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

June 13, 2011

Concepts of Programming Languages
Spring term 2011

Final Exam

Bar Code

Instructions: Read carefully before proceeding.

1) No books or other aids are permitted for this test.
2) This exam booklet contains 14 pages (10 exercises), 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**.
3) 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 three 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**.
4) **Duration of the exam:** 3 hours
5) 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	8	9	10	$\sum$
Possible Marks	8	12	12	8	10	6	8	10	18	8	100
Final Marks											

Exercise 1 (8 Marks)

Write a Prolog predicate `reverse_deep(L1,L2)` that succeeds if and only if `L2` corresponds to list `L1` in reverse order and the elements are also deeply reversed. Assume that you have the `reverse` predicate from the lectures.

```
?- reverse_deep ([1, [[2, 3], 4], [[2], [1, 3]]], X).
X = [[[2],[3, 1]], [4, [3, 2]], 1]
```

Solution:

```
reverse_deep([],[]).
reverse_deep([H|T],R):- \+is_list(H),
                         reverse_deep(T,TR),
                         append(TR,[H],R).

reverse_deep([H|T],R):- is_list(H),
                        reverse_deep(H,HR),
                        reverse_deep(T,TR),
                        append(TR,[HR],R).
```

Exercise 2 (6+6=12 Marks)

Provide a function to split a list of integers into two lists, such that the first list contains all non-negative integers and the second list contains all negative integers. For instance, the list `[1,-2,0,3,-4]` should be split into lists `[1,0,3]` and `[-2,-4]`.

a) Write a Haskell function to support this functionality. The input to the function is the list of integers to be split and the output is a pair of lists, the first containing non-negative numbers and the second containing the negative ones.

Solution:

```
split l = (filter (>=0) l,filter (<0) l)
```

Alternative Solution:

Solution:

```
split [] = ([],[])
split (x:xs) = if x>=0 then (x:l1,l2)
                       else (l1,x:l2)
               where (l1,l2) = split xs
```

b) Write a Prolog predicate `split` that takes three arguments to support this functionality.

Solution:

```
split([],[],[]).
split([X|Y], B, [X|C]) :- X>=0, split(Y,B,C).
split([X|Y], [X|B],C) :- X < 0, split(Y,B,C).
```

Exercise 3 (4+8=12 Marks)

a) Define a function, any function, that has the type shown below.

```
Main> :type foo
foo :: (a,b) -> (a,b) -> Bool -> (a,b)
```

Solution:

```
foo (n,m) (_,_) True  = (n,m)
foo (_,_) (n,m) False = (n,m)
```

b) The questions below apply to the functions defined below. When reading the examples below, do not forget that strings are shorthand notation for lists of characters. The built-in reverse function reverses lists.

```
mystery words = foldr (help) "" words

p x = x == reverse x

help w1 w2
  | p w1 && p w2 = if (length w1 > length w2) then w1 else w2
  | p w1 = w1
  | p w2 = w2
  | otherwise = ""
```

1. What would `help` return when applied to `"foo"` and `"bar"`?

   Solution:

   `""`

2. What would `help` return when applied to `"Haskell"` and `"abba"`?

   Solution:

   `"abba"`

3. What does `mystery` return if passed `["foo", "abba", "aardvark", "tenet"]`?

   Solution:

   `"tenet"`

4. Describe, in English, what the `mystery` function does in general.

   Solution:
   The function mystery returns the longest palindrome in a list of strings.

Exercise 4 (4+4=8 Marks)

a) Define a haskell function `max` which takes two non-negative positive integers and determines the maximum of the two. For example, `max 2 3 = 3` and `max 5 0 = 5.` Here, you may not use any pre-defined functions except for `(+)` and `(-)`. In particular, you may not use `(>)` or any other comparison operator.

Solution:

```
max 0 y = y
max x 0 = x
max x y = 1 + max (x-1) (y-1)
```

b) Use the function `max` from Part a to define a haskell function `maxList` which takes two lists of non-negative integers of same length and builds a list, which at each position contains the maximum of the elements of the two argument lists. For example, `maxList [1,6,3] [2,4,5] = [2,6,5]`.

Solution:

```
maxList = zipWith max
```

Alternative Solution:

```
maxList [] [] = []
maxList (x:xs) (y:ys) = max x y : maxList xs ys
```

Exercise 5 (5+5=10 Marks)

a) Implement a function that represents a given string by a sequence of numbers from $\{-1, 0, 1\}$, where -1 denotes a closing bracket, 1 an opening bracket and 0 is used for all other characters in the string. For instance the string `"a((bc)d)"` is represented by the list `[0,1,1,0,0,-1,0,-1]`.

Solution:

```
represent :: [Char] -> [Int]
represent [] = []
represent (x:xs) =
    if (x == '(') then 1:(represent xs)
    else if (x == ')') then (-1) : (represent xs)
    else 0 : (represent xs)
```

b) Implement a function to check the correct nesting of a given finite string. For instance, the string `"a((bc)d)"` is correctly nested and the string `"a)bc(d"` is not correctly nested.

