COMP3007A (Winter 2019) – "Programming Paradigms"

Quiz 2 of 2 **Model Solution for Version 2**

This is a closed book exam. No calculators, cellphones, laptops, or other aids are permitted. Answer every question in the space that has been provided. You must show all your work without skipping steps; correct answers that are presented without justification may receive a mark of zero.

Student Name ________________________________

Student Number ________________________________

1. [HASKELL] The following data type can be used to represent a list of Integer values…

```
data IList = EmptyList | Single Int | Cons Int Int IList
```

To clarify, using this representation…

`[]`	would be	`EmptyList`
`[1]`	would be	`Single 1`
`[2, 3, 4]`	would be	`Cons 2 3 (Single 4)`

etc.

Write a function that computes the product of a list encoded using this type. To clarify, your function should apply the multiplication operator to all the Ints in the IList. (Please note that the product of an empty list of Ints should be 1.) The type declaration for the function you will write has been provided below. You may not use any built-in functions in the creation of your solution, but the multiplication operator in Haskell is * and you are free to use that. [3.0 marks]

```
multz :: IList -> Int
```

```
multz :: IList -> Int
multz EmptyList = 1
multz (Single x) = x
multz (Cons a b c) = (a * b * (multz c))
```

2. [HASKELL] Consider the following custom data types for storing student data (specifically an association between full name and student number) and telephone numbers (specifically an association between a last name and a phone number).

```
school_database :: [ ( [Char] , Int ) ]
school_database = [ ("Helen Stokes", 100111222), ("Tobey Stokes", 100333444),
                    ("Jax Palmer", 100555666), ("Naomi Greig", 100777888) ]

phone_directory :: [ ( [Char] , [Char] ) ]
phone_directory = [ ("Helen Stokes", "(613) 746-3394"), ("Tobey Stokes", "(613) 616-1155"),
                    ("Landon Fuller", "290-3446"), ("Jax Palmer", "354-3646") ]
```

Write a recursive function that, when passed the phone_directory and a string of three digits, returns a list of every phone number in the directory that ends in those same three digits. Please note that the area code may or may not be recorded in the phone_directory. You may not use any built-in functions, nor are you permitted to use the !! operator, but you may write your own helper functions, if you wish. [3.0 marks]

```
foo :: [([Char],[Char])] -> [Char] -> [[Char]]
foo [] _ = []
foo ((a, b) : t) key
   | phone_match b key = b : (foo t key)
   | otherwise = (foo t key)

phone_match :: [Char] -> [Char] -> Bool
phone_match [] _ = False
phone_match (h:t) key
   | (t == key && length t == 3) = True
   | otherwise = (phone_match t key)

length [] = 0
length (h:t) = 1 + (length t)
```

This is a closed book exam. No calculators, cellphones, laptops, or other aids are permitted. Answer every question in the space that has been provided. You must show all your work without skipping steps; correct answers that are presented without justification may receive a mark of zero.

Student Name ______________________________

Student Number ______________________________

3. [HASKELL] Using the same custom data types that were presented in the previous question, write a function that, when passed the `school_database`, `phone_directory`, and a student number (as an Int), returns the phone number for that student. You must use the Maybe type to account for the possibility that you are passed a student number that does not exist or corresponds to a person who is not in the phone directory, and so the return value for the function you write must be `Maybe [Char]`. [3.0 marks]

```
foo :: [([Char],Int)]->[([Char],[Char])] -> Int -> Maybe [Char]
foo a b c = (qux b (bar a c))

bar :: [ ( [Char] , Int ) ] -> Int -> Maybe [Char]
bar [] _ = Nothing
bar ((name, id) : t) key
  | id == key = (Just name)
  | otherwise = (bar t key)

qux :: [ ( [Char] , [Char] ) ]-> Maybe [Char] -> Maybe [Char]
qux _ Nothing = Nothing
qux [] _ = Nothing
qux ((name, phone) : t) (Just key)
  | name == key = (Just phone)
  | otherwise = (qux t (Just key))
```

