Programmering og Problemløsning, 2017

Input og Output

Martin Elsman

Datalogisk Institut
Københavns Universitet
DIKU

27. November, 2017

Input og Output

Emner for i dag:

1. **Konsol input og output:**
   - Udskrivning af text.
   - Input fra konsollen.
2. **Skrivning og læsning af filer.**
   - Åbning og lukning af filer.
   - Læsning.
   - Skrivning.
3. **Regulære udtryk.**
   - Matching.
4. **Læsning fra internettet.**
   - HTTP requests.

Konsollen

Med begrebet konsol henvises der til at der, når en process startes, etableres (åbnes) tre strømme som konteksten kan benytte til at kommunikere med processen (programmet):

Standard output	`stdout`	`System.Console`
Standard input	`stdin`	`System.Console`
Standard error	`stderr`	`System.Console.Error`

Vi har tidligere set hvordan `printf` (og venner) kan benyttes til at udskrive på `stdout`.

Funktioner i `System.Console`

Udskrivning på `stdout`:

```
val Write     : string -> unit   // Write to the console
val WriteLine : string -> unit   // Write a line
```

Indlæsning fra `stdin`:

```
val Read     : unit -> int      // Read ascii character
val ReadLine : unit -> string   // Write a line
```

Bemærk: Funktionerne `Read` and `ReadLine` "blokerer" indtil data er tilgængelig.

Kommandolinieargumenter

Kommandolinieargumenter kan læses ved at etablere en hovedfunktion af typen `string array -> int` og tilføje en `EntryPoint`-attribut til funktionen:

```
[<EntryPoint>]
let main (args: string array) : int =
  for a in args do printfn "%s" a
  0                                    // status code "ok"
```

Eksempel kørsel — og generering af stand-alone eksekverbar:

```
bash-3.2$ fsharpc --nologo main.fs
bash-3.2$ mkbundle --simple -o main main.exe
bash-3.2$ ./main hi there
hi
there
bash-3.2$
```

Eksempel: Temperatur omregner (`temp.fs`)

```
open System
let fahrenheit (c:float) : unit =
  if c < -273.15 then failwith "input too small"
  else printfn "Fahrenheit: %f" (9.0/5.0*c + 32.0)

do Console.Write "Temperature in degrees Celcius: "
let s = Console.ReadLine()
do try fahrenheit(float(s)) with
     | Failure s -> Console.Error.WriteLine s
     | _ -> Console.Error.WriteLine "Expecting number"
```

Bemærk:

- Vi benytter `Console.Error.WriteLine` til udskrivning af fejlbeskeder.

Eksempel: Gentagne input (`numbers.fs`)

```
open System
let rec loop (a:float) : float =
  match Console.ReadLine() with
    | "" -> a
    | s -> loop (a+float(s))
do Console.WriteLine "Enter numbers (end with empty line):"
do try printfn "Sum: %f" (loop 0.0) with
     | _ -> Console.Error.WriteLine "Expecting numbers"
```

Bemærk:

- Vi benytter en exception "wild card handler" til at fange fejl i input.

Læsning af filer

Operationer til læsning af UTF-8 filer er tilgængelige i modulet `System.IO.File`:

```
type StreamReader = System.IO.StreamReader
    EndOfStream : bool
    Close       : unit -> unit
    ReadToEnd   : unit -> string  // incl. newlines
    ReadLine    : unit -> string  // excl. newlines
    Read        : unit -> int
val OpenText : string -> StreamReader
```

Eksempel: linier i en fil (`lines.fs`)

```
[<EntryPoint>]
let main (args:string array) : int =
  let rec loop n (r:System.IO.StreamReader) =
    if r.EndOfStream then n
    else (ignore(r.ReadLine()); loop (n+1) r)
  in if Array.length args > 0 then
       (printfn "%d" (
          loop 0 (System.IO.File.OpenText args.[0])); 0)
     else (printfn "Expects file name as argument"; 1)
```

Skrivning af filer

Operationer til skrivning af UTF-8 filer er tilgængelige i modulet `System.IO.File`:

```
type StreamWriter = System.IO.StreamWriter
   Close       : unit -> unit
   WriteLine   : string -> unit  // add newline
   Write       : string -> unit  // no newline

val CreateText : string -> StreamWriter
val AppendText : string -> StreamWriter
```

Eksempel: Fibonacci tal i en fil (`fibfile.fs`)

```
let rec fib n = if n <= 2 then 1 else fib(n-1)+fib(n-2)
let rec loop n i (w:System.IO.StreamWriter) =
  if i > n then w.Close()
  else (w.WriteLine(string(fib i)); loop n (i+1) w)
let n = loop 10 1 (System.IO.File.CreateText "out.txt")
```

Læsning og skrivning af bytes

UTF-8 karakterer har variabel vidde (en-fire bytes) hvilket gør indexing i en UTF-8 streng lineær i størrelsen på strengen.

