Problem Set 4
Course **Security Engineering**
(Winter Term 2018)

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

Due Date: 07 Dec 2018, 1:30 PM, via email to
`nathalie.jolanthe.dittrich(at)uni-weimar.de`.

Goals: Getting familiar with test coverage and Hoare logic.

Task 1 – Introduction (No Credits)

Read Chapters 14 - 18 of John English. Inform yourself about **`gcov`** and **`lcov`**:
Quick-start guide for **`gcov`**, More information on **`gcov`**, About **`lcov`**

Task 2 – Mini Project – Testing and Coverage (2+2 Credits)

Examine your test coverage with **`gcov`** and **`lcov`**, if necessary, add tests, to achieve at least 95% line-coverage of your implementations:

a) For your **`vectors`** package.

b) For your **`coffee_machine`** package.

Task 3 – Mini Project – Data Flow and Hoare Logic (5 Credits)

Given the following function **`F`**:

```
function F(N: Natural) return Natural is
    I: Natural := 0;
    X: Natural := 1;
begin
    while I < N loop
        I := I + 1;
        X := X * I;
    end loop;

    return X;
end F;
```

a) Add the correct data-flow annotations (**`def`**, **`p-use`**, **`c-use`**).

b) Derive appropriate pre- and post-conditions, loop variant, and invariant.

c) Use Hoare logic to show its *total* correctness. Denote the known statements at every step, and denote the rules (assignment, condition, conclusion, etc.) you used.

d) **Bonus +1:** Prove the correctness (with statements, rules) and give appropriate pre-/post-conditions also for the function **`G`**.

```ada
function G(N: Natural; K: Natural) return Natural is
    X: Natural;
    Y: Natural;
    Z: Natural;
begin
    X := F(K);
    Y := F(N-K);
    Z := F(N);
    return Z / (X * Y);
end G;
```

Task 4 – Mini-Project – Hoare Logic (4 Credits)

Use Hoare logic to show the *total* correctness of the following code. Derive appropriate pre- and post-conditions, loop variant, and invariant.

```ada
function Fibonacci (N : Natural) return Natural is
   A : Natural := 0;
   B : Natural := 1;
   Sum  : Natural;
begin
   for I in 1..N loop
      Sum := A + B;
      B   := A;
      A   := Sum;
   end loop;
   return A;
end Fibonacci;
```

Task 5 – Mini-Project – Binary Tree (5 Credits)

Inform yourselves about **Access Types** in Ada. Implement the following specification of the packages **Binary_Tree**.

```ada
with Ada.Unchecked_Deallocation;

generic
    type Item_Type is private;
    with function "<"(Left, Right: Item_Type) return Boolean;
    with function "="(Left, Right: Item_Type) return Boolean;
    with procedure Put_Item(Item: Item_Type);
package Binary_Trees is
    Item_Already_In_Tree_Exception: exception;
    Item_Not_Found_Exception: exception;

    type Binary_Tree;
    type Binary_Tree_Access is access all Binary_Tree;
    type Binary_Tree is record
        Item   : Item_Type;
        Left   : Binary_Tree_Access := Null;
        Right  : Binary_Tree_Access := Null;
        Parent : Binary_Tree_Access := Null;
    end record;

    procedure Add_Item(T: in out Binary_Tree_Access; Item: Item_Type);
    -- Adds Item as a leaf node in the tree T at the correct location.
    -- Raises an Item_Already_In_Tree_Exception if Item already is
    -- in the tree T.
    function Create(Item: Item_Type) return Binary_Tree_Access;
    -- Creates a new tree with a single node that contains the given Item.
    function Has_Children(T: Binary_Tree_Access) return Boolean;
```

```ada
    -- Returns True if the tree T possesses any children; False otherwise.
    function Has_Item(T: Binary_Tree_Access; Item: Item_Type) return Boolean;
    -- Returns True if the tree T contains the Item; False otherwise.
    function Get_Height(T: Binary_Tree_Access) return Natural;
    -- Returns the height of the tree T.
    function Get_Num_Leaves(T: Binary_Tree_Access) return Natural;
    -- Returns the number of leaves of the tree T.
    procedure Put(T: Binary_Tree_Access);
    -- Prints all items of the tree T in-order using Put_Item.
    procedure Remove_Item(T: in out Binary_Tree_Access; Item: Item_Type);
    -- Removes the node -- and only that node -- which contains the item
    -- from the tree T. Raises an Item_Not_Found_Exception if the item is not
    -- in the tree T. Frees the memory for the node.
    procedure Remove_All(T: in out Binary_Tree_Access);
    -- Removes all nodes from the tree T and frees their memory.
private
    procedure Free is new Ada.Unchecked_Deallocation(
        Binary_Tree,
        Binary_Tree_Access
    );
end Binary_Trees;
```