

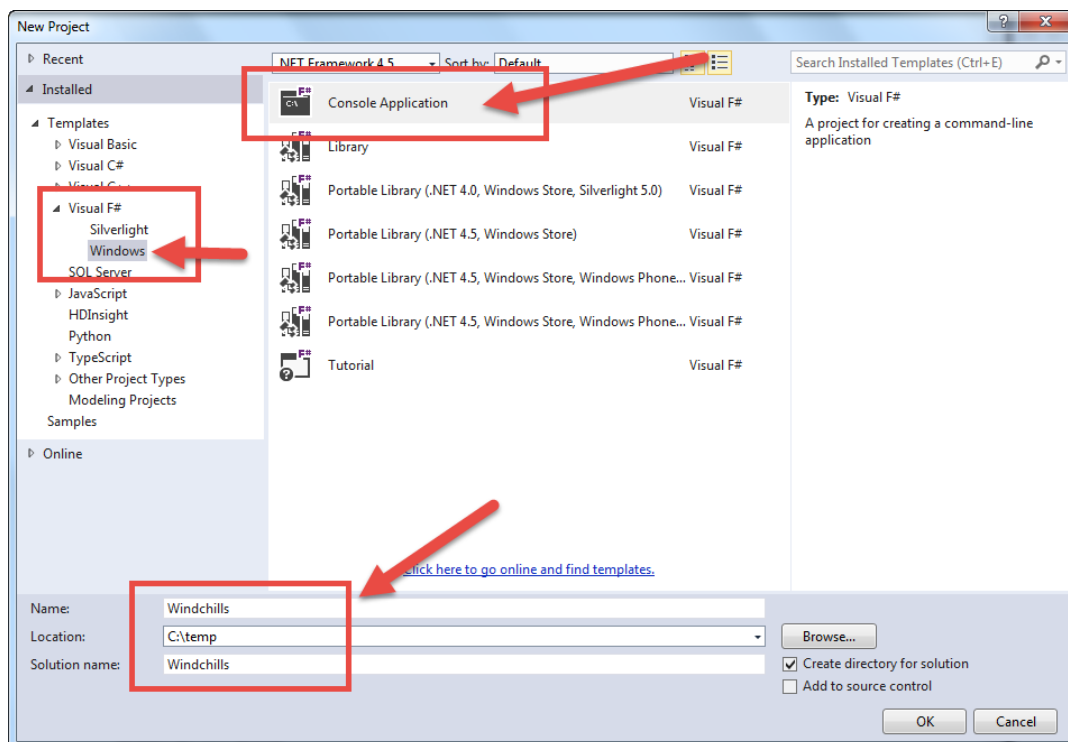
## Homework #4

**Complete By:** Saturday, September 26<sup>th</sup> @ 11:59pm  
**Assignment:** completion of following exercise  
**Policy:** Individual work only, late work *\*is\** accepted  
**Submission:** electronic submission via Blackboard

### Getting Started

The idea is to write a short program in F# to help you get started with functional programming. Note that if you do not yet have access to a computer with Visual Studio installed, you have two options. First, you can try the machines in SELE 2249 or 2249F, they have Visual Studio installed; these are generally available most afternoons and evenings. Alternatively, you can program in <http://ideone.com>, F# is a supported language.

When working in Visual Studio, create a new project as you normally would, except expand the templates for “Visual F#”, select Windows, then Console Application, and give your project a name — “Windchills”.



