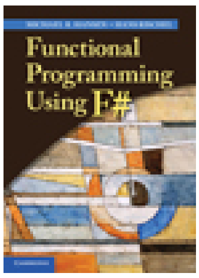

Functional Programming Using F#

Michael R. Hansen, Hans Rischel

Book DOI: http://dx.doi.org/10.1017/CBO9781139093996

Online ISBN: 9781139093996

Hardback ISBN: 9781107019027

Paperback ISBN: 9781107684065

Chapter

Appendix A - Programs from the keyword example pp. 339-345

Chapter DOI: http://dx.doi.org/10.1017/CBO9781139093996.015

Cambridge University Press

Appendix A

Programs from the keyword example

This appendix provides complete programs of the keyword program from the Chapter 10. It consists of

- a section introducing the basic HTML concepts,
- a section containing the complete `IndexGen` program, and
- a section containing the complete `NextLevelRefs` program.

The remaining program for the keyword example: `MakeWebCat`, appears in Table 10.19. The source can also be found on the homepage of the book.

A.1 Web source files

The source of a web-page is a file encoded in the HTML (Hyper Text Mark-up Language) format. This section gives a brief introduction to HTML using the library documentation web-page as an example.

An HTML file is an ordinary text file using a special syntax. Certain characters like `<`, `>`, `&` and `"` are delimiters defining the syntactical structure. The file consists of *elements* of the form <. . . > intermixed with text to be displayed. The following construction will for instance make a button with a link to a web-page:

```
<a href="http://msdn.microsoft.com/en-us/library/ee353439.aspx">
active pattern</a>
```

The text:

```
active pattern
```

is displayed in the button and a click will cause the browser to select the web-page given by the URI:

```
http://msdn.microsoft.com/en-us/library/ee353439.aspx
```

A link is hence defined by a pair of elements: a *start element* `<a...>` and an *end element* `</a>` surrounding the text to be displayed. The construction:

```
href="..."
```

defines the `href` *attribute* of the element `<a...>`. Attributes have special uses and are not displayed text.

Elements in HTML appear in pairs of start and end elements, and some elements may contain attributes. The line break element `<br />` is considered a (degenerated) pair of start and end element `<br></br>`.

Text to be displayed is *encoded* in the HTML encoding with certain characters *encoded* using HTML *escape sequences* like `&lt;` and `&amp;` encoding `<` and `&`. The internet browser performs the corresponding *decoding* when displaying a text.

The HTML notation has developed over time and web-pages around the world follow different standards. The standard is now controlled by the World Wide Web Consortium (W3C) and more recent standards define HTML as a specialization of the XML notation.

The HTML-source of the library keyword index page starts with the *Document type definition* that is an XML `<!DOCTYPE...>` element:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

The start of the HTML part is signalled by:

```
<html>
```

The heading starts with the *title* to be displayed on the boundary of the browser window:

```
<head>
<title>F# Program Library Documentation Keyword Index</title>
```

It is followed by the *style* section:

```
<style type = "text/css">
h1 {color: purple; font-size: x-large; font-family: Verdana}
p  {font-family: Verdana; font-size: large; color: maroon}
a  {font-family: Verdana; text-decoration: none;
    font-size: medium}
</style>
</head>
```

This section defines the appearance of different parts of the web-page:

`h1`: Level 1 heading in purple with x-large Verdana font.
`p`: Paragraphs in large Verdana font in maroon colour
`a`: Links in medium-sized Verdana font without the default underlining.

The reader may consult a Cascading Style Sheet (CSS) manual for further information about styles in HTML.

The *body* starts with a level 1 heading `<h1>...</h1>` and a paragraph `<p>...</p>`:

```
<body>
<h1>F# Program Library Documentation Keyword Index</h1>
<p>Version date: Saturday, August 27, 2011</p>
```

Each link is followed by a line break `<br />`:

```
<a href="http://msdn.microsoft.com/en-us/library/ee353439.aspx">
active pattern</a><br />
```

A empty keyword line is just an extra line break:

```
<br />
```

Document body and entire HTML document are terminated by end elements:

```
</body></html>
```

HTML-sources may contain links with an abbreviated reference. The web-page with URI:

```
http://msdn.microsoft.com/en-us/library/ee353439.aspx
```

do for instance contain a link with a *path* instead of a full URI:

```
href="/en-us/library/ee370230"
```

This path is interpreted relative to the *base URI*:

```
http://msdn.microsoft.com/
```

as pointing to the web-page with URI:

```
http://msdn.microsoft.com/en-us/library/ee370230
```

The reader may consult an HTML (or XHTML) manual for further information.

A.2 The `IndexGen` program

This section contains the complete `IndexGen` program. For the documentation of the program, we refer to Section 10.8. The source code is split into an input and an output parts, that are shown in following the two tables.

```
open System;;
open System.IO;;
open System.Globalization;;
open System.Text.RegularExpressions;;
open Microsoft.FSharp.Collections;;
open System.Web;;
open TextProcessing;;

// Input part

type resType = | KeywData of string * string list
               | Comment
               | SyntError of string;;
let reg = Regex @"\G\s*\042([^\042]+)\042(?:\s+([^\s]+))*\s*$";;
let comReg = Regex @"(?:\G\s*$)|(?:\G//)";;
let tildeReg = Regex @"~";;
let tildeReplace str = tildeReg.Replace(str," ");;
let getData str =
  let m = reg.Match str
  if m.Success then
    KeywData(captureSingle m 1,
             List.map tildeReplace (captureList m 2))
  else let m = comReg.Match str
       if m.Success then Comment
       else SyntError str;;

let enString = orderString "en-US";;

let keyWdIn() =
  let webCat = restoreValue "webCat.bin"
  let handleLine (keywSet: Set<orderString*string>) str =
    match getData str with
    | Comment          -> keywSet
    | SyntError str    -> failwith ("SyntaxError: " + str)
    | KeywData (_,[]) -> keywSet
    | KeywData (title,keywL) ->
        let uri = Map.find title webCat
        let addKeywd kws kw = Set.add (enString kw, uri) kws
        List.fold addKeywd keywSet keywL
  let keyWdSet = Set.empty<orderString*string>
  fileFold handleLine keyWdSet "keywords.txt";;
```

