Lecture 12: File IO in C++

November 3rd, 2018

Topics Introduced: Command line arguments, string formatting, File IO, functions, the pre-processor and `#include`.

0.1 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
```

0.1.1 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;
```

```
20 }
```

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
```

0.2 Formatting

```
1 #include <iostream>
2
3 int main() {
4   double a = 2.;
5   std::cout << "a = " << a << std::endl;
6   return 0;
7 }
```

Output:

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

0.2.1 Showing decimal point

```
1 #include <iostream>
2
3 int main() {
4   double a = 2.;
5   std::cout.setf(std::ios::showpoint);
6   std::cout << "a = " << a << std::endl;
7   return 0;
8 }
```

Output:

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

0.2.2 Showing decimal point

```
1 #include <iostream>
2
3 int main() {
4   double a = 2., b = 3.14;
5   int c = 4;
6
```

```
7     std::cout.setf(std::ios::showpoint);
8
9     std::cout << "a = " << a << std::endl;
10    std::cout << "b = " << b << std::endl;
11    std::cout << "c = " << c << std::endl;
12
13    return 0;
14  }
```

Output:

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

0.2.3 Controlling decimal places

```
1   #include <iostream>
2
3   int main() {
4     double a = 2., b = 3.14;
5     int c = 4;
6
7     //Always show 3 decimal places
8     std::cout.setf(std::ios::fixed, std::ios::floatfield);
9     std::cout.setf(std::ios::showpoint);
10    std::cout.precision(3);
11
12    std::cout << "a = " << a << std::endl;
13    std::cout << "b = " << b << std::endl;
14    std::cout << "c = " << c << std::endl;
15
16    return 0;
17  }
```

Output:

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

0.2.4 Scientific notation

```
1   int main() {
2     double a = 2., b = 3.14;
3     int c = 4;
4
5     std::cout.setf(std::ios::scientific, std::ios::floatfield);
6     std::cout.precision(3);
7
8     std::cout << "a = " << a << std::endl;
9     std::cout << "b = " << b << std::endl;
```

```
10    std::cout << "c = " << c << std::endl;
11
12    return 0;
13  }
```

Output:

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

0.2.5 Field width

```
1   #include <iostream>
2
3   int main() {
4     double a = 2., b = 3.14;
5     int c = 4;
6
7     std::cout.setf(std::ios::scientific, std::ios::floatfield);
8     std::cout.precision(3);
9
10    std::cout << "a = " << a << std::endl;
11    std::cout.width(15);
12    std::cout << "b = " << b << std::endl;
13    std::cout.width(30);
14    std::cout << "c = " << c << std::endl;
15
16    return 0;
17  }
```

Output:

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

0.2.6 Fill character

```
1   #include <iomanip>
2   #include <iostream>
3
4   int main() {
5
6     std::cout.fill('0');
7
8     for(int n = 0; n < 10; n++) {
9       std::cout << std::setw(2) << n << std::endl;
10    }
11
12    return 0;
13  }
```

Output:

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

0.2.7 `cout` and files work the same

```
1  #include <iostream>
2  #include <fstream>
3
4  int main() {
5    double a = 2., b = 3.14;
6    int c = 4;
7
8    std::ofstream f("formatting.txt");
9    f.setf(std::ios::showpoint);
10
11   f << "a = " << a << std::endl;
12   f << "b = " << b << std::endl;
13   f << "c = " << c << std::endl;
14
15   f.close();
16
17   return 0;
18 }
```

Output:

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

0.3 More on reading data

0.3.1 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
```

0.3.2 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
```

0.3.3 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
```

0.3.4 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;
}
```

0.3.5 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
```

0.3.6 Step by step extraction

```
f.open(filename);
```

```
2  if (f.is_open()) {
3    std::string shape;
4    while (f >> shape) {
5      int nval;
6      // Determine the shape and how many values need to be read
7      if (shape == "workspace" or shape == "rectangle")
8        nval = 4;
9      else if (shape == "circle")
10       nval = 3;
11     else if (shape == "triangle")
12       nval = 6;
13     else {
14       std::cerr << "ERROR: Unknown shape '" << shape;
15       std::cerr << "'" << std::endl;
16       return 1;
17     }
18
19   // Read appropriate number of values
20   float val[6];
21   for (int n = 0; n < nval; n++) {
22     f >> val[n];
23   }
```