4. [HASKELL] Prove that `bar n = foo False n` by using structural induction. Use the following implementations for `bar`, and `foo` and refer to the individual lines in these implementations using the labels B1, B2, F1, F2A, and F2B. You are expected to follow the process of structural induction as it was demonstrated in class and you must show all your work (including the line applied during each step of your equational reasoning).

```
        bar :: [Bool] -> Bool
B1      bar [] = True
B2      bar (h:t) = h && (bar t)

        foo :: Bool -> [Bool] -> Bool
F1      foo _ [] = True
        foo x (h:t)
F2A       | h == x = False
F2B       | otherwise = foo x t
```

[5.0 marks]

```
Base Case; Prove:                    bar n = foo False n
```

bar [] = True by B1	foo False [] = True by F1

```
Inductive Case; Prove:
                    bar t = foo False t -> bar (h:t) = foo False (h:t)

Inductive Assumption:         bar t = foo False t

Case 1: h == True
```

bar (h:t) = h && (bar t) by B2 = True && (bar t) = (bar t) = foo False t by IA	foo False (h:t) = foo False t by F2B

```
Case 2: h == False
```

bar (h:t) = h && (bar t) by B2 = False && (bar t) = False	foo False (h:t) = False by F2A

This is a closed book exam. No calculators, cellphones, laptops, or other aids are permitted. Answer every question in the space that has been provided. You must show all your work without skipping steps; correct answers that are presented without justification may receive a mark of zero.

Student Name ______________________________

Student Number ______________________________

5. [HASKELL] Recreate the following function such that it is tail-call optimized. Do not forget to include a helper function (with type declaration `[Int] -> [Int]`) and a new type declaration for your tail-call optimized function. [4.0 marks]

```
foo [] = []
foo (h:t)
    | (odd h) = (foo t)
    | otherwise = ((3 * h) : (foo t))
```

```
foo_helper :: [Int] -> [Int]
foo_helper arg = foo_optimized arg []

foo_optimized :: [Int] -> [Int] -> [Int]
foo_optimized [] accumulator = accumulator
foo_optimized (h:t) accumulator
  | (odd h) = foo_optimized t accumulator
  | otherwise = foo_optimized t (accumulator ++ [(3 * h)])
```

6. [HASKELL] How would you write a SINGLE EXPRESSION using the built-in higher order functions `foldr`, `map`, and/or `filter` to provide the conjunction of the negations of the elements of a list of booleans? You cannot use any functions other than `not` to complete this question, but you can use the conjunction operators (i.e., &&). (n.b., The identity element of conjunction is True.) [4.0 marks]

```
foldr (&&) True (map not x)
```

7. [PROLOG] A person would be considered your "grandniece or grandnephew" if and only if that person can be defined as a child of one of your nieces or nephews. If you already have working predicates for `parent` and `sibling` (i.e., `parent(X,Y)` succeeds if and only if X is a parent of Y and `sibling(X,Y)` succeeds if and only if X and Y are siblings), write a predicate `grandnieceornephew/2` such that `grandnieceornephew (X,Y)` succeeds if and only if Y is the grandniece or grandnephew of x. [3.0 marks]

```
grandnieceornephew(X, Y)
  :- niecenephew(Z, Y), child(X, Z).

child(X, Y)
  :- parent(Y, X).

niecenephew(X, Y)
  :- sibling(Z, Y), child(X, Z).
```

8. [PROLOG] Assuming that the user continues pressing ";" as long as possible (thereby forcing Prolog to identify all possible solutions), what is the exact output provided by the following Prolog program in response to the query `foo(A)`? [3.0 marks]

```
foo(4).
foo(3).
foo(A) :- qux(A).
foo(A) :- bar(A); ack(A).
foo(1).
bar(1).
bar(A) :- ack(A), xyz(A).
qux(1).
qux(2).
ack(3).
xyz(3).
```

Write the Exact Output Here

```
A = 4 ;
A = 3 ;
A = 1 ;
A = 2 ;
A = 1 ;
A = 3 ;
A = 3 ;
A = 1.
```