



# Poll Everywhere

[pollev.com/cse333cosmo](https://pollev.com/cse333cosmo)

**How many hours have you spent on Homework 1 so far?**

- A. [0, 4) hours
- B. [4, 8) hours
- C. [8, 12) hours
- D. [12, 16) hours
- E. [16, 20) hours
- F. 20+ hours
- G. Prefer not to say

# C++ Intro

## CSE 333 Spring 2021

**Guest Instructor:** Cosmo Wang

**Teaching Assistants:**

Atharva Deodhar

Callum Walker

Cosmo Wang

Dylan Hartono

Elizabeth Haker

Kyrie Dowling

Leo Liao

Markus Schiffer

Neha Nagvekar

Nonthakit Chaiwong

Ramya Challa

# Administrivia

- ❖ Homework 1 due yesterday, late submission due Sunday
  - Submit on Sunday by 11:59 pm PDT
- ❖ Exercise 4 due **next Tuesday, April 20<sup>th</sup>** @ 11 am PDT
  - Implement lite versions of C std I/O functions using POSIX
  - Read a directory and open/copy text files found there
- ❖ Exercise 5 released today, due next Wednesday, April 21<sup>st</sup>
  - First exercise in C++!
  - Read an integer from command line and print out all its factors
- ❖ Homework 2 released today, due April 29<sup>th</sup>
  - Fill out partner sign-up form by April 21<sup>st</sup> @ 11:59 PDT
  - Demo in lecture on Monday

# 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

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=c17 -o hello helloworld.c
```
  - Based on what you know now, type in the chat about one thing that goes on in the execution of this “simple” program
    - Be detailed!

# 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 helloworld helloworld.cc
```

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

# Hello World in C++

```
                                iostream.h                                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
  - 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`  
*stdcin*                      *stdcout*                      *stdcerr*

# Hello World in C++

helloworld.cc

```
C++: #include <iostream> // for cout, endl
#include <cstdlib> // for EXIT_SUCCESS
C: #include <stdlib.h>
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 <math.h>` `<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`
  - `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>    // 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++

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 (in C/C++)
  - C++ allows classes and functions 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 ←*

*↓ 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>    // for cout, endl
#include <cstdlib>     // for EXIT_SUCCESS

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

*Handwritten notes:*  
- A red oval highlights the line `std::cout << "Hello, World!" << std::endl;`  
- An arrow points from the text "this line is equivalent to" to the highlighted line.  
- Below the highlighted line, the text "cout" has two arrows pointing to the `std::cout` and `std::endl` parts of the line.

- ❖ 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>    // 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 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!

# Let's Refine It a Bit

helloworld2.cc

```
#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++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>       // 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

*← Do NOT use this in your code!*

-  `using namespace std;` imports all names from `std::`
-  `using std::cout;` imports *only* `std::cout` (used as `cout`)

*↪ use this instead*

# Let's Refine It a Bit

helloworld2.cc

```
#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;
}
```

- ❖ Benefits of `using namespace std;`
  - We can now refer to `std::string` as `string`, `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>       // 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  
*more on these later*
  - `hello` is deallocated (and its destructor invoked) when `main` returns

# Let's Refine It a Bit

helloworld2.cc

```
#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 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<</a>](http://www.cplusplus.com/reference/string/string/operator<</)

# String Concatenation

concat.cc

```
#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;
}
```

- ❖ 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>    // 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;
}
```

- ❖ 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>       // for 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!"));`

*They are just member functions!*

# Stream Manipulators

manip.cc

```
#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
    - <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 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;
}
```

Handwritten diagram illustrating the output of the first line: "Hi! \_ \_ \_ \_ 5 \_ 5 \n". A bracket under the four underscores is labeled "4", indicating the width set by `setw(4)`. An arrow points to the second "5", indicating that `setw` only affects the next field.

- ❖ **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 dec, hex, setw

using namespace std;

int main(int argc, char** argv) {
    cout << "Hi! " << setw(4) << 5 << " " << 5 << endl;
    cout << hex << 16 << " " << 13 << endl; → 10-d\n
    cout << dec << 16 << " " << 13 << endl; → 16-13\n
    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++

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
    - E.g. Use `std::cerr` instead of `fprintf(stderr, ...)`
  - Can mix C and C++ idioms if needed to work with existing code, but avoid mixing if you can
    - **Use C++(17)**

# Reading

echonum.cc

```
#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;
}
```

- ❖ `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

# Poll Everywhere

[pollev.com/cse333cosmo](http://pollev.com/cse333cosmo)

❖ How many *different* versions of << are called?

- Ignore the stream manipulators for now
- Also, what is output?

msg.cc

A. 1

① *std::string*

B. 2

② *unsigned int*

C. 3

③ *int*

D. 4

④ *char\**

E. We're lost...

```

#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;
}
    
```

$160 + 12 = 10(16) + 12 = 0xac$

ignored

Output: "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