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 Boolean values…

```
data BList = EmptyList | Single Bool | Cons Bool BList Bool
```

To clarify, using this representation…

```
                    []   would be   EmptyList
                [True]   would be   Single True
[True, False, False]   would be   Cons True (Single False) False    etc.
```

Write a function that computes the disjunction of a list encoded using this type. To clarify, your function should apply the logical "or" operator to all the Bools in the Blist. (Please note that the logical "or" of an empty list of Bools should be False.) 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 "or" operator in Haskell is || and you are free to use that. [3.0 marks]

```
orz :: BList -> Bool
```

```
orz :: BList -> Bool
orz EmptyList = False
orz (Single x) = x
orz (Cons a b c) = (a || c || (orz b))
```

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 full 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 school_database and a last name, returns a list of every full name in the school_database with that specified last name. You may assume that the full names that appear in the school_database are always exactly two names, separated by a single space. 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] , Int ) ] -> [Char] -> [[Char]]
foo [] _ = []
foo ((a, b) : t) key
  | bar a key = a : (foo t key)
  | otherwise = (foo t key)

bar :: [Char] -> [Char] -> Bool
bar [] _ = False
bar (h:t) key
  | h == ' ' && t == key = True
  | otherwise = (bar t key)
```

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 phone number (as a list of characters), returns the student number for that student. You must use the Maybe type to account for the possibility that you are passed a phone number that does not exist or corresponds to a person who is not in the school database, and so the return value for the function you write must be `Maybe Int`. [3.0 marks]

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

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

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

4. [HASKELL] Prove that `foo n = bar True n` by using structural induction. Use the following implementations for `foo`, and `bar` and refer to the individual lines in these implementations using the labels `F1`, `F2`, `B1`, `B2A`, and `B2B`. 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).

```
        foo :: [Bool] -> Bool
F1      foo [] = False
F2      foo (h:t) = h || (foo t)

        bar :: Bool -> [Bool] -> Bool
B1      bar x [] = False
        bar x (h:t)
B2A       | h == x = True
B2B       | otherwise = bar x t
```

[5.0 marks]

Base Case; Prove: `foo [] = bar True []`

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

Inductive Case; Prove:

`foo t = bar True t -> foo (h:t) = bar True (h:t)`

Inductive Assumption: `foo t = bar True t`

Case 1: `h == True`

`foo (h:t)`	`bar True (h:t)`
`= h || (foo t)  by F2`	`= True  by B2A`
`= True || (foo t)`	
`= True`	

Case 2: `h == False`

`foo (h:t)`	`bar True (h:t)`
`= h || (foo t)  by F2`	`= bar True t  by B2B`
`= False || (foo t)`	
`= foo t`	
`= bar True t  by IA`	

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)
   | (even h) = (foo t)
   | otherwise = (foo t) ++ [(h * 2)]
```

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

foo_optimized :: [Int] -> [Int] -> [Int]
foo_optimized [] accumulator = accumulator
foo_optimized (h:t) accumulator
  | (even h) = foo_optimized t accumulator
  | otherwise = foo_optimized t ((h * 2) : accumulator)
```

6. [HASKELL] How would you write a SINGLE EXPRESSION using the built-in higher order functions `foldr`, `map`, and/or `filter` to provide the sum of the squares of the elements of a list of integers? You cannot use any other functions to complete this question, but you can (of course) use the addition and exponentiation operators (i.e., + and **). [4.0 marks]

`n.b., (**2) is a unary operator for square`

```
foldr (+) 0 (map (**2) x)
```

7. [PROLOG] A person would be considered your "first cousin once removed" if and only if that person can be defined as a grandchild of one of your aunts or uncles. 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 `firstcousinonceremoved/2` such that `firstcousinonceremoved(X,Y)` succeeds if and only if `Y` is the first cousin once removed of x. [3.0 marks]

```
firstcousinonceremoved(X, Y)
  :- auntuncle(Z, Y), grandchild(X, Z).

auntuncle(X, Y)
  :- parent(Z, Y), sibling(X, Z).

grandchild(X, Y)
  :- parent(Z, X), parent(Y, 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(3).
foo(A) :- bar(A), qux(A).
foo(2).
foo(A) :- ack(A).
bar(1).
bar(A) :- ack(A); xyz(A).
qux(1).
qux(3).
ack(2).
ack(3).
xyz(3).
```

Write the Exact Output Here

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