Problem Set 3
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: 23 Nov 2018, 1:30 PM, via email to
`nathalie.jolanthe.dittrich(at)uni-weimar.de`.

Goals: Testing Ada code with testgen/AUnit and Ada2012 Pre- and Post-Conditions.

Task 1 – Introduction (No Credits)
Read Chapters 9 - 13 of John English.

Task 2 – Testing (No Credits)
If not already done, read up on the testing frameworks **`testgen`** and **`AUnit`**:

a) **testgen:** A small, easy-to-learn tool that allows to quickly write a battery of tests called *test driver* - for any Ada packages. Note that **`testgen`** depends on **`tg`** (by André Spiegel) which can also be found here. **`testgen`** can be found here.

b) **AUnit:** The standard testing framework for Ada; read and understand the implementation and the design of test cases, test suites, and test harnesses.

Task 3 – Mini Project 1 – Vectors (4 Credits)
Use **`testgen`** *and* **`AUnit`** to write a test suite for the Vectors package from the second problem set. Test at least all functions from the specification excep t for the output functions.

Task 4 – Mini Project 2 – Coffee Machine (4 Credits)
Implement the following specification and use **`testgen`** *or* **`AUnit`** to write a test suite:

```
1  package Coffee_Machine is
2
3    -- Simulation of a coin-driven coffee machine
4    -- User: - One slot to insert coins (only 10 or 20 cents)
5    --      - One button to press ("money back")
6    -- Machine: one slot to drop coins, the coffee output
7    -- Given 30 cents or more, the coffee is produced immediately
8    -- (Note that Overspending is Possible)
9
10   type State is private;
11   type Action is (Ten_Cent, Twenty_Cent, Button);
12   type Reaction is (Nothing, Drop_All_Coins, Coffee);
13
14   procedure Initialize (X : out State);
15   procedure X(S      : in out State;
16         Act    : in Action;
17         React    : out Reaction);
18
```

```
19    private
20       type State is range 0..2;
21
22 end Coffee_Machine;
```

Task 5 – Mini Project 3 – Implementing a Block Cipher (5 Credits)
In June 2013 the U.S. National Security Agency published the SIMON family of block ciphers on eprint: `http://eprint.iacr.org/2013/404`.

Read and understand the proposed specification of SIMON. Implement the version with 32-bit state size. Use the following specification:

```
1  package Simon32 is
2      type Byte is mod 2**8;
3      type Word is mod 2**16;
4
5      for Byte'Size use 8;
6      for Word'Size use 16;
7
8      type Bytes is array (Integer range <>) of Byte;
9      type Words is array (Integer range <>) of Word;
10
11     type Block_32 is new Bytes(0..3);
12     type Block_64 is new Bytes(0..7);
13
14     Cipher_Not_Initialized_Exception: exception;
15
16     function Decrypt(Ciphertext: in Block_32) return Block_32;
17     -- Decrypts the given ciphertext block and returns the corresponding
18     -- plaintext block. Requires that a key was given by calling Prepare_Key
19     -- before; raises a Cipher_Not_Initialized_Exception otherwise.
20     function Encrypt(Plaintext: in Block_32) return Block_32;
21     -- Encrypts the given plaintext block and returns the corresponding
22     -- ciphertext block. Requires that a key was given by calling Prepare_Key
23     -- before; raises a Cipher_Not_Initialized_Exception otherwise.
24     procedure Prepare_Key(Key: in Block_64);
25     -- Generates the round keys from the given cipher key.
26     -- Must be invoked before any en- or decryption can happen.
27 private
28     Num_Rounds: constant Positive := 32;
29     Num_State_Words: constant Positive := 2;
30
31     type Key_Type is array(1..Num_Rounds) of Word;
32
33     Round_Keys: Key_Type;
34 end Simon32;
35
```

Use either **`testgen`** *or* **`AUnit`** to write a test driver for your implementation of SIMON-32; you can use the official test vectors from the publication.

Task 6 – Mini Project 4 – Inheritance and White-Box Testing (6 Credits)
The package **`Bank_Accounts`** below was slightly modified compared to the previous problem set. Derive another bank-account type in a package **`Bank_Accounts.Overdrawable`** which allows the account to be overdrawn up to a defined limit. Define a third package **`Bank_Accounts.Fees`** which charges a fee for every withdrawal and transfer. Implement test cases for each account type using *either* **`testgen`** *or* **`AUnit`**.

```
package Bank_Accounts is
    subtype Cents_Type is Integer;
    Default_Balance: constant Cents_Type := 0;

    type Account_Type is tagged limited private;

    Overspent_Exception: exception;
    Invalid_Amount_Exception: exception;

    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.
private
  type Account_Type is tagged limited record
    Balance: Cents_Type;
  end record;
end Bank_Accounts;
```