Exercise 1
temp:=x;x:=e;c;x:=temp
VC(let x=e in c, B) = VC(temp:=x, VC(x:=e, VC(c, VC(x:=temp, B))))

= VC(temp:=x, [e/x] VC(c, [temp/x] B))

= [x/temp]([e/x](VC(c, [temp/x]B)))

Note that as opposed to the temp assignment, this can also be written more tersely as
[e/x]VC(c,[x_0/x]B)

Exercise 2
c:
let x:=5 in if x<10 then x:=1 else x:=20 such that
Note that x in the command is the let variable.

$$(let\ x :=\ 5\ in\ if\ x < 10\ then\ x := 1\ else\ x := 20,\ \sigma) \rightarrow \sigma'$$

B:
{x=1}

sigma:
As per the buggy let rule, the VC is [e/x] VC(com, B). c is written as com to avoid ambiguity between c mentioned above and the command inside the let block.

VC(if x<10 then x:=1 else x:=20, (x=1)) = {x<10}

Let sigma be the state after the let variable is initialized.
$\sigma(X) = 5$
[e/x]{x<10}={5<10} = true

The above VC is satisfied in the state sigma.
$$(let\ x :=\ 5\ in\ if\ x < 10\ then\ x := 1\ else\ x := 20,\ \sigma) \rightarrow \sigma'$$

However in the new state sigma-prime, $\sigma'(X)$ will be restored to the value of x before the let block and B is hence false.

Note: The basis of this is that if we were to decompose the let command, the resultant state sigma-prime of the let statement is a consequence of not just the command executed in scope but also a consequence of restoring the old value of the let variable.

Exercise 3
$$\frac{\vdash \{A\}c\{Inv\} \quad \{Inv \wedge b\}c\{Inv\}}{\vdash \{A\}do\ c\ while_{Inv}\ b\{Inv \wedge - b\}}$$

Exercise 4:

$$Inv \wedge VC(c,\ Inv) \wedge (\forall x1..xn\ \ Inv => (b => VC(c, Inv))) \wedge ((Inv \wedge -b) => B)$$

Exercise 5:

Rule mal

A: {(x<10)=>(x=25) ^ -(x<10)=>(x=25)}

This can be simplified to

A: {(x=25)}

B: {(x=25)}

States and c

$\sigma[X] = 25$

c: while x<2 do x:=x+1

In the given state sigma, c's behaviour is

$(while\ x < 2\ do\ x := x + 1,\ \sigma) - > \sigma$

Since c is effectively skip, there is no new state sigma prime.

$$\frac{\vdash \{x = 25\}x := x + 1\{(x < 10) => (x = 25)\Lambda\ -(x < 10) => (x = 25)\}}{\vdash \{(x < 10) => (x = 25)\Lambda\ -(x < 10) => (x = 25)\}while\ x < 10\ do\ x := x + 1\{(x = 25)\}}$$

simplifies to

$$\frac{\frac{---fail---}{\vdash \{x=25\}x:=x+1\{x=25\}}}{\vdash \{x = 25\}while\ x < 10\ do\ x := x + 1\{(x = 25)\Lambda - (x < 10)\}}$$

A and B hold in sigma(there is no sigma-prime).

However it is impossible to prove {A}c{B} as shown above.

Rule: river (This rule is just a special case of the mal rule)

A: {x=25}

B: {(x=25) ^ ~(x<10)}

This can be simplified to {(x=25)}

States and c

$\sigma[X] = 25$

c: while x<2 do x:=x+1

However in the given state sigma, commad c's behaviour is

$(while\ x < 2\ do\ x := x + 1,\ \sigma) - > \sigma$

Since c is effectively skip. Hence there is no new state sigma prime.

sigma (since there is no sigma-prime) makes A and B true

However, {A}c{B} can never be proved

$$\dfrac{\dfrac{---fail---}{\vdash \{x=25\} x:=x+1 \{x=25\}}}{\vdash \{x = 25\} while\ x < 10\ do\ x := x + 1 \{(x = 25) \Lambda - (x < 10)\}}$$

Exercise 6

I spent about 3 hours on the homework. In response to your question on what genre I enjoy, I like to sing Indian classical. I can appreciate jazz as well. I have started on the project. Right now, I am stuck on some inter operablity between C and Ruby. I have decided to keep the same constructs as NESL.

By the way, the new monitor exceeds my expectations #LED #win !! #arrived on Thursday.