Problem Set 2
Course **Security Engineering**
(Winter Term 2021)

Bauhaus-Universität Weimar, Chair of Media Security
Prof. Dr. Stefan Lucks, Jannis Bossert
URL: `http://www.uni-weimar.de/de/medien/professuren/mediensicherheit/teaching/`
Due Date: 28 April 2021, 9.15 AM CEST, via Moodle.

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

General Remarks:

You can sign up for the mini projects via moodle, first come – first served.

The `.ads` files for tasks 3 to 7 can also be found on moodle.

Think of appropriate exceptions and how to handle them.

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.

```
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.

```
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 – Galois Field ($GF(2^n)$) (6 Credits)

The Galois Field $GF(2^n)$ is a finite field. Each element $a = (a_0, \ldots, a_{n-1}) \in GF(2^n)$ can be represented as an n-bit vector, and can be written uniquely in the form

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_0, \quad \text{with } a_i \in \{0, 1\}.$$

Hint:

- A (brief) description of the finite field arithmetic can be found here and here.

Implement the following specification.

```
generic
    -- Example: A type GF_2_4_Type is mod 2**4 means that we consider the
    -- elements of Galois Field (2**8).
    type Element_Type is mod <>;

    -- An irreducible polynomial P required for arithmetics.
    P: Element_Type;
package GF2n is
    function "+"(X: Element_Type; Y: Element_Type) return Element_Type;
    function "-"(X: Element_Type; Y: Element_Type) return Element_Type;
    function "*"(X: Element_Type; Y: Element_Type) return Element_Type;
    function "/"(X: Element_Type; Y: Element_Type) return Element_Type;

```

```
    function Find_Inverse(X: Element_Type) return Element_Type;
    function GCD(X: Element_Type; Y: Element_Type) return Element_Type;
    function Is_Primitive(X: Element_Type) return Boolean;
end GF2n;
```

Task 6 – Mini Project 3 – Graphs (6 (Extra) Credits)

Implement the following Graph package.

```
with Ada.Containers.Vectors;

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
    type Edge_Type is record
        From: Vertex_Type;
        To: Vertex_Type;
        Weight: Integer := 0;
    end record;

    package Edge_Vectors is new Ada.Containers.Vectors(
        Element_Type => Edge_Type,
        Index_Type => Natural);
    package Vertex_Vectors is new Ada.Containers.Vectors(
        Element_Type => Vertex_Type,
        Index_Type => Natural);
    use Edge_Vectors;
    use Vertex_Vectors;

    Edges: Edge_Vectors.Vector;
    Vertices: Vertex_Vectors.Vector;

end Graph;
```

Task 7 – Mini Project 4 – RGB (6 (Extra) Credits)

Implement the following package using saturation arithmetic.

```ada
package RGB is
  type Color_RGB is private;
  type Color_HSV is private;  -- has to be defined in private part
  type Color_CMYK is private; -- has to be defined in private part

  -- subtypes for Color_RGB
  subtype Intensity is Integer range 0..255;

  -- define appropriate subtypes for Color_HSV and Color_CMYK
  -- [...]

  function "+"(Left: Color_RGB; Right: Color_RGB) return Color_RGB;
  function "-"(Left: Color_RGB; Right: Color_RGB) return Color_RGB;
  function "*"(Left: Color_RGB; Right: Color_RGB) return Color_RGB; -- dot product

  function RGB_To_HSV(Item: in Color_RGB) return Color_HSV;
  function RGB_To_CMYK(Item: in Color_RGB) return Color_CMYK;

  procedure Put(Item: in Color_RGB);
  procedure Put(Item: in Color_HSV);
  procedure Put(Item: in Color_CMYK);

private
  type Color_RGB is record
    Red, Green, Blue: Intensity;
  end record;
end RGB;
```