0.3.7 Read line by line

```
1  f.open(filename);
2  if (f.is_open()) {
3    std::string line;
4    while (getline(f, line)) {
5      std::cout << line << std::endl;
6    }
7    f.close();
8  }
9  else {
10   std::cerr << "ERROR: Failed to open file" << std::endl;
11 }
```

0.3.8 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
```

0.3.9 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)
```

0.3.10 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);
```

0.3.11 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.
```

0.3.12 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;
}
```

0.3.13 Testing extraction

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

0.3.14 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;
```

```
12    if (ss >> n)
13      std::cout << "n = " << n << std::endl;
14    else
15      std::cerr << "ERROR: string stream extraction failed" << std::endl;
16
17    return 0;
18  }
```

0.3.15 Extraction failures

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

0.4 Functions

- Functions allow us to decompose a program into smaller components
- It is easier to implement, test, and debug portions of a program in isolation
- Allows work to be spread among many people working mostly independently
- If done properly it can make your program easier to understand and maintain
  - Eliminate duplicated code
  - Reuse functions across multiple programs

0.4.1 C/C++ function

Example:

```
1  int sum(int a, int b) {
2    int c = a + b;
3    return c;
4  }
```

Components:

```
1  return_type function_name(argument_type1 argument_var1, ...) {
2     // function body
3     return return_var; // return_var must have return_type
4  }
```

0.4.2 `sum` function in use

`src/sum1.cpp`

```
#include <iostream>

int sum(int a, int b) {
  int c = a + b;
  return c;
}

int main() {
  int a = 2, b = 3;

  int c = sum(a,b);
  std::cout << "c = " << c << std::endl;

  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion sum1.cpp -o sum1
$ ./sum1
c = 5
```

0.4.3 Order matters

`src/sum2.cpp`:

```
#include <iostream>

int main() {
  int a = 2, b = 3;

  // the compiler does not yet know about sum()
  int c = sum(a,b);
  std::cout << "c = " << c << std::endl;

  return 0;
}

int sum(int a, int b) {
  int c = a + b;
  return c;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion sum2.cpp -o sum2
sum2.cpp: In function 'int main()':
sum2.cpp:7:18: error: 'sum' was not declared in this scope
  int c = sum(a,b);
                  ^
```

0.4.4 Function declarations and definitions

- A function *definition* is the code that implements the function
- It is legal to call a function if it has been defined or *declared* previously
- A function *declaration* specifies the function name, input argument type(s), and output type. The function *declaration* need not specify the implementation (code) for the function.

`src/sum3.cpp`:

```
1  #include <iostream>
2
3  // Forward declaration or prototype
4  int sum(int a, int b);
5
6  int main() {
7    int a = 2, b = 3;
8
9    int c = sum(a,b);
10   std::cout << "c = " << c << std::endl;
11
12   return 0;
13 }
14
15 // Function definition
16 int sum(int a, int b) {
17   int c = a + b;
18   return c;
19 }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion sum3.cpp -o sum3
$ ./sum3
c = 5
```

0.4.5 Data types

`src/datatypes1.cpp`

```
1  #include <iostream>
2
3  int sum(int a, int b) {
4    int c;
5    c = a + b;
6    return c;
7  }
8
9  int main() {
10   double a = 2.7, b = 3.8;
11
12   int c = sum(a,b);
13   std::cout << "c = " << c << std::endl;
14
```

```
15    return 0;
16  }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion datatypes1.cpp -o datatypes1
datatypes1.cpp: In function 'int main()':
datatypes1.cpp:14:18: warning: conversion to 'int' from 'double' may alter its value [-Wcon
  int c = sum(a,b);
                ^
datatypes1.cpp:14:18: warning: conversion to 'int' from 'double' may alter its value [-Wcon
$ ./datatypes1
c = 5
```