Almindelige karakterer i UTF-8 har størrelse 8-bit og i mange tilfælde vil der derfor ikke være forskel på filer i UTF-8 format og filer i simpelt ascii-format.

F# giver mulighed for at arbejde med filer på byte-niveau (8-bits) ved brug af `System.IO.FileStream` klassen.

Regulære udtryk

Regulære udtryk er navnet for et sprog til at klassificere tekststrenge.

En relation kaldet *matching* definerer klassen af strenge specificeret ved et givent regulært udtryk (også kaldet et mønster).

Regulære udtryk i F#

Regulære udtryk er i F# understøttet gennem klassen `System.Text.RegularExpressions`:

```
type Regex = System.Text.RegularExpressions
   IsMatch     : string -> bool
   Match       : string -> Match      // For extraction
   Replace     : string * string -> string

val Regex : string -> Regex
```

Regulære udtryk (del I)

Nedenstående tabel definerer sproget for regulære udtryk p, også kaldet mønstre.

p	**Definition**
`.`	matchet af alle karakterer
c	matchet af karakteren c
`\`c	matchet af den escapede karakter c, hvor c er en af `|`, `*`, `+`, `?`, `(`, `)`, `[`, `]`, `$`, `.`, `\`, `t`, `n`, `v`, `f`, `r`
p_1p_2	matchet af strengen s hvis et prefix af s matcher p_1 og resten af s matcher p_2 (f.eks. matcher strengen `abc` mønsteret `a.c`)
p*	matchet af en streng s hvis s kan deles op i 0, 1, eller flere strenge der hver matcher p (f.eks. matcher strengene `abbbbbba` og `aa` mønsteret `ab*a`)
p+	matchet af en streng s hvis s kan deles op i 1, eller flere strenge der hver matcher p (f.eks. matcher strengen `caaab` mønsteret `ca+b`, men strengen `cb` gør ikke)

Regulære udtryk (del II)

p	**Definition**
(*p*)	matchet af strenge der matcher *p* (f.eks. matcher strengen `cababcc` mønsteret `c(ab)*cc`
$p_1 \mid p_2$	matchet af strenge der enten matcher p_1 eller p_2 (f.eks. matcher strengene `pig` og `cow` mønsteret `pig|cow`)
[*class*]	matchet af karakterer i klassen *class*. Sekvenser af karaktererne a, b, c, 1, 2, 3, 4 matcher mønsteret `[abc1-4]*`; ordningen er underordnet.
[^*class*]	matchet af karakterer der ikke er i klassen *class*.
p?	matchet af den tomme streng eller af en streng der matcher *p* (f.eks. matcher strengene `aa` and `aba` mønsteret `ab?a`, men strengen `abba` matcher ikke mønsteret `ab?a`).

Karakterklasser

En karakterklasse *class* definerer en mængde af karakterer ved at mængden listes. Det er tilladt at benytte *ranges* ved brug af bindestreg; f.eks. kan klassen `0123456789` skrives som `1-9`.

Eksempler på regulære udtryk

- `[A-Za-zÆØÅæøå]` : matches af alle karakterer i det danske alfabet.
- `[0-9][0-9]` : matches af tal bestående af to cifre, hvor begge cifre kan være nul.
- `(cow|pig)s?` : matches af de fire strenge `cow`, `cows`, `pig` og `pigs`.
- `((a|b)a)*` : matches af strengene `aa`, `ba`, `aaaa`, `baaa`, ...
- `(0|1)+` : matches af de binære tal (f.eks. `0`, `1`, `01`, `11`, `011101010`).
- `..` : matches af to vilkårlige karakterer.
- `([1-9][0-9]*)/([1-9][0-9]*)` : matches af positive brøker af hele tal (f.eks. 1/8, 32/5645 og 45/6). Bemærk at mønsteret ikke matches af brøkerne 012/54 og 1/0.
- `<html>.*</html>` : matches af HTML indhold.
- `www\.(di\.ku|diku)\.dk` : matches af de to web-adresser `www.di.ku.dk` og `www.diku.dk`.

