Visual F# Tips: Debugging and using the console window

- For many purposes, you can debug simply by running individual functions in the interactive F# top-level. This is generally the easiest way to debug.
- You can also print values (other than functions) using e.g., `printfn "x=%A y=%A" x y`
- Visual Studio includes a debugger that allows you set breakpoints where the program should stop, examine variables, step through code line by line, etc. Read about it in the help for visual studio.
- To run in the debugger, your whole file/project will be compiled and run, and the output will be sent to a console window.
- When running in this way, it can be useful to add the following at the end of your program to prevent the console window from closing immediately.

```fsharp
printfn "Program finished, press Enter."
System.Console.ReadLine()
```

Lab details

This lab involves some imperative programming and a first look at creating multiple threads that run concurrently (skipping slightly ahead of the lectures).

1. Consider the following types for mutable lists.

```fsharp
/// Mutable lists

type 'a listMut = MutNil | MutCons of 'a ref * 'a mutList
and 'a mutList = ('a listMut) ref // Parens not needed
```

a) Write functions to convert between lists and mutable lists.

```fsharp
let list2mut : 'a list -> 'a mutList
let mut2list : 'a mutList -> 'a list
```

b) Write a function that inserts an element into a sorted mutable integer list in the right position by modifying the list rather than rebuilding.

```fsharp
let mutInsert : 'a -> 'a mutList -> 'a mutList
```

c) Consider the following:

```fsharp
let origEnd = ref MutNil
let mut = ref (MutCons (ref 32, ref (MutCons (ref 42, origEnd) ) ) )
origEnd = !mut
```
This creates a cyclic list, which is generally not possible with non-mutable lists. Write a function that determines whether a list is cyclic by recursively moving two references down the list. The first should take one step each time, the other should take two steps. If the end of the list is reached, then the list is not cyclic. If the two references ever refer to the same cell, then this list is cyclic. Use the function `System.Object.ReferenceEquals : 'a ^ 'b -> bool` to check whether two references are the same.

```csharp
mutCyclic : 'a mutList -> bool
```

[The correctness of this algorithm is slightly subtle – basically if there is a cycle, then eventually both references will reach it. After this, the fast reference will be behind the slow reference, because of the cycle, and will eventually catch it, which we’ll detect because the fast reference only catches up by one each time.]

2. Consider the following code from lectures, similar to that in lectures. It fetches a web page and displays it in a simple GUI textbox via Windows Forms. Fetching is slightly simplified in that it uses a “use” binding for `resp` – this is just like a `let` binding, but automatically closes `resp` when the program leaves the scope of the variable (in this case when the `http` function returns). (More generally `use` causes the Dispose method to be called on the variable.)

```csharp
open System // For System.Net, System.IO and String
let http(url: string) = // Get the contents of a URL via the web
    use resp = Net.WebRequest.Create(url).GetResponse()
    let reader = new IO.StreamReader(resp.GetResponseStream())
    let html = reader.ReadToEnd()
    html

let form = new Form(Visible=true, TopMost=true, Text="My web page html viewer")
let textB = new RichTextBox(Dock=DockStyle.Fill)
form.Controls.Add(textB)
```

```csharp
textB.Text <- http("http://www.csse.uwa.edu.au")
```

Here `new` is being used to construct .NET objects for the user interface, and to read data, just like in Java, although with a special form of constructor arguments that include names.

a) Try running this program to verify that it works. You will need to add a reference to the `System.Windows.Forms` assembly by right clicking on your project in the solution explorer window – this is a Dynamic Link Library (dll) that is part of the .NET distribution.

b) Modify this program so any exceptions while fetching the URL are caught, and then converted to a string which is placed in the text box instead of the web page contents. Test this by changing the URL to something invalid.

[Hint: You can just apply the conversion function `string` to the exception.]

c) Modify this program to reverse the order of the character in the web page contents, and replace each space character by two spaces.

[Hint: use an array comprehension – a string is a subtype of `char seq`, so can be used in a generator. To build a string from an array of characters `cArr` just use `new String cArr`. Also consider the functions available in the `Array` module.]
3. The following code makes use of a background worker thread, which runs at the same time as the main thread. This is a simple example of a concurrent program.

```csharp
open System.ComponentModel
let worker = new BackgroundWorker()
let numIterations = 20

let printRepeatedly who =           /// print a number of lines
  for i in 1..numIterations do
    printfn "Iteration %d for %s" i who

  // Tell the worker what to do when it is told to run
  worker.DoWork.Add(fun _ -> printRepeatedly "worker")

  worker.RunWorkerAsync()          /// Tell the worker to run now in the background
  printRepeatedly "main thread"     /// make the main thread print also
```

a) Try running this program to see what it does. Do you get the same results every time?
b) Modify this program to add another background worker and see how this affects the result.