0.4.6 Implicit casting

src/datatypes2.cpp:

```
1   #include <iostream>
2
3   int sum(int a, int b) {
4     double c = a + b;
5     return c; // we are not returning the correct type
6   }
7
8   int main() {
9     double a = 2.7, b = 3.8;
10
11    int c = sum(a,b);
12    std::cout << "c = " << c << std::endl;
13
14    return 0;
15  }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion datatypes2.cpp -o datatypes2
datatypes2.cpp: In function 'int sum(int, int)':
datatypes2.cpp:6:10: warning: conversion to 'int' from 'double' may alter its value [-Wconv
  return c;
         ^
datatypes2.cpp: In function 'int main()':
datatypes2.cpp:13:18: warning: conversion to 'int' from 'double' may alter its value [-Wcon
  int c = sum(a,b);
                ^
datatypes2.cpp:13:18: warning: conversion to 'int' from 'double' may alter its value [-Wcon
$ ./datatypes2
c = 5
```

0.4.7 Explicit casting

src/datatypes3.cpp

```
#include <iostream>

int sum(int a, int b) {
  double c = a + b;
  return (int)c;
}

int main() {
  double a = 2.7, b = 3.8;

  int c = sum((int)a,(int)b);
  std::cout << "c = " << c << std::endl;

  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion datatypes3.cpp -o datatypes3
```

0.4.8 void

- Use the `void` keyword to indicate absence of data
- src/void1.cpp

```
#include <iostream>

void printHeader(void) {
  std::cout << "-------------------------" << std::endl;
  std::cout << "      MySolver v1.0      " << std::endl;
  std::cout << "-------------------------" << std::endl;
}

int main() {
  printHeader();
  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion void1.cpp -o void1
$ ./void1
-------------------------
      MySolver v1.0
-------------------------
```

0.4.9 `void` and `return`

`src/void2.cpp`:

```
#include <iostream>

void printHeader(void) {
  std::cout << "-------------------------" << std::endl;
  std::cout << "      MySolver v1.0      " << std::endl;
  std::cout << "-------------------------" << std::endl;
  return 0;
}

int main() {
  printHeader();
  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion void2.cpp -o void2
void2.cpp: In function 'void printHeader()':
void2.cpp:8:10: error: return-statement with a value, in function returning 'void' [-fpermi
  return 0;
         ^
```

0.4.10 `void` and `return`

`src/void3.cpp`:

```
#include <iostream>

void printHeader(void) {
  std::cout << "-------------------------" << std::endl;
  std::cout << "      MySolver v1.0      " << std::endl;
  std::cout << "-------------------------" << std::endl;
  return;
}

int main() {
  printHeader();
  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion void3.cpp -o void3
```

0.4.11 Ignoring return value

`src/ignore.cpp`:

```
#include <iostream>
```

```
2
3  int sum(int a, int b) {
4    int c = a + b;
5    return c;
6  }
7
8  int main() {
9    int a = 2, b = 3;
10
11   sum(a,b); // legal to ignore return value if you want
12
13   return 0;
14 }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion ignore.cpp -o ignore
$ ./ignore
```

0.4.12 Function scope

src/scope1.cpp:

```
1  #include <iostream>
2
3  int sum(void) {
4    // a and b are not in the function scope
5    int c = a + b;
6    return c;
7  }
8
9  int main() {
10   int a = 2, b = 3;
11
12   int c = sum();
13   std::cout << "c = " << c << std::endl;
14
15   return 0;
16 }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion scope1.cpp -o scope1
scope1.cpp: In function 'int sum()':
scope1.cpp:5:11: error: 'a' was not declared in this scope
  int c = a + b;
          ^
scope1.cpp:5:15: error: 'b' was not declared in this scope
  int c = a + b;
              ^
...
```

0.4.13 Global scope

src/scope2.cpp:

```
#include <iostream>

// an be accessed from anywhere in the file (bad, bad, bad)
int a;

void increment(void) {
  a++;
}

int main() {
  a = 2;

  std::cout << "a = " << a << std::endl;
  increment();
  std::cout << "a = " << a << std::endl;

  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion scope2.cpp -o scope2
$ ./scope2
a = 2
a = 3
```

