CS2104 Programming Language Concepts
Laboratory Assignment 4 : Parser for Lambda Expression

(Due: 22th October 2013 8pm)

Please submit your solution into IVLE workbin as a single OCaml file. You can use the utilities provided by globals.ml, debug.ml, gen.ml, etc but must not change their implementation. The utility modules need not be submitted.

You are asked to design a parser for lambda calculus with let-construct of the following form:

```
<lam-expr> ::=  <identifier> | <lam-expr> <lam-expr>
              | ( <lam-expr> )  | let <identifier> = <lam-expr> in <lam-expr>
              | \  {<identifier>}+ . <lam-expr>
```

For applications, you must make sure that it associates to the left. (Hint: you may make use of repeat1 to try get a number of lambda terms before forming the nested binary applications.) For lambda abstraction, you must make sure that it extends as far as possible to the right. We have already provided an abstract syntax for your lambda term, namely:

```
type lambda =
      | Var of string
      | App of lambda * lambda
      | Lam of string * lambda
      | Let of string * lambda * lambda;;
```

Your task is the complete the parser Lambda.lam_expr and also to support multi-arguments lambda abstraction of the form (\x y z. x (y z)) which will be parsed as:

Lam (“x”, Lam(“y”, Lam(“z”, App(Var ”x”,App(Var ”y”,Var ”z”))))))

We have given you some test cases. Please design additional test cases for your parser.
You must also complete a free variable function, called fv, that would return a set of free variables for each given lambda expression. Some sample trace of this function is captured below:

```
fv@3
fv inp1 :((\x.((\z.z) x)) ((\z.z) (\x.(x x))))
fv@3 EXIT:[]

fv@4
fv inp1 :(y (x y))
fv@4 EXIT:[y,x]

fv@5
fv inp1 :let y=(\z.z) in (\x.y) end
fv@5 EXIT:[]
```