C++ Intro
CSE 333 Winter 2020

Instructor: Justin Hsia

Teaching Assistants:
Andrew Hu       Austin Chan       Brennan Stein
Cheng Ni        Cosmo Wang       Diya Joy
Guramrit Singh  Mengqi Chen      Pat Kosakanchit
Rehaan Bhimar   Renshu Gu        Travis McGaha
Zachary Keyes   

Exercise 8 released today, due Wednesday
- First C++ exercise!
- Parallels to ex0 – compare user input checking between C/C++

Homework 2 due next Thursday (2/6)
- File system crawler, indexer, and search engine
- **Note**: Make sure a copy of `libhw1.a` is in the `hw1/` directory
  - Either your own (run `make`) or ours (copy from `hw1/solution_binaries`)
- **Demo**: use Ctrl-D to exit `searchshell` gracefully, test on directory of small self-made files
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 sections in the *C++ Primer*
  - It’s hard to learn the “why is it done this way” from reference docs, and even harder to learn from random stuff on the web
  - Lectures and examples will introduce the main ideas, but aren’t everything you’ll want need to understand
Hello World in C

```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`:
    ```bash
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++

#include <iostream>    // for cout, endl
#include <cstdlib>      // for EXIT_SUCCESS

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

Looks simple enough...

- Compile with `g++` instead of `gcc`:

  ```bash
  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++

```cpp
#include <iostream> // for cout, endl
#include <cstdlib> // for EXIT_SUCCESS

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 write “.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`
Hello World in C++

```cpp
#include <iostream> // for cout, endl
#include <cstdlib>  // for EXIT_SUCCESS

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

- `cstdlib` is the C standard library’s `stdlib.h`
  - Nearly all C standard library functions are available to you
    - For C header `foo.h`, you should `#include <cfoo>`
  - We include it here for `EXIT_SUCCESS`, as usual
Hello World in C++

```cpp
#include <iostream> // for cout, endl
#include <cstdlib> // for EXIT_SUCCESS

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`
  - `std::cout` is an object of class `ostream`
  - Used to format and write output to the console
  - The entire standard library is in the namespace `std`
Hello World in C++

```cpp
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

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!)
    - Use it!
Hello World in C++

```
#include <iostream>  // for cout, endl
#include <cstdlib>   // for EXIT_SUCCESS

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 and functions to overload operators!
    - Here, the `iostream` class overloads "<<"
      - *i.e.* it defines different **member functions** (methods) that are invoked when an `iostream` is the left-hand side of the `<<` operator
Hello World in C++

```
#include <iostream>  // for cout, endl
#include <cstdlib>   // for EXIT_SUCCESS

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

- **`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++

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++

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 writes newline (`'\n'`) to the `ostream` it is invoked on and then flushes the `ostream`’s buffer
  - This enforces that something is printed to the console at this point

```cpp
#include <iostream>  // for cout, endl
#include <cstdlib>    // for EXIT_SUCCESS

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

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 can get really hard to know what’s actually happening!

```cpp
#include <iostream>  // for cout, endl
#include <cstdlib>   // for EXIT_SUCCESS

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

helloworld.cc
Let’s Refine It a Bit

```cpp
#include <iostream> // for cout, endl
#include <cstdlib>  // for EXIT_SUCCESS
#include <string>   // for 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) – but include it explicitly anyway if you use it
Let’s Refine It a Bit

```cpp
#include <iostream>   // for cout, endl
#include <cstdlib>    // for EXIT_SUCCESS
#include <string>     // for 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::`
  - `using std::cout;` imports *only* `std::cout` (used as `cout`)
Let’s Refine It a Bit

⚠ Benefits of `using namespace std;`

- We can now refer to `std::string` as `string`, `std::cout` as `cout`, and `std::endl` as `endl`

```cpp
#include <iostream> // for cout, endl
#include <cstdlib>  // for EXIT_SUCCESS
#include <string>   // for string

using namespace std;

int main(int argc, char** argv) {
  string hello("Hello, World!");
  cout << hello << endl;
  return EXIT_SUCCESS;
}
```
Let’s Refine It a Bit

```cpp
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <string>      // for 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 (an ordinary local variable)
  - Passing the C string "Hello, World!" to its constructor method
  - `hello` is deallocated (and its destructor invoked) when `main` returns
Let’s Refine It a Bit

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`
String Concatenation

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

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

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:
  ```cpp
  hello.operator=(hello.operator+(", World!"));
  ```

```cpp
#include <iostream> // for cout, endl
#include <cstdlib> // for EXIT_SUCCESS
#include <string> // for string

using namespace std;

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

```cpp
#include <iostream>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS
#include <iomanip>     // for dec, hex, setw

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
Stream Manipulators

- **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

```cpp
#include <iostream>   // for cout, endl
#include <cstdlib>    // for EXIT_SUCCESS
#include <iomanip>    // for dec, hex, setw

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;
}
```

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

```c
#include <cstdio>    // for printf
#include <cstdlib>   // for EXIT_SUCCESS

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

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

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

```cpp
#include <iostream>  // for cout, endl
#include <cstdlib>   // for EXIT_SUCCESS

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;
}
```

Polling Question

- How many *different* versions of `<<` are called?
  - For now, ignore manipulator functions
  - Also, what is output?

A. **1**
B. **2**
C. **3**
D. **4**
E. We’re lost…

```cpp
#include <iostream>
#include <cstdlib>
#include <string>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    int n = 172;
    string str("m");
    str += "y";
    cout << str << hex << setw(2) << 15U << n << "e!" << endl;
    return EXIT_SUCCESS;
}
```
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