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

June 16, 2008

# Concepts of Programming Languages
# Spring term 2008
## 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, 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 | ∑ |
|---|---|---|---|---|---|---|---|---|---|---|
| Possible Marks | 6 | 12 | 10 | 8 | 8 | 4 | 6 | 6 | 10 | 70 |
| Final Marks | | | | | | | | | | |

**Exercise 1** (6 Marks)

a) What does it mean that a grammar is ambiguous?

**Solution:**

A grammar is ambiguous if it is possible to construct a sentence that is valid according to the grammar and has more than one derivation tree.

b) Explain what it means for a program to be declarative.

**Solution:**

A program is declarative if it specifies what should be done rather than how to do it.

c) Explain what the benefits and drawbacks of using a declarative model of computation are.

**Solution:**

Easier reasoning about programs and robustness are among the major features of a declarative model of computation. Unnatural code and difficulties in simulation of real-life stateful objects are among the major drawbacks of declarativeness.

**Exercise 2** (12 Marks)

a) Write a Prolog predicate `subset(R,L)` that succeeds if and only if `R` is a subset of `L`. Do not use any predefined predicates.

```
?- subset(R, [1,2]).
R = [] ;
R = [2];
R = [1];
R = [1,2]
```

**Solution:**

```
subset([], []).
subset(S, [_|T]) :- subset(S,T).
subset([H|T1],[H|T2]) :- subset(T1,T2).
```

b) Write a Prolog predicate `findAllSubsets` that generates all possible subsets of a list.

```
?- findAllsubsets([1,2],L)
L = [[], [2], [1], [1,2]]
```

**Solution:**

```
findAllsubsets(R,L) :-
    setof(L1, subset(L1,R),L).
```

**Exercise 3** (10 Marks)

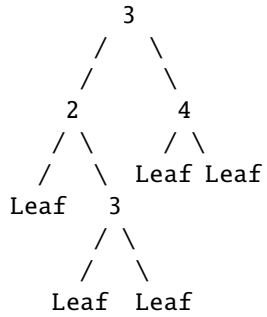
Given the following datatype

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

The tree `t1` defined by the following expression

```
t1 = Node 3 (Node 2 Leaf (Node 3 Leaf Leaf)) (Node 4 Leaf Leaf)
```

produces the following tree structure.




We call elements like `3` or `4` that are stored in the tree nodes of the tree

a) The *fringe* of a tree is defined as all the nodes that have two empty subtrees, that is, whose both subtrees are Leaf. Define a Haskell function `fringe` that computes the fringe of a tree as a list of nodes in left-to-right order.

   For example `fringe t1 = [3,4]`.

   Give the type of the function.

   **Solution:**

```
fringe :: Tree a -> [a]
fringe Leaf = []
fringe (Node x Leaf Leaf) = [x]
fringe (Node x l r) = fringe l ++ fringe r
```

b) Define a Haskell function `level` that yields the list of nodes at a particular level in a tree. For example, `level 0 t` the function should always yield a list containing just the root of `t`, and `level 1 t` yields all children of the root of `t`.

   For example, `level 2 t1 = [3]`.

   Give the type of the function.

   **Solution:**

```
level :: Int -> Tree a -> [a]
level 0 (Node x _ _) = [x]
level n (Node x l r) = level (n-1) l ++ level (n-1) r
level _ _ = []
```

**Exercise 4** (8 Marks)

A formula of the form 3 + (x * y) can be represented as follows:

```
Sum (Constant 3) (Product (Variable "x") (Variable "y"))
```

a) Define in Haskell a datatype that corresponds to the above representation of a formula.

b) Define a Haskell function `sumAtoms` that calculates the total number of constants and variables in a formula. For example.

```
sumAtoms (Sum (Constant 3) (Product (Variable "x") (Variable "y"))) = 3
sumAtoms (Sum (Variable x) (Product (Variable "x") (Variable "x"))) = 3
sumAtoms (Variable "x") = 1
```

Determine the type of the function `sumAtoms`.

**Solution:**

```
data Formula =
Constant Int |
Variable String |
  Sum Formula Formula |
Product Formula Formula

sumAtoms :: Formula -> Int

sumAtoms (Constant a) = 1
sumAtoms (Variable a) = 1
sumAtoms (Sum a a1) = sumAtoms a + sumAtoms a1
sumAtoms (Product a a1) = sumAtoms a + sumAtoms a1
```

**Exercise 5** (8 Marks)

Write a function riffle that takes a list as a single argument. It is assumed that the list contains an equal number of even and odd integers. The function produces a rearrangement of the elements in the list so that odd and even number alternate. If the list contains a different number of even and odd integers, the list returned will contain twice as many integers as there are odd numbers, if there are less odd numbers or twice as many integers as there are even numbers, if there are less even numbers. For example,

```
riffle [2,4,5,3,1,6] [2,5,4,3,6,1]
riffle [2,4,5,3,1,6,7] [2,5,4,3,6,1]
riffle [2,4,5,3,1,6,8] [2,5,4,3,6,1]
```

