Part 1 and 2

- topic
- important

- C++ is a compiled language

Std::cout

- std::cout is a type iostream
- endl causes a newline and stream flush
- Shift operator << overloaded
- Operator overloading is a major feature of C++

const and constexpr

- whenever possible, good to mark object as constant so that any usage of the object cannot modify it
- Eliminates "accidental modification" bugs
- Non-constant vs constant variables
- constexpr must be initialized at compile time by constant expression

- Possible for function, not just return value, to be const

References and pointers

- Pointers are like the same in C
- References are just pointers that are automatically dereferenced when used as a variable
- T & = reference to

  int & x = reference to int

  T* = pointer to

  int* x = pointer to an int

& = reference

* = pointer

- Const can be used in combination w/ references

Classes versus structs

- Classes are stronger than structs in C++
- They have better info hiding properties
- struct, any declared variables are default public
- class, any declared variables are default private

Constructors

- are called when you create an object or when you make a variable with an initializer list
- Result is the same - class constructor is called when the object is constructed

KISS

- "Keep it Simple and Stupid"
- Use const to mark things:
  - Returns or methods you don't want to modify or BOTH

- If you want to modify anything you are passing around
  - use non-const but be careful of "leaky state"
- Return a const reference if you don't want to modify it
- If you don't want "leaky state" and don't want to use const reference, return value and it will copied on the stack

nullptr

- use this rather than null

sizeof

- returns the size of class or primitive in bytes

Abstract class

- Any intermediate or root class which defines an interface but doesn't say how it is implemented
- Implementation left for subclass

=0

- Means that you are not implementing it there but a subclass will
- If you don't implement it somewhere in the subclass, a compiler error will occur

Overide and Virtual

- If you use virtual to define a prototype or other implementations for a function in a base class, also mark all implementations with override

Public, private, and protected visibility

- public: anyone with a pointer or reference to an instance to the class can see it
- private: only the class can see the data
- protected: only the class and the subclasses can see the data

Smart pointers

- classes that wrap a pointer or scoped pointer
- Wrapper class over a real pointer
- Wrapper classes are just a class which takes functionality and "wraps it up" into something with widespread applicability

Debug Methods

- Explanation / Eyeballing
- Dry running
- using a debugger
- using printing and logging
- Binary subdivision

Part 3

Data-Driven Pattern

- A "pattern" is a recipe or method we can use to deal with a particular situation
- MMOs and games usually use the data driven pattern

- Most of the behavior of the simulation is defined by data, code provides framework

Canonical entities

- Implemented as classes
- Examples are like NPCs, Players, Items

Commonality

- Look for commonality –
  - NPCs and Players are similar – but Players have a real person operating them
  - Thus we look for a superclass for NPC and Player - Character

Pattern (more detailed)

- software design pattern is a general reusable solution
- Patterns occur within a given context
- They are not a finished design
- They are a description on how to solve a problem in a way it can be reused
- They are formulized best practices that can be used to solve problems

- Patterns give us a language to discuss more advanced topics
- We use them to establish a common baseline and skip a lot of explaining

Relationship with OO design

- OO design patterns typically show interactions and relationships between classes and objects
- They don't specify final classes but a common language to making software
- Patterns typically have a Problem and a Solution

Polymorphism

- The provision of a single interface to entities of different types or the use of a single symbol to represent multiple types

The OO way

- OO encourages moving functionality into the actual object
- The object has sovereignty over its own data
- Simpler in long term

Delegation

- shifted a behavior specific to an object to that object
- Code uses object in a generic way
- "special purpose" stuff is delegated to the object to take care of
- Loader code is generic, "special purpose" is inside the subclass
- loader has less "special cases"

Part 4

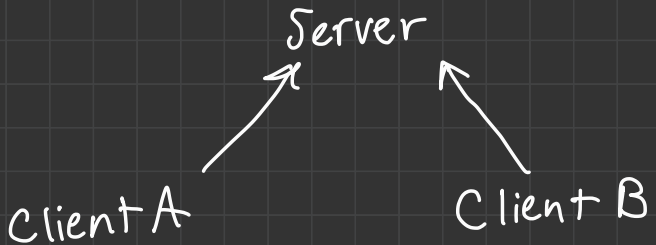
Message

- If we encapsulated all of the state for an operation into an instance of a class that can be passed around between objects, it would be a message or a task

The Queue Pattern

Producer → Queue → Consumer

Client Server Model w/ messages



Message Queues

- They decouple the producers of the messages from the consumers of the messages
- They buffer work between threads/processes and seperate computers
- They can smooth spikey workflows
- In client server, there would be a message queue to which messages are added and removed

