Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 9 - individuel opgave

Jon Sporring

25. november - 29. november.
Afleveringsfrist: lørdag d. 30. november kl. 23.59.

Emnerne for denne arbejdsseddel er:

- undtagelser (exceptions),
- input fra tastatur og output til skærm,
- input og output til filer,
- input fra internettet.

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

Øveopgaver

9ø0 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.

9ø1 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.

9ø2 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 or 0 depending on whether there was an error or not when running the program.

9ø3 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.

9ø4 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.

9ø5 Make a program with the function,

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

which initiates a dialogue with the user using `filenameDilaogue` from Exercise 9ø4. 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 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.

9ø6 Make a program with the function,

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

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

9ø7 Make a program with a function,

```
fileReplace :
  filename:string -> needle:string -> replace:string -> unit
```

which replaces all occurrences of the string `needle` with the string `replace` in the file `filename`. Your solution must use the `System.IO.File.OpenText`, `ReadLine`, and `WriteLine` functions.

9ø8 Make a calculator program

```
simpleCalc : unit -> unit
```

which starts an infinite dialogue with the user. The user must be able to enter simple expressions of positive numbers. Each expression must consist of a value, one of the binary operators `+`, `-`, `*`, `/`, and a value. When the user presses `<enter>`, the the expression is evaluated and the result is written as `ans = <result>` with the correct result entered. The input-values can either be a positive integer or the string “ans”, and the string “ans” should be the result of the previous evaluated expression or 0, in case this is the first expression typed. As an example, a dialogue could be as follows:

```
$ simpleCalc
>3+5
ans=8
>ans/2
ans=4
```

Here we used the character > to indicate, that the program is ready to accept input.

If the input is invalid or the evaluation results in an error, then the program should give an error message, and the input should be ignored.

9ø9 Make implementations of the following functions:

```
safeIndexIf : arr:'a [] -> i:int -> 'a
safeIndexTry : arr:'a [] -> i:int -> 'a
safeIndexOption : arr:'a [] -> i:int -> 'a option
```

Each of them must return the value of `arr` at index `i`, when `i` is a valid index, and otherwise handle the error-situation. The error-situations must be handled in different ways:

- `safeIndexIf` must not make use of `try-with` and must not cast an exception.
- `safeIndexTry` must use `try-with`, and it must call `failwith` when there is an error.
- `safeIndexOption` must return `None` in case of an error.

Make a short test of all 3 functions, by writing the content of an array to the screen (and not as an option type). The tests must also include examples of error situations and must be able to handle possible exceptions casted. In your opinion, is any of the above method superior or inferior in how they handle errors and why?

Afleveringsopgaver

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. UNIX also has an inverse version of `cat`, called `tac`, which prints the files in reverse order and reverses their content line-by-line. For example, if the file `a.txt` contains the characters `aaa\nbbb\n` and the file `b.txt` contains the characters `ccc\nddd\n` with `\n` being the newline character, then

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

will output `aaa\nbbb\nccc\nddd\n` to the screen. In contrast,

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

will output `ddd\nccc\nbbb\naaa\n` to the screen.

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

9i0 Make a 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`. The function should be placed in the implementation-file `readNWrite.fs`.

9i1 Make a function,

```
printFile : filenname:string -> bool
```

which prints the content of the file with the name `filename` to the screen. If no error occurs, then the function must return `true`, and otherwise `false`. The function should be placed in the implementation-file `readNWrite.fs`.

9i2 First extend the implementation-file `readNWrite.fs` with a function,

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

which takes a list of filenames. The function should use `readFile` (Exercise 9i0) 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 a program, `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 using `printFile` (Exercise 9i1). The program must return 0 or 1 depending on whether the operation was successful or not.

9i3 First extend the implementation-file `readNWrite.fs` with a function,

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

which takes a list of files, reads their content with `readFile` (Exercise 9i0), concatenates them, and returns the result as a string in reverse order line-by-line (i.e. the opposite of `cat` on a line-by-line basis). If any of the files do not exist, then the function should return `None`.

Then write a program, `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 using `printFile` (Exercise 9i1). The program must return 0 or 1 depending on whether the operation was successful or not.

9i4 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="https://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.

Afleveringen skal bestå af

- en zip-fil, der hedder `9i_<navn>.zip` (f.eks. `9i_jon.zip`)

Zip-filen `9i_<navn>.zip` skal indeholde en og kun en mappe `9i_<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 svarende til hver af delopgaverne. De skal kunne oversættes med `fsharpc`, og de oversattte 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 nogle få eksempler på kørsler af hvert program, der illustrerer at og hvordan de virker.

God fornøjelse.