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

May 14, 2015

CSEN403: Concepts of Programming Languages
Spring 2015
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 14 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	12	8	15	8	8	9	12	72
Final Marks								

Exercise 1 (12 Marks)

Define a Prolog predicate `memCount(Element, List, Count)` that is true if `Element` occurs `Count` times within `List`.

Note that `List` could consist of other lists and those lists could consist of lists too and so on. Therefore, you should check the existence of `Element` recursively in those lists.

```
?- memCount(a,[b,a],N).
N = 1
?- memCount(a,[b,[a,a,[a],c],a],N).
N = 4
?- memCount([a],[b,[a,a,[a],c],a],N).
N = 1
?- memCount([a],[b,[a,a,[a],c],[a,[[a],b]]],N).
N = 2
```

Solution:

```
memCount(_,[],0).

memCount(X,NL,0):- X\= NL, \+is_list(NL).

memCount(H, [H|T], N):- memCount(H,T,N1),
                        N is N1+1.

memCount(X,[H|T],N):- X\= H,
                      memCount(X,T,N1),
                      memCount(X,H,N2),
                      N is N1+N2.
```

Exercise 2 (8 Marks)

The product of the ages in years of three teenagers is 4590. None of the teens are the same age.

A teenager is a person whose age is between 13 and 20 inclusive.

Write a CLP program that calculates the ages of the three teenagers.

Solution:

```
:-use_module(library(clpfd)).
findAges(L):- L = [A,B,C],
              L ins 13..20,
              A*B*C #= 4590,
              all_different(L),
              labeling([],L).
```

Exercise 3 (2+3+4+6=15 Marks)

Suppose the playing cards in a standard deck are represented by characters in the following way: `'2'` through `'9'` plus `'0'` (zero) stand for the number cards, with `'0'` representing the 10, while the `'A'`, `'K'`, `'Q'` and `'J'` stand for the face cards, i.e. the ace, king, queen and jack, respectively. Let's call these the *card characters*. The other characters, including the lowercase letters `'a'`, `'k'`, `'q'`, and `'j'`, and the digit `'1'`, are not used to represent cards.

a) Write `isFaceCard :: Char -> Bool` to test whether a character stands for a face card.

```
isFaceCard 'A' = True
isFaceCard '5' = False
isFaceCard 'a' = False
```

Solution:

```
isFaceCard :: Char -> Bool
isFaceCard x = x == 'A' || x == 'K' || x == 'Q' || x == 'J'
```

b) Write `isCard :: Char -> Bool` to test whether a character stands for a card. For example,

```
isCard 'A' = True
isCard '5' = True
isCard 'a' = False
isCard '1' = False
```

Hint: You can use the function `isDigit :: Char -> Bool`.

```
isDigit '6' = True
isDigit '1' = True
isDigit 'A' = False
```

Solution:

```
isCard :: Char -> Bool
isCard x = isFaceCard x || (isDigit x && x /= '1')
```

c) Write a function `f :: String -> Bool` to test whether all card characters in a string represent face cards. For example:

```
f "ABCDE" = True
f "none here" = True
f "4 Aces" = False
f "01234" = False
f "" = True
f "1 Ace" = True
```

You could use the two function `isCard` and `isFaceCard`. You are not allowed to use any other library functions.

Solution:

```
f :: String -> Bool
f [] = True
f (c:str) | isCard c  = isFaceCard c && f str
          | otherwise = f str
```

d) Write a function `g :: String -> Bool` that behaves like `f` but using one or more of the following higher-order functions:

```
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
foldr :: (a -> b -> b) -> b -> [a] -> b
```

You are not allowed to use recursion in the implementation

Solution:

```
h :: String -> Bool
h str = foldr (&&) True (map isFaceCard (filter isCard str))
```

Exercise 4 (8 Marks)

Write a function `f :: [Int] -> Bool` that, given a non-empty list of numbers, returns True if each successive number (except the first) is at least twice its predecessor in the list. The function should give an error if applied to the empty list. For example:

```
f [1,2,7,18,47,180] = True
f [17] = True
f [1,3,5,16,42] = False
f [1,2,6,6,13] = False
```

Solution:

```
f :: [Int] -> Bool
f [] = error "Cannot apply function to an empty list"
f [_] = True
f (x:x':xs) = x' >= 2*x && f (x':xs)
```

Exercise 5 (8 Marks)

Write a function `p :: [Int] -> Int` that computes the sum of the cubes of the **positive** numbers in a list. For example:

```
p [-13] = 0
p [] = 0
p [-3,3,1,-3,2,-1] = 36
p [2,6,-3,0,3,-7,2] = 259
p [4,-2,-1,-3] = 64
```

Note: Do not use recursion. Use only the following higher-order library functions:

```
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
foldr :: (a -> b -> b) -> b -> [a] -> b
```

Solution:

```
cube x =  x*x*x
r xs = foldr (+) 0 (map cube (filter (>0) xs))
```

Exercise 6 (9+6(Bonus)=9 Marks)

A boolean expression/proposition is an expression that results in a boolean value; either true or false.