Std::thread

- Thread is smallest sequence that can be managed independently by a scheduler
- Threads allow multiple functions to execute

- std::thread provides support for threads
- Execution order running inside the threads is indeterminate
- Indeterminate means its not possible to determine which order things happen, can vary each time you run multithreading
- TLDR: Any state "leaking" between the threads at all weird things can happen
- NOTHING is thread safe unless it is in the "concurrency support library" and "atomic"

Atomic

- A fundamental operation is making something a tomic
- Atomic provides a way to easily thread safe the shared state

Mutexes and Conditional Variables

- std::mutex and std::conditional_variable
- Mutexes solve the problem of contention for shared resources (could be anything)

Synchronization Primitives

- Mutexes
  - Mutual Exclusion - only one thing can hold the mutex at a time
- Locks
  - Enforces access to critical sections based on using any type of synchronization variable

- Atomic objects or variables
  - Can only be in one state or another; thread safe "variables" that can be read/written by multiple threads
- Conditional Variables
  - used for signalling between threads. Using these with atomic objects allows us to talk to the threads

Mutex definition

- object in a program used to negotiate mutual exclusion among threads
- Only the holder to the mutex can execute code

Lock definition

- Wrapper class around mutexes
- Do the lock work and prevent access to critical sections

sleep() and usleep()

- sleep() sleeps the current thread for a # of seconds
- usleep() sleeps for a # of microseconds
- Both sleep times are approximate
- During sleep, operation is suspended

Socket()

- A method of communicating between disparate machines

- Client reaches out across network to server it is connected to
- Server processes request and then communication happens across the socket

- Domain: AF_LOCAL for same host, AF_INET for ipv4, AF_INET6 for ipv6

- Type: socket type, either SOCK_STREAM_TCP (reliable) SOCK_DGRAM_UDP (unreliable)
- Protocal: 0 for internet protocal (IP)

4 Stage process

- Create socket
- Figure out where to connect
- Make connection
- Send data

Creating a server

- Create a socket
- Set the socket options
- Put the socket on the network interface
- Listen for connections
- Accept() connections and establish socket
- Read as much as we want from socket
- Close socket and clean up resources

Select() and pselect()

- Monitors a set of file descriptors for activity
- Allows a program to monitor multiple file descriptors waiting until one or more file descriptor is ready (e.g. input possible)

Part 5

Cross Platform issues

- There are differences in C++ compilers
- OS / vendor differences
- Platform specific differences (e.g GUIs)
- Cross platform building (CMake)
- Cross platform libraries

Developmental packages and package management

- Using apt-get installs precompiled versions of whatever package the upstream package maintainer thought made sense in the context of whatever version of Ubuntu (in our case)

OS Package manager issues

- Don't always work well for development
- Such dependencies/packages are stable versions which are years older than what you can find on GitHub
- Sometimes packages are not configured to provide appropriate options

Better Things exist

- vcpkg and conan (dependency managers)
- Google has their own internal version
- The job of providing a developmental environment is beginning to be the province of "dev ops" staff

CMakeLists.txt

- Analog of a makefile
- contains info on what youre building
- Expected at least in the root of the project
- Contains list of instructions which are used to generate a makefile or project file for platform

Generators

- cmake supports working with platform specific tools using generators
- Generators allow us to tell CMake what environment to generate project files for
- These generated project files can then be loaded into the appropriate IDE

vcpkg

- Free C/C++ package manager
- Install is easy, once installed you tell CMake to use it

Conan

- Install conan, installs stuff in cmake install dir
- Make a conanfile.txt in top level directory
- Populate with dependencies

Protocol buffer objects

- Protocol buffer classes are basically data holders (like structs in C) that don't provide additional functionality
- Wrapping the generated protocol buffer class in an application specific class adds richer behavior
- NEVER add behavior to the generated class by inheriting from them

Propertied Objects

- Wrap generated protobuf message object inside another class
- Wrapper class would have OO functionality
- Protobuf object are really just data holders

Protobuf and inheritance

- It does not do it
- Can be used in languages that don't support inheritance like duck-typed and functional languages
- Single "flat" object per message

Serialization

- SerializeToString: serializes message and stores bytes in string
- ParseFromString: parses a message from given string
- SerializesToOstring: writes a message to C++ ostream

- ParseFromIstream: parses a message from given C++ istream

Evaluating open source projects

- Talked more in details in lecture
- Make sure you're able to have an opinion about this and write it down
- Broadly
  - what a project does relative to what you want it to do
  - How it will work
  - Ease of use
  - Cross platformers

Dependency Managers

- How you get dependencies in your project
- Always a better solution than making your own ball of dependencies using cmake or whatever
- Use the dependency manager for what it's good for and let it manage your dependencies for you