

CS350 Lab 4: Abstract Syntax

Change the definition of `which_cs350_lab` in `tiger.cc` so that it has the value 4.

Modify the file `tiger-grammar.y` (and, if necessary, other files) to build an abstract syntax tree for tiger programs. This is discussed at length in Chapter 4 of Appel's book, but his approach involves a lot of work to build AST's in C rather than C++. You may either use my C++ versions of the AST files (`AST.h` and `AST.cc`) or use `AST_appel.h` (which should be compatible with what is in the textbook) or remove these files from the project and replace them with your own files, which you can build as extensions of your CS245 abstract syntax trees.

Your AST should be an attribute of the parse tree. You may either keep the attributes you used for detection of illegal lvalues and breaks in Lab 3, or get rid of these old attributes and check for illegal lvalues and breaks some other way (by traversing the AST, perhaps). The root of the complete AST for the entire program (which would presumably be an attribute of the expression in your starting production, `program` \rightarrow `expression`) can be communicated to the test program by setting the value of the global variable `AST_root`.

As with prior labs, this lab will have a combination of straightforward elements (those for which Appel's AST corresponds closely to your grammar) and things that are trickier (in this case, because the abstract syntax differs from the most straightforward concrete syntax). Many of the latter are discussed in the end of Appel's Chapter 4 (such as how to deal with `&` and `|`) or the start of my `AST.h` (such as how to deal with `if`'s that have no `else`). In particular, collecting consecutive sequences of function (or type) declarations into groups can be rather challenging. Fortunately, this will save you significant work in Lab 5.

As with all large programming projects, this one is probably best taken in a sequence of well-planned steps. Note that you can compile a program that would set at AST node to 0, as long as you don't then run that and try to use the AST — thus, you can do a first version of this lab in which you do all the things that aren't tricky and just set all the other AST's to 0 (or perhaps even leave them blank, if that will compile). So you could start with just integer literals and operations (`+`, `-`, `*`, etc.) and run some tests, and then add another kind of expression, and thus proceed in testable steps.

Record important design decisions for this lab in the file `Lab4-AbstractSyntax` in the `Design_Documents` subdirectory.

For 80% partial credit, do everything but arrays, records, and type declarations.