.niallmadden.ie/2425-CS319/2425-CS319-Projects.pdf

2. Recall: objects

Last week we learned that

- A **class** is a general form of data type that we can create;
- An **object** is an instance of a particular class. E.g,

- A **method** is a member of a class that is a function. E.g.,

Note: The notes for last week were updated after the class to give a more coherent view on constructors and destructors. See `https://www.niallmadden.ie/2425-CS319/Week07/CS319-Week07.pdf`

. . . . . .

Before we continue with writing our own classes, we can now visit some important related topics in C++:

`strings` * **input and output streams** * **files**

3. Strings

A **string** is a collection of characters representing, for example, a word or a sentence.

In C++, a `char` array can be used to store a string. That approach is called a "C string", since it is inherited from an older language, C.

Such "C strings" are not so easy to work with, so C++ provides its own `string` class. The class can be accessed once the `string` header file is included. It is part of the `std` namespace.

```
#include <string>
.
.
.
std::string name;
```

3. Strings

We have used `string` before (Weeks 1 and 2), but have not thought of it as a class.

Since it is a class, it has some methods, including:

- `length()` and `size()` which both return the number of characters in the string;
- `substr(i,l)` with returns a substring of length `l`, starting at position `i`.
- `find()` which finds the first occurence on one substring in another.
- `c_str()` return the "C string" version. (Need this when working with files).

Example

Write a short C++ program that defines a `string` containing a sentence, and then extract the first word as another `string`.

00substring.cpp

```
#include <iostream>
#include <string>

int main(void)
{
  std::string
    sentence="Ada Lovelace was the first programmer",
    first_word;
  int space_loc = sentence.find(" ");     // Find first space
  first_word = sentence.substr(0,space_loc); // extract substring

  std::cout << "sentence is: " << sentence << std::endl;
  std::cout << "first word is: '" << first_word << "'\n";
  return(0);
}
```

3. Strings

Expected output:

```
sentence is: Ada Lovelace was the first programmer
first word is: ‘Ada’
```

With numbers, we are used to working with special functions called **operators**, which are usually represented by a mathematical symbol, such as `+`, `-`, `=`, `*`, `/`, etc.

When writing our own `class`, we can overload some of these (more about the details later).

The `string` class overloads several operators:

- Assignment: `=`
- Relational: `==`, `>`, `<`, etc;
- Arithmetic: `+`, `+=`

01string-operators.cpp

```
#include <iostream>
#include <string>

int main(void)
{
  std::string name[3], // array of names
    long_name="";
  name[0]="Augusta";
  name[1]="Ada";
  name[2]="King";

  long_name = name[0] + " " + name[1] + " " + name[2];

  std::cout << "long_name: " << long_name << std::endl;
  return(0);
}
```

Output

```
  long_name: Augusta Ada King
```

I/O means "Input/Output. So far, we have taken input from the keyboard, typically using cin, and sent output to a terminal window, using cout.

These are examples of **streams**: flows of data to or from your program. Moreover, they are examples of **objects** in C++.

In fact cout and cin are **objects** and are manipulated by their **methods**, i.e., public member functions and operators. (We saw this in Week 3)

Methods:

- `width(int x)` – minimum number of characters for next output,
- `fill(char x)` – character used to fill with in the case that the width needs to be elongated to fill the minimum.
- `precision(int x)` – sets the number of significant digits for floating-point numbers.

Code – width, fill

```
std::cout.fill('0');
for (int i=0; i<8; i++)
{
  std::cout.width(6);
  std::cout << rand()%200000
     << std::endl;
}
```

Output

```
089383
130886
092777
036915
147793
038335
085386
160492
```

Code – precision

```
double Pi=3.1415926535;
for (int i=1; i<=8; i++)
{
  std::cout.precision(i);
  std::cout << "Pi (correct to "<< i << " digits) is "
            << Pi << std::endl;
}
```

Output

```
Pi (correct to 1 digits) is 3
Pi (correct to 2 digits) is 3.1
Pi (correct to 3 digits) is 3.14
Pi (correct to 4 digits) is 3.142
Pi (correct to 5 digits) is 3.1416
Pi (correct to 6 digits) is 3.14159
Pi (correct to 7 digits) is 3.141593
Pi (correct to 8 digits) is 3.1415927
```

- `setw` – like `width`
- `left` – Left justifies output in field width. Used after `setw(`*n*`)`.
- `right` – right justify.
- `endl` – inserts a newline into the stream and calls flush.
- `flush` – forces an output stream to write any buffered characters
- `dec` – changes the output format of number to be in decimal format
- `oct` – octal format
- `hex` – hexadecimal format
- `showpoint` – show the decimal point and some zeros with whole numbers

Others: setprecision(n), fixed, scientific, boolalpha, noboolalpha, ...

Need to include `iomanip`

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 inteter: ";
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 );
```

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

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

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

Close the files:
`InFile.close()`,
`OutFile.close()`

01CountChars.cpp

```
   while( ! InFile.eof() ) {
28   i++;
     InFile.get( c );
30 }

32 OutFile <<
     "CPlusPlusTerms.txt contains "
34   << i << " characters \n";

36 InFile.close();
   OutFile.close();

   return(0);
40 }
```

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 characters. More precisely, it is of type `const char*`.

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

```
  char InFileName[20]="CPlusPlusTerms.txt";
2 ...
  std::cout << "Processing the contents of "
4           << InFileName << std::endl;
  ...
6 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.

```
std::ifstream InFile;
std::string InFileName;

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 begining.
```

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  }
```

Alternatively, we could 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 make 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 vary 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-big 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
   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  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  1  0  0  1  0  0  1  0
   0  1  0  0  0  0  1  0  0  0  0  0  0  0  0  1  0  0  1  0  0  1  0  0  1  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
   0  1  0  0  0  0  0  0  0  1  0  0  0  0  1  0  0  1  0  0  0  0  0  1  0
9  0  1  0  0  0  0  0  0  0  1  0  0  0  0  1  0  0  1  0  0  0  0  0  1  0
   0  1  1  1  1  0  1  1  1  1  0  1  1  1  1  0  1  1  1  0  0  0  0  1  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  0
```

- 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

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

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

6. Portable Bitmap Format (pbm)

03FlipPBM.cpp

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

6. Portable Bitmap Format (pbm)

03FlipPBM.cpp

```
    // Open the output file
34  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);
```