Solution:

```
checkBracketing l = check [] l

check [] [] = True
check _ [] = False
check l (x:xs) = if x == '(' then check ('(':l) xs
                             else if x == ')'
                             then if l == [] then False
                                  else check (tail l) xs
                             else check l xs
```

Exercise 6 (4+2=6 Marks)

Consider the following function:

```
mystery f = foldr (help f) []

help f x xs = f x : xs
```

where `foldr` is implemented as presented in the lectures as follows:

```
foldr g h [] = h
foldr g h (x:xs) = g x (foldr g h xs)
```

a) What is the output of the following expression? Trace your execution.

```
mystery (+3) [1,2,4,7]
```

Solution:

```
> mystery (+3) [1,2,4,7]
[4,5,7,10]
```

b) What is the functionality of `mystery f ys` for all functions `f` of type `(a->b)` and `ys` of type `a`. Justify your answer.

Solution:

```
mystery f ys = map f ys
```

Exercise 7 (8 Marks)

Write a higher-order function called `duplicateSome` that takes a predicate and a list of items, and adds duplicates to the output list of those items for which the predicate returns True. For full credit, your solution should be explicitly recursive.

```
Main> duplicateSome even [1..10]
[1,2,2,3,4,4,5,6,6,7,8,8,9,10,10]
Main> duplicateSome (<5) [1..10]
[1,1,2,2,3,3,4,4,5,6,7,8,9,10]
Main> duplicateSome even [1,3,5]
[1,3,5]
Main> duplicateSome even []
[]
```

Solution:

```
duplicateSome _ [] = []
duplicateSome p (x:xs) = if p x then x:x:duplicateSome p xs
                                else x:duplicateSome p xs
```

Exercise 8 (6+4=10 Marks)

Consider the following definition of a datatype of bits:

```
data Bit = O | I    deriving Show
```

This datatype has two different values, written `O` and `I`, which we will use to represent the bits 0 and 1. Now we can define a type of binary numbers:

```
type BinNum = [Bit]
```

For convenience, we will assume that the least significant bit is stored at the head of the list so that, for example, `[O, O, I]` represents the number 4 and `[O, I, I, O, I, O]` represents 22.

a) Define the functions:

```
toBinNum   :: Integer -> BinNum
fromBinNum :: BinNum -> Integer
```

that convert backwards and forwards between Integers and their corresponding `BitNum` representations.

Solution:

```
fromBinNum l = fromBinNumH l 0

fromBinNumH [] _ = 0
fromBinNumH (O:xs) index = fromBinNumH xs (index+1)
fromBinNumH (I:xs) index = (2^index) + fromBinNumH xs (index+1)

toBinNum 0 = []
toBinNum n = if (mod n 2)==0 then O : toBinNum (div n 2)
                             else I : toBinNum (div n 2)
```

b) Define a `BinNum` increment function

```
inc :: BinNum -> BinNum
```

without using either `toBinNum` or `fromBinNum`.

For example, `inc [I,I,O,I,O,I]` should yield `[O,O,I,I,O,I]`.

Solution:

```
inc [I]=[O,I]
inc (O:xs) = I:xs
inc (I:xs) = O:inc xs
```

Exercise 9 (6+6+6=18 Marks)

- Define a datatype in C for a `Person`. A `Person` is defined by his name, age and an array of `Person` objects. These objects correspond to his family members arranged as father then mother then siblings. The siblings are sorted in descending order according to their age.

  Solution:

```
typrdef struct{
    char* name;
    int age;
    Person* family;
    int familyCount;
}Person;
```

- Define the function `Person* createPerson(char* n,int a,int r)` which returns a pointer to a new `Person` object, where `n` is the name of the person, `a` is his/her age and `r` is the number of family members.

  Solution:

```
Person* createPerson(int n,int a,int r){
    Person* p = (Person*) malloc(sizeof(Person));
    (*p).name = n;
    (*p).age = a;
    (*p).familyCount = r;
    (*p).family = (Person*) malloc(sizeof(Person)*r);
    return p;
}
```

- Define the function `char* elderBrotherName(Person p)` that returns the name of the elder brother of the `p`.

  Solution:

```
char* elderBrotherName(Person p){
     return (p.family[2]).name;
}
```

Exercise 10 (8 Marks)

Implement a **non-iterative** C function `void rightshift(int* arr,int itemCount)` which performs right shift of the elements of the array `arr`. The variable `itemCount` represents the number of items in the array `arr`. A right shift of the items of the array

```
{2,3,6,8,11,23,56,89}
```

changes the array to:

```
{3,6,8,11,23,56,89,2}
```

Another right shift of the produced array:

```
{6,8,11,23,56,89,2,3}
```

Solution:

```
void rightshift(int* arr,int itemCount){
     int* temp = arr;
     arr++;
     *(arr+itemCount) = *temp;
}
```

Extra Sheet

Extra Sheet

Extra Sheet