http://www.se.rwth-aachen.de/

Department of Software Engineering
RWTH Aachen University
Prof. Dr.-Ing. Manfred Nagl, Emeritus,
Dipl.-Math. Michael von Wenckstern
(vonwenckstern@se-rwth.de)

Exercise course: *Ada*
WS 2014 / 15
November 5, 2014

Exercise Sheet 4

Exercise on November 12th,2104 does not take place. This is why this exercise sheet contains so many tasks and points.

Submission:

When:	**Thu, November 18th**, 2014. 11:55 pm
Where:	L^{2}P-eLearning room of Ada or e-mail

Organization

Exercise sheets must be submitted in groups of two to four students. The submission must be delivered electronically via the L^{2}P-eLearning room of Ada.

Exercise 4.1 Programming Task – Matrix (8 points)

a) Define a data type for matrices as a two-dimensional real array with unspecified bounds.
b) Write a procedure "Init" which initializes a matrix with values 0.0. The procedure should not contain a loop.
c) Implement functions for addition and multiplication of matrices with type Matrix. For that, overload the appropriate arithmetic operators. Which exception occurs, if matrices do not match?
d) Provide a procedure "Read" which reads matrices (by rows with "arbitrary" number of rows and characters per row) via standard input.
e) Write a procedure "Sort" which sorts a matrix as follows: The smallest value should be at position (1,1). All other values should be inserted into the column with increasing values. By this, the first column contains the smallest values of the matrix. Specify the interface, so that the implementation of an arbitrary sorting algorithm can be used in the procedure body.

Illustrate the procedures by some input and output examples.

Exercise 4.2 Discriminated Record Types (4 points)

a) Two use cases are introduced for discriminated record types: (1) The discriminants are used to define the size of an array component of a record and (2) to construct a variant part in a record. There are two further use cases for discriminants. It is possible to define default values and discriminant constraints for record components. Explain these two cases and give an example for them.

b) Are the following declarations valid?

```
type R (J: positive) is record
      S: String (1..J);
end record;

A: array(1..10) of R;
```

Exercise 4.3 Types, Derived Types, Subtypes (4 points)

a) What is wrong in the following declaration?

```
type Week_Number_Type is Integer range 1..52;
```

Illustrate how a valid integer type declaration, a derived type declaration, and a subtype declaration can be made for the declaration above.

b) Consider the following declarations:

```
type Hour_Number_Type is range 1..12;
subtype Late_Hour_Number_Type is Hour_Number_Type range 7..12;

type Month_Number_Type is range 1..12;
subtype Late_Month_Number_Type is Month_Number_Type range 7..12;

type One_To_Twelve_Type is range 1..12;
Hour_Number         : Hour_Number_Type;
Late_Hour_Number    : Late_Hour_Number_Type;
Month_Number        : Month_Number_Type;
Late_Month_Number  : Late_Month_Number_Type;
One_To_Twelve       : One_To_Twelve_Type;
```

Which of the following assignments are legal (requires a runtime-check)?

1. `Hour_Number := Late_Hour_Number;`
2. `Late_Hour_Number := Hour_Number;`
3. `Hour_Number := Month_Number;`
4. `Month_Number := Late_Month_Number;`

5. `Hour_Number := Late_Month_Number;`
6. `One_To_Twelve := Hour_Number;`

Exercise 4.4 Pointer and Access Types (4 points)

a) Explain the differences of the following declarations:

```
type IntAccess is access Integer;

type IntAccessAll is access all Integer;

type IntAccessConst is access constant Integer;
```

Explain the meaning of the key words `aliased` and `aliased constant`, as used as follows:

```
a: Integer :=7;  b: aliased Integer :=7;  c: aliased constant Integer :=8;

p: IntAccess;    q: IntAccessAll;         r: IntAccessConst;
```

Which of the following assignments are allowed? Give reasons for your answer.

```
1) a:=b;            2) q := c'Access;    3) b:=c;    4) r:= b'Access;

5) c:=a;            6) r:= new Integer; 7) a:=p;    8) r.all := a;

9) p := a'Access; 10) p := b'Access;    11) q :=p; 12) p:= q'Access;
```

b) Data structures using pointers are often compared with spaghetti code which is the outcome of intensive use of unconditional jumps (gotos). What is the reason for that comparison? What arguments are there why this comparison is unstable? What Ada 95 does to “ensure” the use of pointers in programs?

Exercise 4.5 Snake-Video-Game using GtkAda (15 points)

Introduction to Ada packages.
An explanation of the game is given at http://de.wikipedia.org/wiki/Snake.

a) Create a ADA package `Snake`. It should be an Abstract Data Object.
   a. The record type `Point` contains the x and `y` position of one snake body part.
   b. The package has a `root` pointer for the first body part.
   c. The record type `Element` contains the record type `Point` and a reference to the previous and next `Element` (body parts).

   d. This class also has a Food variable of the type `Point` containing the position of the food.
   e. Create function which returns true, if the Snake eats itself (The head has the same position like any other body part)

```
function EatingMySelf return Boolean;
```

f. Create an enumeration type `Direction_Type` with the following elements `Left`, `Right`, `Up`, `Down`
g. The procedure `Move` moves the snake in the desired direction. This is done by deleting the last body part and inserting a new body part at the beginning of the "body part list". But if the new head is at the same position of the food, then the snake is growing (meaning the last body part will not be deleted) and a new food position will be chosen randomly. If the snake did move 50 times and did not eat the food, then also a new food position will be chosen (maybe the food is at a position where the snake cannot go).

```
procedure Move(Direction: in Direction_Type);
```

b) Create a ADA package `Environment`. It should be an Abstract Data Type.
   a. All procedures have as parameter a `100x100 matrix of Boolean` elements (`Matrix100_Type)`. At the beginning the matrix is empty (all elements are false). If an element being true means that it is an obstacle and the snake cannot go there.
   b. Add procedures to add the following obstacle types:

```
procedure Rectangle(Matrix: Matrix100_Type; X, Y, width,
                    height: Natural100);

procedure Circle(Matrix: Matrix100_Type; X, Y,
                    radius: Natural100);

procedure Line(Matrix: Matrix100_Type; X1, Y1, X2,
                    Y2: Natural100);
```

   c. Create a function which returns true, when the snake hits an obstacle.

```
function TouchedObstacle(Matrix: Matrix100_Type; S: Snake)
return Boolean;
```

c) Create a ADA package for the User (using GtkAda). It should contain of a `Gtk_Window` displaying the content (Snake and obstacle) to the user.
   a. And has the following procedures:

```
procedure Draw(Obstacle: Matrix100_Type);
-- obstacles will be drawn black
-- each matrix element has a draw size of 5x5 pixel
-- --> The playing field is 500x500 pixels

procedure Draw(S: Snake);
-- Snake will be drawn green, the head (root element) will be read
```

   b. This package also registers the handlers for the key events (using the arrows on the keyboard). Use a variable which has the actual direction and change the direction, when the user presses any arrow key. Every 50ms invoke the

`Snake’s Move` procedure. After the move procedure, it also checks whether the snake hits an obstacle or is eating itself. If this happens the game is over.

d) Start the game with a snake of the length four and at least one rectangle, circle and line obstacle.
e) If the game is over, then the actual screenshot should be frozen as background. In the foreground a text displaying the snake’s length and the used timed in seconds will be shown.