Second Assignment in PoP

Jakob H. Schauser, pwn274

September 2021

1 Describing the three ways of running f# code

As with so many things in life, there are no clear-cut rights or wrongs when it comes to running f# code. The gray area that is code-compilation has multiple possible solutions - each with their own advantages and drawbacks. The three main ways are comprised of:

1.1 Compiling and running with a single command

The first way of bringing f# code to life is via the "fsharpi"-command. This command interprets the f# code in a file and runs it. One might suspect the command "fsharpi" to be the best, as it certainly is the most convenient. But this method of interpretation certainly has some disadvantages. Every time the program has to be run, it will be interpreted anew, making this method slower in the long run as the work done by the intepreter has already been done and is therefore utterly reduntant.

1.2 Compiling and running with two seperate commands

The choice could also fall on first compiling the code before running it on the computer. This is typically done via the "fsharpc" command, which uses dotnet to create an executable .exe file. This file can then, through use of mono, be understood by the computer and runned. The smart thing here is that once the executable is produced, it can easily be run at your heart's content without need for new compiling. The inevitable drawback of this method is, that the compilation and subsequent running of the program takes longer than the direct interpretation as described in the last section.

1.3 Running interactively

You can also run interactively. This is not a smart way to program bigger projects, but when simply needing to run a single line of code, this is the way to go.

2 Converting between bases

2.1 The whole grid

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

2.2   From binary to decimal

I use the trusty doubling technique. Given $(10101)_2$, moving from left to right I double the current total and add the next. This gives:

- (+1) 1
- (+0) 2
- (+1) 5
- (+0) 10
- (+1) 21

Meaning the final answer is $(10101)_2 = (21)_{10}$

2.3   From decimal to binary

Here I use the "opposite" halving technique. I take the number and continously halve it, keeping track of the remainder. Once finished, the remainedrs are read from bottom up. I am using the number $(10)_{10}$. Here under I keep track of the current number on the left and the remainder on the right:

- 5 - 0
- 2 - 1
- 1 - 0
- 0 - 1

Giving $(10)_{10} = (1010)_2$.

2.4   From binary to hex

Moving from binary to hex also has a smart trick. Looking in groups of four from right to left, simply convert to each digit seperately using mental arithmetic. I am once again using $(10101)_2$:

(1)(0101) = (1)(5) = 15

So $(10101)_2 = (15)_{16}$

2.5   From hex to binary

Now we just do the pposite, again converting each digit in to a group of four. I will be using the number $(2F)_{16}$: (2)(F) = (0010)(1111) = 101111

2.6   From binary to octal

Now, we simply do as with the binary to hexadecimal conversion exept in groups of three, again using $(10101)_2$: (10)(101) = (2)(5) = 25

2.7   From octal to binary

Once again the same as from hex to binary but in groups of three, now using
$(73)_8$: $(7)(3) = (111)(011) = 111011$