COMP 212 Spring 2015
Homework 08

This short homework will be a chance to get used to using modules, which will be used heavily in the final assignment next week.

1 Modules

Recall from last semester that a pixel can be represented by four 8-bit numbers, which represent the α (transparency), red, green, and blue values.

The following signature presents an interface to pixels:

```
signature PIXEL =
sig
    type pixel
    (* show_pixel p returns the (alpha,red,green,blue)
       values of the pixel.
       If show_pixel p == (a,r,g,b) then
       all four numbers are in the range the range [0,256) *)
    val show_pixel : pixel -> int * int * int * int
    (* Make a pixel from its (alpha,red,green,blue) values.
       Assumes the integers are in the range [0,256) *)
    val make_pixel : int * int * int * int -> pixel
end
```

This says that a pixel can be converted to and from a tuple of (α,red,green,blue) values.

One simple implementation of pixels is as a tuple of (α,red,green,blue) values:

```
structure PairPixel : PIXEL =
struct
    datatype pixel = P of int * int * int * int
    fun make_pixel x = P x
    fun show_pixel (P (a,r,g,b)) = (a,r,g,b)
end
```

The hidden datatype constructor makes the type of pixels abstract.

This implementation works fine, but is space-inefficient, because each of the four `int`s are stored using a whole word (32 bits), but we only need 32 total to represent a pixel.

Task 1.1 (20 pts). Give structure named `WordPixel` that implements the `PIXEL` signature using a **single** `Word32.word` to represent a pixel—this will use a quarter of the space of the integer implementation. You can use the functions described in the following signature: http://sml-family.org/Basis/word.html as well as the helper functions in `WordUtil` in the homework file. Make sure to make the type `pixel` abstract!

Task 1.2 (5 pts). We have provided a functor

```
functor Test(P : PIXEL)
```

that takes an implementation of pixels and runs a few tests on it (because the tests are `val` declarations in the module, the tests are evaluated when the functor is applied). Call this functor on your module to test your code.

Task 1.3 (20 pts). Write a client function `remove_red` that takes a sequence of pixels and removes the red from each one. Your code should be in a functor so that it works for any implementation of pixels.

```
functor Images(P : PIXEL) : IMAGE_TRANSFORMATIONS
```

Task 1.4 (5 pts). We have provided a functor `RemoveRedTest` that tests a remove red function on a few pixels. Test your code by calling this functor on your remove red using your word-based pixel implementation.