0.4.14 Passing arguments

src/passing1.cpp:

```
#include <iostream>

void increment(int a) {
  a++;
  std::cout << "a = " << a << std::endl;
}

int main() {
  int a = 2;

  increment(a);
  std::cout << "a = " << a << std::endl;

  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion passing1.cpp -o passing1
$ ./passing1
a = 3
a = 2
```

0.4.15 Passing arguments

src/passing2.cpp:

```
#include <iostream>

void increment(int a[2]) {
  a[0]++;
  a[1]++;
}

int main() {
  int a[2] = {2, 3};

  std::cout << "a[0] = " << ", " << "a[1] = " << std::endl;
  increment(a);
  std::cout << "a[0] = " << ", " << "a[1] = " << std::endl;

  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion passing2.cpp -o passing2
$ ./passing2
a[0] = 2, a[1] = 3
a[0] = 3, a[1] = 4
a[0] = 3, a[1] = 4
```

0.4.16 Pass by value

- C/C++ default to pass by value, which means that when calling a function the arguments are copied
- However, you need to be careful and recognize what is being copied
- In the case of a number like `int a`, what is being copied is the value of the number
- For a static array like `int a[2]`, what is being passed and copied is the location in memory where the array data is stored
- Will discuss pass by reference when we get to data structures

0.4.17 Towards modularity

src/main4.cpp:

```
#include <iostream>

int sum(int a, int b);

int main() {
```

```
6     int a = 2, b = 3;
7
8     int c = sum(a,b);
9     std::cout << "c = " << c << std::endl;
10
11    return 0;
12  }
```

src/sum4.cpp:

```
1   int sum(int a, int b) {
2     int c = a + b;
3     return c;
4   }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion main4.cpp sum4.cpp -o sum4
$ ./sum4
c = 5
```

0.4.18 Maintaining consistency

src/main5.cpp:

```
1   #include <iostream>
2
3   int sum(int a, int b);
4
5   int main() {
6     int a = 2, b = 3;
7
8     int c = sum(a,b);
9     std::cout << "c = " << c << std::endl;
10
11    return 0;
12  }
```

src/sum5.cpp:

```
1   \begin{Highlighting}[]
2   \DataTypeTok{double}\NormalTok{ sum(}\DataTypeTok{double}\NormalTok{ a, }\DataTypeTok{double}\NormalTok{ b)
3     \DataTypeTok{double}\NormalTok{ c = a + b;}
4     \ControlFlowTok{return}\NormalTok{ c;}
5   \NormalTok{\}}
6   \end{Highlighting}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion main5.cpp sum5.cpp -o sum5
/tmp/ccCKlsvX.o: In function main':
main5.cpp:(.text+0x21): undefined reference to sum(int, int)'
collect2: error: ld returned 1 exit status
```

0.5 The preprocessor and `#include`

- We have used functionality from the C++ standard library for output to the screen using `cout`, performing I/O with files, using the string object, etc.
- A library is a collection of functions, data types, constants, class definitions, etc.
- Somewhat analogous to a Python module
- At a minimum, accessing the functionality of a library requires `#include` statements

0.5.1 `#include`

- So what actually happens when you put something like `#include <iostream>` in your file?
- `<iostream>` is a way of referring to a file called iostream that is part of the compiler installation and on the corn machines is found at `/usr/include/c++/4.8/iostream`
- These types of files are called include or header files and contains forward declarations (prototypes) of functions, class definitions, constants, etc.

