CME 211: Lecture 12

Topics:

- Additional file I/O options in C++
- Functions
- Preprocessor & `#include` statements

Command line arguments

```
#include <iostream>

int main(int argc, char *argv[]) {
  // Display the command line arguments
  for (int n = 0; n < argc; n++) {
    std::cout << n << " " << argv[n] << std::endl;
  }
  return 0;
}
```

Output:

```
$ ./argv1 hello.txt 3.14 42
0 ./argv1
1 hello.txt
2 3.14
3 42
```

Command line arguments

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

int main(int argc, char *argv[]) {
  if (argc < 4) {
    std::cout << "Usage:" << std::endl;
    std::cout << " " << argv[0] << " <filename> <param1> <param2>" << std::endl;
    return 0;
  }

  std::string filename = argv[1];
  double param1 = std::stof(argv[2]);
  int param2 = std::stoi(argv[3]);

  std::cout << "filename = " << filename << std::endl;
  std::cout << "param1 = " << param1 << std::endl;
  std::cout << "param2 = " << param2 << std::endl;

  return 0;
}
```

Output:

```
$ g++ -std=c++11 -Wall -Wconversion -Wextra argv2.cpp -o argv2
$ ./argv2 hello.txt 3.14 42
```

```
filename = hello.txt
param1 = 3.14
param2 = 42
```

Formatting

```
#include <iostream>

int main() {
  double a = 2.;
  std::cout << "a = " << a << std::endl;
  return 0;
}
```

Output:

```
$ ./formatting1
a = 2
```

Showing decimal point

```
#include <iostream>

int main() {
  double a = 2.;
  std::cout.setf(std::ios::showpoint);
  std::cout << "a = " << a << std::endl;
  return 0;
}
```

Output:

```
$ ./formatting2
a = 2.00000
```

Showing decimal point

```
#include <iostream>

int main() {
  double a = 2., b = 3.14;
  int c = 4;

  std::cout.setf(std::ios::showpoint);

  std::cout << "a = " << a << std::endl;
  std::cout << "b = " << b << std::endl;
  std::cout << "c = " << c << std::endl;

  return 0;
}
```

Output:

```
$ ./formatting3
a = 2.00000
b = 3.14000
c = 4
```

Controlling decimal places

```
#include <iostream>

int main() {
  double a = 2., b = 3.14;
  int c = 4;

  //Always show 3 decimal places
  std::cout.setf(std::ios::fixed, std::ios::floatfield);
  std::cout.setf(std::ios::showpoint);
  std::cout.precision(3);

  std::cout << "a = " << a << std::endl;
  std::cout << "b = " << b << std::endl;
  std::cout << "c = " << c << std::endl;

  return 0;
}
```

Output:

```
$ ./formatting4
a = 2.000
b = 3.140
c = 4
```

Scientific notation

```
int main() {
  double a = 2., b = 3.14;
  int c = 4;

  std::cout.setf(std::ios::scientific, std::ios::floatfield);
  std::cout.precision(3);

  std::cout << "a = " << a << std::endl;
  std::cout << "b = " << b << std::endl;
  std::cout << "c = " << c << std::endl;

  return 0;
}
```

Output:

```
$ ./formatting5
a = 2.000e+00
b = 3.140e+00
c = 4
```

Field width

```
#include <iostream>

int main() {
  double a = 2., b = 3.14;
  int c = 4;

  std::cout.setf(std::ios::scientific, std::ios::floatfield);
  std::cout.precision(3);

  std::cout << "a = " << a << std::endl;
  std::cout.width(15);
  std::cout << "b = " << b << std::endl;
  std::cout.width(30);
  std::cout << "c = " << c << std::endl;

  return 0;
}
```

Output:

```
$ ./formatting6
a = 2.000e+00
           b = 3.140e+00
                          c = 4
```

Fill character

```
#include <iomanip>
#include <iostream>

int main() {

  std::cout.fill('0');

  for(int n = 0; n < 10; n++) {
    std::cout << std::setw(2) << n << std::endl;
  }

  return 0;
}
```

Output:

```
$ ./formatting7
00
01
02
...
```

`cout` and files work the same

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

```
int main() {
  double a = 2., b = 3.14;
  int c = 4;

  std::ofstream f("formatting.txt");
  f.setf(std::ios::showpoint);

  f << "a = " << a << std::endl;
  f << "b = " << b << std::endl;
  f << "c = " << c << std::endl;

  f.close();

  return 0;
}
```

Output:

```
$ ./formatting8
$ cat formatting.txt
a = 2.00000
b = 3.14000
c = 4
```

More on reading data

Loading a table

Remember the Movielens data?

```
$ cat u.data
196 242 3   881250949
186 302 3   891717742
22  377 1   878887116
244 51  2   880606923
166 346 1   886397596
298 474 4   884182806
115 265 2   881171488
253 465 5   891628467
305 451 3   886324817
6   86  3   883603013
```

Same data on each line

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

int main() {

  std::ifstream f;
  f.open("u.data");
  if (f.is_open()) {
    int uid, mid, rating, time;
```

```
    while (f >> uid >> mid >> rating >> time) {
      std::cout << "user = " << uid;
      std::cout << ", movie = " << mid;
      std::cout << ", rating = " << rating << std::endl;
    }
    f.close();
  }
  else {
    std::cerr << "ERROR: Failed to open file" << std::endl;
  }
  return 0;
}
```

Output:

```
$ ./file1
user = 196, movie = 242, rating = 3
user = 186, movie = 302, rating = 3
user = 22, movie = 377, rating = 1
user = 244, movie = 51, rating = 2
user = 166, movie = 346, rating = 1
user = 298, movie = 474, rating = 4
user = 115, movie = 265, rating = 2
user = 253, movie = 465, rating = 5
user = 305, movie = 451, rating = 3
user = 6, movie = 86, rating = 3
```

