Problem Set 2
Course **Safe and Secure Software**
(Winter Term 2016)

Bauhaus-Universität Weimar, Chair of Media Security
Prof. Dr. Stefan Lucks, Eik List
URL: `http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching/`

Due Date: 07 Nov 2016, 1:30 PM, via email to `eik.list(at)uni-weimar.de`.

Goal of This Problem Set: Learn packages, types, records, Pre-/Post-conditions, and exception handling in Ada.

I will notify (via e-mail) those students who will present a mini-project.

Task 1 – Introduction (No Credits)

Read Chapters 4 to 8 of John English.

Task 2 – Randomizing, Enums and Types (4 Credits)

Implement Task 5.4 of John English.

Task 3 – Pre- and Post-Conditions (4 Credits)

Implement the following specification and add appropriate pre- and postconditions.

```ada
package Bank_Accounts is
    subtype Cents_Type is Integer;
    type Account_Type is record
        Balance: Cents_Type := 0;
    end record;

    function Get_Balance(Account: Account_Type) return Cents_Type;
    -- Returns the current Balance from Account.
    procedure Deposit(Account: in out Account_Type; Amount: Cents_Type);
    -- Deposits Amount at the given Account.
    procedure Withdraw(Account: in out Account_Type; Amount: Cents_Type);
    -- Withdraws Amount from the given Account.
    procedure Transfer(From: in out Account_Type;
                       To: in out Account_Type;
                       Amount: in Cents_Type);
    -- Transfers Amount from Account From to Account To.
end Bank_Accounts;
```

Task 4 – Mini Project 1 – Vectors (6 Credits)

Implement the following package of Vector arithmetic.

```ada
package Vectors is
    type Vector is record
        X: Float := 0.0;
        Y: Float := 0.0;
        Z: Float := 0.0;
    end record;

    function "+"(Left: Vector; Right: Vector) return Vector;
    -- Adds two vectors dimension-wise.
    function "-"(Left: Vector; Right: Vector) return Vector;
    -- Subtracts the right vector from the left one dimension-wise.
    function "*"(Left: Vector; Right: Float) return Vector;
    -- Multiplies all dimensions of Left by Right.
    function "*"(Left: Vector; Right: Vector) return Float;
    -- Computes the scalar product.
    function "="(Left: Vector; Right: Vector) return Boolean;
    -- Returns True if all dimensions of Left are equal to that of Right;
    -- Returns False otherwise.
    function Are_Orthogonal(Left: Vector; Right: Vector) return Boolean;
    -- Determines if both vectors stand orthogonal to each other or not.
    function Cross_Product(Left: Vector; Right: Vector) return Vector;
    -- Computes the cross product.
    function Distance(Left: Vector; Right: Vector) return Float;
    -- Computes the distance between both vectors.
    function Distance_To_Origin(Item: Vector) return Float;
    -- Computes the distance to the origin of the coordinate system.
    procedure Put(Item: Vector);
    -- Prints the vector in the format (X, Y, Z).
end Vectors;
```

Task 5 – Mini Project 2 – Graphs (6 Credits)

Implement the following Graph package.

```
generic
    type Vertex_Type is private;
    with function "="(Left: Vertex_Type; Right: Vertex_Type) return Boolean;
package Graph is
    Edge_Not_Found_Exception: exception;
    Vertex_Already_In_Graph_Exception: exception;

    type Edge_Type is private;
    type Vertex_Array is array(Natural range <>) of Vertex_Type;

    procedure Add_Vertex(Vertex: Vertex_Type);
    -- Stores the Vertex in the Graph. Raises a
    -- Vertex_Already_In_Graph_Exception if it is already in the graph.
    procedure Add_Edge(From: Vertex_Type; To: Vertex_Type; Weight: Integer);
    -- Stores a new edge in the Graph from From to To that has the given
    -- Weight assigned to it. If an edge from From to To is already stored
    -- in the Graph, this function only re-assigns the given Weight to it
    -- and does nothing beyond.
    procedure Clear;
    -- Removes all vertices and edges from the graph.
    function Get_Edge_Weight(From: Vertex_Type; To: Vertex_Type) return Integer;
    -- Returns the weight of the edge, if it is stored in the graph.
    -- Raises an Edge_Not_Found_Exception otherwise.
    function Has_Edge(From: Vertex_Type; To: Vertex_Type) return Boolean;
    -- Returns True if an edge from From to To is stored in the graph.
    -- Returns False otherwise.
    function Remove_Edge(From: Vertex_Type; To: Vertex_Type) return Boolean;
    -- Removes the edge in the Graph from From to To, if existing;
    -- Raises an Edge_Not_Found_Exception otherwise.
    function To_Vertex_Array return Vertex_Array;
    -- Returns an array containing exactly all current vertices of the graph.
private
    -- TODO
end Graph;
```