0.5.2 Preprocessor

- Before files are processed by the compiler, they are run through the C preprocessor, `cpp`
- What does the preprocessor do?
- For one thing it processes those `#include` statements

0.5.3 Hacking the preprocessor

```
$ cpp -P goodbye.txt
Hello!

Goodbye!


$ cat hello.txt
Hello!
$ cat goodbye.txt
#include "hello.txt"
Goodbye!
```

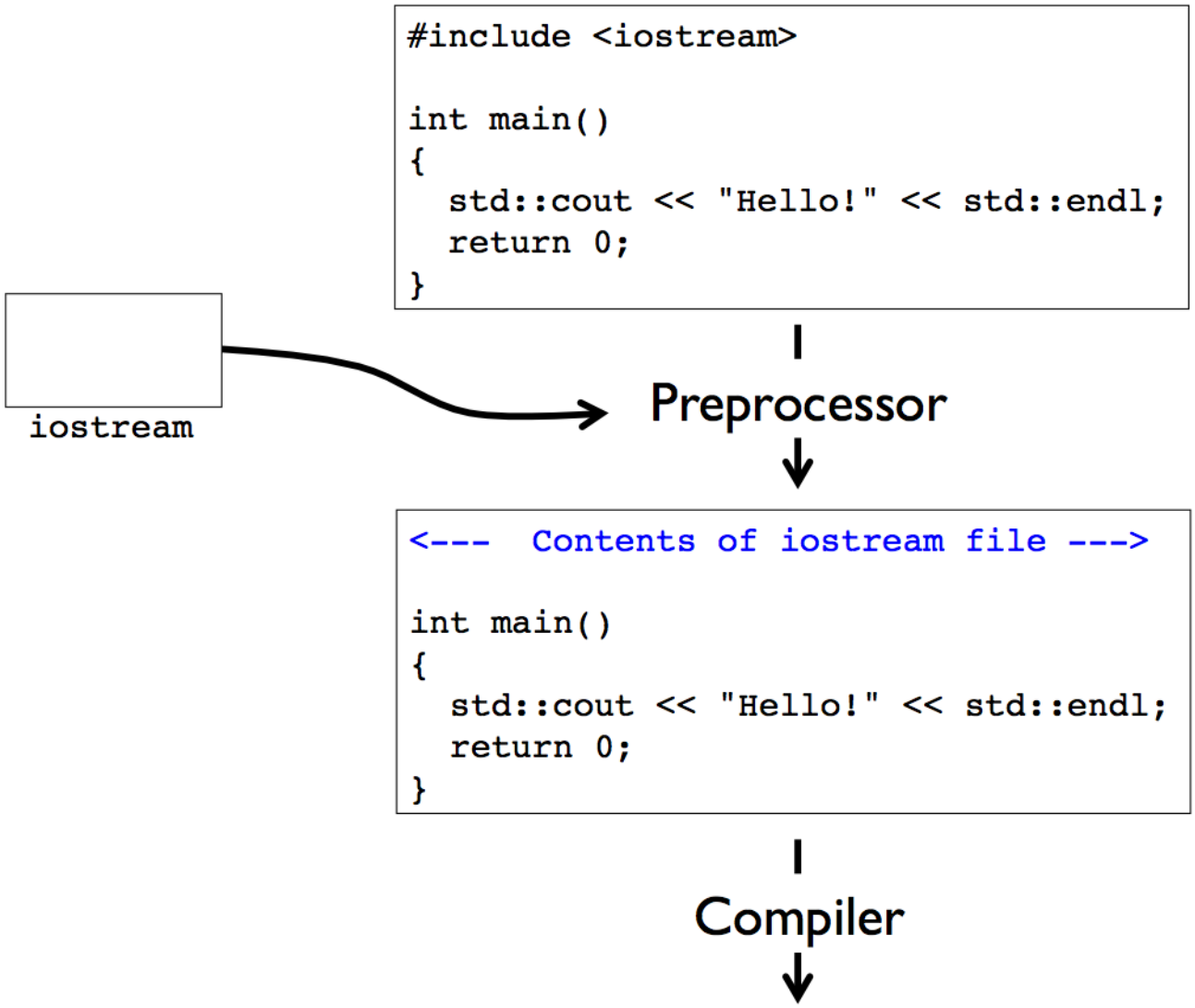

Figure 1: fig

```
$ cpp -P goodbye.txt
Hello!

Goodbye!
```

0.5.4 Compilation process

0.5.5 Standard decomposition

- Function (and type) *declarations* go in header (`.hpp`) files
- Function *definitions* go in source (`.cpp`) files

