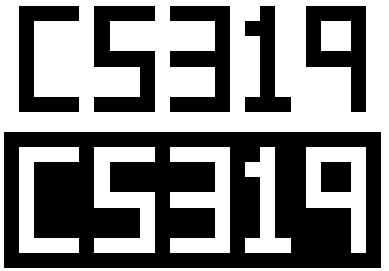
CS319: Scientific Computing

Week 9: Strings; Files and Streams; Vectors and Matrices

Dr Niall Madden

9am and **4pm**, 06 March, 2024



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

Note: some of these slides for for SELF STUDY: read through them all!

Outline

News and Updates

- Lab 4: grades for Lab 4 have been posted.
- Class test: grades have been posted.
- Lab 6: Deadline pushed out to 17:00, Thursday 7 March.
- Project topics: deadline pushed out to 17:00, Wed 6 March (today!). Anything submitted after that will score zero.
- Project proposals due **17:00, Tuesday 12 March**. See Slide 9 of niallmadden.ie/2324-CS319/2324-CS319-Projects.pdf
- Submit the proposal at `https://universityofgalway.instructure.com/courses/12359/assignments/65516`

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,

```
stack s1, s2; // stack is the class and
                 s1, s2 are objects.
```

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

```
s1.pop(); s2.push();   // pop & push are methods.
```

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

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

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 no so easy to work with, so C++ provides its one `string` class.

The `string` class is one that is "built-in" to the C++ language, and can be accessed once the `string` header file is included.

We have used it before, 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`. Eg s.substr(2,3) returns "llo".
- `find()` which finds the first occurence on one substring in another. s.find("l") retuns 2.
- `c_str()` return the "C string" version.

Eg: std::string s = "Hello";
then s.length() returns 5.

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);
}
```

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: `=` Eg `s1 = "hello"; s2 = "goodbye".`
- Relational: `==`, `>`, `<`, etc; Eg `(s1 > s2)` is true because `s1` comes after `s2` lexiographically ("alphabetically").
- Arithmetic: `+`, `+=`

Eg `s3 = s1 + " and " + s2;`

Set `s3` to be "hello and goodbye"

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);
}
```

I/O streams as objects `cout, cin`

`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`

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.

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 start by looking at a simple example:

(i) open a file,

(ii) count the number of characters,

(iii) save this number to a new file.

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

Care is needed to make sure we download "CPlusPLusTerms.txt" to the correct location.

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

To **read** from a file, declare an object of type `ifstream`; to **write** to a file, declare an object of type `ofstream`.

Open the file by calling the `open()` member function.

02

02CountChars.cpp

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

12  int main(void )
    {
14    std::ifstream InFile;
      std::ofstream OutFile;
16    char c;

18    std::cout << "Processing ..."
           << " CPlusPlusTerms.txt";
20    std::cout << "See file Output.txt for"
           << " more information.";
22    InFile.open("CPlusPlusTerms.txt");
      OutFile.open("Output.txt");

      int i=0;
26    InFile.get( c );
```

New objects InFile, of type ifstream

OutFile, of type ofstream

get() is a method.

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()`

eof = "End of file"

01CountChars.cpp

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

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

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

38   return(0);
}
```

just like cout

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";
...
std::cout << "Processing the contents of "
          << InFileName << std::endl;
...
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);
}
```

cout

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

Finished here at 9am