Different data types

See `src/dist.female.first`:

```
MARY           2.629  2.629      1
PATRICIA       1.073  3.702      2
LINDA          1.035  4.736      3
BARBARA        0.980  5.716      4
ELIZABETH      0.937  6.653      5
JENNIFER       0.932  7.586      6
MARIA          0.828  8.414      7
SUSAN          0.794  9.209      8
MARGARET       0.768  9.976      9
DOROTHY        0.727 10.703     10
LISA           0.704 11.407     11
NANCY          0.669 12.075     12
KAREN          0.667 12.742     13
BETTY          0.666 13.408     14
```

Be careful with data types

```
std::ifstream f;

f.open("dist.female.first");
if (f.is_open()) {
  std::string name;
  double perc1, perc2;
```

```
  int rank;
  while (f >> name >> perc1 >> perc2 >> rank) {
    std::cout << name << ", " << perc1 << std::endl;
  }
  f.close();
}
else {
  std::cerr << "ERROR: Failed to open file" << std::endl;
}
```

Step by step extraction

What if lines have a varying amount of data to load?

```
$ cat geometry1.txt
workspace 0 0 10 10
circle 3 7 1
triangle 4 6 8 6 5 7
rectangle 1 1 8 2
$ cat geometry2.txt
workspace 0 0 10 10
circle 3 7 1
line 0 0 3 2
rectangle 1 1 8 2
```

Step by step extraction

```
f.open(filename);
if (f.is_open()) {
  std::string shape;
  while (f >> shape) {
    int nval;
    // Determine the shape and how many values need to be read
    if (shape == "workspace" or shape == "rectangle")
      nval = 4;
    else if (shape == "circle")
      nval = 3;
    else if (shape == "triangle")
      nval = 6;
    else {
      std::cerr << "ERROR: Unknown shape '" << shape;
      std::cerr << "'" << std::endl;
      return 1;
    }

  // Read appropriate number of values
  float val[6];
  for (int n = 0; n < nval; n++) {
    f >> val[n];
  }
```

Read line by line

```
f.open(filename);
if (f.is_open()) {
  std::string line;
  while (getline(f, line)) {
    std::cout << line << std::endl;
  }
  f.close();
}
else {
  std::cerr << "ERROR: Failed to open file" << std::endl;
}
```

Read line by line

```
$ ./file4 geometry1.txt
workspace 0 0 10 10
circle 3 7 1
triangle 4 6 8 6 5 7
rectangle 1 1 8 2
$ ./file4 geometry2.txt
workspace 0 0 10 10
circle 3 7 1
line 0 0 3 2
rectangle 1 1 8 2
```

String stream

```
f.open(filename);
if (f.is_open()) {
  // Read the file one line at a time
  std::string line;
  while (getline(f, line)) {
    // Use a string stream to extract text for the shape
    std::stringstream ss;
    ss << line;
    std::string shape;
    ss >> shape;

    // Determine how many values need to be read
    int nval;
    if (shape == "workspace" or shape == "rectangle")
    nval = 4;
    ...
else {
  std::cerr << "ERROR: Unknown shape '" << shape;
  std::cerr << "'" << std::endl;
  return 1;
}
// Read appropriate number of values
float val[6];
```

```
for (int n = 0; n < nval; n++)
  ss >> val[n]
```

Output:

```
$ ./extraction1
Usage:
./extraction1 <name data> [nnames]

Read at most nnames (optional)
```

Convert argument to number

```
#include <limits>

int main(int argc, char *argv[]) {
  if (argc < 2) {
    std::cout << "Usage:" << std::endl;
    std::cout << " " << argv[0] << " <name data> [nnames]" << std::endl << std::endl;
    std::cout << " Read at most nnames (optional)" << std::endl;
    return 0;
  }

  // Setup string for the filename to be read
  std::string filename = argv[1];

  // Determine maximum number of names to read
  int nnames = std::numeric_limits<int>::max();
  if (argc == 3) {
    nnames = std::stoi(argv[2]);
  }

  std::ifstream f;
  f.open(filename);
```

Convert argument to number

```
$ ./extraction1 dist.female.first
Read 10 names.
$ ./extraction1 dist.female.first 7
Read 7 names.
$ ./extraction1 dist.female.first 3
Read 3 names.
```

Testing extraction

```
#include <iostream>
#include <sstream>

int main(int argc, char *argv[]) {
  // Setup a string stream to access the command line argument
  std::string arg = argv[1];
  std::stringstream ss;
```

```
  ss << arg;

  // Attempt to extract an integer from the string stream
  int n = 0;
  ss >> n;
  std::cout << "n = " << n << std::endl;

  return 0;
}
```

Testing extraction

```
$ ./extraction2 42
n = 42
$ ./extraction2 -17
n = -17
$ ./extraction2 hello
n = 0
```

Extraction failures

```
#include <iostream>
#include <sstream>

int main(int argc, char *argv[]) {
  // Setup a string stream to access the command line argument
  std::string arg = argv[1];
  std::stringstream ss;
  ss << arg;

  // Attempt to extract an integer from the string stream
  int n = 0;
  if (ss >> n)
    std::cout << "n = " << n << std::endl;
  else
    std::cerr << "ERROR: string stream extraction failed" << std::endl;

  return 0;
}
```

Extraction failures

```
$ ./extraction3
n = 42
$ ./extraction3
n = -17
$ ./extraction3
ERROR: string stream extraction failed
$ ./extraction3
n = 3
```

Reading

- **C++ Primer, Fifth Edition** by Lippman et al.
- Chapter 8: The IO Library
- Chapter 17: Specialized Library Facilities: Section 17.5.1