Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 9 - gruppeopgave

Jon Sporring

Rettet udgave
23. november - 28. november.
Afleveringsfrist: lørdag d. 28. november kl. 22.00.

It computerprogram kan kommunikere med omverdenen på mange måder. Programmet kan f.eks. læse og skrive til filsystemet, læse fra tastaturet og skrive til skærmen, eller føre en dialog med andre computere via en protokol såsom html over internettet. Processen bliver ofte kaldet input/output eller blot I/O. I I/O opstår der ofte uventede begivenheder, såsom brugeren indtaster forkert information, en ønsket fil findes ikke, eller netværket går ned, og en typisk måde at håndtere sådanne uventede begivenheder er via et undtagelsessystem også kaldet exception. En exception bliver kastet og grebet (cast and handled) og F# (og de fleste andre sprog) har faste strukturer til at kaste og gribe undtagelser. Undtagelser kan også bruges til andre (uventede) begivenheder, såsom indicering af strenge og lister udenfor det lovlige indiceringsinterval, men for disse tilfælde findes der ofte mere elegante fejlhåndteringsmekanismer. Alt dette er temaet for denne arbejdsseddel.

Emnerne for denne arbejdsseddel er:

- at kunne håndtere fejlsituationer med undtagelser (exceptions) og option typen,
- at kunne læse fra tastaturet og fra kommandolinjen og skrive til skærmen,
- at kunne læse til og skrive fra filer,
- at kunne læse fra internettet.

Opgaverne er opdelt i øve- og afleveringsopgaver. I denne periode skal I arbejde i grupper med jeres afleveringsopgaver. Regler for gruppe- og individuelle afleveringsopgaver er beskrevet i "'Noter, links, software m.m."'→"'Generel information om opgaver"'.

Øveopgaver (in English)

9ø0 Implement the faculty function $n! = \prod_{i=1}^{n} i$, $n > 0$ as `fac : n:int -> int`. The function must cast a `System.ArgumentException` exception, if the function is called with $n < 1$. Call `fac` with the values $n = -4, 0, 1, 4$, and catch possible exceptions.

9ø1 Add a new and selfdefined exception `ArgumentTooBig of string` to `fac` in Assignment 9ø0, and cast it with the argument `"calculation would result in an overflow"`, when n is too large for the `int` type. Call the function with a small and a large value of n, catch the possible exception and handle it in case by writing the exception message to the screen.

9ø2 Make a new faculty function `facFailwith : n:int -> int`, as `fac` in Assignment 9ø1, but where the 2 exceptions are replaced with `failwith` with the arguments `"argument must be greater than 0"` and `"calculation would result in an overflow"` respectively. Call `facFailWith` with $n = -4, 0, 1, 4$, catch possible exceptions with the `Failure` pattern, and write the returned message from `failwith` to the screen.

9ø3 Write a new faculty function as in Assignment 9ø1 but with the name and type `facOption : n:int -> int option`, which returns `Some m`, when the result is computable and `None` otherwise. Call `fac` with the values $n = -4, 0, 1, 4$, and write the result to the screen.

9ø4 Write a program `myFirstCommandLineArg` which takes an arbitrary number of arguments from the command line and writes each argument as, e.g.,

```
$ mono myFirstCommandLineArg.exe a sequence of args
4 arguments received:
0: "a"
1: "sequence"
2: "of"
3: "args"
```

The program must exit with the status value 0.

9ø5 Make a program `myFirstReadKey` which continuously reads from the keyboard using the `System.Console.ReadKey()` function. The following key-presses must result in the following:

'a' writes "left" to the screen
's' writes "right" to the screen
'w' writes "up" to the screen
'z' writes "down" to the screen
shift+'q' quits the program

All other key-presses must be ignored. When the program exits, the exit status must be 0.

9ø6 Make a program `myFirstReadFile` which

(a) opens the text file "myFirstReadFile.fsx" as a stream using the `System.IO.File.OpenText` function,

(b) reads each individual character using the `System.IO.StreamReader.Read` function,

(c) writes each character to the screen using the `printf` function, and

(d) closes the stream using `System.IO.FileStream.Close`.

The *program's* exit status must be 1 in case of error and 0 otherwise.

9ø7 Make a program `myFirstWriteFile` which

(a) opens a new text file "newFile.txt" as a stream using the `System.IO.File.CreateText` function,

(b) writes the characters 'a' ... 'z' one at a time to the file using `System.IO.StreamReader.Write`, and

(c) closes the stream using the `System.IO.FileStream.Close` function.

The program's exit status must be 1 or 0 depending on whether there was an error or not when running the program.

9ø8 Write a function,

```
filenameDialogue : question:string -> string
```

which initiates a dialog with the user using the `question`. The function should return the filename the user inputs as a string. If the user wishes to abort dialogue, then the user should input an empty string.