- Source files that want to use the functions must `#include` the header

`src/main6.cpp`:

```
\begin{Highlighting}[]

\end{Highlighting}
```

`src/sum6.hpp`

```
double sum(double a, double b);
```

`src/sum6.cpp`:

```
#include "sum6.hpp"

double sum(double a, double b) {
  double c = a + b;
  return c;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion main6.cpp sum6.cpp -o sum6
$ ./sum6
c = 5
```

0.5.6 `#include` syntax

- The `.hpp` file extension denotes a C++ header file
- < > around the file name means that the preprocessor should search for an include file in a system dependent or default directory
- These are typically include files that come with the compiler like `iostream`, `fstream`, `string`, etc.
- Usually these files are somewhere in `/usr/include` with the GNU compilers on Linux
- `"header.hpp"` means that the preprocessor should first search in the user directory, followed by a search in a system dependent or default directory if necessary

0.5.7 `#define`

`src/define1.cpp`:

```
// define ni and nj to be 16

```

```
3  #define ni 16
4  #define nj 16
5
6  int main() {
7    int a[ni][nj];
8
9    for(int i = 0; i < ni; i++) {
10     for(int j = 0; j < nj; j++) {
11       a[i][j] = 1;
12     }
13   }
14
15   return 0;
16 }
```

Pass the code through the preprocessor:

```
$ cpp -P define1.cpp
// define ni and nj to be 16

int main() {
  int a[16][16];

  for(int i = 0; i < 16; i++) {
    for(int j = 0; j < 16; j++) {
      a[i][j] = 1;
    }
  }

  return 0;
}
```

0.5.8 Macros

- Real power of `#define` is in setting up macros
- Similar to functions but handled by the preprocessor

0.5.9 `#define` macro

`src/define2.cpp`

```
1  #include <iostream>
2
3  #define sqr(n) (n)*(n)
4
5  int main() {
6    int a = 2;
7
8    int b = sqr(a);
9    std::cout << "b = " << b << std::endl;
10
```

```
11    return 0;
12  }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion define2.cpp -o define2
$ ./define2
b = 4
```

0.5.10 Be careful

src/define3.cpp:

```
1   #include <iostream>
2
3   #define sqr(n) n*n
4
5   int main() {
6     int a = 2;
7
8     int b = sqr(a+3);
9     std::cout << "b = " << b << std::endl;
10
11    return 0;
12  }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion define3.cpp -o define3
$ ./define3
b = 11
```

0.5.11 Predefined macros

src/define4.cpp:

```
1   \#include <iostream>
2
3   int main() {
4     std::cout << "This line is in file " << __FILE__
5               << ", line " << __LINE__ << std::endl;
6     return 0;
7   }
```

Output:

```
$ g++ -Wall -Wextra -Wconversion define4.cpp -o define4
$ ./define4
This line is in file define4.cpp, line 5
```

0.5.12 Conditional compilation

src/conditional.cpp:

```
#include <iostream>

#define na 4

int main() {
  int a[na];

  a[0] = 2;
  for (int n = 1; n < na; n++) a[n] = a[n-1] + 1;

#ifdef DEBUG
  // Only kept by preprocessor if DEBUG defined
  for (int n = 0; n < na; n++) {
    std::cout << "a[" << n << "] = " << a[n] << std::endl;
  }
#endif

  return 0;
}
```

Output:

```
$ g++ -Wall -Wextra -Wconversion conditional.cpp -o conditional
$ ./conditional
$ g++ -Wall -Wextra -Wconversion conditional.cpp -o conditional -DDEBUG
$ ./conditional
a[0] = 2
a[1] = 3
a[2] = 4
a[3] = 5
```

0.5.13 Reading

- **C++ Primer, Fifth Edition** by Lippman et al.
- Chapter 6: Functions: Sections 6.1 - 6.3
- Chapter 8: The IO Library
- Chapter 17: Specialized Library Facilities: Section 17.5.1