POP Assignment 2i

Thomas Haulik Barchager - jxg170

September 2021

1 Assignment

1.1 2i0

Describe the 3 ways an F# program can be run from the command line (terminal), and discuss the advantages and disadvantages of each method.

1.1.1 (2i0)

The first way to run F# code is by using the interpreter in the terminal/CMD by typing **"fsharpi"**. this allows you to type F# code into the terminal and run the code directly from the terminal by ending the code with "**;;**". This is a easy and quick way to run small amount of code, if for example you want to make some calculation by using F#. Furthermore by running the code this way you are giving more information about the variables you initializing, what type it is and what value it has. The disadvantages to this is that the code need to compile every time you run it, which can make for a slow process. Plus the code that is typed in the terminal doesn't get saved.

The second way to run F# code is to use the interpreter on a already created .fsx file. This is done by typing "fsharpi filname.fsx". By running the code this way you are not giving all the extra information as the first way, instead the code just runs normal and the only things that shows up in the terminal is the things you told the code to print out. As the first way to run the code, this way can also be considered slow because the code need to compile every time you run it.

The third way to run F# code is to manually compile the .fsx file to a .exe and then run that file. This is done by first typing "fsharpc filname.fsx" which will compile the code and created a .exe file, that can be executed by typing "mono filname.exe". By running the code this way, you like the second way, are not giving all the extra information as the first way, but only what you told the code to print out. The advantages to manually Compile the .fsx file to a .exe, is that the code no longer need to compile every time to run the code and instead runs the code immediately. The downside to this is then you want to make some small changes to the code, you will have to write them in the .fsx file and then compile the file again to create a new .exe file.

1.2 2i1

Using slicing, write an expression in F# which extracts the first and the second word from the string "hello world". Enter the expression in an .fsx file and compile and run it. Does the program produce output? Explain why or why not.

1.2.1 (2i1)

```
let string = "Hello world"
let stringcut = string.IndexOf(" ")

printfn "%A" string
printfn "%A" string.[0..stringcut-1i]
printfn "%A" string.[stringcut+1..]
```

Figure 1: The slicing code

The program does produce an output as expected. The code starts by initialize two variables. One that contains the string "Hello world" and one that contains the index of the space between the two words. By using the **Printfn** function I tell the code first to print the string "Hello world", then the first word of the string: "Hello" and then the last word of the string: "World".

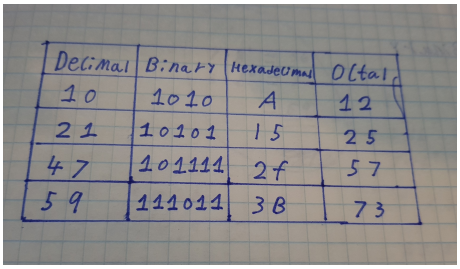
1.3 2i2

Use pen and paper to complete the following table

Decimal	Binary	Hexadecimal	Octal
10			
	10101		
		2f	
			73

such that every row represents the same value written on 4 different forms. Include a demonstration of how you converted binary to decimal, decimal to binary, binary to hexadecimal, hexadecimal to binary, binary to octal, and octal to binary.

1.3.1 (2i2)



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

Figure 2: Completed table

Hex	Bin
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Figure 3: Hexadecimal to Binary

1.3.2 (2i2: binary to decimal)

$Binary = 111011_2$
$1 * 2^0 = 1$
$1 * 2^1 = 2$
$0 * 2^2 = 0$
$1 * 2^3 = 8$
$1 * 2^4 = 16$
$1 * 2^5 = 32$
$Decimal = 59_{10}$

1.3.3 (2i2: decimal to binary)

$Decimal = 21_{10}$
$21/2 = 10\ Rem = 1$
$10/2 = 5\ Rem = 0$

$5/2 = 2\ Rem = 1$
$2/2 = 1\ Rem = 0$
$1/2 = 0\ Rem = 1$
$Binary = 10101_2$

1.3.4 (2i2: binary to hexadecimal)

By using the table in figure 3, I'm able to split the binary up in 4 digits number and translate the binary number to Hexadecimal. The reason behind this is because $2^4 = 16$

$Binary = 101111_2$
$0010 = 2$
$1111 = F$
$Hexadecimal = 2F_{16}$

1.3.5 (2i2: hexadecimal to binary)

The same idea as the above one, just the other way around.

$Hexadecimal = 3b_{16}$
$3 = 0011$
$b = 1011$
$Binary = 111011_2$

A bigger number for fun:

$Hexadecimal = 562A_{16}$
$5 = 0101$
$6 = 0110$
$2 = 0010$
$A = 1010$
$Binary = 101011000101010_2$

1.3.6 (2i2: binary to octal)

Again the same idea as the above one, but this time splitting the binary up in 3 digits number instead. $2^3 = 8$

$Binary = 111011_2$
$111 = 7$
$011 = 3$
$Octal = 73_8$

1.3.7 (2i2: octal to binary)

$octal = 57_8$
$5 = 101$

$7 = 111$
$Binary = 101111_2$

A bigger number for fun:

$octal = 645_8$
$6 = 110$
$4 = 100$
$5 = 101$
$Binary = 110100101_2$