CEng 242 Homework 2

Due: 3^{rd} April 2004

You are given a datatype `let_expr` with the following definition:

```
datatype let_expr =
         const of int | var of string | plus of let_expr * let_expr |
         multiply of let_expr * let_expr |
         letexpr of (string*int) list * let_expr;
```

A `let_expr` is one of the following:

- An integer constant
- A string representing a variable
- `plus` or `multiply` operation of two `let_expr`'s
- A `letexpr` defining a list of strings as local variable names and values on the enclosed `let_expr`.

`let_expr`$([(x,3),(y,1),(z,0)], expr)$ defines the variables $[x, y, z]$ as values $3, 1, 0$ respectively in the scope of the *expr*. Redeclaration of same variable is possible within *expr* and local declaration overrides the previous one. If the same variable name is repeated in the definition list, the last definition overwrites the former one. Note that the body of these variable definitions are just simple integers.
As an example, the following `let_expr`:
`letexpr([("x",1),("y",2)], letexpr([("x",3)], multiply(var "x", const 4)))`
Corresponds to the following expression in ML:

```
let val x = 1;
    val y = 2
in
    let val x = 3
    in  x * 4;
    end;
end;
```

A variable reference `var "x"` is called **bound** if `"x"` is listed in any of the upper level `letexpr`'s in the same path. If a variable reference is not bound it is called a **free** variable.

a) Write a function `FV` that will get a `let_expr` and return the list of free variables in the let expression. Multiple free occurrences of the same variable symbol should be listed once. Function will have the following type signature:

```
FV : let_expr -> string list
```

Examples:

```
- FV(letexpr([("x",1)],plus(var "x",const 1)));
- val it = [] : string list;

- FV(plus(var "x",plus(letexpr([("y",0)],
                          plus(var "z",var "z")),
                       var "z")));
- val it = ["x","z"] : string list;

- FV(letexpr([("x",1)],
       plus(letexpr([("w",2)],
              plus(var "w", multiply(var "x", const 3))),
            var "w")));
- val it = ["w"] : string list;

- FV(letexpr([("w",1)],
       plus(letexpr([("x",5)],
              plus(var "w", multiply(var "x",const 3))),
            var "w")));
- val it = [] : string list;
```

b) Write a function `substitute` which will take a `let_expr` parameter $expr_1$. It will return a new `let_expr` $expr_2$ where all *applied* occurrences of variables will be replaced by the constant values in their binding occurrences. Resulting value should not contain any `letexpr` tagged values. Free variables should left as they are.

`substitute : let_expr -> let_expr`

Example:

```
- substitute (letexpr([("w",1)],
                plus(letexpr([("x",3)],
                       plus(var "w",multiply(var "x", const 3))),
                     var "w")));

val it = plus (plus (const 1,multiply (const 3,const 3)),const 1) : let_expr


- substitute (letexpr([("x",12)],
                plus(letexpr([("w",7)],
                       plus(var "w",multiply(var "x", const 3))),
                     var "w")));

val it = plus(plus(const 7,multiply(const 12, const 3)), var "w") : let_expr



- substitute (letexpr([("z",10)],
                plus(var "x",plus(letexpr([("y",3),("z",0)],
                                          plus(var "z",var "z")),
                     var "z"))));
val it = plus(var "x",plus(plus(const 0,const 0), const 10)) : let_expr
```

c) Write a function `evaluate` which will take a `let_expr` parameter and evaluate it as an integer value if there is no free variable in it. If there is at least one free variable in the expression, it will return `NONE`. Otherwise it will evaluate the expression after the variable substitution and return the integer as the `SOME` *`intvalue`*.

```
evaluate : let_expr -> int option
```

`option` type is a built-in type in ML and defined as:

```
datatype 'a option = NONE | SOME of 'a;
```

Example:

```
- evaluate (letexpr([("w",1)],
                 plus(letexpr([("x",3)],
                         plus(var "w",multiply(var "x", const 3))),
                      var "w")));

val it = SOME 11 : int option


- evaluate (letexpr([("x",12)],
                 plus(letexpr([("w",7)],
                         plus(var "w",multiply(var "x", const 3))),
                      var "w")));

val it = NONE : int option
(* w is free *)

- evaluate (letexpr([("z",10),("x",5)],
                 plus(var "x",plus(letexpr([("y",3),("z",0)],
                                      plus(var "z",var "z")),
                      var "z"))));
val it = SOME 15 : int option
```

Follow the newsgroup for submission details. Cheaters will get 0 from all of the 6 homeworks.