Domácí úkoly -- grep

```
open System

let args = Environment.GetCommandLineArgs()
let neg = args.[1] = "-v"
let re = new Text.RegularExpressions.Regex(if neg then args.[2] else args.[1])

let rec process_line() =
  match Console.ReadLine() with
    | null -> ()
    | line -> if re.IsMatch line <> neg then Console.WriteLine line
              process_line()

process_line()
```

Domácí úkoly -- hist

```
open System

let lines = new System.Collections.Generic.Dictionary<string, int>()
let add line = if lines.ContainsKey line then lines.[line] <- lines.[line] + 1
                                         else lines.Add(line, 1)

let write_result() =
  lines |> Seq.sort_by (fun line -> -line.Value)
        |> Seq.iter (fun line -> printf "%d\t%s\n" line.Value line.Key)

let rec process_line() = match Console.ReadLine() with | null -> write_result()
                                                       | line -> add line
                                                                 process_line()
process_line()
```

Domácí úkoly -- sed

```
//#r "FSharp.Compiler.CodeDom"
open System
open System.Reflection
open System.CodeDom.Compiler

let sed src =
  let compiler = new Microsoft.FSharp.Compiler.CodeDom.FSharpCodeProvider()
  // Díky Michalovi Pavelčíkovi za následující swičík
  let cParams = new CompilerParameters(GenerateInMemory=true)
  let compile2Assembly src =
    let cResult = compiler.CompileAssemblyFromSource(cParams, [|src|])
    if cResult.Errors.HasErrors then failwith "Chyba v dané funkci"
    else cResult.CompiledAssembly

  let F =
    let src' = sprintf "#light\nnamespace FNS=type FT()=
                        static member F(s:string):string=s|>(%s)" src
    (compile2Assembly src').GetType("FNS.FT").GetMethod("F")
  let runF (input:string) = F.Invoke(null, [|box input|]) :?> string

  let rec processLine() =
    match Console.ReadLine() with | null -> ()
                                  | line -> line |> runF |> Console.WriteLine
                                            processLine()
  processLine()

let args = System.Environment.GetCommandLineArgs()
if args.Length > 1 then sed args.[1]
                   else printfn "Usage: sed fsharp-function"
```

Domácí úkoly -- dehash

```
let dehash f =
  let rec cycle x y = let x', y' = f x, f (f y)
                      if x' = y' then x' else cycle x' y'
  let rec steps n x y = let x', y' = f x, f y
                        if x' = y' then (n+1) else steps (n+1) x' y'
  let num_values = steps 0 0 (cycle 0 0)
  let block_len = num_values |> float |> sqrt |> int
  let num_blocks = num_values / block_len

  let fblock x = let rec f' x = function | 0 -> x
                                         | n -> f' (f x) (n-1)
                 f' x block_len

  let blocks = seq {1..num_values} |> Seq.scan (fun x _ -> fblock x) 0
  let blocks = Seq.zip (Seq.skip 1 blocks) blocks |> dict

  fun x ->
    let rec prev_block y = if blocks.ContainsKey y then blocks.[y]
                                                   else prev_block (f y)
    let rec prev_value y = if f y = x then y else prev_value (f y)
    prev_value (prev_block x)
```

Background worker

```
System.ComponentModel.BackgroundWorker
  RunWokerAsync : [ unit | obj ] -> unit
  CancelAsync : unit -> unit
  CancellationPending : unit -> bool
  ReportProgress : int [ -> ojb ] -> unit
  events OnDoWork, OnProgressChanges, OnRunWorkerCompleted
DoWorkEventArgs má Argument, Cancel, Result
```

Thread pool

```
System.Threading.ThreadPool.QueueUserWorkItem : WailCallback [ * obj ] -> unit
type WaitCallback = delegate of obj -> unit

System.Threading.ThreadPool.RegisterWaitForSingleObject :
  WaitHandle -> WaitOrTimerCallback -> obj -> [int|TimeSpan] -> (rpt:bool)->unit
type TimerOrWaitCallback = delegate of obj * (timeOut : bool) -> unit

System.Threading.ThreadPool.Set{Min,Max}Threads : (wrk:int) * (IO:int) -> unit
```

Asynchronní výpočty

```
#r "FSharp.Powerpack.dll"
open System.IO
open System.Net
let pages = ["http://moma.org/"; "http://www.thebritishmuseum.ac.uk/"; ... ]

let AsyncFetch (url:string) = async {
  let! resp = WebRequest.Create(url).AsyncGetResponse()
  use reader = new StreamReader(resp.GetResponseStream())
  let! html = reader.AsyncReadToEnd()
  do printfn "Read %d chars from %s" html.Length url }
let work() = pages |> List.iter (AsyncFetch >> Async.Spawn)

let AsyncProcessImage i = async {
  use inStream = File.OpenRead(sprintf "image%d.in" i)
  let! pixels = inStream.AsyncRead(numPixels)
  let pixels' = TransformImage(pixels)
  use outStream = File.OpenWrite(sprintf "image%d.out" i)
  do! outStream.WriteAsync(pixels') }
let tasks = [ for i in 1..numImages] -> AsyncProcessImage i]
Async.Run (Async.Parallel tasks)
```

