(Context: C - yacc - lex)

Frequently Asked Questions about the ANSI C Yacc Grammar

Q: How current is this? Are you still maintaining this?

As of 2012, yes. There are some versions of the grammars floating around in various states of disrepair that I can't do anything about. The current one is at www.quut.com/c/ANSI-C-grammar-y.html. (I didn't own the previous hosts, but do control quut.com, so there's some chance of this URL being more stable than the others.)

Q: So, can I use this to make derived work?

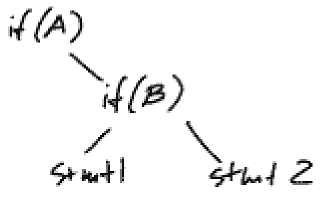
Yes, you can use this to make derived work with or without attribution; both I and the grammar's original poster are fine with that. (It's just a grammar. We've saved you some typing, but that's about it; the bulk of the work is still left.)

Q: Your grammar is broken! I'm getting a "shift-reduce conflict"!

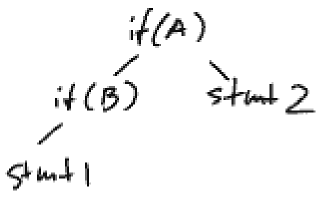
This is known as the *dangling else problem*. Theoretically, there are two possible parse trees for

```
if (A) if (B) stmt1; else stmt2;
```

In the correct parse tree, the "else" belongs to the "if (B)".



In the incorrect but theoretically possible parse tree, the else belongs to the if (A).



Programming languages usually* resolve this by attaching the else to the innermost open if. (This is the way many human languages deal with similar ambiguities as well.) C does it that way, too, and so do yacc and its descendants - it's safe to ignore the warning message, as long as you know your grammar and know what it means.

A second shift-reduce conflict was introduced in C11 with the new "_Atomic" type-specifier (if followed by parentheses) or type-qualifier.

You can make Yacc not print messages about shift/reduce conflicts by announcing how many there are in the header:

```
%expect 2
```

tells it to ignore one such warning. (So why didn't I include one in the grammar? Because this isn't a finished tool, just a part. I expect users to either just read or to heavily edit the grammar, and there's no point in polishing it beforehand.)

---

* Every once in a while, a language designer is fed up with the ambiguity and decides to eradicate it once and for all. Some otherwise Algol-like languages solve the problem by always requiring curly braces around their if/else parts (perl, sieve), or by using indentation itself as a bracketing mechanism (python).

Q: Your grammar is broken! It doesn't parse this C program here!

To parse full-fledged C source code, you'll need at least a preprocessor and a semantic analyser. For example, you'll need to parse C declarations to keep track of which words are names of types and which ones are still available as new identifiers. You'll need to keep track of scopes, and update your tables of what identifiers mean as their declarations move in and out of scope.

Writing a semantic analyser for C isn't easy.

From far away, the grammar looks like all you need to do is to compile and run it (surely, with this much Yacc input, how much work can there be left?), but that's misleading.

If you find anything else wrong, send email to *jutta@pobox.com*. Thanks!