The following datatype represents boolean expressions/propositions.

```
data Proposition = Var String                       -- Atom
                 | F                                -- False
                 | T                                -- True
                 | Not Proposition                  -- Negation
                 | Proposition :|: Proposition      -- Disjunction (OR)
                 | Proposition :&: Proposition      -- Conjuction (AND)
```

Note that the expressions are written in infix notation using the constructors `:|:` and `:&:`. This is similar to the colon constructor `:` in the built-in list datatype.

a) Write a function `isNorm :: Proposition -> Bool` that returns true when a proposition is in negation normal form. A proposition is in negation normal form if the only occurrences of logical negation (Not) are applied to variables. For example,

```
isNorm (Var "p" :&: Not (Var "q")) = True
isNorm (Not (Var "p" :|: Var "q")) = False
isNorm (Not (Not (Var "p")) :|: Not T) = False
isNorm (Not (Var "p" :&: Not (Var "q"))) = False
isNorm T = True
```

Solution:

```
isNorm :: Proposition -> Bool
isNorm (Var x)           =  True
isNorm T                 =  True
isNorm F                 =  True
isNorm (Not (Var x))     =  True
isNorm (Not p)           =  False
isNorm (p :|: q)         =  isNorm p && isNorm q
isNorm (p :&: q)         =  isNorm p && isNorm q
```

b) Write a function `norm :: Proposition -> Proposition` that converts a proposition to an equivalent proposition in negation normal form. A propositon may be converted to normal form by repeated application of the following equivalences:

```
Not F <-> T
Not T <-> F
Not (Not p) <-> p
Not (p :|: q) <-> Not p :&: Not q
Not (p :&: q) <-> Not p :|: Not q
```

For example,

```
norm (Var "p" :&: Not (Var "q")) = (Var "p" :&: Not (Var "q"))
norm (Not (Var "p" :|: Var "q")) = Not (Var "p") :&: Not (Var "q")
norm (Not (Not (Var "p")) :|: Not T) = (Var "p" :|: F)
norm (Not (Var "p" :&: Not (Var "q"))) = Not (Var "p") :|: Var "q"
norm (Var "p") = Var "p"
norm T = T
```

Solution:

```
norm :: Proposition -> Proposition
norm (Var x)          =  Var x
norm T                =  T
norm F                =  F
norm (Not (Var x))    =  Not (Var x)
norm (Not T)          =  F
norm (Not F)          =  T
norm (Not (Not p))    =  norm p
norm (Not (p :|: q))  =  norm (Not p) :&: norm (Not q)
norm (Not (p :&: q))  =  norm (Not p) :|: norm (Not q)
norm (p :|: q)        =  norm p :|: norm q
norm (p :&: q)        =  norm p :&: norm q
```

Exercise 7 (12 Marks)

a) Consider the following code:

```
int a;
int *y;
int *x = &a;

*x = 15;
*x += 2;
y = x;
*y *= 2;
```

What are the values of `*x` and `*y` after executing this code?

Solution:

*x = 34 and *y = 34

b) Consider the following code

```
int name[] = {5,23,119};
int *p, *q;

p = name;
q = name + 1;

printf("%i %i %i \n", *name, *p, *q);

*(p++);
(*q)++;

printf("%i %i\n", *p, q[0]);
```

1. What does the code print?

   Solution:

   ```
   5 5 23
   24 24
   ```

2. What is the content of array `name` once the fragment is executed?

   Solution:

   ```
   {5, 24, 119}
   ```

c) What does the program print?

```
int main()
{
int a = 5;
int b = 23;

printf("%i %i\n", a, b);

scramble(b, &a);
printf("%i %i\n", a, b);

scramble(a, &b);
printf("%i %i\n", a, b);

return 0;



}

void scramble(int x, int *y)
{
int temp;

printf("%i %i\n", x, *y);

temp = x + 1;
x = *y;
*y = temp;

printf("%i %i\n", x, *y);
}
```

Solution:

```
5 23
23 5
5 24
24 23
24 23
23 25
24 25
```

Scratch paper

Scratch paper

Scratch paper