C++ Intro

CSE 333 Spring 2018

Instructor: Justin Hsia

Teaching Assistants:

Danny Allen
Kevin Bi
Michael Poulain
Waylon Huang

Dennis Shao
Jack Xu
Renshu Gu
Wei Lin

Eddie Huang
Matthew Neldam
Robby Marver

Administrivia

- Exercise 7 posted yesterday, due Monday
  - Read a directory and open/copy text files found there
  - Good warm-up for…

- Homework 2 due in two weeks (4/26)
  - File system crawler, indexer, and search engine
  - Spec and starter files will be pushed out today

Today's Goals

- An introduction to C++
  - Give you a perspective on how to learn C++
  - Kick the tires and look at some code

- **Advice:** Read related section in the *C++ Primer*
  - It's hard to learn the "why is it done this way" from reference docs
  - Lectures and examples will introduce the main idea, but aren't everything you'll ~~want~~ need to understand

Hello World in C

helloworld.c

```
#include <stdio.h>    // for printf()
#include <stdlib.h>   // for EXIT_SUCCESS

int main(int argc, char** argv) {
  printf("Hello, World!\n");
  return EXIT_SUCCESS;
}
```

- You never had a chance to write this!
  - Compile with **gcc**:

```
gcc -Wall -g -std=c11 -o hello helloworld.c
```

  - Based on what you know now, describe to your neighbor everything that goes on in the execution of this “simple” program
    - Be detailed!

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

- Looks simple enough…
  - Compile with g++ instead of gcc:

```
g++ -Wall -g -std=c++11 -o helloworld helloworld.cc
```

  - Let's walk through the program step-by-step to highlight some differences

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

- `iostream` is part of the ***C++*** standard library
  - Note: you don't include "`.h`" when you include C++ standard library headers
    - But you *do* for local headers (*e.g.* `#include "ll.h"`)
  - `iostream` declares stream *object* instances in the "`std`" namespace
    - *e.g.* `std::cin`, `std::cout`, `std::cerr`

      stdin stdout stderr

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

C++: ⟷ C: #include <stdlib>

- cstdlib is the *C* standard library's stdlib.h
  - Nearly all C standard library functions are available to you
      math.h <cmath>
    - For header foo.h, you should #include <cfoo>
  - We include it here for EXIT_SUCCESS, as usual

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

- `std::cout` is the "`cout`" object instance declared by `iostream`, living within the "`std`" namespace
  - C++'s name for `stdout` FILE *
  - `std:cout` is an object of class `ostream`
    - http://www.cplusplus.com/reference/ostream/ostream/
  - Used to format and write output to the console
  - The entire standard library is in the namespace `std`

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

- C++ distinguishes between objects and primitive types
  - These include the familiar ones from C: `char`, `short`, `int`, `long`, `float`, `double`, etc.
  - C++ also defines `bool` as a primitive type (woo-hoo!)

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

- "<<" is an operator defined by the C++ language
  - Defined in C as well: usually it bit-shifts integers (in C/C++)
  - C++ allows ***classes*** to overload operators!
    - Here, the `ostream` class overloads "<<"
    - *i.e.* it defines different member functions (methods) that are invoked when an `ostream` is the left-hand side (LHS) of the << operator

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

ostream

char*

- `ostream` has many different methods to handle `<<`
  - The functions differ in the type of the right-hand side (RHS) of `<<`
  - *e.g.* if you do `std::cout << "foo";`, then C++ invokes `cout`'s function to handle << with RHS `char*`

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

cout

this single line is equivalent to:

std::cout << "Hello, World!";
std::cout << endl;

- The `ostream` class' member functions that handle `<<` return a reference to themselves
  - When `std::cout << "Hello, World!";` is evaluated:
    - A member function of the `std::cout` object is invoked
    - It buffers the string `"Hello, World!"` for the console
    - And it returns a reference to `std::cout`

Hello World in C++

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

- Next, another member function on **`std::cout`** is invoked to handle **<<** with RHS **`std::endl`**
  - **`std::endl`** is a pointer to a "manipulator" function
    - This manipulator function ① writes newline (`'\n'`) to the **`ostream`** it is invoked on and then ② flushes the **`ostream`**'s buffer
    - This *enforces* that something is printed to the console at this point

Wow…

helloworld.cc

```
#include <iostream>
#include <cstdlib>

int main(int argc, char** argv) {
  std::cout << "Hello, World!" << std::endl;
  return EXIT_SUCCESS;
}
```

- You should be surprised and scared at this point
  - C++ makes it easy to hide a significant amount of complexity
    - It's powerful, but really dangerous
    - Once you mix everything together (templates, operator overloading, method overloading, generics, multiple inheritance), it gets *really* hard to know what's actually happening!

