German University in Cairo June 24, 2012
Faculty of Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Concepts of Programming languages
Spring Term 2012

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 15 pages (9 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	$\sum$
Possible Marks	10	8	8	6	6	14	9	17	5	83
Final Marks										

Exercise 1 (2+2+2+4=10 Marks)

Consider a database about courses, class times, classrooms, and student enrollment, given in the form of the following Prolog facts.

```
when(275,10).
when(261,12).
when(381,11).
when(398,12).
when(399,12).

where(275,owen102).
where(261,dear118).
where(381,cov216).
where(398,dear118).
where(399,cov216).

enroll(mary,275).
enroll(john,275).
enroll(mary,261).
enroll(john,381).
enroll(jim,399).
```

Define the following Prolog predicates by one or more rules. Note that the shown goals are just examples. You should define the predicates so that it is possible to formulate goals with variables or constants at any argument position. **Hint**: The inequality of, say two variables `X` and `Y`, can be expressed using the subgoal `X \= Y`.

a) Define a predicate `schedule/3` that gives for a student the classrooms and times of his or her taken courses, that is, if you evaluate the goal `schedule(mary,P,T)`, Prolog should give the following result.

```
?- schedule(mary,P,T).
P = owen102
T = 10 ;
P = dear118
T = 12 ;
```

As another application of the schedule predicate, consider the goal `schedule(S,cov216,T)` that shows all students that are in the classroom `cov216` together with the corresponding time.

```
?- schedule(S,cov216,T).
S = john
T = 11 ;
S = jim
T = 12 ;
```

Solution:

```
schedule(X,P,T) :- enroll(X,C), where(C,P), when(C,T).
```

b) Define a predicate `usage/2` that gives for a classroom all the times it is used. For example, the goal `usage(cov216,T)` should yield the following result.

```
?- usage(cov216,T).
T = 11 ;
T = 12 ;
```

The goal `usage(X,11)` should list all classrooms that are used at 11.

Solution:

```
usage(P,T) :- where(C,P), when(C,T).
```

c) Define a predicate `conflict/2` that can compute conflicts in the assignment of courses to classrooms. A conflict exists if two different courses are assigned to one classroom for the same time. The arguments of the conflict predicate are two course names. You can use the goal `conflict(275,X)` to find out any courses that are in conflict with the course 275.

```
?- conflict(275,X).
No
```

The goal `conflict(X,Y)` determines all pairs of possible conflicts.

Solution:

```
conflict(C,D) :- where(C,P), where(D,P), when(C,T), when(D,T), C\=D.
```

d) Define a predicate meet/2 that can determine pairs of students that can meet in a classroom by either attending the same course or by having classes that are back to back in one classroom. The last condition means that a student Jim can meet any student who has a course that is in the same classroom and immediately follows Jim's course. (Note that your definition of meet doesn't have to be symmetric, that is, if students `A` and `B` can meet, then your implementation has to return Yes for `meet(A,B)` or `meet(B,A)`, but not necessarily for both calls. You can ignore the case when students are enrolled in conflicting courses.)

Solution:

```
meet(X,Y) :- enroll(X,C), enroll(Y,C), X\=Y.
meet(X,Y) :- enroll(X,C), enroll(Y,D),
             where(C,P), where(D,P),
             when(C,T1), when(D,T2),
             T2 is T1+1, X\=Y.

meetSym(X,Y) :- meet(X,Y); meet(Y,X).
```

Exercise 2 (8 Marks)

Bubble sort is a traditional sort algorithm which is not very effective. The bubble sort works by passing sequentially over a list, comparing each value to the one immediately after it. If the first value is greater than the second, their positions are switched. Implement a predicate `bubble_sort(List,Result)` that holds if `Result` is the sorted sequence of `List`. **Hint:** Use an accumulator to implement bubble sort

