C++ Parsing (Or, How to Induce Heartattack)

C++ cannot be parsed without

- Context-sensitive parsing
- Unbounded lookahead
- A great deal of pain

C++ has a few constructs that are ambiguous and are resolved arbitrarily by the C++ standards committee:

```
int b;
float a(float(b)); // func decl or var def?
```

Maybe 3 parser generators can handle C++ right now.

Most C++ front ends are hand-built (and purchased).

C++ Trouble Spots

- Type names versus idenifiers

```
T(*a); // function call if T is ID
T(*a); // def of var “a is ptr to T”
```

- Qualified type names versus qualified identifiers

```
A::B::foo
A::B::T
```

or same as above but with arbitrary lookahead required

- Constructor versus member declaration

```
class T {
   T *a; // member variable definition
   T() {} // constructor definition
};
```

Yet More C++ Ick

This is the standard ambiguity listed in Ellis and Stroustrup.

```
T(*a)->m = 37; // statement
T(*a)(int); // ptr to func declaration
```

- Function-style initializers versus variable definition

```
T t(id);// func decl for t if id is a type
T t(id);// var def for t initialized to id if id is non-type
```

- Qualified pointer versus declaration

```
// have to see past A::B:: to the '*'
int A::B::*foo();
```

If the '*' is present, 'foo' is a ptr to a member function of A::B.

One Last Surprise

- `sizeof(expr)` vs `sizeof(abstract type)`

What does this mean? Unambiguous, but hard to parse.

```
// T(a) is type conversion or an expr
sizeof(T(a));
```

Resolving Symbols

C++ has 3 more dimensions than C. When looking up a symbol in the symbol table you must worry about:

- Local scope: could be a local variable
- Global/File scope; could be a global variable
- Inheritance; could be a member of class on path from current class to root class
- Function overloading; could be one of multiple functions with the same name; there could be a function defined in the file and one in the class hierarchy as well!
- User-defined type-casting; the type of an argument could be different than specified, thus, making a previously-invisible overloaded-function visible.

How Not To Parse C++

For context-sensitivity:

> Do not modify the lexer to return different token types depending on what the symbol table says an identifier is.

For unbounded lookahead:

> Do not hack the parser so that you can make it backtrack upon failure.

Do not write the parser by hand.