German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

May 13 2017

Concepts of Programming Languages
Spring term 2017

Final Exam

Bar Code

Instructions: Read carefully before proceeding.

1) Duration of the exam: 3 hours (180 minutes).
2) (Non-programmable) Calculators are allowed.
3) No books or other aids are permitted for this test.
4) This exam booklet contains 13 pages, including this one. Three extra sheets of scratch paper are attached and have to be kept attached. **Note that if one or more pages are missing, you will lose their points. Thus, you must check that your exam booklet is complete**.
5) Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem or on the four extra sheets and make an arrow indicating that. **Scratch sheets will not be graded unless an arrow on the problem page indicates that the solution extends to the scratch sheets**.
6) When you are told that time is up, stop working on the test.

Good Luck!

Don't write anything below ;-)

Exercise	1	2	3	4	5	6	7	$\sum$
Possible Marks	10	14	8	10	12	8	16	78
Final Marks								

Exercise 1 (10 Marks)

Write the definition of the Prolog predicate `common(L1,L2,C)` which holds if `C` is the number of common elements between `L1` and `L2`. **You cannot use any predefined predicates. If you need to use any, implement it from scatrch.**

```
?- common([1,2,1],[1,2],C).
C = 3 ;
false.

?- common([1],[1,2],C).
C = 1 ;
false.

?- common([],[1,2],C).
C = 0.
```

Solution:

```
common([],_,0).
common([H|T],List,C):-
                my_member(H,List),
                common(T,C1),
                C is C1 + 1.
common([H|T],List,C):-
                \+my_member(H,List),
               common(T,C).


my_member(X,[X|T]).
my_member(X,[Y|T]):- my_member(X,T).
```

Exercise 2 (10+4=14 Marks)

A 4-digit number is lucky if the sum of the first two digits equals the sum of the last two digits.

a) Write a CLP predicate `lucky(A,B,C,D,N)` that holds if `N` represents a 4-digit lucky number consisting of the digits `A, B, C, D` where `A` is the left most digit and `D` is the right most digit.

Use the clpfd library.

Solution:

```
lucky(A,B,C,D,N) :-
    [A,B,C,D] ins 0..9,
    A+B #= C+D,
    N #= 1000*A+100*B+10*C+1*D.
```

b) Write a predicate `sum_all(Sum)` that computes the sum of all existinng lucky numbers. **You can use lucky/5**.

Solution:

```
sum_all(Sum):-
            set_of(N,lucky(A,B,C,D,N),List),
            sum_list(List,Sum).

sum_list([],0).
sum_list([H|T],Sum):-
                    sum_list(T,S1),
                    Sum is S1 + H.
```

Exercise 3 (2+4+2=8 Marks)

Given the following definition of the predicates `mystery/3` and `mystery/4`.

```
mystery(L1,N,L2) :- mystery(L1,N,L2,N).
mystery([],_,[],_).
mystery([_|Xs],N,Ys,0) :- mystery(Xs,N,Ys,N).
mystery([X|Xs],N,[X|Ys],K) :-
                             K > 0,
                             K1 is K - 1,
                             mystery([X|Xs],N,Ys,K1).
```

a) What is the result of the query `mystery([1,2],0,L).`

Solution:

```
L=[]
```

b) What is the result of the query `mystery([1,2,3],2,L).`

Solution:

```
L=[1,1,2,2,3,3]
```

c) What is the functionality of `mystery(L1,N,L2)` for any `L1,N,L2`.

Solution:

`L2` contains the elements in `L1` where every element is duplicated `N` times.

Exercise 4 (8+2=10 Marks)

a) Write a Haskell function

```
changeFirst p val xs
```

that returns a list that looks like `xs` except that the leftmost item in `xs` that satisfies predicate `p` is replaced by `val`. If none of the elements of `xs` satisfy `p`, the `xs` is returned.

Here are some examples,

```
changeFirst even 33 [1, 7 , 4, 8, 2] = [1, 7, 33, 8, 2]
changeFirst even 33 [1,3,7,9] = [1,3,7,9]
```

Solution:

```
changeFirst _ _ [] = []
changeFirst p val (x:xs)
    | p x = val : xs
    |otherwise = x : (changeFirst p val xs)
```

b) Give the type of the function `changeFirst`.

Solution:

```
changeFirst :: (a -> Bool) -> a -> [a] -> [a]
```

Exercise 5 (10+2=12 Marks)

a) Write a function `f :: [Int] -> [Int] -> Int` that computes the sum of the numbers in its first argument that are divisible by the number at the corresponding position in its second argument. If the lengths of the two lists do not match, the extra elements in the longer list are ignored. Assume that none of the numbers in the second argument are 0. For example:

Use basic functions, and higher order functions, but not recursion. **i.e. do not use recursion.**

```
f [6,9,2,7] [2,3,5,1] = 22
22 is computed as follows: 6+9+7
f [6,9,2] [2,3,5,1] = 15
f [1,2,3,4,5] [5,4,3,2,1] = 12
f [10,20,30,40] [3,4,5,6,7] = 50
```

You can implement any helper non-recursive function that you need. **You can use the built-in function zip which pairs up every two corresponding elements in a list as follows:**

```
>zip [1,2] [3,4]
[(1,3),(2,4)]

>zip [1,2,5] [3,4]
[(1,3),(2,4)]

>zip [1,2] [3,4,5]
[(1,3),(2,4)]
```