Solution:

```
bubble_sort(List,Sorted):-b_sort(List,[],Sorted).
b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-bubble(H,T,NT,Max),b_sort(NT,[Max|Acc],Sorted).

bubble(X,[],[],X).
bubble(X,[Y|T],[Y|NT],Max):-X>Y,bubble(X,T,NT,Max).
bubble(X,[Y|T],[X|NT],Max):-X=<Y,bubble(Y,T,NT,Max).
```

Exercise 3 (6+2=8 Marks)

a) Implement a function called bubble that takes a list of numeric values `l` and returns a pair `(m,l1)` where `m` is the smallest element of `l` and `l1` is the list `l` after removing `m` from it. The elements of `l1` should be in the same order as the one for `l`.

```
Main> bubble [1,2,4,5,3]
(1,[2,4,5,3])

Main> bubble [4,5,2,3]
(2,[4,5,3])

Main> bubble [1]
(1,[])
```

Solution:

```
bubble :: Ord a => [a] -> (a,[a])
```

b) What is the type of the function `bubble`?

Solution:

```
bubble [x]     = (x,[])
bubble (x:xs) | m < x     = (m,x:ys)
              | otherwise = (x,xs)
                where (m,ys) = bubble xs
```

Exercise 4 (6 Marks)

Implement a Haskell function `sublist lis1 lis2` turns `true` if the elements of `lis1` occur in `lis2` in the same order and `false` otherwise. The elements of `lis1` do not have to be consecutive in `lis2`. So `sublis "bd" "abcde"` is `true`, but `sublist "db" "abcd"` is `false`.

Solution:

```
sublist [] _ = True
sublist _ [] = False
sublist (x:xs) (y:ys) = (x == y && sublist xs ys) || (sublist (x:xs) ys)
```

Exercise 5 (6 Marks)

Implement a Haskell function `sumSquarePos lis` that returns the sum of the squares of the positive integers in a list of integers `lis`.

The function should be implemented using the higher-order functions that are introduced in the lectures. No recursion is allowed.

Solution:

```
sumSquarePos :: [Int] -> Int
sumSquarePos lis = foldr (+) 0 (map square (filter positive lis))
  where
  positive x = x > 0
  square x = x * x
```

Exercise 6 (3+3+(3+2+3)=14 Marks)

a) Given this definition:

```
q1 [] x = x
q1 (x:xs) y = q1 xs (x-y)
```

Write the most general type of `q1`. (In other words, write what Hugs would type in response to `:t q1`.) Justify your answer.

Solution:

```
q1 :: Num a => [a] -> a -> a
```

b) Given this definition:

```
q2 x y z w = w (x y) (z y)
```

Write the most general type of `q2`.Justify your answer.

Solution:

```
q2 :: (a -> b) -> a -> (a -> c) -> (b -> c -> d) -> d
```

c) Given this definition:

```
mystery 0 _ x = x
mystery n f x = mystery (n-1) f (f x)
```

1. Write the most general type of `mystery`. Justify your answer.

   Solution:

   ```
   mystery :: Num a => a -> (b->b) -> b -> b
   ```

2. What is the output of the following term? Justify your answer.

   ```
   Main> mystery 3 (*2) 4
   ```

   Solution:

   ```
   32
   ```

3. What is the functionality of the function `mystery` for any function `f`?

Exercise 7 (2+2+5=9 Marks)

Given the following data type in Haskell:

```
data Instruction = PUSH Int  -- push an integer on the stack
                 | ADD       -- add the top two elements of the stack
                 | SUB      -- substract the top element from the second element
                 | MUL       -- multiply the top two elements of the stack
                   deriving (Show)
```

a) Define a type `Stack` that consists of a lists of integers

Solution:

```
type Stack = [Int]
```

b) Define a type `Program` that consists of a lists of instructions.

Solution:

```
type Program = [Instruction]
```

c) Define a function `calc` with the following type

```
calc :: (Program, Stack) -> Stack
```

that performs the set of instructions defined in `Program` on a stack. Make sure that the operations `ADD`, `SUB` and `MUL` will replace the top two elements of the stack by the result of the operation performed, respectively.

