Problem Set - Bonus Edition
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`

**Mini-Project 1: SPARK-Proof The Gaussian sum formula (4 Points)**
Write a program in SPARK that computes the Gaussian sum formula

$$1 + 2 + 3 + ... + n = \sum_{i=1}^{n} \frac{n(n+1)}{2}.$$

Make use of the data type `Positive`. The goal of this task is to proof the correctness of your program using the SPARK proof checker. Your solution must include beside the SPARK package (`ads` and `adb` file) the following ones: the proof log (`plg` file), the proof reviews (`prv` file), and the proof summary (`sum` file). Below you will find a few helpful hints.

- Up to which value of $n$ is the Gaussian sum computed correctly?
- Specify a new subtype of `Positive`, according to your observation, and use this subtype as the in-parameter of your subprogram.

**Mini-Project 2: SPARK-Proof Faculty (4 Points)**
Write a program in SPARK that computes the faculty of n. Make use of the data type `Positive`. The goal of this task is to proof the correctness of your program using the SPARK proof checker. Your solution must include beside the SPARK package (`ads` and `adb` file) the following ones: the proof log (`plg` file), the proof reviews (`prv` file), and the proof summary (`sum` file). Below you will find a few helpful hints.

- Up to which value of $n$ is $n!$ computed correctly?
- Specify a new subtype of `Positive`, according to your observation, and use this subtype as the in-parameter of your subprogram.
- You can proof the correctness by using a proof function and the SPARK simplifier. Do not forget that you need a `cfg`-file for run-time checks.

**Mini-Project 3: Whitebox testing: Statement coverage (4 Points)**
Consider reasonable test cases for the vectors package, and then write a `testgen` test driver for that package. Use `gcov` to ensure that all statements of the package are covered by your test. Achieve a perfect score by covering all lines of code (100 percent statement coverage.) Your solution must include beside your test driver the gcov counting (`.gcov` file).

```
package Vectors is
   type Float_Vector is array (Positive range <>) of Float;
```

```
   procedure Vector_Put (X : Float_Vector) ;

   -- cross product
   function "*" (Left, Right : Float_Vector) return Float_Vector;

    -- scalar product
   function "*" (Left, Right : Float_Vector) return Float;

   -- stretching
   function "*" (Left : Float_Vector; Right : Float) return Float_Vector;
end Vectors;

with Ada.Text_IO;

package body Vectors is
   package Float_IO is new Ada.Text_IO.Float_IO (Float);

   procedure Vector_Put (X : Float_Vector) is
   begin
      Ada.Text_IO.Put ("(");
      for I in X'Range loop
         Float_IO.Put (X (I), Aft => 1, Exp => 0);
         if I /= X'Last then
            Ada.Text_IO.Put (", ");
         end if;
      end loop;
      Ada.Text_IO.Put (")");
   end Vector_Put;

   -- cross product
   function "*" (Left, Right : Float_Vector) return Float_Vector is
   begin
      if Left'Length /= Right'Length then
         raise Constraint_Error with
           "vectors of different size in dot product";
      end if;
      if Left'Length /= 3 then
         raise Constraint_Error with
           "dot product only implemented for R**3";
      end if;
      return Float_Vector'(Left (Left'First + 1) * Right (Right'First + 2) -
                             Left (Left'First + 2) * Right (Right'First + 1),
                           Left (Left'First + 2) * Right (Right'First) -
                             Left (Left'First) * Right (Right'First + 2),
                           Left (Left'First) * Right (Right'First + 1) -
                             Left (Left'First + 1) * Right (Right'First));
   end "*";
```

```ada
   -- scalar product
   function "*" (Left, Right : Float_Vector) return Float is
      Result : Float := 0.0;
      I, J : Positive;
   begin
      if Left'Length /= Right'Length then
         raise Constraint_Error with
           "vectors of different size in scalar product";
      end if;
      I := Left'First; J := Right'First;
      while I <= Left'Last and then J <= Right'Last loop
         Result := Result + Left (I) * Right (J);
         I := I + 1; J := J + 1;
      end loop;
      return Result;
   end "*";

   -- stretching
   function "*" (Left : Float_Vector; Right : Float) return Float_Vector is
      Result : Float_Vector (Left'Range);
   begin
      for I in Left'Range loop
         Result (I) := Left (I) * Right;
      end loop;
      return Result;
   end "*";
end Vectors;
```

**Mini-Project 4: Parallel-Mini-RC4 key extraction (4 Points)**

Write a program that computes the Key for a given Mini-RC4 keystream. The program takes one command-line arguments, the keystream represented as a hex string. Your implementation should use **four tasks** to compute the key in parallel. Furthermore, pressing the "q" key should immediately quit the program. This can be realized using the procedure `Ada.Text_IO.Get_Immediate`. The program – if not interrupted – should output a candidate key represented as hex string.

```ada
package Mini_RC4 is
   type Byte is mod 2**8;
   for Byte'Size use 8;
   type Byte_Array is array (Natural range <>) of Byte;

   subtype Key_Type is Byte_Array(0..3);

   type Context_Type is private;

   procedure Key_Scheduler(Key : in Key_Type; Ctx : out Context_Type);
```

```
   procedure Get_Keystream(Ctx : in out Context_Type;
                           Keystream : out Byte_Array);

private
   subtype Expanded_Key_Type is Byte_Array(0..255);

   type Context_Type is record
      S : Expanded_Key_Type;
   end record;
end Mini_RC4;

package body Mini_RC4 is
   procedure Swap(Ctx : in out Context_Type; I, J : Integer) is
      T : Byte;
   begin
      T := Ctx.S(I);
      Ctx.S(I) := Ctx.S(J);
      Ctx.S(I) := T;
   end Swap;

   procedure Key_Scheduler(Key : in Key_Type; Ctx : out Context_Type) is
      J : Byte := 0;
   begin
      for I in  Expanded_Key_Type'Range loop
         Ctx.S(I) := Byte(I);
      end loop;

      for I  in  Expanded_Key_Type'Range loop
         J := J +
           Ctx.S(I) +
           Key((I mod (Key_Type'Last+1)));
         Swap(Ctx,I,Integer(J));
      end loop;
   end Key_Scheduler;

   procedure Get_Keystream(Ctx : in out Context_Type;
                           Keystream : out Byte_Array) is
      I : Byte := 0;
      J : Byte := 0;
   begin
      for K in Keystream'Range loop
         I := I + 1;
         J := J + Ctx.S(Integer(I));
         Swap(Ctx, Integer(I), Integer(J));
         Keystream(K) := Ctx.S( Integer( (Ctx.S(Integer(I))
                                         + Ctx.S(Integer(J))) ));
      end loop;
```

```
   end Get_Keystream;
end Mini_RC4;
```