# Problem Set 7
# 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`

**Problem 1: JE-Tasking (4 Points)**
Read Chapter 19 of JE, and solve Exercises 19.1-19.4.

**Mini-Project 1: Parallel-Hofstadter Q sequence (4 Points)** Write a program that computes the Hofstadter Q sequence. The program takes two command-line arguments. The first parameter is the length of the Hofstadter Q sequence and the second a timeout that determines the maximum lifetime of the program. Your implementation should use **four tasks** to compute the Hofstadter Q sequence in parallel. Furthermore, pressing the "q" key should immediately quit the program. This can be realised using the procedure `Ada.Text_IO.Get_Immediate`.

**Example:** `# ./hofstadter_sequence 4 2`.
The Output of the example above should be either `1, 1, 2, 3` or a real subset of `1, 1, 2, 3` if and only if the timeout is triggered before computation of the complete sequence.

**Mini-Project 2: Parallel-Merge-Sort (4 Points)** Write a program that computes a list of numbers using the merge sort algorithm. The program takes two command-line arguments. The first parameter is name of a file that contains a list of numbers and the second a timeout that determines the maximum lifetime of the program. Your implementation must use at least **two tasks** using the divide-and-conquer approach. Furthermore, pressing the "q" key should immediately quit the program. This can be realised using the procedure `Ada.Text_IO.Get_Immediate`.

**Mini-Project 3: Paralle-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 realised using the procedure `Ada.Text_IO.Get_Immediate`. The program – if not interrupted – should output a candidate key represented as hex string.

```
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;
```