Eksempler på regulære udtryk

Det er ganske nemt at gøre brug af regulære udtryk i F#:

```
> open System.Text;;
> let r = RegularExpressions.Regex "^[1-9][0-9]*$";;
val r : RegularExpressions.Regex = ^[1-9][0-9]*$

> r.IsMatch "2320";;
val it : bool = true

> r.IsMatch "23d20";;
val it : bool = false
```

Bemærk:

- `IsMatch(s)` undersøger om en delstreng i *s* matcher mønsteret.
- Ved at tilføje karaktererne `^` samt `$` først og sidst i mønsteret er det hele strengen der betragtes.

Udtrækning af data med regulære udtryk

Metoden Match på et Regex object giver mulighed for at udtrække data fra en større streng:

```
type Match = System.Text.RegularExpressions.Match
   Success     : bool
   Groups      : Group seq
   NextMatch   : unit -> Match
```

Eksempel:

```
open System.Text.RegularExpressions
let extract (r:Regex) (s:string) : string option =
  let m = r.Match s
  in if m.Success then Some (string(m.Groups.[1]))
     else None
let r = Regex "is ([1-9][0-9]*) years"
let text = "Hans is 34 years old"
do printfn "%A" (extract r text)
```

Læsning fra internettet

Det er muligt at foretage HTTP requests fra F# programmer til web-servere på internettet.

Hjælpefunktion:

```
open System.Net
open System
open System.IO

let fetchUrl (url:string) : string =
  let req = WebRequest.Create(Uri(url))    // make request
  use resp = req.GetResponse()
  use stream = resp.GetResponseStream()
  use reader = new IO.StreamReader(stream)
  in reader.ReadToEnd()
```

Bemærk:

- Den specielle F# **use**-binding sikrer at resourcer der er knyttet til "disposible" objekter automatisk bliver frigivet når scope forlades.

"Scraping" af valutakurser

Hvis vi kan hente en vilkårlig internetside kan vi benytte denne feature til, f.eks., at hente valutakurser.

Her følger kildekoden for et udsnit af `valutakurser.dk`'s hjemmeside:

```
<div id="cdditem_1_EUR" ... val="744.2100" ...>
  <img src="http://i.euroinvestor.com/flags/blank.png" .../>
  <span>Euro</span>
</div>
```

Regulært udtryk til at udtrække værdien

```
let floatregex = "([1-9][0-9]*([.][0-9]+)?)"
let regex = "EUR.*val\\=\\\"" + floatregex + "\\\""
```

"Scraping" af valutakurser – Implementation (`fetchfx.fs`)

```
// Find FX rate for a particular currency
let findfx cur s =
  let floatregex = "([1-9][0-9]*([.][0-9]+)?)"
  let regex = cur + ".*val\\=\\\"" + floatregex + "\\\""
  in match extract (Regex regex) s with
      | Some v -> v
      | None -> "error"

do printf "Enter currency (CUR): "
let cur = System.Console.ReadLine()
let cur_regex = Regex "USD|EUR|DKK|CHF|SEK|NOK|GBP|AUD|JPY|CAD"

if cur_regex.IsMatch cur then
  let s = fetchUrl "http://www.valutakurser.dk"
  let fx = findfx cur s
  do printfn "%sDKK=%s" cur fx
else
  do printfn "Currency '%s' not supported" cur
```

"Scraping" af valutakurser — Brug af løsningen

```
bash-3.2$ fsharpc --nologo fetchfx.fs
bash-3.2$ mono fetchfx.exe
Enter currency (CUR): USD
USDDKK=626.6000
bash-3.2$ mono fetchfx.exe
Enter currency (CUR): CAD
CADDKK=493.0800
bash-3.2$
```

Spørgsmål: Virker programmet efter hensigten på alle slags input?

Bemærk:

- Dette eksempel er kun vist som et eksempel på hvad der er teknisk muligt.
- Der kan være juridiske problemer med at "scrape" data fra web-sites...