Table A.1 *The* `IndexGen` *program: Input part*

```
// Output part

let preamble =
  "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01//EN\"
   \"http://www.w3.org/TR/html4/strict.dtd\">
<html>
<head>
<title>F# Program Library Documentation Keyword Index</title>
<style type = \"text/css\">
h1 color: purple; font-size: x-large; font-family: Verdana
p  font-family: Verdana; font-size: large; color: maroon
a  font-family: Verdana; text-decoration: none;
    font-size: medium
</style>
</head>
<body>
<h1>F# Program Library Documentation Keyword Index</h1>
<p>Version date: "
   + (String.Format(CultureInfo "en-US","0:D",DateTime.Now))
   + "</p>" ;;

let postamble = "</body></html>" ;;

let webOut(keyWdSet) =
  use webPage = File.CreateText "index.html"
  let outAct oldChar (orderKwd: orderString,uri: string) =
    let keyword = string orderKwd
    let newChar = keyword.[0]
    if (Char.ToLower newChar <> Char.ToLower oldChar
       && Char.IsLetter newChar)
    then webPage.WriteLine "<br />"
    else ()
    webPage.Write "<a href=\""
    webPage.Write uri
    webPage.WriteLine "\">"
    webPage.Write (HttpUtility.HtmlEncode keyword)
    webPage.WriteLine "</a><br />"
    newChar

  webPage.WriteLine preamble
  Set.fold outAct 'a' keyWdSet |> ignore
  webPage.Close()

[<EntryPoint>]
let main (param: string[]) =
  let keyWdSet = keyWdIn()
  webOut keyWdSet
  0;;
```

Table A.2 *The* `IndexGen` *program: Output part*

A.3 The `NextLevelRefs` program

This section contains the complete `NextLevelRefs` program. For the documentation of the program, we refer to Section 10.9.

```
open System     ;;
open System.IO  ;;
open System.Net ;;
open System.Collections.Generic ;;
open System.Text.RegularExpressions ;;
open System.Web ;;
open System.Xml ;;
open TextProcessing ;;

type infoType = StartInfo of int | EndDiv of int
                | RefInfo of string * string | EndOfFile;;

let rec nextInfo(r:XmlReader) =
  match r.Read() with
  | false -> EndOfFile
  | _     ->
  match r.NodeType with
  | XmlNodeType.Element ->
    match r.Name with
    | "div" when (r.GetAttribute "class" =
                     "toclevel2 children")
          -> StartInfo (r.Depth)
    | "a" -> let path = r.GetAttribute "href"
             ignore(r.Read())
             RefInfo(r.Value,path)
    |  _  -> nextInfo r
  | XmlNodeType.EndElement when r.Name = "div"
          -> EndDiv (r.Depth)
  |  _    -> nextInfo r ;;

let rec anyRefs(r:XmlReader) =
  match nextInfo r with
  | StartInfo n -> Some n
  | EndOfFile   -> None
  | _           -> anyRefs r ;;

let regQuote = Regex @"\042"  ;;
let quoteReplace str = regQuote.Replace(str,"’") ;;
let cStr s = quoteReplace(HttpUtility.HtmlDecode s) ;;
let getWEBrefs(uri: string) =
  let baseUri = Uri uri
  let webCl = new WebClient()
  let doc = webCl.DownloadString baseUri
  use docRd  = new StringReader(doc)
  let settings =
      XmlReaderSettings(DtdProcessing = DtdProcessing.Ignore)
  use reader = XmlReader.Create(docRd,settings)
  let rec getRefs(n) =
    match nextInfo reader with
    | RefInfo(t,path) ->
      let pathUri = Uri(baseUri,path)
      (cStr t, pathUri.AbsoluteUri) :: getRefs(n)
    | EndDiv m       ->
      if m <= n then [] else getRefs n
    | p              -> failwith ("getRefs error: " + (string p))
  match anyRefs reader with
  | None   -> []
  | Some n -> getRefs n

open System ;;
open System.IO ;;

let outputRef (output:StreamWriter) (title:string, uri:string) =
  output.WriteLine title
  output.WriteLine uri;;

let expandUri (output:StreamWriter) uri =
  let lst = getWEBrefs uri
  List.iter (outputRef output) lst  ;;

let handleLinePair (output:StreamWriter) (rdr: StreamReader) =
  ignore(rdr.ReadLine())
  expandUri output (rdr.ReadLine()) ;;

[<EntryPoint>]
let main (args: string[]) =
  if Array.length args < 2 then
    failwith "Missing parameters"
  else
  if File.Exists args.[1] then
    failwith "Existing output file"
  else
    use output = File.CreateText args.[1]
    fileXiter (handleLinePair output) args.[0]
    output.Close()
    0 ;;
```

Table A.3 *The* `NextLevelRefs` *program*