

C++ Intro Continued

CSE 333

Instructor: Alex Sanchez-Stern

Teaching Assistants:

Audrey Seo

Deeksha Vawani

Derek de Leuw

Katie Gilchrist

Administrivia

- ❖ Exercise 8 due on **Wednesday**
 - Using C++ knowledge
- ❖ HW2 is pushed
 - Due **one week from Thursday (7/24)**; get started ASAP!

Today

- ❖ “Hello world” in C++
 - Program setup
 - Using the C++ standard library
 - I/O

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;
}
```

- ❖ Compile with gcc:

```
gcc -Wall -g -std=c17 -o hello helloworld.c
```

- ❖ You should be able to describe in detail everything in this code

Hello World in C++

helloworld.cc

```
#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:

```
g++ -Wall -g -std=c++17 -o hello helloworld.cc
```

.cc is one of many extensions used for c++ files, but it's the one we'll use in this class.

Hello World in C++

helloworld.cc

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

Hello World in C++

helloworld.cc

```
#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++

helloworld.cc

```
#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`
 - Used to format and write output to the console
 - `std::cout` is an object of class `ostream`
 - <http://www.cplusplus.com/reference/ostream/ostream/>

Hello World in C++

helloworld.cc

```
#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
 - 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 of the << operator

Hello World in C++

helloworld.cc

```
#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++

helloworld.cc

```
#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;
}
```

- ❖ The `ostream` class' member functions that handle `<<` return **a reference to themselves**
 - When `std::cout << "Hello, World!"` is evaluated, it returns a reference to `std::cout`

Hello World in C++

helloworld.cc

```
#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;
}
```

- ❖ 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

Wow...

helloworld.cc

```
#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;
}
```

- ❖ You should be a little worried 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!

C++ Features

- ❖ So far:
 - (Slightly) different include syntax
 - Namespaces
 - Stream objects
 - Operator overloading
- ❖ Now let's add:
 - `using` keyword
 - Constructing objects
 - More operators

Let's Refine It a Bit

helloworld2.cc

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

using namespace std;

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

Google style guide says don't do this, but for CSE333, using namespace std is ok

- ❖ 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

helloworld2.cc

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

using namespace std;

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

❖ Benefits of `using namespace std;`

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

Let's Refine It a Bit

helloworld2.cc

```
#include <iostream>    // for cout, endl
#include <cstdlib>      // for EXIT_SUCCESS
#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++17) – but include it explicitly anyway if you use it
 - <http://www.cplusplus.com/reference/string/>

Let's Refine It a Bit

helloworld2.cc

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

using namespace std;

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

❖ C++ distinguishes between objects and **primitive types**

- Primitive types 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!
 - (but `bool` and `int` values silently convert types for compatibility)

Let's Refine It a Bit

helloworld2.cc

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

using namespace std;

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

We'll talk about constructors and destructors later

- ❖ Here we are instantiating a `std::string` object *on the stack* (an ordinary local variable)
 - Passing the C string `"Hello, World!"` to construct it
 - `hello` is deallocated when `main` returns

Let's Refine It a Bit

helloworld2.cc

```
#include <iostream>    // for cout, endl
#include <cstdlib>      // for EXIT_SUCCESS
#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 overloads the “<<” operator
 - Defines a function that is invoked when the LHS is `ostream` and the RHS is `std::string`
 - [21](http://www.cplusplus.com/reference/string/string/operator<</div><div data-bbox=)

String Concatenation

concat.cc

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

using namespace std;

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

- ❖ Let's make it a bit more complicated

String Concatenation

concat.cc

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

concat.cc

```
#include <iostream>    // for cout, endl
#include <cstdlib>      // for EXIT_SUCCESS
#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>    // for cout, endl
#include <cstdlib>      // for EXIT_SUCCESS
#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!"));
```

Formatting

- ❖ What about all the string formatting that `printf` let us do?
 - More string manipulator functions!

Stream Manipulators

manip.cc

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

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

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

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 *integer* output to the stream
 - Stays in effect until you set the stream to another base (*i.e.* it is persistent)

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

C and C++

helloworld3.cc

```
#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++(17)

Don't forget

- ❖ Exercise 8 due on **Wednesday**
- ❖ HW2 due **one week from Thursday (7/24)**

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