1(a)

```
type exp =
      Var of string
    | Int of int
    | Add of exp * exp
    | Mul of exp * exp;;

let rec deriv e x =
    match e with
          Var y -> if y = x then Int 1 else Int 0
        | Int _ -> Int 0
        | Add (e1, e2) -> Add (deriv e1 x, deriv e2 x)
        | Mul (e1, e2) -> Add (Mul (deriv e1 x, e2), Mul(e1, deriv e2 x));;

deriv (Mul (Add (Var "x", Var "y"), Add (Var "y", Int 2))) "x";;
deriv (Mul (Add (Var "x", Int 1), Add(Var "x", Int 2))) "x";;
```

実行結果

```
type exp = Var of string | Int of int | Add of exp * exp | Mul of exp * exp
val deriv : exp -> string -> exp = <fun>
- : exp =
Add (Mul (Add (Int 1, Int 0), Add (Var "y", Int 2)),
 Mul (Add (Var "x", Var "y"), Add (Int 0, Int 0)))
- : exp =
Add (Mul (Add (Int 1, Int 0), Add (Var "x", Int 2)),
 Mul (Add (Var "x", Int 1), Add (Int 1, Int 0)))
```

1(b)

```
type exp =
      Var of string
    | Int of int
    | Add of exp * exp
    | Mul of exp * exp;;

let rec simplify e =
    match e with
          Add (Int 0, e0) -> e0
        | Add (e0, Int 0) -> e0
        | Mul (Int 0, e0) -> Int 0
        | Mul (e0, Int 0) -> Int 0
        | Mul (e0, Int 1) -> e0
        | Mul (Int 1, e0) -> e0
        | Add (e1, e2) -> Add (simplify e1, simplify e2)
        | Mul (e1, e2) -> Mul (simplify e1, simplify e2)
        | Var y -> Var y
        | Int e0 -> Int e0;;

let rec deriv' e x =
    match e with
          Var y -> if y = x then Int 1 else Int 0
        | Int _ -> Int 0
        | Add (e1, e2) -> simplify(Add (simplify (deriv' e1 x), simplify (deriv' e2 x)))
        | Mul (e1, e2) -> simplify(Add (simplify (Mul (deriv' e1 x, e2)), simplify (Mul(e1, deriv'
e2 x))));;

deriv' (Mul (Var "x", Var "y")) "x";;
deriv' (Mul (Add (Var "x", Int 1), Add(Var "x", Int 2))) "x";;
deriv' (Mul( Mul( Add (Var "x", Int 1), Add (Var "x", Int 2)), Mul( Add (Var "x", Int 3), Add (Var
"x", Int 4)))) "x";;
```

実行結果

```
type exp = Var of string | Int of int | Add of exp * exp | Mul of exp * exp
val simplify : exp -> exp = <fun>
val deriv' : exp -> string -> exp = <fun>
- : exp = Var "y"
- : exp = Add (Add (Var "x", Int 2), Add (Var "x", Int 1))
- : exp =
Add
 (Mul (Add (Add (Var "x", Int 2), Add (Var "x", Int 1)),
   Mul (Add (Var "x", Int 3), Add (Var "x", Int 4))),
 Mul (Mul (Add (Var "x", Int 1), Add (Var "x", Int 2)),
  Add (Add (Var "x", Int 4), Add (Var "x", Int 3))))
```

1(c)

```
type exp =
      Var of string
    | Int of int
    | Add of exp * exp
    | Mul of exp * exp
    | Exp of exp * int;;

let rec deriv e x =
    match e with
          Var y -> if y = x then Int 1 else Int 0
        | Int _ -> Int 0
        | Add (e1, e2) -> Add (deriv e1 x, deriv e2 x)
        | Mul (e1, e2) -> Add (Mul (deriv e1 x, e2), Mul(e1, deriv e2 x))
        | Exp (e0, n)  -> Mul (Mul (Int n, Exp (e0, n-1)), (deriv e0 x));;

deriv (Exp (Add (Var "x", Int 1), 2)) "x";;
```

実行結果

```
type exp =
    Var of string
  | Int of int
  | Add of exp * exp
  | Mul of exp * exp
  | Exp of exp * int
val deriv : exp -> string -> exp = <fun>
- : exp =
Mul (Mul (Int 2, Exp (Add (Var "x", Int 1), 1)), Add (Int 1, Int 0))
```