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

May 26, 2010

Concepts of Programming Languages
Spring 2010
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 13 pages (8 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	$\sum$
Possible Marks	6	12	10	14	9	12	7	15	85
Final Marks									

Exercise 1 (6 Marks)

Write a Prolog predicate `translate(L1,L2)` that succeeds if `L1` is a list of digits (numbers from 0 to 9) and `L2` is a list of the corresponding words.

For example

```
?- translate([1,3,5,1],L2).
L2 = [one, three, five,one]
```

Solution:

```
translate([],[]).
translate([X|T],[Y|T1]): -
    means(X,Y),
    translate(T,T1).

means(1,one).
means(2,two).
means(3,three).
means(4,four).
means(5,five).
means(6,six).
means(7,seven).
means(8,eight).
means(9,nine).
```

Exercise 2 (8 + 4 = 12 Marks)

a) Implement a Prolog predicate `take/3` such that `take(N,L,M)` is true if, and only if, `M` is the longest prefix of length at most `N` of the list `L`.

For example, the query

```
?- take(2, [6,3,4], M).
M = [6,3]
?- take(4, [6,3,4], M).
M = [6,3,4]
```

Solution:

- Possible solution:

```
take(N,L,M) :- length(L,O), P is min(O,N), append(M,Q,L), length(M,P).
```

- Alternative Solution:

```
take(0,_,[]).
take(_,[],[]).
take(N,[X|Xs],[X|Ys]) :- N > 0, O is N-1, take(O,Xs,Ys).
```

b) Implement a Prolog predicate `firstHalf/2` such that `firstHalf(L, M)` is true if, and only if, `M` is the list that contains exactly the first half of the elements of the list `L`. For example,

```
?- firstHalf([6,3,4,5], M).
M = [6,3]
firstHalf([6,3,4], M).
M = [6]
```

Note: You have to use the predicate `take` from part a).

Solution:

```
firstHalf(L,M) :-
       length(L,N),
       O is N//2,
       take(O,L,M).
```

Exercise 3 (4+4+2=10 Marks)

a) Implement a Haskell function `snoc` that adds an integer to the end of a list.

Solution:

```
snoc :: a -> [a] -> [a]
snoc i []     = [i]
snoc i (x:xs) = x:snoc i xs
```

b) Implement a function `rev` that reverses the elements in a list. You should use the function `snoc`

Solution:

```
rev :: [a] -> [a]
rev []     = []
rev (x:xs) = snoc x (rev xs)
```

c) What is the type of the function

```
f = map rev
```

Solution:

```
f :: [[a]] -> [[a]]
```

Exercise 4 (8+2+4=14 Marks)

a) Implement a higher-order function `span` that takes a predicate `p` and a list `l` as input and returns a pair of lists where the first list consists of the first elements of `l` that satisfy `p` and in the second list all remaining elements of `l`. The list will be split into two parts, where the second part starts with the first item that does not satisfy `p` and consists of all elements of `l` that come after this item (including the item itself).

CHECK THE EXAMPLES FOR A BETTER UNDERSTANDING OF THE FUNCTION!

For example:

```
> span even [2,4,6,7,8,10,11,12]
([2,4,6],[7,8,10,11,12])
> span (>0) [-1,2,3,4,5]
([],[-1,2,3,4,5])
> span (>0) [6,10,-1,2,3,4,5]
([6,10],[-1,2,3,4,5])
```

Solution:

```
span p [] = ([],[])
span p (x:xs)
     |p x = (x:ys,zs)
     |otherwise = ([],(x:xs))
     where (ys,zs) = span p xs
```

b) Give the datatype of `span`.

Solution:

```
span :: (a -> Bool) -> [a] -> ([a],[a])
```

c) Implement a a higher-order function `break` that takes a predicate `p` and a list `l` as input and returns a pair of lists where the first list consists of the first elements of `l` that satisfy the negation of `p` and in the second list all remaining elements of `l`.

```
> break even [1,3,5,6,8,9,10]
([1,3,5],[6,8,9,10])
```

Note: You have to use `span`.

```
break p =
```

Solution:

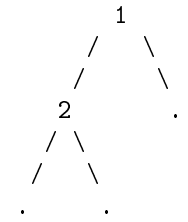
```
break p = span (not . p)
```

Exercise 5 (3+6+6(Bonus)=9 Marks)

Consider the following polymorphic datatype, representing binary trees where every node (including leaf nodes) are labeled:

```
data Tree a = Leaf  | Node (Tree a) a (Tree a)
```

For example given the following tree `t`



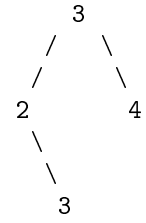
The representation of `t` as an object of type `Tree Int` in Haskell would be:

```
Node (Node Leaf 2 Leaf) 1 Leaf
```

a) Draw the tree for the following expression:

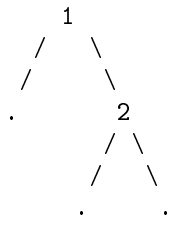
```
ex = Node (Node Leaf 2 (Node Leaf 3 Leaf)) 3 (Node Leaf 4 Leaf)
```

Solution:



b) Implement the function `swapTree` which returns the tree where all left children have been swapped with the corresponding right children.