Solution:

```
divides (a,b) = mod a b==0
* 1 mark

f x y =  foldr (+) 0 (map (\(x,y)->x) (filter divides (zip x y )))

* 1 mark for correct use of foldr
* 1 mark for (+) 0
* 1 mark for correct use of map
* 1 mark for defining a map function
* 1 mark for getting single "x" value as output of map function
* 1 mark for correct use of filter function
* 1 mark for calling filter using a "divides" function
* 1 mark for correct use of zip function
* 1 mark for passing x and y to zip function

ALTERNATE SOLUTION

foldr (+) 0 (map selectX (zip x y))

selectX (x,y) = if mod x y == 0 then x else 0
```

b) Write a second function `g :: [Int] -> [Int] -> Int` that behaves like `f`, this time using basic functions and recursion but not higher order library functions. **i.e. Do not use any higher order library functions.**

Solution:

```
g :: [Int] -> [Int] -> Int
g [] _ = 0
g _ [] = 0
g (n:ns) (m:ms) |mod n m ==0 = n + g ns ms
| otherwise = g ns ms

⋆ 0.5 marks for base cases
⋆ 0.5 marks for checking modulus = 0
⋆ 0.5 marks for n + g ns ms
⋆ 0.5 marks for otherwise = g ns ms
```

Exercise 6 (4+2+2=8 Marks)

Given the following Haskell function:

```
mystery i p f =
    if p i then i: mystery (f i) p f
           else []
```

a) What is the output of the following call

```
mystery 0 (< 15) (+4)
```

Solution:

```
[0,4,8,12]
```

b) What is the type for the `mystery` function.

Solution:

```
mystery :: a -> (a -> Bool) -> (a -> a) -> [a]
```

c) What does `mystery 0 (<=j) (+1)` does for any `j`?

Solution:

It outputs a list containing all elements starting from 0 till j.

Exercise 7 (3+3+3+3+4=16 Marks)

Here is the definition of a data type for representing a few basic shapes. A figure is a collection of shapes.

```
type Number = Int

type Point = (Number,Number)

type Length = Number

data Shape = Pt Point
           | Circle Point Length
           | Rect Point Length Length
           deriving Show

type Figure = [Shape]

type BBox = (Point,Point)
```

For Example:

- `Circle (10,10) 5` represents a circle centered at (10,10) with radius 5.
- `Pt (8,10)` represents a point with an x-coordinate of `8` and y-coordinate of `10`.
- `Rect (5,5) 10 6` represents a rectangle with a lower left corner at `(5,5)`. The width of the rectangle is `10` and its length is `6`.

The type `BBox` represents bounding boxes of objects by the points of the lower-left and upper-right hand corners of the smallest enclosing rectangle.

a) Define the functions `width`, `bbox`, and `minX` that compute the width, bounding box, and minimum `x` coordinate of a shape, respectively.

1. `width :: Shape -> Length`

   ```
   > width (Pt (4,5))
   0
   > width (Circle (5,5) 3)
   6
   > width (Rect (3,3) 7 2)
   7
   ```

   Solution:

   ```
   width (Pt(x,y)) = 0
   width (Circle (x,y) r) = 2*r
   width (Rect (x,y) w h) = w

   1 mark for each
   - 0.5 for Circle/Rect if written as Pt(x,y)
   -0.5 for each if result is prefixed with "Length"
   ```

2. `bbox :: Shape -> BBox`

   ```
   > bbox (Pt (5,4))
   ((5,4),(5,4))
   > bbox (Circle (5,5) 3)
   ((2,2),(8,8))
   > bbox (Rect (3,3) 7 2)
   ((3,3),(10,5))
   ```

   Solution:

```
bbox (Pt(x,y)) = ((x,y),(x,y))
bbox (Circle (x,y) r) = ((x-r,y-r),(x+r,y+r))
bbox (Rect (x,y) w h) = ((x,y),(x+w,y+h))

1 mark for each
-0.5 mark for each for writing BBox as prefix of body of the functions
-0.5 mark for each if result has prefix "Pt"
```

3. `minX :: Shape -> Number`

```
> minX (Pt (4,4))
4
> minX (Circle (5,5) 3)
2
> minX (Rect (3,3) 7 2)
3
```

Solution:

```
%  minX shape = minx where ((minx,miny),(maxx,maxy))=bbox shape

minx (Pt (x,y)) = x
minX (Circle (x,y) r) = x-r
minx (Rect (x,y) w h) = x

1 mark for each
-0.5 for each if circle/rect have Pt(x,y)
-0.5 for each if output has prefix "Number"
```

b) Define a function `addPt`, which adds two points component wise.

```
addPt :: Point -> Point -> Point
>addPt (1,2) (3,4)
(4,6)
```

Solution:

```
addPt (x,y) (x1,y1) = (x+x1,y+y1)

1 mark for (x,y) (x1, y1)
2 marks for (x+x1, y+y1)

-0.5 for each if "Pt" is used as prefix
```

c) Define a function `move` that moves the position of a shape by a vector given by a point.

```
move :: Shape -> Point -> Shape
>move (Pt (1,2)) (3,4)
Pt (4,6)
>move (Circle (5,5) 3) (1,2)
Circle (6,7) 3
>move (Rect (1,6) 3 7) (10,22)
Rect (11,28) 3 7
```

Solution:

```
move (Pt(x,y)) (px,py)= Pt(x+px,y+py)
move (Circle (x,y) r) (px,py) = (Circle (x+px,y+py) r)
move (Rect (x,y) w h) (px,py)= (Rect (x+px,y+py) w h)

1 mark for heads (Pt/Circle/Rect, no prefix for (px, py))
2 marks for (x+px, y+py)
1 mark for the including the rest of Circle (r) and Rect (w h)
```

Scratch paper

Scratch paper

Scratch paper