Co to je Async<'a>

```
Async.Run : Async<'a> -> 'a
Async.Spawn : Async<unit> -> unit
Async.SpawnThenPostBack : Async<'a> * ('a -> unit) -> unit
Async.SpawnFuture : Async<'a> -> AsyncFuture<'a> Má member Value : 'a

type AsyncBuilder with
  member Return : 'a -> Async<'a>
  member Delay : (unit -> Async<'a>) -> Async<'a>
  member Using, For, While, TryWith, TryFinally, Bind : ...
    let bindPrimA p1 f  =
      P (fun args ->
        hijack args (fun () ->
          let cont a = protect args.econt f a (fun p2 -> invokeA p2 args)
            invokeA p1 { cont=cont;ccont=args.ccont;econt=args.econt;
                         blocked=args.blocked;group=args.group }))

type Async<'a> = Async of ('a -> unit) * (exn -> unit) -> unit
Async.Primitive : ('a -> unit) * (exn -> unit) -> Async<'a>
Async.Parallel : seq<Async<'res>> -> Async<'res array>
Async.Catch : Async<'a> -> Async<Choice<'a, exn>>

let trylet f x = try Choice2_1 (f x) with exn -> Choice2_2 exn
let protect cont econt f x =
  match trylet f x with | Choice2_1 v -> cont v
                        | Choice2_2 exn -> econt exn
type System.IO.Stream with
  member this.AsyncRead (buffer, offset, count) =
    Async.Primitive (fun (cont, econt) ->
      stream.BeginRead(buffer, offset,count,
                       AsyncCallback(protect cont econt this.EndRead)
                       null) |> ignore
Async.BuildPrimitive : 'a [ * 'b [ * 'c]] *
  ('a [->'b[->'c]]->AsyncCallback->obj->IAsyncResult) * (IAsyncResult->'res) ->
  Async<'res>

let Parallel taskSeq = Async.Primitive (fun (cont, econt) ->
    let tasks = Seq.to_array taskSeq
    let cnt = ref tasks.Length
    let results = Array.zero_create tasks.Length
    tasks |> Array.iteri (fun i p ->
      Async.Spawn ( async { let! res = p
                            do results.[i] <- res
                            let n = System.Threading.Interlocked.Decrement(cnt)
                            do if n=0 then cont results }))
```

SynchronizationContext

```
System.Threading.SynchronizationContext.Current
```

metoda `Post : SendOrPostCallback * obj -> unit` asynchronně
metoda `Send : SendOrPostCallback * obj -> unit` synchronně
kde `type SendOfPostcallback = delegate of obj -> unit`

Předávání a zpracování asynchronních zpráv

```
let counter =
  MailboxProcessor.Start(fun inbox ->
    let rec loop n = async { do printfn "Have %d..." n
                             let! msg = inbox.Receive()
                             return! loop (n+msg)
                           }
    loop 0
  )

counter.Post(4)
counter.Post(3)
```

```
type internal msg = Inc of int | Fetch of AsyncReplyChannel<int> | Stop
type AsyncCounter() =
  let counter =
    MailboxProcessor.Start(fun inbox ->
      let rec loop n = async {
        let! msg = inbox.Receive()
        match msg with
          | Inc d -> return! loop (n+d)
          | Fetch chann -> chann.Reply n
                           return! loop n
          | Stop -> return ()
      }
      loop 0
    )
  member this.Inc n = counter.Post (Inc n)
  member this.Fetch() = counter.PostAndReply (Fetch)
  member this.Stop() = counter.Post (Stop)


                              Active Patterns
                              ---------------
type Complex(x, y) =
    let mutable x = x
    let mutable y = y
    member this.XY with get() = x, y and set((x', y')) = x<-x'; y<-y'
    member this.RO with get() = sqrt (x * x + y * y), atan2 y x
                   and  set((r, o)) = x <- r * cos o; y <- r * sin o
    static member fromXY xy = new Complex(xy)
    static member fromRO ro = let c = new Complex(0., 0.) in c.RO <- ro; c
let (|XY|) (c:Complex) = c.XY
let (|RO|) (c:Complex) = c.RO

let add a b = match (a, b) with
               XY (x1,y1), XY (x2,y2) -> Complex.fromXY (x1+x2, y1+y2)
let mul a b = match (a, b) with
               RO (r1,o1), RO (r2,o2) -> Complex.fromRO (r1*r2, o1+o2)


let (|Sude|Liche|) n = if n&&&1 = 0 then Sude else Liche
match 3 with | Sude -> printfn "sude"
             | Liche -> printfn "liche"


let (|Nasobek5|_|) n = if n%5 = 0 then Some (n/5) else None
match 10 with | Nasobek5 n -> printfn "10=5*%d" n
              | _ -> printf "neni nasobek 5"
let (|NasobekK|_|) k n = if n%k = 0 then Some (n/k) else None
match 21 with | NasobekK 4 n -> printfn "21=4*%d" n
              | NasobekK 3 n -> printfn "21=3*%d" n
```