**Hint:** Use (a similar function to) the predefined function `zip` and the higher order function `filter`.

**Solution:**

```
riffle :: [Int] -> [Int]
riffle as = myzip evens odds
    where
    evens = filter even as
    odds  = filter odd as

myzip :: [a] -> [a] -> [a]
myzip as [] = []
myzip [] bs = []
myzip (a:as) (b:bs) = a:b:(myzip as bs)
```

**Exercise 6** (4 Marks)

Given the following Haskell function:

```
interleave x []     = [[x]]
interleave x (y:ys) = [x:y:ys] ++ map (y:) (interleave x ys)
```

Give the type of the function and the result of the reduction of

```
interleave 3 [1,2,3]
```

Justify your answer by tracing your program

**Solution:**

```
interleave :: a -> [a] -> [[a]]
interleave 3 [1,2,3]  =>  [[3,1,2,3],[1,3,2,3],[1,2,3,3],[1,2,3,3]]
```

**Exercise 7** (6 Marks)

Given the following C program:

```
#include <stdio.h>
int x;
int main(void)

{
            int y = 100;
            x = 200;
            y = shuffle(&x, &y);
            printf("x = %d, y = %d\n", x, y);

            x = 300;
            y = 200;
            shuffle(&y, &x);
            printf("x = %d, y = %d\n", x, y);

            x = 500;
            printf("Function returns %d.\n", shuffle(&x, &x));
            return 0;
}

int shuffle(int *p1, int *p2)
{
            int y = 300;
            *p2 = 400;
            x = x + 300;
            return (*p1 + 100);

}
```

What would you expect to see on standard output when this program is executed? Justify your answer.

**Solution:**

```
x = 500, y = 600
x = 700, y = 200
Function returns 800.
```

**Exercise 8** (6 Marks)

a) Consider a structure defined as follows:

```
struct item
{
     char name[50];
     int price;
};
```

Let's say an array of 100 items (numbered 0 through 99) is declared as follows:

```
struct item items[100];
```

1. Write a statement that will assign the integer 25 to the price of the first item in the array.

   **Solution:**

   ```
   items[0].price = 25;
   ```

2. Write a statement that will assign the string `"jukebox"` to the name of the last item in the array.

   **Solution:**

   ```
   strcpy(items[99].name, "jukebox");
   ```

   Note that you cannot use

   ```
   items[99].name = "jukebox";
   ```

b) Your program uses the variable `pf` which has been declared as a pointer to float as follows:

```
float *pf;
```

The user has entered a value into an integer variable `x`. Your code has already checked to ensure that `x` is positive.

You want to allocate memory to hold `x` floats and to have the variable `pf` point to the allocated memory. Write a statement of code to do this.

**Solution:**

```
pf = (float *) malloc(sizeof(float) * x);
```

**Exercise 9** (10 Marks)

Write a function called `check_strings` that accepts two strings as parameters. The function should return 1 if the two strings are the exact reverse of each other, and it should return 0 otherwise. For example, if the two strings are `"Hello World!"` and `"!dlroW olleH"`, the function should return 1, but if the first of these strings is the same and the second is anything else, it should return 0. You can assume that the appropriate header files have been included at the top of the file.

Write two versions of the function one that uses arrays of characters and one that uses pointers.

**Solution:**

```
int check_strings1(char str1[], char str2[])
{
  int length, i;
  /* If strings have different lengths,
     they can't be exact reverses of each other. */
  if (strlen(str1) != strlen(str2))
    return 0;
  /* Compare all characters that should be equal. */
  length = strlen(str1);
  for (i = 0; i < length; i++)
    if (str1[i] != str2[length-1-i])
      return 0;
  return 1;

}


int check_strings2(char *str1, char *str2)
{
  char *p1 = str1;
  char *p2 = str2 + strlen(str2) - 1;
  /* If strings have different lengths,
     they can't be exact reverses of each other. */

  if (strlen(str1) != strlen(str2))
    return 0;
  /* Walk along both strings in opposite directions. */

  for (p1 = str1, p2 = str2 + strlen(str2) - 1;
       p2 > str2;
       p1++, p2--)
    {
      if (*p1 != *p2)
        return 0;
    }
  return 1;

}

/* make a reverse copy of the first string and compare
   this created string with the second. you must use "malloc"
   to allocate memory for the created string */

int check_strings3(char *str1, char *str2)
{
  char *str3, *pc1, *pc2;
  /* Allocate space for reverse string
     (including null character). */
  str3 = (char *) malloc(sizeof(char) * (strlen(str1) + 1));
```

```
  /* Make reverse copy of first string. */
  for (pc1 = str3, pc2 = str1 + strlen(str1) - 1;
       pc2 >= str1;
       pc2--, pc1++)
    {
      *pc1 = *pc2;
    }
  *pc1 = '\0';

  /* Compare reverse of first string with second string. */

  if (strcmp(str3, str2) == 0)
    {
      free (str3);
      return 1;
    }

  else
    {
      free (str3);
      return 0;
    }

}
```

**Extra Sheet**

**Extra Sheet**

**Extra Sheet**