CSE 341 **Assignment 6**
November 27, 1998 **Solution**

1. Suppose that the following Miranda script has been filed in (it is on `orcas` in `~borning/miranda/assign6.m`).

```
cube x = x*x*x
my_map f [] = []
my_map f (x:xs) = f x : my_map f xs
my_append [] x = x
my_append (x:xs) ys = x : my_append xs ys
my_map2 f [] [] = []
my_map2 f (a:as) (b:bs) = f a b : my_map2 f as bs
rev f x y = f y x
alligator x y = 3+y
```

What is the result of evaluating the following Miranda expressions? (If there is a compile-time type error, or a run-time error, or a non-terminating computation, say so.) If the expression is followed by `::` then give the type, instead of the value.

`cube 3`	`27`
`cube ::`	`num->num`
`my_map ::`	`(*->**)->[*]->[**]`
`my_map cube ::`	`[num]->[num]`
`my_map cube [1..]`	`[1,8,27,64,125,216,343,512,729,1000,...]`
`my_append ::`	`[*]->[*]->[*]`
`my_map2 alligator [1..]  [10..]`	`[13,14,15,16,17,18,19,20,21,22,...]`
`my_map2 ::`	`(*->**->***)->[*]->[**]->[***]`
`my_map2 alligator ::`	`[*]->[num]->[num]`
`my_map2 (rev alligator) ::`	`[num]->[*]->[num]`
`alligator (1/0) (6/2)`	`6.0`
`alligator (6/2) (1/0)`	`<program error:  attempt to divide by zero>`
`alligator ::`	`*->num->num`
`(rev alligator) 10 20`	`13`
`rev ::`	`(*->**->***)->**->*->***`
`rev my_map2::`	`[*]->(*->**->***)->[**]->[***]`

2. Write and test a Miranda function `pythagoras` that takes the lengths of the two legs of a right triangle and returns the length of the hypotenuse. For example:

```
pythagoras 3.0 4.0  => 5.0
```

Solution:

```
pythagoras x y = sqrt (x*x + y*y)
```

Output:

```
pythagoras 4 5     6.403124237433
pythagoras 3 4     5.0
pythagoras 1 1     1.414213562373
pythagoras 1.5 2   2.5
```

3. Write and test a Miranda function `no_duplicates` that takes a list of numbers as an argument, and returns a new list which is like the argument but with duplicate elements dropped. The order of the elements in the returned list doesn't matter. For example:

```
no_duplicates [1, 3, 3, 8, 5, 3, 5]  => [1, 3, 8, 5]
no_duplicates [1, 3, 5]  => [1, 3, 5]
no_duplicates [ ]  => [ ]
```

Solution 1: (without using builtin functions)

```
no_duplicates [] = []
no_duplicates (a:as) = a : no_duplicates (remove_a as)
                       where
                       remove_a [] = []
                       remove_a (b:bs) = remove_a (bs), b = a
                                       = b : remove_a (bs), otherwise
```

Output 1:

```
no_duplicates [1,1,1,1,1]                       [1]
no_duplicates [1]                               1
no_duplicates "hello there"                     helo tr
no_duplicates [1,2,2,4,2,3,2,3,4,5,3,2,3,7]     [1,2,4,3,5,7]
no_duplicates []                                []
```

Solution 2: (using builtin function `filter`)

```
no_duplicates [] = []
no_duplicates (a:as) = a : no_duplicates (filter ((~=) a) as)
```

Output 2:

```
no_duplicates [1,1,1,1,1]                       [1]
no_duplicates [1]                               1
no_duplicates "hello there"                     helo tr
no_duplicates [1,2,2,4,2,3,2,3,4,5,3,2,3,7]     [1,2,4,3,5,7]
no_duplicates []                                []
```

Solution 3: (using builtin function `member`)

```
no_duplicates [] = []
no_duplicates (a:as) = no_duplicates as, member as a
                     = a : no_duplicates as, otherwise
```

Output 3:

`no_duplicates [1,1,1,1,1]`	[1]
`no_duplicates [1]`	1
`no_duplicates "hello there"`	lo thre
`no_duplicates [1,2,2,4,2,3,2,3,4,5,3,2,3,7]`	[1,4,5,2,3,7]
`no_duplicates []`	[]

4. Write and test a Miranda function `deriv` that takes a function f as an argument and returns a function that is (approximately) its derivative f'. Thus the type of `deriv` should be

   ```
   (num -> num) -> (num -> num)
   ```

   For example

   ```
   deriv sin
   ```

   should return a function. If you apply this function to 0, it should return (approximately) 1, since the derivative of sin is cos, and cos 0 is 1. As another example, suppose we define

   ```
   double x = 2*x
   ```

   Then `deriv` double should return a function. If you apply this function to 100, it should return (approximately) 2.

   Exact solution? It is not possible to get an exact solution to the derivative of an arbitrary `num->num` Miranda function. Instead we must approximate.

   Solution: (Note: your solution does not have to use `limit`)

   ```
   epsilon = 1.0e-8

   deriv_approx f x 0 = error "tried to approximate with h=0"
   deriv_approx f x h = (f(x+h) - f(x-h) )/(2*h)

   deriv_list f x = map (deriv_approx f x) [h | h <- epsilon, h/2.0 .. ]

   deriv f x = limit (deriv_list f x)
   ```

   Output: (some answers have been truncated (not rounded))

`map (deriv sin) [1,2,3,4]`	[0.5403,-0.4161,-0.9899,-0.6536]
`deriv sin 0`	1.0
`deriv sin 3.14159`	-0.999999993919
`deriv ((*) 5) 10`	5.000001124245
`deriv ((*) 5) 99`	4.999992597732
`deriv ((*) 5) 0`	5.0
`deriv cube 0`	0.0
`deriv cube 1`	2.999999981768