Solution:

```
calc ([],s) = s
calc (PUSH n : rest, s) = calc(rest, n:s)
calc (ADD : rest, m:n:s) = calc(rest, (m+n):s)
calc (SUB : rest, m:n:s) = calc(rest, (m-n):s)
calc (MUL : rest, m:n:s) = calc(rest, (m*n):s)
calc _ = error "stack underflow"
```

Exercise 8 (2+3+4+4+4=17 Marks)

Given the following data structure for defining a bank account in C:

```
typedef struct {
        int number;
        }BankAccount;
```

a) Define a data structure for representing a transaction. Each transaction is related to one bank account. It is either a deposit transaction or a non-deposit transaction concerned with a specific amount of money.

Solution:

```
typedef struct {
        BankAccount *account;
        int deposit;
        int amount;
        }Transaction;
```

b) Define a data structure for representing a customer. Each customer has a name. Each customer has only one bank account. An account has transactions performed on it. Each account has a maximum number of transactions that could be performed.

Solution:

```
typedef struct {
        char *name;
        BankAccount *account;
        Transaction *transactions;
        int maxNumberOfTransaction;
        int currentNumberOfTransactions;
        }Customer;
```

c) Define a function `Customer* addANewCustomer(char* cname, BankAccount* bk, int max)` that creates a new customer with name `cname` where `bk` is a pointer to his/her bank account and `max` is the maximum number of transactions that could be performed.

Solution:

```
Customer* addANewCustomer(char* cname, BankAccount* bk, int max)
{
    Customer *newC=(Customer*)malloc(sizeof(Customer));
    (*newC).name=cname;
    (*newC).maxNumberOfTransaction=max;
    (*newC).currentNumberOfTransactions=0;
    (*newC).transactions=(Transaction*)malloc(max*sizeof(Transaction));
    return newC;
}
```

d) Define a function `void addNewDepositTransactionTo(Customer* customerToAdd, int moneyAmount)` that adds a new deposit transaction to the customer with amount `moneyAmount` where `customerToAdd` is a pointer to the customer.

Solution:

```
     Transaction *new_t=malloc(sizeof(Transaction));
     (*new_t).account=(*customer_to_add).account;
```

```
    (*new_t).deposit=1;
    (*new_t).amount=money_amount;

    int index=(*customer_to_add).currentNumberOfTransactions;
    Transaction *transactions=(*customer_to_add).transactions;
   *(transactions+index)=(*new_t);
    (*customer_to_add).currentNumberOfTransactions++;
```

e) Define a function `int checkAllNonDeposit (Customer customersArray[], int len)` that returns 1 if all cutomers in the array `customersArray`, that has length `len`, have only performed non-deposit transactions.

Solution:

```
int checkAllNonDeposit(Customer customersArray[], int len)
{
    int i;
    for(i=0;i<len;i++)
    {
                  int len2=(customers_array[i]).currentNumberOfTransactions;
                  Transaction *transactions=(customers_array[i]).transactions;
                  int j;
                  for(j=0;j<len2;j++)
                  {
                             Transaction current_transaction=*(transactions+j);
                             if(current_transaction.deposit)
                                     return 0;
                  }
    }
    return 1;

}
```

Exercise 9 (5 Marks)

Define a function `isSubString` in C that accepts two strings as parameters and returns 1 if the first string is a substring in the second one and 0 otherwise.

Solution:

```
int is_sub_string(char* subString,char* initialString)
{
    int i=0;
    int j=0;
    char* initialSubString = subString;
    while(*(initialString+i)!=’\0’)
    {
        if(*(initialSubString+j)==’\0’)
        return 1;
        else
         {
            if(*(initialSubString+j)==*(initialString+i))
              j++;
            else
             j=0;
         }

          i++;
    }
    if(*(initialSubString+j)==’\0’)
                                   return 1;
    return 0;
}
```

Extra Sheet

Extra Sheet

Extra Sheet