When computing `swapTree t` one would obtain the following tree:



Apart from the function declaration, also give the most general type declaration for `swapTree`. Do not use any higher-order functions.

Solution:

```
swapTree :: Tree a -> Tree a
swapTree Leaf = Leaf
swapTree (Node l x r) =
             Node (swapTree r) x (swapTree l)
```

c) This part is a bonus question.

Define a haskell function `paths` which takes a `Tree` a and produces the list of paths from the root to a Node that has two Leaf children.

For example, `paths ex` should return `[[3,2,3], [3,4]]`.

Solution:

Possible Solution:

```
paths = p [] where
p path Leaf = []
p path (Node Leaf x Leaf) = [path++[x]]
p path (Node left x right) = p (path++[x]) left ++ p (path++[x]) right
```

Alternative Solution:

```
paths t = map reverse (p [] t) where
p path Leaf = []
p path (Node Leaf x Leaf) = [x:path]
p path (Node left x right) = p (x:path) left ++ p (x:path) right
```

Exercise 6 (6+6=12 Marks)

Polynomials over a carrier set a are defined inductively as follows:

- Any constant (`Const`) from a is a polynomial.
- Any variable (`Var`) is a polynomial. (One can represent such variables in Haskell using the data type `String`).
- If p and q are polynomials, then also the `Sum` of p and q and the `Product` of p and q are polynomials.

a) Define a polymorphic Haskell data structure `Polynomial a` for polynomials over an (arbitrary) type `a`. Its data constructors should be `Const`, `Var`, `Sum`, and `Product`.

Solution:

```
data Polynomial a = Const a
                  | Var String
                  | Sum (Polynomial a) (Polynomial a)
                  | Product (Polynomial a) (Polynomial a)
```

b) Implement a Haskell function that multiplies a polynomial by a factor.

For example if we multiply $2 \times x + 3 \times y$ by z, the result will be $2 \times x \times z + 3 \times y \times z$.

Solution:

```
mult x (Sum y z) =
      Sum (mult x y) (mult x z)
mult x y = (Product x y)
```

Exercise 7 (2+2+3=7 Marks)

a) Which operator is used to get value at address stored in a pointer variable?

Solution:

```
*
```

b) Combine the following two statements into one?

```
char *p;
p = (char*) malloc(100);
```

Solution:

```
char *p = (char*)malloc(100);
```

c) What will be displayed? Justify your answer!

```
#include<stdio.h>

int main()
{
    int a = 10, b = 10;
    int c,d;
    int *ptra = &a;
    int *ptrb = &b;

    ++*ptra;
    (*ptrb)++;

    printf("\n A = %d , B = %d", a , b);

    c = ++*ptra;
    d = (*ptrb)++;

    printf("\n A = %d , B = %d", a , b);
    printf("\n C = %d , D = %d", c , d);

    return 0;
}
```

Solution:

```
Output:
 A = 11 , B = 11
 A = 12 , B = 12
 C = 12 , D = 11
```

Exercise 8 (3+4+4+4=15 Marks)

a) Implement a data structure in `C` for representing circles. A circle is defined by the `X` and `Y` coordinates of its center and its radius.

Solution:

```
typedef struct{
    int x;
    int y;
    int r;
}Circle;
```

b) Implement the function `Circle* createCircle(int x,int y,int r)` that creates a circle with center at the point `(x,y)` with radius `r`.

Solution:

```
Circle* createCircle(int cx,int cy,int cr){
        Circle* newCircle = (Circle*) malloc(sizeof(Circle));
        (*newCircle).x = cx;
        (*newCircle).y = cy;
        (*newCircle).r = cr;
        return newCircle;
}
```

c) Implement the function `int hasConcentric(Circle x[],int len)` which takes an array of circles (of length `len`) and check whether any two circles in the array are concentric. Concentric circles have the same center point.

Solution:

```
int hasConcentric(Circle x[],int len){
    int i;
    for(i=0;i<len;i++){
         int j;
         for(j=i;j<len;j++){
                 if(x[i].x==x[j].x && x[i].y==x[j].y){
                          return 1;
                 }
         }
    }
    return 0;
}
```

d) Implement the function `Circle*removeFirst(Circle x[])` that removes the first item in the array of circles `x`, and returns a pointer to the next one.

Solution:

```
Circle* removeFirst(Circle x[]){
     Circle* nxt = x+1;
     free(x);
     return nxt;
 }
```

Extra Sheet

Extra Sheet

Extra Sheet