Problem Set 4
Safe and Secure Software (WS 11/12)

Bauhaus-University Weimar, Chair of Media Security
Prof. Dr. Stefan Lucks, Christian Forler
Url: `http://www.uni-weimar.de/cms/medien/mediensicherheit.html`

Problem 1: Let's SPARK
Download and install the SPARK GPL Edition from http://libre.adacore.com/libre/download/. Then read "Let's SPARK!" Part 1 and 2:

- http://www.adacore.com/2009/06/29/gem-68/,
- http://www.adacore.com/2009/09/07/gem-69/.

Problem 2: Let's SPARK RGB (3 Points)
Enrich the package specification with SPARK annotations to allow a complete

- exception freeness,
- data flow, and
- information flow analysis

using the SPARK `Examiner`.

```
package RGB is
   type Color is private;
   subtype Intensity is Integer range 0..255;

   function To_Color(Red   : Intensity;
                     Green : Intensity;
                     Blue  : Intensity)
                    return Color;


   -- Returs True if "Item" is a valid color, otherwise False
   -- Implementation notes:
   -- "Item" is valid if and only if Item.Valid = True,
   -- otherwise "Item" is invalid.
   function Is_Valid(Item : Color) return Boolean;

   -- Return only True if "Left" and "Right" is a valid color, and
   -- "Left" and "Right" represent the same color in the RGB color model,
   function "="(Left : Color; Right : Color) return Boolean;

   -- Perform saturation arithmetic if "Left" and "Right" is a valid color,
```

```
   -- otherwise return any invalid color.
   function "+"(Left : Color; Right : Color)  return Color;
   function "-"(Left : Color; Right : Color)  return Color;
   function "*"(Left : Color; Right : Color)  return Color;

   -- Perform saturation arithmetic if "Left" and "Right" is a valid color,
   -- otherwise return an invalid color.
   -- If there is a division by zero, then an invalid number is returned.
   Returns any invalid number if division by zero occurs.
   function "/"(Left : Color; Right : Color)  return Color;

   -- Print the hex string reprensentation of Item if valid,
   -- otherwise print "Invalid Color".
   procedure Put(Item : in Color);

private
   type RGB is (Red, Green, Blue);
   type RGB_Color is array (RGB) of Intensity;
    type Color is record
      Value : RGB_Color;
      Valid : Boolean := True;
   end record;
end RGB;
```

Problem 2: Coffee Machine'12 (2 Points)
Enrich the package specification with resonable pre- and post-conditions (using the dynamic DbC feature of Ada'12).

```
package Coffee_Machine is
   -- Simulation of a coin-driven coffee machine
   -- User: - One slot to insert coins (only, 10 or, 20 cents)
   --       - One button to press (''money back'')
   -- Machine: one slot to drop coins, the coffee output
   -- Given 30 cents or more, the coffee is produced immediately
   -- (Note that Overspending is Possible)

   type State is private;
   type Action is(Ten_Cent, Twenty_Cent, Button);
   type Reaction is(Nothing, Drop_All_Coins, Coffee);

   procedure Initialize( X : out State);
   procedure X(S     : in out State;
               Act   : in Action;
               React : out Reaction);

private
   type State is range 0..2;
```

```
end Coffee_Machine;
```

Mini-Project 1: Graph Algorithms (4 Points)
Implement the following specification.

```
with Mark_Graph;

generic
   with package Graph  is new Mark_Graph(<>);
package Graph_Algorithms is
   -- Marks all Vertices with Vertex_Mark'First that are reachable from
   -- vertex "Source" using the "Breadth_First_Search" algorithm.
   -- Raises Constraint_Error if Vertex_Mark enumeration has less then two
   -- entries, or Source is not a vertex of G.
   procedure Breadth_First_Search(G      : in out Graph;
                                  Source : in out Vertex_Type);

   -- Marks all Vertices with Vertex_Mark'First that are reachable from
   -- vertex "Source" using the "Depth_First_Search" algorithm.
   -- Raises Constraint_Error if Vertex_Mark enumeration has less then three
   -- entries, or Source is not a vertex of G.
   procedure Depth_First_Search(G      : in out Graph;
                                Source : in out Vertex_Type);


   -- Returns the minimal spanning tree (MST) that includes vertex
   -- source.
   -- Return an empty graph if Source is not a vertex of G.
   function Minimal_Spanning_Tree(G      : in Graph;
                                  Source : in Vertex_Type) return Graph;


   -- Returns the shortest path form source to Destination using the
   -- Dijkstra's algorithm. Raises Constraint_Error if
   -- Source or Destination is not a vertex of G.
   function Shortest_Path(G           : in Graph;
                          Source      : in Vertex_Type;
                          Destination : in Vertex_Type) return Vertex_Array;
end Graph_Algorithms;
```

Mini-Project 2: Graph Algorithms Tests (4 Points)
Consider reasonable test cases for the graph algorithms package specification. Think about equivalence classes, limits, invariants etc. Then write a `testgen` "test driver" and justify each test. Finally, convince your fellow students and lecturer that your test-driver is a good one.

Mini-Project 3: Let's SPARK the Correct Ballot Box. (4 Points)
Replace the Ada'12 pre- and post-conditions with SPARK annotations to allow a complete

- exception freeness,
- data flow, and
- information flow analysis

using the SPARK `Examiner`

```
package Correct_Ballot_Box is

   Number_Of_Options: constant Positive := 10;
   -- parties / candidates to choose from

   Number_Of_Voters: constant Positive := 1000;

   subtype Options is Positive range 1 .. Number_Of_Options;
   subtype Voters  is Natural  range 0 .. Number_Of_Voters;

   Remaining_Voters: Voters := Voters'Last;
   -- number of people who did not yet vote

   Votes: array(Options) of Voters := (others => 0);

   function Is_Empty return Boolean is
      ((Remaining_Voters = Voters'Last)
       and (for all I in Options => Votes(I) = 0));

   procedure Vote_For(Vote: Options) with
     Pre  => Remaining_Voters > 0,
     Post => (Remaining_Voters = Remaining_Voters'Old-1) and
       (for all I in Options => (if I = Vote
                                 then Votes(I) = Votes'Old(I) +1
                                 else Votes(I) = Votes'Old(I)));

end Correct_Ballot_Box;
```