Click OK and Visual Studio will create an F# project and generate a main program for you:

```
// Learn more about F# at http://fsharp.net
// See the 'F# Tutorial' project for more help.
```

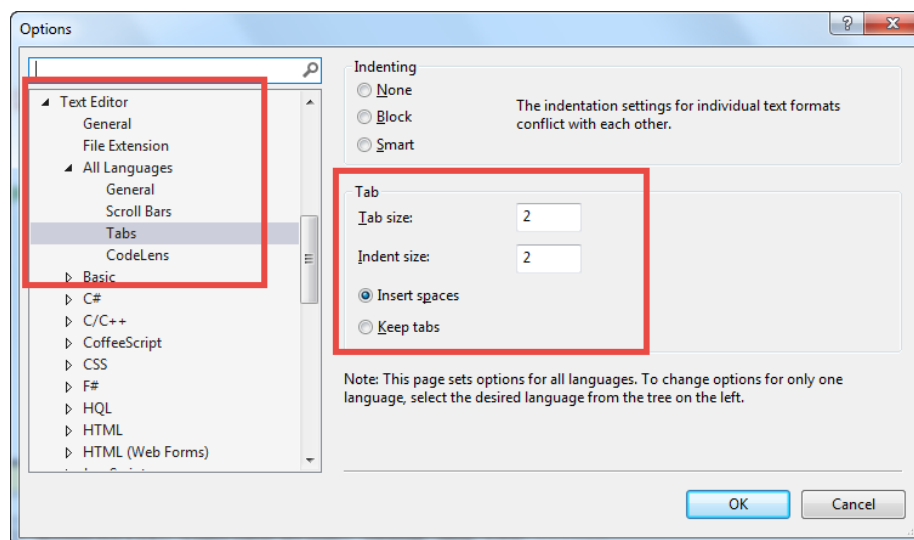
```
[<EntryPoint>]
let main argv =
    printfn "%A" argv
    0 // return an integer exit code
```

Change the program by deleting the header comment, then adding the **#light** directive so that you enable “indentation” mode — this allows us to program without having to use { ... }. Also modify the program to output “Hello world!” like a C programmer:

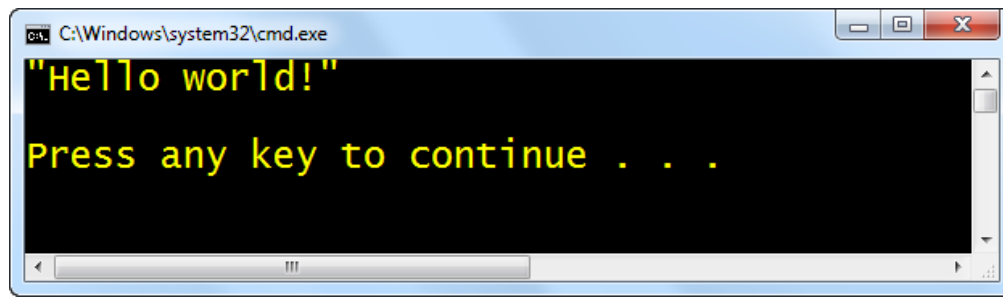
```
#light

[<EntryPoint>]
let main argv =
    let msg = "Hello world!"
    printfn "%A" msg
    printfn ""
    0 // success
```

Two things about programming in “indentation” mode: (1) you must indent consistently within a function (notice that each statement under main has 2 spaces in front), and (2) you must use spaces, not tabs. For this reason, I configure Visual Studio to convert tabs to spaces: Tools menu, Options, scroll down and expand “Editor”, expand “All Languages”, select “Tabs”, and configure:



