Matt Leifer 4/25/16

My implementations of the functions in expr.ml and evluation follow the spec with only one real significant different. I have to functions that can convert an expression into a string: exp_to_string_AST and exp_to_string_CON which will turn an expression into either an abstract syntax tree or the concrete syntax respectively.

So for my extension of **MiniMl** I added floats, strings, the unit type, and lists to the language. Floats works exactly as they do in standard OCaml. A float is a number from 0 to 9 followed by a '.' and then possible another number. I also added the division operator / for both floats and ints.

Strings can contain any sequence of alphanumeric characters and _ but notably not spaces and are demarcated by a pair of " marks. So the format is, "<string>". The empty string is "".

Units can be used in functions, as arguments to functions, and in let expressions but not let rec expressions. Units are represented by () and a function that accepts a unit must take a unit as an argument. By which I mean, that if a user enters `let f = (fun () -> fun x -> x + x) in f 42 3)` miniml will throw and error because the first argument of f has to be of unit type (solution: change 42 to ()). This is the only time when any of the types are checked in MiniMl.

Lists work similarly to lists in Ocaml − expressions (lists or otherwise) can be added to another list using the :: operator and two lists can be concatenated using the @ operator. The lists however can contain expressions that OCaml would consider to have different types. The concrete syntax of a valid list in Miniml would be `[1; 2; "a"; 3+4; 4.8; let f = fun x -> x + 1 in f 4]` even though this list wouldn't be valid in OCaml because each element of the list has a different type. Since these are all represented as "expressions" they are all effectively the same type. The format of a correct list is `[<exp>;<exp>;...]`, with expressions separated by ';'and enclosed by '['and ']'. An empty list is just []. Calling one of the evaluate functions on the list will result in a list with all the internal expressions evaluated. So the list above would evaluate to [1; 2; "a"; 7; 4.8; 5]. Because MiniMl lacks pattern matching statements it's impossible to implement functions that map lists to other lists or get elements out of a list once they've been put in. The last element in a list cannot be followed by a semicolon like it can be in OCaml.

One other difference between lists in OCaml and my implementation of MiniMl is that in OCaml a "let" or "let rec" can only come at the end of a list because in OCaml, if there is a "let" or "let rec" expression followed by a semi−colon, even one meant to act as a list separater must return unit. In my implementation, even if you have a let or let rec expression followed by a semi− colon, MiniMl still processes it. So for example in OCaml, the list:
`[let rec f = fun x -> if x = 0 then 1 else x * f (x - 1) in f 4; 10]`
is not a syntactically correct list because the let rec expression does not evaluate to unit and OCaml just returns [10] but throws a warning about the let rec not evaluating to unit. In my implementation, this same input would become [24; 10].