9ø9 Make a program with the function,

```
printFile : unit -> unit
```

which initiates a dialogue with the user using `filenameDilaogue` from Exercise 9ø8. The function must ask the user for the name of a file, and if it exists, then the content is to be printed to the screen. The program must return 0 or 1 depending on whether the specified file exists or not.

The internet is a great source of information, and many files are published as html-files. In the following assignment(s), you are to work with html-files on the internet.

Note that most internet pages requires a valid certificate before they will allow your program to access it. By default, Mono has no certificates installed. One way to install useful certificates is to use `mozroots`, which is a part of the Mono package. On Linux/MacOS you do the following from the console:

```
mozroots --import --sync
```

On Windows you type the following (on one line)

```
mono "C:\Program Files (x86)\Mono\lib\mono\4.5\mozroots.exe" --import
--sync
```

Note that your installation of `mozroots` may be in a different path, and you may have to adapt the above path to your installation. After running the above, your program should be able to read most pages without being rejected.

9ø10 Make a program with the function,

```
printWebPage : url:string -> string option
```

which reads the content of the internetpage `url` and returns its content as a string option.

Afleveringsopgaver (in English)

The program `cat` is a UNIX-program, which concatenates (i.e. joins) files. The program exists on both Linux and macOS. When passing two text files to `cat`, e.g. `a.txt` and `b.txt`, then the program prints the contents of file `a.txt` followed by the contents of `b.txt` to the screen. Consider an inverse version of `cat`, called `tac`, which prints the files in reverse order and prints each file from the last to the first character. For example, if the file `a.txt` contains the characters `abc\ndef\n` and the file `b.txt` contains the characters `123\n456\n` with `\n` being the newline character, then

```
cat a.txt b.txt
```

will output `abc\ndef\n123\n456\n` to the screen. In contrast,

```
tac a.txt b.txt
```

will output `654\n321\nfed\ncba\n` to the screen.

In the following assignments you are to write a (functional) implementation of `cat` and `tac` in F#.

9g0 Make the library `readNWrite.fs` with the function,

```
readFile : filename:string -> string option
```

which takes a filename and returns the contents of the text file as a `string option`. If the file does not exist, the function should return `None`.

9g1 First extend the library `readNWrite.fs` with a function,

```
cat : filenames:string list -> string option
```

which takes a list of filenames. The function should use `readFile` (Exercise 9g0) to read the contents of the files. The contents of the files should be merged into a single `string option`, which the function returns. If any of the files do not exist, then the function should return `None`.

Then write an application, `cat`, which takes a list of filenames as command-line arguments, calls the `cat` function with this list and prints the resulting string to the screen. The program must return 1 in case of an error and 0 otherwise.

9g2 First extend the library `readNWrite.fs` with a function,

```
tac : filenames:string list -> string option
```

which takes a list of files, reads their content with `readFile` (Exercise 9g0), reverses the order of each file in a line-by-line manner and reverses each line (i.e. the opposite of `cat`) and concatenates the result. If any of the files do not exist, then the function should return `None`.

Then write an application, `tac`, which takes a list of filenames as command-line arguments, calls the `tac` function with this list and prints the resulting string to the screen. The program must return 0 or 1 depending on whether the operation was successful or not.

9g3 In the html-standard, links are given by the `<a></a>` tags. For example, a link to Google's homepage is written as `<a href="http://google.com">Press to go to Google</a>`.

Make a program `countLinks` which includes the function

```
countLinks : url:string -> int
```

The function should read the page given in `url` and count how many links that page has to other pages. You should count by counting the number of `<a` substrings. The program should take a url, pass it to the function and print the resulting count on the screen. In case of an error, then the program should handle it appropriately.

Krav til afleveringen

Afleveringen skal bestå af:

- en zip-fil, der hedder `9g_<(gruppe)navn>.zip` (f.eks. `9g_jon.zip`)

Zip-filen `9g_<(gruppe)navn>.zip` skal indeholde en og kun en mappe `9g_<(gruppe)navn>`. I den mappe skal der ligge en `src` mappe og filen `README.txt`.

I `src` skal der ligge følgende og kun følgende filer:

- readNWrite.fs, cat.fsx, tac.fsx, countLinks.fsx,

som beskrevet i opgaveteksten. Programmerne skal kunne oversættes med `fsharpc`, og de oversatte filer skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden som minimum ved brug af `<summary>`, `<param>` og `<returns>` XML-tagsne. Filen `README.txt` skal ganske kort beskrive, hvordan koden oversættes og køres.

README.txt filen skal også inkludere et eller flere få eksempler på kørsler af hvert program, der illustrerer at og hvordan de virker.

God fornøjelse.