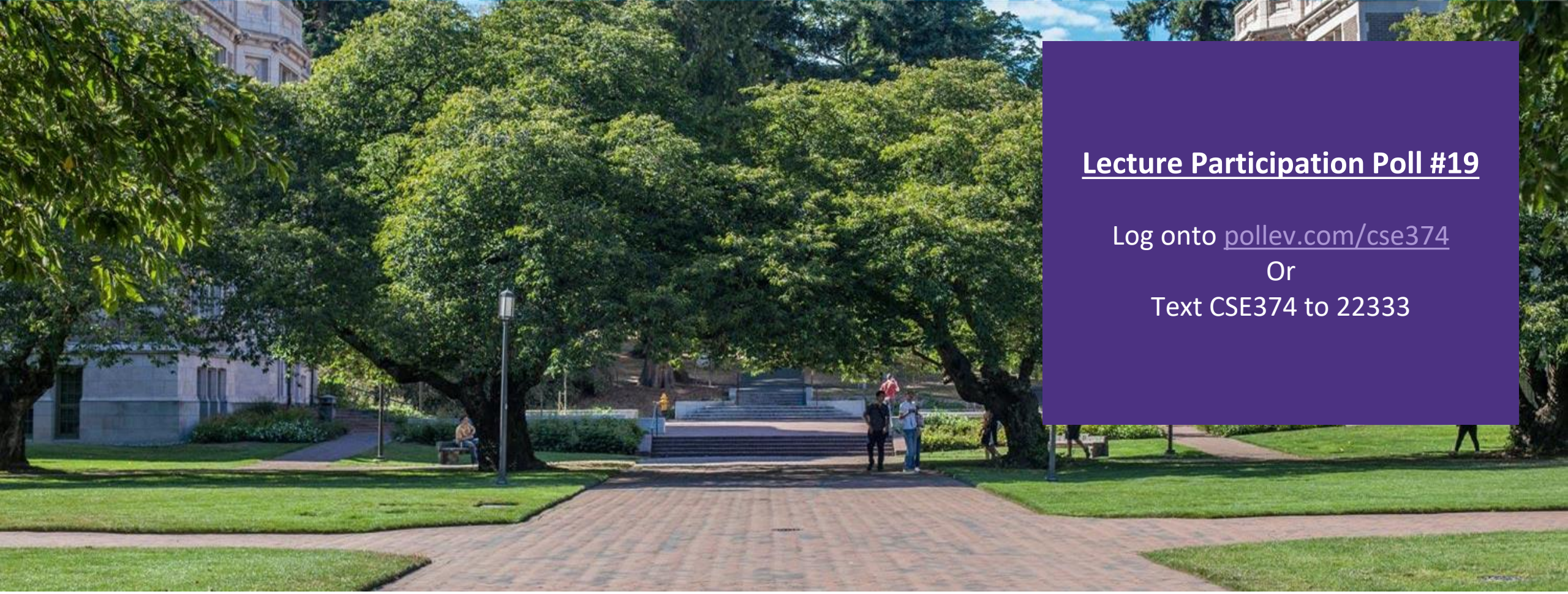


Lecture Participation Poll #19

Log onto pollev.com/cse374
Or
Text CSE374 to 22333

Lecture 19: Intro to C++

CSE 374: Intermediate Programming Concepts and Tools

Administrivia

HW4 late turn in locks today

Reminder: on Friday I gave everyone 3 more late days

Object Oriented Programming

- **Encapsulation**
  - discrete portions of code keep state and implementation private while providing public interfaces
- **Abstraction**
  - the high level interface is exposed to users without detailing underlying code
- **Inheritance**
  - classes can be derived from other classes allowing for shared code
- **Polymorphism**
  - subclasses implement methods of superclasses to allow for a consistent interface

Meet C++

- C++ is a general-purpose programming language created as an **extension** of the C programming language
  - Sometimes referred to "C with Classes"
  - Includes object-oriented, generic and functional features in addition to facilities for low-level memory manipulation
  - Designed with a bias towards system programming and embedded, resource-constrained software
- C is (roughly) a subset of C++, a C program can be compiled as a C++ program
  - 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
- 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++ is considered a "middle level" language
  - can do both system programming and OOP