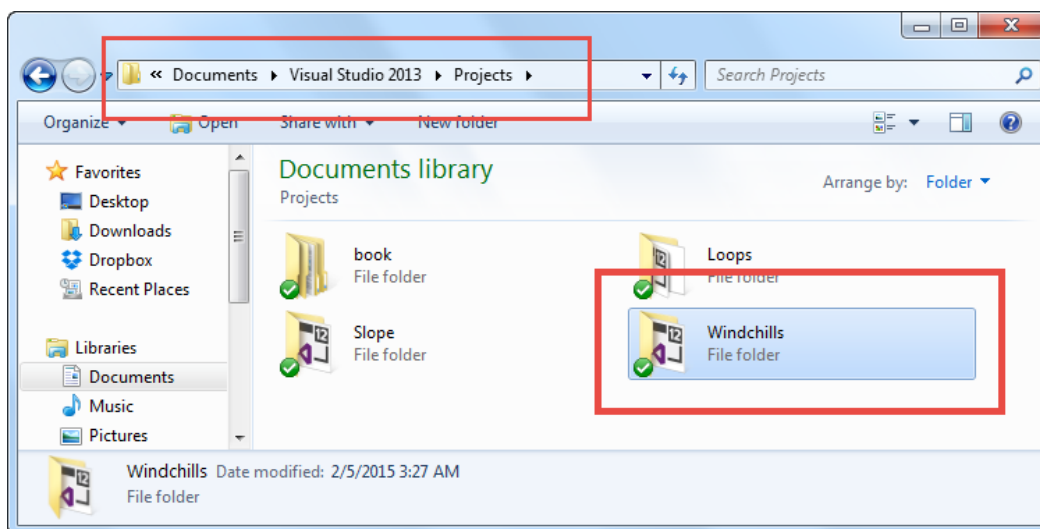
If necessary, you may need to retype the indentation to eliminate any tabs you generated before configuring. Now go ahead and run your program — use Ctrl+F5 (or Debug menu, Start without debugging). You should see a simple console window open:



All is well!

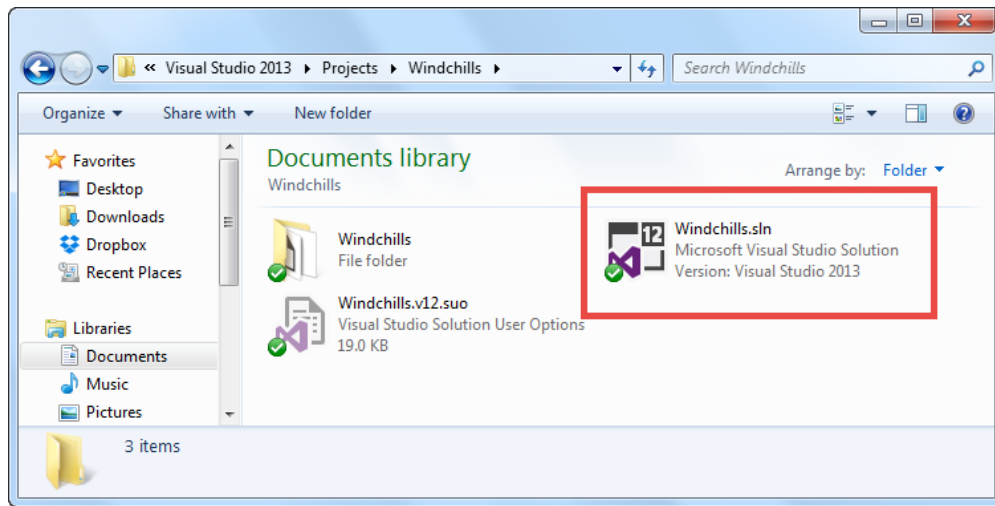
## Working with Visual Studio

Since many of you are new to Visual Studio, it's probably time to give you some tips on how to work with it. If you have Visual Studio open, go ahead and close it. Where is your program? By default, your program can be found in Documents, Visual Studio 2013 / 2015, Projects. If you named the project “Windchills”, you’ll see a **Windchills** folder:

