University of Computing Science

Department of Computer Science
Functional and Logic Programming Exam

Duration: 2 hours
Reading Time: 15 minutes
Total Marks: 100

Instructions:

1. Answer all questions on the exam paper itself.
2. The number of marks for each question is given; use it as a guide for the time to spend on each question.
3. No electronic devices or additional materials are permitted.

Question 1: Haskell List Operations [15 marks]

Define a Haskell function `removeDuplicates :: Eq a => [a] -> [a]` that removes all duplicate elements from a list, keeping only the first occurrence of each element.

For example:

```
removeDuplicates [1, 2, 2, 3, 4, 4, 5] = [1, 2, 3, 4, 5]
removeDuplicates "mississippi" = "misp"
```

Question 2: Haskell Map and Filter [15 marks]

Define a Haskell function `squareOdds :: [Int] -> [Int]` that squares each odd number in a list and leaves even numbers unchanged.

For example:

```
squareOdds [1, 2, 3, 4] = [1, 2, 9, 4]
squareOdds [2, 4, 6] = [2, 4, 6]
```

Question 3: Haskell Trees [20 marks]

Using the following binary tree definition:

```haskell
data Tree a = Empty | Node a (Tree a) (Tree a)
```

Write a Haskell function `treeProduct :: Num a => Tree a -> a` that calculates the product of all values in the tree. Return 1 if the tree is empty.

For example:

```haskell
treeProduct (Node 2 (Node 3 Empty Empty) (Node 4 Empty Empty)) = 24
treeProduct Empty = 1
```

Question 4: Prolog List Membership [15 marks]

Define a Prolog predicate `contains_duplicates(L)` that succeeds if a list `L` contains any duplicate elements.

For example:

```
?- contains_duplicates([1, 2, 3, 4]).
false.

?- contains_duplicates([1, 2, 2, 3]).
true.
```

Question 5: Prolog List Manipulation [20 marks]

Define a Prolog predicate `split_even_odd(L, Evens, Odds)` that splits a list `L` into two lists: `Evens` for even numbers and `Odds` for odd numbers.

For example:

```
?- split_even_odd([1, 2, 3, 4, 5, 6], Evens, Odds).
Evens = [2, 4, 6],
Odds = [1, 3, 5].
```

Question 6: Haskell Higher-Order Functions [15 marks]

Define a Haskell function `applyTwice :: (a -> a) -> a -> a` that applies a given function twice to an argument.

For example:

```
applyTwice (+3) 7 = 13
applyTwice (*2) 4 = 16
```

Question 7: Prolog Tree Depth [15 marks]

Using the following tree structure in Prolog:

```
tree(nil).
tree(node(Left, Value, Right)) :- tree(Left), tree(Right).
```

Define a Prolog predicate `max_depth(Tree, Depth)` that calculates the maximum depth of a binary tree `Tree`.

For example:

```
?- max_depth(node(node(nil, 1, nil), 2, node(nil, 3, nil)), Depth).
Depth = 2.
```