POP assignment 2i

Mikkel *Luc* Carlsen

13. september 2020

Task 2i0

The three different methods that can be used to run an Fsharp program is(assuming that you have navigated into the folder that you are working in):

- 1) Write "fsharpi" in the terminal to launch Fsharp in the terminal and then write the code directly into the terminal
- 2) Write "fsharpi *Fsharp file name*" in the terminal, E.g. "fsharpi 2i1.fsx", where 2i1.fsx is the name of my Fsharp file
- 3) Write "fsharpc *Fsharp file name*" in the terminal. After the command has loaded type "mono *Fsharp file name, but now ending with* ***.exe*** *instead of* ***.fsx***" into the terminal E.g. my case would be "fsharpc 2i1.fsx" and the "mono 2i1.exe"

The first listed method of launching an Fsharp program is advantageous, when testing small fragments of code. This is because "fsharpi" launches Fsharp directly in the terminal, I.e it executes the written code as soon as you type ";;" and press enter, instead of having to compile it first then execute it. On the other hand it's disadvantageous when creating a program, because it's necessary to save the program as a file, I.e you are forced to write the code in a file and then either rewrite it in the terminal or copy paste all of the code into the terminal.

The second listed method of launching an Fsharp program grants the advantage of being able to write bigger piles of code in a file and then test it directly - noticable that its advantage is exactly what was the first methods disadvantage. So if the command "fsharpi *Fsharp file name*" were to be executed, it would execute the program without having to copy paste it. The disadvantage of this method is that it has a overall slow running time, but it's slighty faster than the third way if the purpose is to run the code only once or if it's necessary to recompile everytime before running.

The third listed method is advantageous when running the program multiple times without having to recompile. This is due to the fact that it takes almost no time to run the program with mono when it's already compiled I.e. when the program is already compiled it's possible to just use the mono command without reusing the fsharpc command. But even so, it's slower to use the fsharpc and mono command together, than it's to use the fsharpi *Fsharp file name* command. Therefore it's only advantageous when the program is complete and does not require further changes.

Task 2i1

The goal of the second task was to create an expression in Fsharp, that could extract the words ”hello” and ”world” from a string saying ”hello world”.
The way i solved this task was by writing the following in Fsharp:

```
let a = ''Hello world''\\
let b = a.[0..4]\\
let c = a.[6..10] \\
printfn ''%A'' b
\\
printfn ''%A'' c
\\
```

What the code does is that it defines the variable a as "hello world"Then it defines the variable b as the lenght of a from 0-4, I.e. it defines b as the values in a[0,1,2,3,4] The same goes for c, but for the values a[6,7,8,9,10] afterwards it prints b and then c

Task 2i2

For task 2i2 i completed it with pen and paper, but i have added the table to assigntment as seen below.

Decimal	binary	Hexadecimal	Octal
10	1010	A	12
21	10101	15	25
47	101111	2f	57
59	111011	3b	73

The way i completed the table was by filling out the decimal row first. so step by step, i started out with converting the binary number 10101_2 to a Decimal. This was done by saying $d_n * 2^n + d_{n-1} * 2^{n-1}$ e.g. if the binary number is 10101_2 then it would be $1*2^4+0*2^3+1*2^2+0*2^1+1*2^0 -- > 16+0+4+0+1 = 21_{10}$. After that i converted the hexadecimal $2f_{16}$ to decimal by using the power 16^n method starting from the right. e.g. $2f_{16}$ would be converted by saying $16^0 * f_{16} + 16^1 * 2 -- > 1*15+32 = 47_{10}$. Then i converted octal to decimal taking each digit in the octal number from the left to right and multiplying it with its corresponding 8^n. e.g. $73_8 -- > 7*8^1 + 3*8^0 = 56+3 = 59_{10}$ Now that i had filled the decimal row, i used it to convert decimal to the remaining missing values of binary, Hexadecimal and octal numbers.

Decimal –> binaryinary: Done with the integer divide and remainding method e.g.
$10/2 = 5, 10 rem 2 = 0$
$5/2 = 2, 5 rem 2 = 1$
$2/2 = 1, 2 rem 2 = 0$
$1/2 = 0, 1 rem 2 = 1.$
From here you can read the remainding numbers from bottom to top so it says that the decimal 10_{10} is equal to the binary 1010_2.

Decimal –> hexadecimal: Done by integer dividing the decimal with 16, then add that to the result and take the remaining numbers from the devision and convert to hexadecimal. E.g. $59_{10}/16 = 3, rem = 11$ which is equal to $3b_{16}$.

Decimal –> octal: Was done with a homemade version. For every 10th decimal i would say $12 * n$, where n is the amount of 10ths in the decimal number. Then if the number i ended up with would have the first digit as either 8 or 9, then if the single digit was 8 i would round up to the next 10th else if it was 9 i would round up to the next 10th+1. after that that i would finally add the remaining single digits from the decimal. e.g. if the decimal is 47_{10} i would notice that the 10th is 40, I.e. that n=4. So i would say $12*4 = 48$ and then see that the first digit is 8, which means i have to round up to the nearest 10th, which is 50. Now i add the remaining single digits from the decimal so that i get my octal number $50 + 7 = 57_8$