HW 5 **Prof. Caldwell**
Due: 18 September 2007 **COSC 3015**

1

We defined `list_sum` and `map` in class as follows:

```
list_sum [] = 0
list_sum (h:t) = h + list_sum  t

map f [] = []
map f (h:t) = (f h) : (map f t)
```

We used `map` to apply the function cirlce area to the elements of a list so we could apply the `list_sum` function to the result. *i.e.* we replaced the rather unwieldy expression

```
total_area = (pi * r1 ^ 2) + (pi * r2 ^ 2 ) + (pi * r3 ^ 2)
```

by the following:

```
total_area = list_sum (map circle_area [r1,r2,r3])
                where circle_area r  = pi * r ^ 2
```

Note that `list_sum` could be generalized since we could pass the operator (instead of having `+` built-in) and could pass the identity for the operator (instead of having `0` built-in.)

Problem 1.1. Write a function `apply_op` whose type is given as

```
apply_op :: (a -> a -> a) -> a -> [a] -> a
```

The first argument of type `(a -> a -> a)` is the type of the operator (*e.g.* the type of `(+)`). The second argument is the identity for the specified operator and the third is a list of elements to apply the operator to.

Hint: You can use the `list_sum` function as a model – nothing much has to change except you need to give it two more arguments. Also, a function (say `f x y = x + y`) can be applied in infix for by writing `4 ‘f‘ 5` instead of `f 4 5`.

You should be able to get the following behavior from your function.

```
Hugs>:t apply_op
:t apply_op
apply_op :: (a -> b -> b) -> b -> [a] -> b
Hugs> apply_op (+) 0 [1,2,3,4]
apply_op (+) 0 [1,2,3,4]
10
Hugs> apply_op (+) 0 [-1,-2,-3,-4]
apply_op (+) 0 [-1,-2,-3,-4]
-10
Hugs> apply_op (-) 0 [1..10]
apply_op (-) 0 [1..10]
-5
```

```
Hugs> apply_op (+) 0 [1..10]
apply_op (+) 0 [1..10]
55
Hugs> apply_op (*) 1 [1..10]
apply_op (*) 1 [1..10]
3628800
Hugs>
```