322 Compilers: Assignment 1a
Test Cases for a Tiger Parser

Design (at least) 25 passing and 5 failing test cases for parsing Tiger expressions. For each passing test case, hand in two files, one called *file*`.tig` containing a tiger program that should parse along with *file*`.sxp` showing how it parses (according to the left-hand column below). For each failing test case, hand in one file called *file*`.tig` containing input the parser should reject and a *file*`.sxp` file containing `#illegal`.

Submit a single `zip` file containing your test cases in a directory called `1a`.

Parsed Tiger expressions:

exp	←	(*biop exp exp*)
	/	(`:=` *lvalue exp*)
	/	*lvalue*
	/	*num*
	/	*str*
	/	`nil`
	/	`()`
	/	(`new` *id* (*id exp*) ⋯)
	/	(`new-array` *id exp exp*)
	/	(`let` (*dec* ⋯) *exp*)
	/	(`begin` *exp exp exp* ⋯)
	/	(`when` *exp exp*)
	/	(`while` *exp exp*)
	/	(`if` *exp exp exp*)
	/	(`for` (*id exp exp*) *exp*)
	/	(`break`)
dec	←	(`var` *id exp*)
	/	(`var` *id id exp*)
	/	(`type` *id ty*)
biop	←	*relop* / `+` / `-` / `*` / `/`
relop	←	*eqop* / `<=` / `>=` / `<` / `>`
eqop	←	`=` / `<>`
lvalue	←	*id*
	/	(`dot` *lvalue id*)
	/	(`aref` *lvalue exp*)
ty	←	*id*
	/	(`record` (*id id*) ⋯)
	/	(`array` *id*)
num	←	a series of digits
str	←	a string, in any valid PLT Scheme string notation; see `http://docs.plt-scheme.org` for details, e.g., `"abc"` or `"two\nlines"`
id	←	a series of letters, numbers, and underscores that begins with a letter

Use `(call-with-input-file "`*file*`.sxp" read)` in PLT Scheme to be sure your *exp*s are well-formed.

Changes to Tiger from the text:

- omit function declarations
- omit function calls from expressions
- change the two-arm'd `if` to: `when` *exp* `do` *exp*
- Add a `new` keyword before record creation and array creation, e.g.,

```
let type t = {x:int,y:int}
in new t {x=1,y=2} end
```

- ignore the `\^c` escapes in strings
- the "f" escapes in strings should only contain newlines, tab characters, return characters and spaces, i.e., ASCII codes 9, 10, 13, and 32.
- `let` expressions with no expressions in the body should be parsed as if they had `()` in the body; with two or more expressions should be parsed with a `begin` expression in the body.
- The expression

  ```
  if 1 then 2 else 3 + if 4 then 5 else 6
  ```

  is illegal, but adding parens should make it parse, i.e:

  ```
  if 1 then 2 else 3+(if 4 then 5 else 6)
  (if 1 then 2 else 3)+(if 4 then 5 else 6)
  ```

  Also, other expression forms that do not have a closing token (i.e., `while`, `when`, etc) followed by an infix operator (i.e., `+`, `=`, `:=`, etc) require parentheses.
- Similar to the above, expression forms that do not have a closing token (i.e., `if`, etc) must be parenthesized if they follow an infix operator (i.e., `+`, `=`, `:=`, etc)