CS 301 Homework 4 – due 3:30pm, Wednesday, March 10, 1999

Typechecking

In this assignment, you will build a type-checker for the PCAT abstract syntax trees generated in assignment 3. To do this, you'll process declarations to extract type information about identifiers and process statements and expressions to make sure that identifiers and literals are used correctly.

For this assignment, assume that any use of real numbers or records in PCAT programs is *illegal*: that is, issue an error if either of these features is used. In addition, assume that type definitions are *not* recursive, i.e.,

```
TYPE t1 = ...
     t2 = ...
```

should be handled as completely equivalent to

```
TYPE t1 = ...
TYPE t2 = ...
```

However, you *should* handle recursive procedure declarations correctly.

Your typechecker should detect all semantic errors, including:

- incompatible or illegal operand types in expressions, statements, or declarations;
- uses of undefined symbols;
- multiple declarations of a symbol within a single scope;
- attempting to assign to a constant;
- incorrect number or types of arguments in procedure call;
- `EXIT` statements in inappropriate contexts;
- `RETURN` statements in inappropriate contexts, or lacking a return value when one is required or vice-versa;
- (for purposes of this assignment only) use of constructs involving real numbers or records. (Any such use should produce *some* error message; the message text need not specifically complain about reals or records, however.)

Do *not* attempt to check that function procedures always execute a `RETURN`, or that array references are within bounds. Consult the PCAT Language Reference Manual for detailed rules about type compatibility.

Your typechecker should be an executable called `check`, which reads input from `stdin`. The input consists of a PCAT program in externalized AST format, as produced by the `parser` of assignment 3. (Thus, the output of `parser` can be piped into `check`.) When it detects an error, `check` should issue a suitable message to `stderr` citing the nature of the error and the line number associated with the AST construct where the error occurred, and then exit gracefully. It should *not* attempt error recovery. If there are no errors, `check` should produce no output at all. As usual, the "correct" behavior of the the typechecker is specified by the behavior of `/u/cs301acc/4/check`. The precise text of your error messages does not have to match the "official" executable, however.

Implementation and Program Submission

Several support files are provided to simplify the implementation:

- As in assignment 3, `ast.[hc]` contain suitable type and constructor definitions for AST trees.
- `parse_ast.y` defines a function `parse_Program` that reads externalized AST and builds an internal tree using the constructor functions.
- `types.[ch]` defines suitable type and constructor definitions for describing PCAT types.

You will need to use a global symbol table or an environment parameter (as in assignment 1's `check.c`) to keep track of the definitions currently in scope. There is no need to engineer this mechanism to handle large numbers of symbols, nor to worry about deallocation of symbol information.

Prepare a `makefile` that builds your checker and produces an executable `check`.

Submit your program by `mailing` a `shar` "bundle" containing the relevant files to `cs301acc`.