C++: Input/Output (I/O)

ens-lal

2013

Outline

- Input/Output on screen
  - inputs
  - outputs
- files
- buffered streams
- stream manipulators

Output

```
#include <iostream>
int main(int argc, char **argv)
{
  std::cout << "Hello world" << std::endl;
  return 0;
}
```

- notice the `#include <iostream>`
- necessary to use I/O from the STL (Standard Template Library)
- notice the std:: namespace
  - all the components from STL are neatly folded under std::
- std::cout is the standard output stream (to the screen)
- the « operator says: *send the right-handside to the left-handside*
  - one can desy-chain the outputs
- std::endl applies 2 operations:
  - carriage return + new line
  - flush the buffer(s)

Output - for classes

How to define the << operator for a class ?

```
#include <iostream>

class A {
  int m_data;
  friend
  std::ostream& operator<<(std::ostream& out, const A& a)
  { out << "A{data=" << a.m_data << "}"; return out; }
 public:
  A(int i=0) : m_data(i) {}
};
```

Output - for classes - II

```
#include <iostream>
#include <string>
int main(int argc, char **argv)
{
  int i = 1;                    float f = 2.3f;
  std::string s = "abcd";       A a(42);
  std::cout << " i = " << i << " ; f = " << f << std::endl
            << " s = " << s << " ; a = " << a << std::endl;
  return 0;
}
```

- the operator<< is clever enough to "recognize" and "adapt" to the type of the object being printed (`string`, `float`, `int` and `class A`)

```
$ ./io-1
 i = 1 ; f = 2.3
 s = abcd ; a = A{data=42}
```

Input - keyboard

- std::cin is the standard input
- operator» is the insertion operator and means *send into ...*
- desy-chained inputs are separated by a separation character (space, tabulation, carriage return)

```
#include <iostream>
#include <string>
int main(int argc, char **argv)
{
  int i = 1; double d = 2.3; std::string s = "abcd";
  std::cin >> i >> d >> std::ws;
  std::getline( std::cin, s );

  std::cout << "[" << i << "|" << d << "|" << s << "]"
            <<  std::endl;
  return 0;
}
```

Input - keyboard - II

```
#include <iostream>
#include <string>
int main(int argc, char **argv)
{
  int i = 1; double d = 2.3; std::string s = "abcd";
  std::cin >> i >> d >> std::ws;
  std::getline( std::cin, s );

  std::cout << "[" << i << "|" << d << "|" << s << "]"
            <<  std::endl;
  return 0;
}
```

- one can also use a stream manipulator
  - std::ws : extracts as many white-spaces as possible
- note that one may need to process by-hand the strings which might contain themselves separators

Input - keyboard - III

```cpp
#include <iostream>
#include <string>
int main(int argc, char **argv)
{
  int i = 1; double d = 2.3; std::string s = "abcd";
  std::cin >> i >> d >> std::ws;
  std::getline( std::cin, s );

  std::cout << "[" << i << "|" << d << "|" << s << "]"
            <<  std::endl;
  return 0;
}
```

```
$ ./io-2
42 66.6 asd easdf
[42|66.6|asd easdf]
```

Files

```
#include <fstream>
#include <string>
int main(int argc, char **argv)
{
  int i = 1;   double d = 2.3;
  std::string s = "abcd efgh";
  std::ofstream f( "file.txt" );

  if (f.is_open()) {
    f << i << " " << d << std::endl << s << std::endl;
    f.close();
  }
  return 0;
}
```

Files

```
#include <fstream>
...
std::ofstream f("file.txt"); // open a file (for writing)
```

- std::ifstream and std::ofstream are (resp.) the input and output file streams
- operator to write to and operator to read from.

```
f << i; // writes an integer into f
...
f.close()
```

- f.close() is needed because the destructor of f won't do it...

Files (cont'd)

```
#include <iostream>
#include <fstream>
#include <string>
int main(int argc, char **argv) {
  int i; double d; std::string s;
  std::ifstream f( "file.txt" );
  if (f.is_open()) {
    if (!f.eof()) { f >> i; }
    if (!f.eof()) { f >> d; }
    if (!f.eof()) { f >> std::ws; }
    if (!f.eof()) { std::getline( f, s ); }
    f.close();
  }
  std::cout << "[" << i << "|" << d << "|" << s << "]"
            << std::endl;
  return 0;
}
```

Files (cont'd)

```
if (!f.eof()) /* ... */
```

- always test whether the stream has not reached the end of file (EOF)
- generally speaking, it is also usually necessary to special-case by hand, after the line-by-line reading of the inputs...

```
$ ./io-3
$ cat file.txt
1 2.3
abcd efgh

$ ./io-4
[1|2.3|abcd efgh]
```

Buffered streams

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
int main(int argc, char **argv) {
  int i; double d; std::string s;
  std::string input = "123 1.23 abcd def";

  std::istringstream b (input);
  b >> i >> d >> std::ws;
  std::getline( b, s );

  std::cout << std::hex
            << "[" << i << "|" << d << "|" << s << "]"
            << std::endl;
  return 0;
}
```

Buffered streams

- std::istringstream and std::ostringstream are (resp.) input and output buffered streams which (resp.) read-from/write-to a std::string
- as for the other streams, once can use the << and >> operators to (resp.) write into and read from them.
- all these operations are applied to a dynamic std::string
- one can convert these streams into a std::string
  - using the std::string constructor
  - or by calling the str() method

```
$ ./io-5
[7b|1.23|abcd def]
```

Manipulators

Inject behavior modifications into the streams.

- `boolalpha`: display the booleans as `true` or `false`
- `dec`: use decimal base
- `fixed`: use fixed-point notation (w/o exponant)
- `hex`: use hexadecimal base
- `internal`: adjust field by inserting characters at an internal position
- `left`: adjust output to the left
- `oct`: use octal base
- `right`: adjust output to the right
- `scientific`: use scientific notation (w/ exponant)
- `showbase`: show numerical base prefixes
- `showpoint`: show decimal point
- `showpos`: show positive signs
- `skipws`: skip whitespaces
- `unitbuf`: flush buffer after insertions
- `uppercase`: generate upper-case letters

Manipulators (cont'd)

```
int i = 1234;
std::cout << std::hex << "0x" << i << std::endl;
```

```
$ ./io-6
0x4d2
```

Questions ?

Questions ?