322 Compilers: Assignment 3
Translation to an intermediate langauge

Your Job

Design two new (mutually referential) AST data structures that represent `tree-exp` and `tree-stm`. Implement a transformation that turns your AST into a `tree-exp` and implement a function (or method) that prints your `tree-exp`s according to the grammar given in `tree-exp-sem.pdf`.

If the program indexes past the bound of an array, the generated code should print `out of bounds` and terminate. If the program accesses a field of the nil value, then the generated code should print the message `nil dereference` and terminate.

Submission Instructions

Submit a single `zip` file containing your test cases in a directory called `3a` and your code in a directory called `3b`. The `3b` directory should contain a script called `toil` that accepts a filename on the commandline and then prints out the `tree-exp` corresponding to the Tiger program in the file.

Your `zip` file should also contain subdirectories `1a`, `1b` `2a`, and `2b` containing either your submissions from last time, or fixed versions of them. The revised implementations will be used when we re-run the parsing and typechecking test fests (but not the revised test cases).

The test cases should all have type `int`, `string`, or `array of int` (note that this is different from the type checker assignment).

The test case files should be named according to their types. Specifically, the test case filenames should begin with `int`, `str`, or `ant` (for array of int). Your program may assume that the names match up to the types properly. Conversely, your test cases must be named properly to be used in the test fest. Beware: your commandline tool might be passed relative path names, e.g.,

```
toil ../robbys-tests/int1.tig
```

To run the test fest, we will run each test case in the evaluator and compare that result to the result of running `evalil` on the output of your translator.

Tips

- Extend environments to map identifiers to the temporaries used to store their values.
- Watch out for lvals – the might get used in an assignment, so they better return something suitable for the first argument of a `move` statement.
- Have a series of code at the beginning of your program that initializes the string values and gets their locations in memory. Store those in some special temporary values and then use the temporaries where the strings actually were in the program.
- Instead of passing around a boolean to indicate if you are in a `while` loop (like you did in your type checker), pass around a label to jump to when you are in the body of a `while` loop.
- Use the `allocate` function to create arrays, records and strings (local variables do not need to be allocated).
- Insert a call to one of the printing functions (`printint` for integers, `printstr` for strings, and `printant` for arrays of integers) around your entire program to see the results.
- Build your translator in stages, making sure that you are running and passing all of your test cases after each stage. Organize the stages by the different forms in the language. Before implementing any of the actual translation code, set it all up to so that it just signals errors. Next, build up your testing harness. Then add tests, observe that they fail (i.e., run the entire test suite), fix them, and iterate. (The point here is that if you have to make some sweeping change to your code, you can be sure that you don't miss anything.)