Ideal Gas Pt. 2 AMA

March 19, 2021
Cinder Code Design

• Ensure you are still designing Cinder applications with the following in mind
  • Object Decomposition: where do you render?
  • Data Flow: are all of the parameters of object methods in the correct context? Do they expose too much?
  • Modularity: what can be split out / reused?
Namespaces

• In C++, you have more options than putting every method in some class
  • Some helpers still belong in classes if they work on an instance
  • Consider namespaces for functions that multiple classes use

• Namespaces can be nested:

```cpp
namespace pictionary {
    namespace utils {
        int foo();
    }
}

// Usage
pictionary::utils::foo();
```
## C++ Variable Passing Semantics

<table>
<thead>
<tr>
<th>Pass by Value (copies)</th>
<th>Pass by Const Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void foo(int n);</strong></td>
<td><strong>void bar(const ComplicatedObject&amp; object);</strong></td>
</tr>
<tr>
<td>• Copies the parameter into the scope of the function</td>
<td>• Passes in a reference to the object (like Java), but does not allow changes to the object</td>
</tr>
<tr>
<td>• All primitives should be passed in this way (const reference provides no benefit to these)</td>
<td>• Can only call const methods on these objects</td>
</tr>
<tr>
<td>• Small objects that need to be copied also work for this</td>
<td>• Faster than passing by value</td>
</tr>
</tbody>
</table>
Same For Foreach Loops!

Pass by Value (copies)
for (int n : numbers)

Pass by Const Reference
for (const vector& vec : vectors)
What about normal references?

• Mutable parameters in functions are rarely needed
  • If you don't need to mutate the object, then pass in a const reference
  • If you do, consider making the function a method of the object
  • If the method is updating two objects, consider whether the objects can be merged, or whether the method could be split up

• Beware passing by non-const reference, as the syntax is the exact same: `foo(my_vector)`

• The Google Style Guide allows for reference params for mandatory mutable arguments, and pointer params for optional
Memory Errors

• SIGSEGV (11) / EXC_BAD_ACCESS
  • Your program attempted to access memory outside of its allowed boundaries
  • Most commonly, dereferencing a null pointer (0x0)
  • Also thrown when dereferencing previously de-allocated memory

• Debugging w/ the CLion debugger can help you isolate where this occurs
Visit the pinned thread on Campuswire to ask your questions. If you see a question you'd like to hear the answer to, upvote it.