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 (do not use pencils).

• Write neatly and clearly. The clarity of your explanations will affect your grade.

• 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>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>14</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 concept of abstract data types and the advantages they introduced with respect to imperative/procedural programming.

b. (4 marks) What does the following Ruby-code print?

```ruby
def like_map(array)
  result = []
  array.each do |element|
    result << (yield element)
  end
  result
end

x = like_map([1, 2, 3]) do |number|
  number * 2
end

print x
```

c. (4 marks) The box model of Prolog execution is a simple way to show the control flow. Briefly sketch and describe the box model.

d. (4 marks) Consider the Haskell function `plus x y = x + y`. How is the function call `plus 20 4` evaluated? What is the name of this evaluation concept?

e. (4 marks) Briefly explain how in Erlang synchronous messaging between two processes is achieved?
Exercise 2 (8 marks) Write a Ruby function `prime_numbers` that has one input parameter `n` and returns an array containing the first `n` prime numbers (2 is the first prime number). Your program should use at least one code block.

Exercise 3 (8 marks) Write a Ruby class `Animal` with a property

- `kind`: a string that holds the type of the animal

and the following instance methods:

- `eat`: takes a parameter `food` and prints a message that the animal is eating `food`
- `sleep` and `wake`: these two methods do not have any arguments; instead, they will set an instance variable `@state` to the string "asleep" and "awake", respectively

Write a second Ruby class `Person` with the following characteristics:

- Inherits from `Animal`
- Automatically sets `@kind` to "person"
- Adds 3 new instance variables: `age`, `gender`, `name`
- Overrides the `eat` method so that a person cannot eat a "person"

Exercise 4 (10 marks) Write a Prolog program `drop_kth(K, L, R)`, which removes the element at position `K` from the list `L` and returns the resulting list in `R`. For instance, `drop_kth(3, [a,b,c,d,e], R)` succeeds with `R = [a,b,d,e]`.

Exercise 5 (10 marks) The following Prolog knowledge base describes a small social network using the `friend` relation that represents a direct friendship between two persons.

```
friend(tom,tim).
friend(tom,alf).
friend(alf,ann).
friend(alf,joe).
friend(joe,sue).
friend(joe,tim).
friend(sue,ann).
```

Write a predicate `friends_dist(X,Y,D)` which tells whether `X` and `Y` are connected by friendship relations at a distance of `D`. For instance, `friends_dist(tom,Y,2)` succeeds and instantiates `Y=ann` and `Y=joe`.
Exercise 6 (14 marks) Write a Haskell module that exports a function `split`, which splits a list at a given position. The list and the split position are given as input parameter; the function returns a pair consisting of the two parts of the list. For instance, `split [1,2,3,4,5] 2` returns the two parts ([1,2], [3,4,5]).

Exercise 7 (16 marks) Write a Haskell module that exports a function `diffAB` which takes as input a list and two elements `a`, `b` of the list and returns the difference between the number of occurrences of `a` and `b` in the list. For instance, `diffAB [3,4,2,3,3] 3 4` returns 2.

Exercise 8 (14 marks) Write an Erlang module that exports a function `loop` for a process that implements an accumulator for numbers and reacts as follows to 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, an error message is shown. In all cases, a corresponding message is printed.

Moreover, show the following steps:

- Start the process
- Send a message to increment the accumulator by 10
- Show the value of the accumulator
- Stop the process