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

May 24, 2014

CSEN403: Concepts of Programming Languages Spring 2014

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 12 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	$\sum$
Possible Marks	10	5	10	10	18	7	60
Final Marks							

Exercise 1 (10 Marks)

Define a Prolog predicate `addUpList( L, K)` that holds if `L` and `K` are lists of integers where every element in `K` is the sum of all the elements in `L` up to the same position. Assume that `L` is a bound variable

```
?- addUpList( [1], K).
K = [1];
no
?- addUpList( [1, 2], K).
K=[1,3] ;
no
?- addUpList([ 1, 3, 5, 7], K).
K=[1, 4, 9, 16];
no
```

Solution:

```
addUpList(L,K):- h(L,K,0).

h([],[],_).
h([H|T], [S|ST], A):- S is H+A,
                      h(T,ST,S).
```

Exercise 2 (5 Marks)

What is the type of the following values? If the expression is invalid, briefly state why. Assume numbers have the type `Int`. Remember that the type of `True` is `Bool`.

a) `"x"`

   Solution:

   `[Char]` or `String`

b) `(1,'x',[True])`

   Solution:

   `(Int, Char, [Bool])`

c) `["x":[]]`

   Solution:

   `[[[Char]]]` or `[[String]]`

d) `map not` where the type of `not` is `Bool -> Bool`.

   Solution:

   `[Bool] -> [Bool]`

e) `[1,'2',"3"]`

   Solution:

   Invalid, due to multiple types. Lists in Haskell are homogeneous.

Exercise 3 (10 Marks)

Write a function `rotabc` that changes `a`'s to `b`'s, `b`'s to `c`'s and `c`'s to `a`'s in a string. Only lowercase letters are affected.

```
> rotabc "abc"
"bca"
> rotabc "test"
"test"
> rotabc "aaaaa"
"bbbbb"
> rotabc "ababab"
"bcbcbc"
> rotabc "cabinet"
"abcinet"
> :t rotabc
rotabc :: [Char] -> [Char]
```

You are not allowed to implement an explicit recursive function `rotabc`. However, you may use any predefined (higher-order) functions or define any helper functions that are not recursive.

Solution:

```
help c | c == 'a' = 'b'
       | c == 'b' = 'c'
       | c == 'c' = 'a'
       | otherwise = c

rotabc = map help
```

Exercise 4 (10 Marks)

Consider a function separate s that returns a pair with the digits and non-digits in the string s separated, with the initial order maintained.

```
> separate "July 4, 1776"
("41776","July , ")
> separate "Problem 7: (10 points)"
("710","Problem : ( points)")
```

Here is a partial implementation of separate, using `foldr`:

```
separate s = foldr f ([],[]) s
```

Your task on this problem is to write the folding function `f`. Use the following `isDigit` function to test for a digit.

```
isDigit c = c >= '0' && c <= '9'
```

Solution:

```
f x (digs, non) = if isDigit x then ((x:digs),non)
                  else (digs, (x:non))
```

Exercise 5 (6+12=18 Marks)

Consider the following data types:

```
data Genre = Nonfiction | Novel | Biography deriving (Eq, Show)
type Name = (String, String)
type Date = (Int, Int, Int) -- day, month, year
data Book = ABook Genre
                  Name -- name of the author
                  String -- title of the book
                  Date -- date of publication
                  Int -- number of pages
            deriving Show
```

Note that `--` stands for a comment in Haskell

a) Write a function `genre` that given a book returns its genre.

```
genre :: Book -> Genre
```

Solution:

```
genre (ABook g _ _ _ _) = g
```

b) Write a function `title` that given a book returns its title.

```
title :: Book -> String
```

Solution:

```
title (ABook _ _ t _ _) = t
```

c) Write a function `date` that given a book returns its date.

```
date :: Book -> Date
```

Solution:

```
date (ABook _ _ _ d _) = d
```

d) Write a function `pages` that given a book returns its number of pages.

```
pages :: Book -> Int
```

Solution:

```
pages (ABook _ _ _ _ p) = p
```

e) Write a function `year` that given a book returns its year of publication.

```
year :: Book -> Int
```

Solution:

```
year b = y where (_, _, y) = date b
```

Or

```
year (ABook _ _ _ (_,_,y) _) = y
```

For the rest of the exercise and as an example, consider the following books where `breakingNews`, `snowden`, and `futureShock` are contants:

```
breakingNews = ABook Novel ("Schaetzing", "Frank") "Breaking News"
                     (06, 03, 2014) 976
snowden = ABook Nonfiction ("Harding", "Luke") "The Snowden Files"
               (06, 02, 2014) 346
futureShock = ABook Nonfiction ("Toffler", "Alvin") "Future Shock"
                    (01, 06, 1984) 576
```

Implement the following functions. Your implementations will be short if you use higher-order functions `map`, `filter`, and `foldr`. You are allowed to define any helper functions. You are also allowed to use any of the functions you defined in the previous page.

a) `publishedIn :: Int -> [Book] -> [Book]` evaluates to a list that contains only those books of the input list that have been published in the year specied by the Int-argument. For example,

```
publishedIn 2014 [breakingNews, snowden, futureShock]
```

evaluates to

```
[ABook Novel ("Schaetzing", "Frank") "Breaking News" (06, 03, 2014) 976,
 ABook Nonfiction ("Harding", "Luke") "The Snowden Files" (06, 02, 2014) 346]
```

Solution:

```
helpIn y b = y == year b
publishedIn y l = filter (helpIn y) l
```

b) `totalPages :: [Book] -> Int` counts the number of pages of all books of the input list. For example,

```
totalPages [breakingNews, snowden, futureShock]
```

evaluates to `1898`.

Solution:

```
totalPages l = foldr (+) 0 (map pages l)
```

c) `titlesOf :: Genre -> [Book] -> [String]` computes a list of all titles of the given genre. For example,

```
titlesOf Nonfiction [breakingNews, snowden, futureShock]
```

evaluates to `["The Snowden Files","Future Shock"]`.

Solution:

```
eqG g b = g == genre b
titlesOf g l = map title (filter (eqG g) l)
```

Exercise 6 (3+4=7 Marks)

In the following, if an address will be printed then you should just write `Address`

a) What will be output of following program?

```
#include<stdio.h>
int main(){
   int i = 3;
   int *j;
   int **k;
   j=&i;
   k=&j;
   printf("%u %u %d ",k,*k,**k);
   return 0;
}
```

Solution:

Address Address 3

b) What will be output of following program?
What is the functionality of `mystery` in general?

```
#include<stdio.h>

int main()
{
    char* s = "nonon";
    int m = mystery(s);
    printf("m = %i\n", m);
    getchar();
    return 0;
}

int mystery(char* s){
    char* o = s;
    while(*s != '\0'){
             s++;
    }
    s--;
    while(o<s){
               if(*o != *s) return 0;
               o++;
               s--;
    }
    return 1;
}
```

Solution:

m = 1

In general, mystery checks whether a string (array of characters terminated by a null character) is a palindrome.

Scratch paper

Scratch paper

Scratch paper