CMSC 22620 Spring 2009	Implementation of Computer Languages	Homework 1 Due April 14, 2009

Recall the CPS conversion from class that translates from untyped λ-calculus to a *continuation-passing style* version of the λ-calculus. We can represent these languages with the following SML modules:

```
structure Lambda : sig
    type var
    datatype exp = Var of var
                 | Abs of var * exp
                 | App of exp * exp
  end = ...

structure CPS : sig
    type var = Lambda.var
    datatype value = Var of var
                   | Abs of (var list * exp)
    and exp = Value of value
            | App of value * value list
  end = ...
```

Note that the CPS representation allows multiple-argument functions. The Danvy-Filinski CPS conversion that handles tail recursion is implemented as follows:

```
structure L = Lambda
structure C = CPS

fun cvt (L.Var x, k : C.value -> C.exp) = k (C.Var x)
  | cvt (L.Abs(x, e), k) = let
      val k' = C.fresh()
      in
        k (C.Abs([x, k'], tailCvt(e, C.Var k')))
      end
  | cvt (L.App(e1, e2), k) = let
      val a = C.fresh()
      in
        cvt(e1, fn m => cvt(e2,
          fn n => C.App(m, [n, C.Abs([a], k(C.Var a))])))
      end
```

```
and tailCvt (L.Var x, k : C.value) = C.App(k, [C.Var x])
  | tailCvt (L.Abs(x, e), k) = let
      val k' = C.fresh()
      in
        C.App(k, [C.Abs([x, k'], tailCvt(e, C.Var k'))])
      end
  | tailCvt (L.App(e1, e2), k) = let
      val a = C.fresh()
      in
        cvt(e1, fn m => cvt(e2, fn n => C.App(m, [n, k])))
      end

fun transform e = cvt (e, fn x => C.Value x)
```

Assume that we extend these languages with constants and conditionals. For the `Lambda` representation, we add the following constructors to the `exp` type:

```
datatype exp = If of exp * exp * exp
             | Const of const
             | ...
```

The new CPS representation is then

```
datatype value = Var of var
               | Abs of (var list * exp)
               | Const of const
and exp = Value of value
        | App of value * value list
        | If of value * exp * exp
```

Extend the `cvt` and `tailCvt` functions to handle constants and conditionals.