Instructions for Students

- Write your name and student number on the exam sheet and on every solution sheet you hand in and also sign them.
- This is a closed book exam: the only resources allowed are blank paper and pens.
- Use a pen and not a pencil.
- Write neatly and clearly. The clarity of your explanations will affect your grade.
- At least 51 points are required to pass the exam.
- The duration of the exam is 2 hours.

Good luck!

Do not write in this space

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Marks</th>
<th>Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 1 (20 marks)

a. (4 marks) Briefly describe the imperative programming paradigm and the key elements.

b. (4 marks) Briefly describe the concept of mixins in Ruby.

c. (4 marks) Consider the following Prolog program:

```
    a(1).
    a(z).
    b(3).
    b(z).
    c(A,B) :- b(B), !, a(A).
    d(A,B) :- b(B), a(A).
```

How many solutions are produced by each of the two queries `c(A,B)` and `d(A,B)`? Briefly explain your answer.

d. (4 marks) Briefly describe the concept of guards in Haskell and how they can be used.

e. (4 marks) Briefly explain how in Erlang synchronous messaging between two processes is achieved?

Exercise 2 (8 marks) Write a class Vehicle with a field `year` and a method `printStats` that prints the value of `year`. The value of `year` is given as an argument to the constructor of Vehicle. Next, write a class Car that extends Vehicle and has two fields `model` and `brand`. The constructor of car has three arguments, and it assigns values to `model` and `brand` and calls the constructor of the superclass Vehicle to initialize `year`. Furthermore, create a method `printStats` which prints `model` and `brand`, and calls the parent method in Vehicle to print `year`. Finally, assume that there exists a module FourWheeled. Use this module to create a mixin with the class Car.

Exercise 3 (12 marks) The following Ruby function computes \( \lceil \log_2 n \rceil \) recursively:

```
def log( n )
    return 0 if n == 1
    if n.even?
        n = n / 2
        return log( n ) + 1
    else
        n += 1
        return log( n )
    end
end
```
a Rewrite this function into a tail-recursive function.

b Rewrite the above function to compute $\lceil \log_x n \rceil$, i.e., logarithm to base $x$.
(Hint: $n \mod x$ computes $n \mod x$)

Exercise 4 (16 marks) Write a Prolog program `dropevery_kth(K,L,R)`, which removes every $K$-th element in a list $L$ and returns the resulting list in $R$. For instance, `dropevery_kth(2,[a,b,c,d,e],R)` succeeds with $R = [a,c,e]$ and `dropevery_kth(1,[a,b,c,d,e],R)` succeeds with $R = []$.

Exercise 5 (8 marks) Let the variables $X$, $Y$ and $Z$ represent three different dices. Write a Prolog predicate `dice(X,Y,Z,N)`, that uses the generate and test pattern to compute all possible combinations of the three dices such that their numbers sum up to $N$. For instance, there is exactly one solution that sums up to 3, namely $X = 1$, $Y = 1$ and $Z = 1$, whereas for $N = 4$ we get 3 solutions.

Exercise 6 (10 marks) Look at the following Haskell program:

```haskell
afun :: [Int] -> Bool
afun l = (anotherfun l) == 0

anotherfun :: [Int] -> Int
anotherfun [] = 0
anotherfun (h:t) = h + (anotherfun t)
```

a. (4 marks) Briefly describe what the program does.

b. (6 marks) Transform the program into a tail-recursive one.

Exercise 7 (14 marks) Write a Haskell module which exports a function `balancedexpr` that checks whether a string containing an arithmetic expression with open and closed parentheses is balanced. An expression is balanced when every open parenthesis has a corresponding closed one and at any point there are not more closed ones than open ones. Remember that in Haskell a string is internally processed as a list of characters. For example,

```haskell
balancedexpr "" returns True
balancedexpr "(4*(3+2))" returns True
balancedexpr "(4*(3+2))" returns False
```

Exercise 8 (12 marks) Write an Erlang module which exports a function `loop` that implements an accumulator for numbers and reacts to the following messages: if a number is received, it is added to the accumulator; if "reset" is received, the accumulator is reset to zero; if "sum" is received, the value of the accumulator is printed out; if "exit" is received, the process is stopped (for all other messages, the accumulator continues to run). In all cases, a corresponding message is printed.
Solution 1

a. Imperative programs define sequences of statements that change the program state (i.e., set of variable). The most important concepts are:

- Assignment statement: assigns values to memory locations and changes the current state of a program
- Variables: refer to memory locations
- Step-by-step execution of commands
- Control-flow statements: Conditional and unconditional (GOTO) branches and loops to change the flow of a program

b. A mixin in Ruby is a combination of modules and classes. More specifically, a module can be included in a class definition. By doing so, all methods of the module are added and available to the class. Mixins have some similarity to the concept of multiple inheritance.

c. (a) $c(A, B)$ has 2 solutions: $A = 1, B = 3; A = z, B = 3$.

   The cut operator (!) stops backtracking and prevents $b(B)$ from being matched with the second b-fact.

