Introduction to Scientific Programming with C++

Session 4: Input, output and text manipulation

Martin Uhrin

UCL

February 11-13th 2013

Table of Contents

1 Output

2 Input

3 Strings revisited

Writing to a file

Let's say we wanted to gift someone with an ASCII caterpillar, here's how:

---

[1]For full list see http://en.cppreference.com/w/cpp/language/escape

Writing to a file

Let's say we wanted to gift someone with an ASCII caterpillar, here's how:

```cpp
#include <iostream>
#include <fstream>

int main () {
  std::ofstream myFile("caterpillar.txt");
  myFile << "Here, have a caterpillar:\n\n";
  myFile << "\\ \n";
  myFile << "  '-.__.-' \n";
  myFile << " /oo |--.--,--,--. \n";
  myFile << " \\_.-'._i__i__i_.' \n";
  myFile << "       \"\"\"\"\"\"\"\" \n";
  if(myFile.is_open())
    myFile.close();
  return 0;
}
```

../code/4_io/lectures/simple_output.cpp

[1] For full list see http://en.cppreference.com/w/cpp/language/escape

Writing to a file

Let's say we wanted to gift someone with an ASCII caterpillar, here's how:

```cpp
#include <iostream>
#include <fstream>

int main () {
  std::ofstream myFile("caterpillar.txt");
  myFile << "Here, have a caterpillar:\n\n";
  myFile << "\\ \n";
  myFile << " '-.__.-' \n";
  myFile << " /oo |--.--,--,--. \n";
  myFile << " \\_.-'._i__i__i_.' \n";
  myFile << "        \"\"\"\"\"\"\"\"\"\"\"\"\"\" \n";
  if(myFile.is_open())
    myFile.close();
  return 0;
}
```

../code/4_io/lectures/simple_output.cpp

Doesn't look like much does it? That's because I've had to use escape characters to print some of the symbols: \ is \\ and " is \" [1]

[1]For full list see http://en.cppreference.com/w/cpp/language/escape

Introduction to ofstream

Output **f**ile **stream**

All of the action happens with the ofstream object. Let's have a closer look.

---

[2]See http://en.cppreference.com/w/cpp/io/ios_base/openmode for a full list.

Introduction to ofstream
Output **f**ile **stream**

All of the action happens with the ofstream object. Let's have a closer look.

Constructing ofstream

```
ofstream ();
ofstream ( const char * filename , ios_base :: openmode mode =
    ios_base :: out );
```

Two ways to construct an ofstream. Second is equivalent to:

```
std :: ofstream myFile ;
myFile . open ( filename , std :: ios_base :: out );
```

The openmode tells ofstream how you want to open the file[2].

---

[2]See http://en.cppreference.com/w/cpp/io/ios_base/openmode for a full list.

Introduction to `ofstream`

Opening a file stream

```
void open(const char * filename, ios_base::openmode mode
    = ios_base::out);
```

Does what it says on the tin: opens a file called `filename` with mode `mode`.

---

[3]See http://en.cppreference.com/w/cpp/io/basic_ostream/operator_ltlt for full list.

Introduction to `ofstream`

Opening a file stream

```cpp
void open(const char * filename, ios_base::openmode mode
    = ios_base::out);
```

Does what it says on the tin: opens a file called `filename` with mode mode.

Writing to a file: the insertion operator

```cpp
ostream & operator <<(/*..stuff..*);
```

Use this to operator to insert text, numbers, strings and other things[3] into a file.

---

[3]See http://en.cppreference.com/w/cpp/io/basic_ostream/operator_ltlt for full list.

Introduction to `ofstream`

Closing a file

```
void close();
```

Tells the operating system to close the file because we no longer need it.

Introduction to `ofstream`

Closing a file

```
void close();
```

Tells the operating system to close the file because we no longer need it.

Warning!

- You do not need to close files, the fstream destructor will do it for you when you leave the current scope.

Introduction to `ofstream`

Closing a file

```
void close();
```

Tells the operating system to close the file because we no longer need it.

Warning!

- You do not need to close files, the fstream destructor will do it for you when you leave the current scope.

- If you want to close a file manually check to see if the file is open before trying to close it:

  ```
  if(myFile.is_open()){
    myFile.close();
  }
  ```

  Calling close on a file that is not open is an error!

Reading from a file

Reading from a text file is relatively easy. Let's dive in:

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main () {
  std::string line;
  std::ifstream
      myFile("caterpillar.txt");

  if(myFile.is_open())
  {
    while(myFile.good())
    {
      std::getline(myFile, line);
      std::cout << line << std::endl;
    }
    myFile.close();
  }
  else
    std::cout << "Unable to open";

  return 0;
}
```

../code/4_io/lectures/simple_input.cpp

Reading from a file

Reading from a text file is relatively easy. Let's dive in:

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main () {
  std::string line;
  std::ifstream
      myFile("caterpillar.txt");

  if(myFile.is_open())
  {
    while(myFile.good())
    {
      std::getline(myFile, line);
      std::cout << line << std::endl;
    }
    myFile.close();
  }
  else
    std::cout << "Unable to open";

  return 0;
}
```

../code/4_io/lectures/simple_input.cpp

Output:

```
Here, have a caterpillar:

    \
     '-.__.-'
    /oo |--.--,--,--.
    \_.-'._i__i__i_.'
        """""""""
```

Introduction to `ifstream`

The `ifstream` (input file stream) class is where most of the magic happened. Let's take a closer look:

Introduction to `ifstream`

The `ifstream` (input file stream) class is where most of the magic happened. Let's take a closer look:

Constructing ifstream

```
ifstream ();
ifstream ( const char * filename , ios_base :: openmode mode =
    ios_base :: in );
```

To create one we give it a file name. Second argument tells `ifstream` how to open the file.

