C/C++ 2015/16 programming exercise 2

This exercise is about representing the syntax of programming languages as C data structures, namely parse trees, and processing such trees.

Here is the grammar of a simple language of expressions (loosely based on Lisp syntax):

E	$\rightarrow$	n	(constant)
E	$\rightarrow$	x	(variable)
E	$\rightarrow$	`(+` L`)`	(operator application for addition)
E	$\rightarrow$	`(*` L`)`	(operator application for multiplication)
E	$\rightarrow$	`(=` x E E`)`	(let binding)
L	$\rightarrow$	$E\,L$	(expression list)
L	$\rightarrow$		

Constants can be integers. Variable names must start with a letter and can be up to 7 characters long. Operator application applies to a list of arguments, e.g.

```
(+ 2 3 5)
```

evaluates to 10.

In a let-binding

```
(= x E1 E2)
```

the variable `x` is bound to the value of E1 in the evaluation of E2. For example,

```
(= x (+ 2 3 5) (* x x (+ x x))
```

should evaluate to 2000. Note that there could be different scopes for the same variable, so that

```
(= y 2 (+ y (= y 10 y) y))
```

should evaluate to 14 (and not 22).

In C, the above grammar can be represented by the types in
`http://www.cs.bham.ac.uk/~hxt/2015/c-plus-plus/evalexp.h`

Your task

Write an evaluation function

```
int evalexp(struct exp *e)
```

that evaluates an expression as outlined above. You may also write a helper function that evaluates an expression list.

Given that an expression may contain variables, you will need to write a helper function

```
int evalexpenv(struct exp *e, struct env *env)
```

In addition to an expression, this evaluation function takes an *environment* as a parameter. An environment in this context is a data structure that binds variables to their values. You need to define a suitable data structure. One possibility is a list where each element contains a variable and its corresponding value.

2 points are given if your `evalexp` function works on tests without let bindings.

3 more points are given if your `evalexp` function also works on tests with let bindings.

As this exercise is not about memory allocation, you are not required to deallocate anything. However, tests need to terminate successfully. Segfault and other errors get 0 marks.

Your submission to Canvas should be a single .c file implementing `evalexp` along with any helper functions and structs, but without a main function.

A main file is here:
`http://www.cs.bham.ac.uk/~hxt/2015/c-plus-plus/evalexpmain.c`

Stretch exercise: 1 bonus point

Write a function

```
int parseandeval(char *p)
```

that parses a string in the language above and evaluates it (in the empty environment). For exampe,

```
parseandeval("(= x (+ 2 3 5) (* x x (+ x x))")
```

should return 2000.

It is recommend that you have a look at this file to see how to parse with lookahead:
`http://www.cs.bham.ac.uk/~hxt/2015/c-plus-plus/ParserTree.c`