Our Sucky Translator
Once upon a time, an evil professor told everyone to hang themselves... Rather, he gave them enough rope to do so.

Here's XTC

Love,

Prof. Grimm
And being the suicidal maniacs that they were, or rather, still are, they considered the option.
“If there is anything we are more of than being emo, it’s being indecisive!” said our protagonists.

I will always decide not to decide, unless of course I decide to change my mind!

So at the last minute they changed their minds.
So with:

- A lack of Sleep
- A complete disregard of any of their health
- A king's worth of coffee
- The new Jay Chou album
- ... and a blue screen of death!!! ...

We present to you our feature presentation
Be considerate. Please silence all pagers and cellular phones.
We started off by salvaging what we could from our midterm presentation...
We decided to implement our Translator as a Visitor

Translator{
1. Parse files
2. Makes minor changes to field types and collect all the fields of the class
3. Visit AST to modify tree’s new class expressions and new array expressions
4. Create Header
}

Header:

• Instantiates everything
• Checks for extensions
  – If there is an extension, it reads the header of the super class (it collects parent fields and parent methods, stores those as Linked Lists)
  – If there is none, then it hard codes java.lang.Object
• Traverses tree for its own methods and fields
  – Do something if it’s overwriting a super method
  – Do something else if it’s overloading any existing method
• Writes to file classname.h
Translator{
1. Parse files
2. Makes minor changes to field types and collect all the fields of the class
3. Visit AST to modify tree nodes’ new class expressions and new array expressions
4. Create Header
5. Header then returns list of methods, combine with fields to create a classNode
6. We resolve overloaded methods in the tree.
}
Overloaded method visitor

• For call expressions, e.g. `o.toString()`, we then take the primary identifier `o` and find the type of that by checking in the field table.
• Once we have the class, we pull the `classNode`
• We search for a method that has the name and same arguments
• Replace the call in the tree with the official C++ method name found in the `classNode`
Translator{
1. Parse files
2. Makes minor changes to field types and collect all the fields of the class
3. Visit AST to modify tree nodes’ new class expressions and new array expressions
4. Create Header
5. Header then returns list of methods, combine with fields to create a classNode
6. We resolve overloaded methods in the tree.
7. We’re ready for printing!!!
}
Our Printer

• Create 2 writers for everything in the object and the other for static things.

• In every class:
  – We ignore the constructor
  – Write static fields
  – For every method **main is treated slightly differently
    • Write the signature as it would appear in the header.
    • Write code block
  – Write to classname.cc
*Our Real Printer*

• For call expressions:
  – If it’s `system.out.print` use standard `cout`.
  – If it’s not, we print out “->_vptr->” instead of “.” and the identifier as the first argument
The End
Next season on Object Oriented Programming:

Java Compiler  Java Code
C++ Compiler

C++ Code
C++ Compiler

Java Code
FIT IN THE @#$%ING HOLE!!!