Introduction to `ifstream`

The `ifstream` (input file stream) class is where most of the magic happened. Let's take a closer look:

Constructing ifstream

```
ifstream ();
ifstream ( const char * filename , ios_base :: openmode mode =
    ios_base :: in );
```

To create one we give it a file name. Second argument tells `ifstream` how to open the file.

Checking if the file is ready to be read from

```
bool good () const ;
```

If there has been an error or the end of file has been reached, this will be false, otherwise true.

```
while ( myFile . good ())
```

Introduction to `ifstream`

Reading a line from the file

```
ifstream & getline ( istream & is , string & str , char delim );
ifstream & getline ( istream & is , string & str );
```

This function reads characters from an input stream (is) until a delimiter (delim) is found. The result is placed in the string str. If you call it again it will continue from where it left off.

---

[4] http://www.cplusplus.com/reference/iostream/ifstream/

Introduction to `ifstream`

Reading a line from the file

```
ifstream & getline ( istream & is , string & str , char delim );
ifstream & getline ( istream & is , string & str );
```

This function reads characters from an input stream (is) until a delimiter (delim) is found. The result is placed in the string str. If you call it again it will continue from where it left off.
The overloaded version with no third parameters assumes the delimiter to be \n (i.e. the new line character).

---

[4]http://www.cplusplus.com/reference/iostream/ifstream/

Introduction to `ifstream`

Reading a line from the file

```
ifstream & getline ( istream & is , string & str , char delim );
ifstream & getline ( istream & is , string & str );
```

This function reads characters from an input stream (is) until a delimiter (delim) is found. The result is placed in the string str. If you call it again it will continue from where it left off.
The overloaded version with no third parameters assumes the delimiter to be \n (i.e. the new line character).
This is *not* part of ifstream, it's actually in <string>.

```
std :: getline ( myFile , line );
```

These are just some of the many things `ifstream` can do, see [4] for a complete look at the class.

---

[4] http://www.cplusplus.com/reference/iostream/ifstream/

Input
Reading from a file

Let's have another look:

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main () {
  std::string line;
  std::ifstream
      myFile("caterpillar.txt");

  if(myFile.is_open())
  {
    while(myFile.good())
    {
      std::getline(myFile, line);
      std::cout << line << std::endl;
    }
    myFile.close();
  }
  else
    std::cout << "Unable to open";

  return 0;
}
```

Output:

```
Here, have a caterpillar:

    \
     '_.__.-'
    /oo |--.--,--,--.
    \_.-'._i__i__i_.'
         """""""""
```

Strings revisited

How long is a piece of string? myString.length() of course

`strings` that we saw earlier are actually classes. There are many useful things we can do with strings. Here are a couple:

Strings revisited

How long is a piece of string? myString.length() of course

`strings` that we saw earlier are actually classes. There are many useful things we can do with strings. Here are a couple:

Get the length

```
size_t size();
size_t length();
```

Get the number of characters in a string. `size_t` is a special type that roughly corresponds to an unsigned integer.

Strings revisited

How long is a piece of string? myString.length() of course

`strings` that we saw earlier are actually classes. There are many useful things we can do with strings. Here are a couple:

Get the length

```
size_t size ();
size_t length ();
```

Get the number of characters in a string. `size_t` is a special type that roughly corresponds to an unsigned integer.

Find content in the string

```
size_t find (const string & str , size_t pos = 0) const ;
```

Search for `str` within the current string, optionally starting after position pos, returning the position of the first occurrence in the string. If `str` is not found `string::npos` is returned.

Splitting up a strings
Step 1

Say we have a string made up of a series of numbers:

```
const std::string line("3.51 9.23 1.25 4.51");
```

How do we get the numbers out so we can deal with them as `double`s and, say, calculate their sum?

Splitting up a strings
Step 1

Say we have a string made up of a series of numbers:

```
const std::string line("3.51 9.23 1.25 4.51");
```

How do we get the numbers out so we can deal with them as `double`s and, say, calculate their sum?
By using the ever handy `stringstream` class:

Step 1: Split the string up using space as a delimiter

```
const std::string line("3.51 9.23 1.25 4.51");
std::string numString;
std::stringstream stream(line);

double num, sum = 0.0;
while(std::getline(stream, numString, ' '))
{
```

Splitting up a strings
Step 2

Step 2: Convert each number string to a double

```
{
   std::stringstream numStream(numString);
   if(!(numStream >> num)){
     num = 0.0;
   }
   sum += num;
}
```

Here we're using the extraction operator (>>) to extract a `double` type
from the `stringstream`.

Splitting up a strings
Step 2

Step 2: Convert each number string to a double

```
{
   std::stringstream numStream(numString);
   if(!(numStream >> num)){
     num = 0.0;
   }
   sum += num;
}
```

Here we're using the extraction operator (>>) to extract a `double` type from the stringstream.

We can check if it failed by using the not operator (!). Remember because this is a class operator we could write:

```
numStream.operator >>(num);
```

To use the stringstream class make just include the <sstream> header.

Splitting up a string

```cpp
#include <iostream>
#include <string>
#include <sstream>

int main()
{
  const std::string line("3.51 9.23 1.25 4.51");
  std::string numString;
  std::stringstream stream(line);

  double num, sum = 0.0;
  while(std::getline(stream, numString, ' '))
  {
    std::stringstream numStream(numString);
    if(!(numStream >> num)){
      num = 0.0;
    }
    sum += num;
  }
  std::cout << "Sum is: " << sum << "\n";

  return 0;
}
```

Output:

Sum is: 18.5

../code/4_io/lectures/parse_numbers.cpp