Features of C++

- Machine independent, but platform dependent
  - C++ executable is not platform-independent but they are machine independent
    - ie executable compiled on one windows machine can run on all windows machines but not on Linux machines
- rich library support
  - standard built in libraries have data structures and algorithms support
  - lots of 3rd party libraries
- Compiled language
  - does not include modern features like garbage collection, dynamic typing
  - makes C++ SUPER fast
- Pointer support and direct memory access
  - enables direct memory manipulation for low-level programming
- Object oriented
  - includes classes, objects, inheritance, polymorphism
  - namespaces allow for intelligent code grouping
- strongly typed
  - specific data types that are not interoperable
  - allows for both static and dynamic type checking
    - types checked at either compile or runtime

C++ Resources

- Best place to start: C++ Primer, Lippman, Lajoie, Moo, 5th edition
- Good Online Source: cplusplus.com
- Serious C++ programmers should read:
  - Effective C++, Meyers, 3rd Edition
    - Best practices for standard C++
  - Effective Modern C++,. Meyers, O'Reilly
    - Additional "best practices" for C++11/C++14

Basic Differences between C and C++

- File names end with *.cc or *.cpp or *.cxx
  - Still use *.h for header files
- Use a different compiler: g++ instead of gcc
- C++ uses C preprocessor but libraries are different
  - #include <cstdlib>
  - basically the same as <stdlib.h>

Hello World

helloworld.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;
}
```

```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

Hello World C++ iostream

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`

Hello World C++ cstdlib

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 C++ std::cout

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`

- 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

Hello World C++ ostrem

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*`

- 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`

I/O in C++

- "<<" 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
- Std library include a cout and a cin function
- Operators >> and << act like shell redirection
- Operators >> and << take left and right operands and return a stream
- use namespace std or
- use std::cout & std::cin

```
using namespace std

cout << "what is your name";
string name;
cin >> name;

cout << "when were you born?";
int year;
cin >> year;
```

Namespaces

- Groups code logically
- can reuse names for each namespace
- Disambiguate with `::` syntax
- Can avoid using the prefix with
  - `using namespace foo`
  - `doSomething(3)`
- if you are using a namespace in a header, you must also use the name space in the source code

```
namespace foo {
    int doSomething(int x);
}
name space bar {
    int doSomething(int x);
}
int main() {
    foo::doSomething(3);
    bar::doSomething(3);
}
```

Cout and Types

- 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!

- `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*`

Refined Hello World

```
#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
- http://www.cplusplus.com/reference/string/

Refined Hello World

```
#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`)

- Benefits of `using namespace std;`
  - We can now refer to `std::string` as `string`, `std::cout` as `cout`, and `std::endl` as `endl`

Refined Hello World

```
#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

- The C++ string library also overloads the << operator
  - Defines a function (*not* an object method) that is invoked when the left hand side is `ostream` and the right hand side is `std::string`

http://www.cplusplus.com/reference/string/string/operator<</

String Manipulation

```
#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;
}
```

String Concatenation

The string class overloads the "+" operator

- Creates and returns a new string that is the concatenation of the left and right

String Assignment

The string class overloads the "=" operator

- Copies the right and replaces the string's contents with it

- 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!"));`

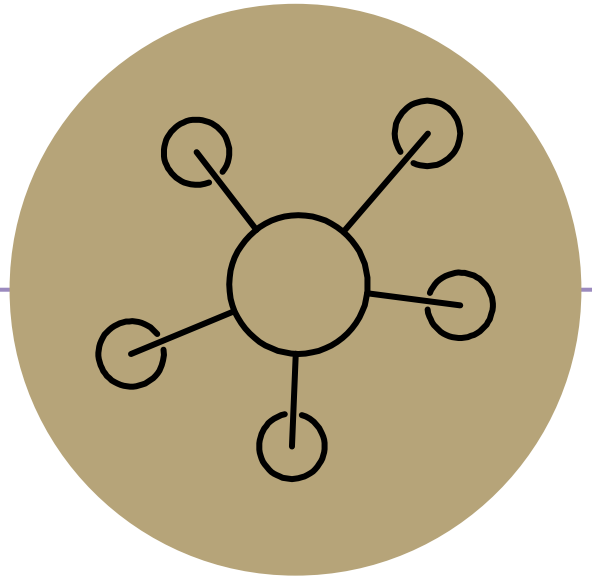
Const

New keyword indicating “this thing must not change”

used for global constants:

```
const int CURRENT_YEAR = 2021;
```

global constants look like global variables, but they are OK stylistically



Questions