   (b) $d(A, B)$ has 4 solutions: $A = 1, B = 3; A = z, B = 3; A = 1, B = z; A = z, B = z$ (or $A = B, B = z$).

d. Guards are boolean conditions that can be used to check/constrain the arguments in function definitions, and hence influence the pattern matching process. They follow the parameters of a function and begin with a pipe symbol "|". If the guard is satisfied, the corresponding function body is executed; otherwise, pattern matching jumps to the next guard.

e. At the receiver side:

   - Each receive clause will have to match the process ID of the requesting sender (in addition to the content of the message).
   - Each receive clause has to send a response to the sender (instead of/in addition to printing some result).

At the sender side:

   - After sending a message, the sender has to wait for a response.

Solution 2
class Vehicle
  def initialize(year)
    @year = year
  end

  def printStats
    print "Year constructed: #{@year}\n"
  end
end

class Car < Vehicle
  include FourWheeled
  def initialize(brand, model, year)
    super(year)
    @brand = brand
    @model = model
  end

  def printStats
    print "Brand: #{@brand}\n"
    print "Model: #{@model}\n"
    super
  end
end

Solution 3

a  Tail recursive function for ⌈log₂ n⌉

  def log_tail_recursive( n )
    return log_tr( n, 0 )
  end

  def log_tr( n, r )
    return r if n == 1
    if n.even?
      n = n / 2
      return log_tr( n, r+1 )
    else
      n += 1
      return log_tr( n, r )
    end
  end

Function for $\lceil \log_x n \rceil$

```python
def logx( n, x )
    return 0 if n == 1
    j = n % x
    if j == 0
        n = n / x
        return logx( n, x ) + 1
    else
        n += (x - j) // n += 1 would also work
        return logx( n, x )
end
end
```

Solution 4

dropevery_kth( K, L, R ) :-
    dropevery_kth( K, K, L, R ).

dropevery_kth( _, _, [ ], [ ] ).
dropevery_kth( 1, K, [ _|Xs], Y ) :-
    !,
    dropevery_kth( K, K, Xs, Y ).
dropevery_kth( N, K, [X|Xs], [X|Ys]) :-
    N > 1,
    N1 is N - 1,
    dropevery_kth( N1, K, Xs, Ys ).

Alternative solution with modulo:

dropevery2_kth( K, L, R ) :-
    dropevery2_kth( 1, K, L, R ).

dropevery2_kth( _, _, [ ], [ ] ).
dropevery2_kth( N, K, [ _|Xs], Y ) :-
    X is N mod K,
    X = 0,
    !,
    N1 is N + 1,
    dropevery2_kth( N1, K, Xs, Y ).
dropevery2_kth( N, K, [X|Xs], [X|Ys]) :-
    N1 is N + 1,
    dropevery2_kth( N1, K, Xs, Ys ).
Solution 5

dice( X, Y, Z, N ) :-
    member( X, [1,2,3,4,5,6] ),
    member( Y, [1,2,3,4,5,6] ),
    member( Z, [1,2,3,4,5,6] ),
    N is X + Y + Z.

Solution 6

a. The program checks whether the elements in an array sum up to zero, that is, it
returns true if the sum is zero, and false otherwise.

b. Tail-recursive version: pass the sum of the elements seen so far as an accumulator;
when the list is empty, the accumulator contains the sum of all elements.

zerosumtr :: [Int] -> Bool
zerosumtr l = (sumuptr l 0) == 0

sumuptr :: [Int] -> Int -> Int
sumuptr [] a = a
sumuptr (h:t) a = sumuptr t (a+h)

Solution 7

module BalancedExpr ( balancedexpr ) where

balancedexpr :: [Char] -> Bool
balancedexpr e = balanced e 0

balanced :: [Char] -> Int -> Bool
balanced [] 0 = True
balanced [] _ = False
balanced (h:t) x
    | x < 0             = False
    | h == '('          = balanced t (x+1)
    | h == ')'         = balanced t (x-1)
    | otherwise        = balanced t x

-- Alternative implementation of balanced with if
balanced2 :: [Char] -> Int -> Bool
balanced2 [] 0 = True
balanced2 [] _ = False
balanced2 (h:t) x = 
  if x < 0 then
    False
  else
    if h == '(' then
      balanced2 t (x+1)
    else
      if h == ')' then
        balanced2 t (x-1)
      else
        balanced2 t x

Solution 8

-module(accumulator).
-export([loop/0]).

loop() -> loop(0).

loop( Sum ) ->
  receive
    "sum" ->
      io:format( "Sum ~p~n", [Sum] ),
      loop( Sum );
    "reset" ->
      io:format( "Reset to 0 ~n" ),
      loop( 0 );
    "exit" ->
      io:format("Exit~n");
    N if is_number(N) ->
      io:format( "Increment by ~p~n", [N] ),
      loop( Sum + N );
    _ ->
      io:format("Invalid message~n"),
      loop( Sum )
  end.