Let’s Refine It a Bit

helloworld2.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello, World!");
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- C++’s standard library has a `std::string` class
  - Include the `string` header to use it
    - Seems to be automatically included in `iostream` on CSE Linux environment (C++11)
  - http://www.cplusplus.com/reference/string/

Let’s Refine It a Bit

helloworld2.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello, World!");
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- The “`using`” keyword introduces a namespace (or part of) into the current region
  - `using namespace std;` imports all names from `std::`

    linter will complain, but we will ignore
  - `using std::cout;` imports *only* `std::cout` (used as `cout`)

Let’s Refine It a Bit

helloworld2.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello, World!");
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- Note the benefits of `using namespace std;`
  - Can now refer to `std::string` by `string`, `std::cout` by `cout`, and `std::endl` by `endl`

Let's Refine It a Bit

helloworld2.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello, World!");
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- Here we are instantiating a `std::string` object *on the Stack*
  - Passing the C string `"Hello, World!"` to its constructor method
  - `hello` is deallocated (and its destructor invoked) when **`main`** returns

more details on these later

Let’s Refine It a Bit

helloworld2.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello, World!");
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- The C++ string library also overloads the `<<` operator
  - Defines a function (*not* an object method) that is invoked when the LHS is `ostream` and the RHS is `std::string`
    - http://www.cplusplus.com/reference/string/string/operator<</

String Concatenation

concat.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello");
  hello = hello + ", World!";
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- The string class overloads the “+” operator
  - Creates and returns a new string that is the concatenation of the LHS and RHS

string char* ← in this example

String Assignment

concat.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello");
  hello = hello + ", World!";
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- The string class overloads the “=” operator
  - Copies the RHS and replaces the string’s contents with it

String Manipulation

concat.cc

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

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello");
  hello = hello + ", World!";
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```

- This statement is complex!
  - First "+" creates a string that is the concatenation of `hello`'s current contents and `", World!"`
  - Then "=" creates a copy of the concatenation to store in `hello`
  - Without the syntactic sugar:
    - `hello.operator=(hello.operator+(", World!"));`

operators are just member functions

Stream Manipulators

manip.cc

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

using namespace std;

int main(int argc, char** argv) {
  cout << "Hi! " << setw(4) << 5 << " " << 5 << endl;
  cout << hex << 16 << " " << 13 << endl;
  cout << dec << 16 << " " << 13 << endl;
  return EXIT_SUCCESS;
}
```

- `iomanip` defines a set of stream manipulator functions
  - Pass them to a stream to affect formatting
    - http://www.cplusplus.com/reference/iomanip/
    - http://www.cplusplus.com/reference/ios/

Stream Manipulators

manip.cc

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

using namespace std;

int main(int argc, char** argv) {
  cout << "Hi! " << setw(4) << 5 << " " << 5 << endl;
  cout << hex << 16 << " " << 13 << endl;
  cout << dec << 16 << " " << 13 << endl;
  return EXIT_SUCCESS;
}
```

- `setw(x)` sets the width of the *next* field to `x`
  - Only affects the next thing sent to the output stream (*i.e.* it is not persistent)

Stream Manipulators

manip.cc

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

using namespace std;

int main(int argc, char** argv) {
  cout << "Hi! " << setw(4) << 5 << " " << 5 << endl;
  cout << hex << 16 << " " << 13 << endl;
  cout << dec << 16 << " " << 13 << endl;
  return EXIT_SUCCESS;
}
```

→Hi!____5_5

→10_d

→16_13

- hex, dec, and oct set the numerical base for *integers* output on the stream
  - Stays in effect until you set the stream to another base (*i.e.* it is persistent)

C and C++

helloworld3.cc

C++:

```
#include <cstdio>
#include <cstdlib>

int main(int argc, char** argv) {
  printf("Hello from C!\n");
  return EXIT_SUCCESS;
}
```

⟷ C: #include <stdio>

- C is (roughly) a subset of C++
  - You can still use **printf** – but bad style now!
  - Can mix C and C++ idioms if needed to work with existing code, but avoid mixing if you can
    - Use C++(11)

Reading

echonum.cc

```
#include <iostream>
#include <cstdlib>

using namespace std;

int main(int argc, char** argv) {
  int num;
  cout << "Type a number: ";
  cin >> num;
  cout << "You typed: " << num << endl;
  return EXIT_SUCCESS;
}
```

- `std::cin` is an object instance of class `istream`
  - Supports the `>>` operator for "extraction"
    - Can be used in conditionals – `(std::cin>>num)` is `true` if successful
  - Has a `getline()` method and methods to detect and clear errors

Peer Instruction Question

- How many *different* versions of **<<** are called?
  - For now, ignore manipulator functions
  - Vote at http://PollEv.com/justinh
  - Also, what is output?

cout << str → std::string (1)
cout << 15U → unsigned... int? (2)
cout << n → int (3)
cout << "e!" → char* (4)

A. 1
B. 2
C. 3
[✓ **D.** 4]
E. We're lost…

msg.cc

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <iomanip>

using namespace std;

int main(int argc, char** argv) {
  int n = 172;  160+12 = 10(16)+12 = 0xac
  string str("m");
  str += "y";                               (not counting these)
  cout << str << hex << setw(2)
       << 15U << n << "e!" << endl;
  return EXIT_SUCCESS;
}          "my_face!"
```

Extra Exercise #1

- Write a C++ program that uses stream to:
  - Prompt the user to type 5 floats
  - Prints them out in opposite order with 4 digits of precision