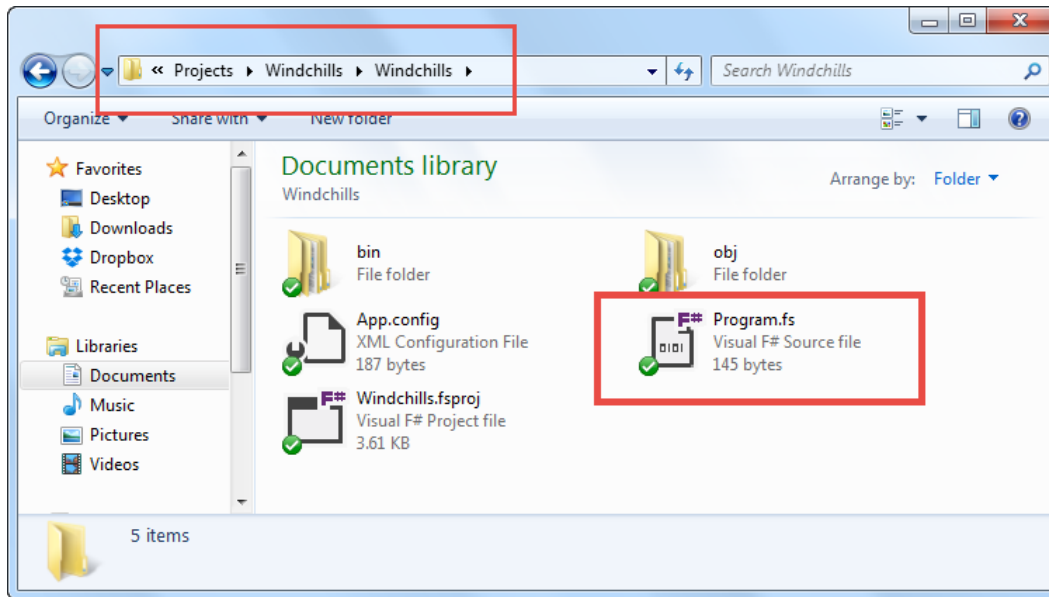


That folder is your entire program. If you want to copy your program and work on another machine, make a copy of this folder (or right-click, “Send to”, “Compresses (zipped) folder”, and you’ll have a single .zip file you can email / move around). When I want to experiment with a program I’ve already written, I make a copy of the folder and open the copy — that way I don’t disturb the original. When I’m done, I can keep the copy, or throw it away.

What’s the proper way to open a project with Visual Studio? Open the folder, and double-click on the “Solution” file:



Finally, where is your actual F# source file that you will submit to BB for grading? Go down into the “Windchills” sub-folder, and you’ll see the F# source file, in this case “Program.fs”:



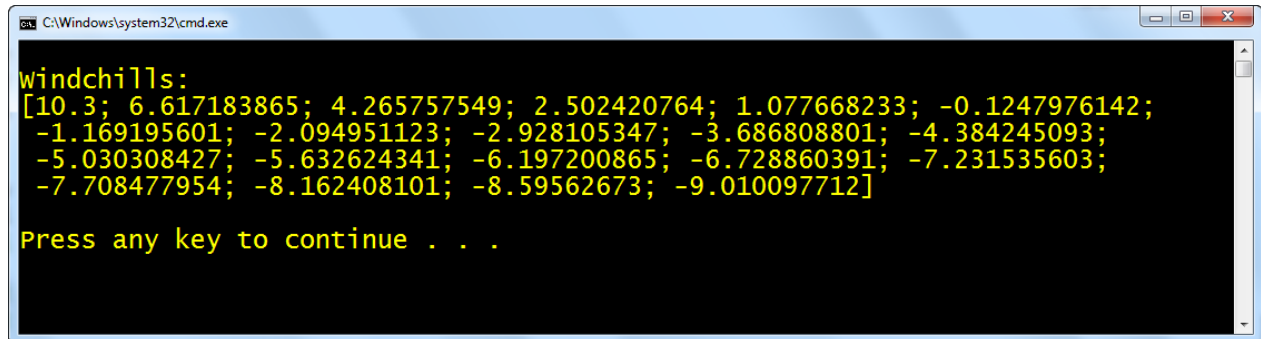
F# source files end in “.fs”.

## Assignment, Part 1

Part 1 of the assignment is to write a program to compute windchills for a temperature of 10 degrees F, across the range of wind speeds 1-20 mph. First, here’s the equation for computing windchill, given a temperature  $T$  (in degrees F) and a wind speed  $W$  (in mph):

$$\text{windchill} = 35.7 + 0.6T - 35.7W^{0.16} + 0.43TW^{0.16}$$

Here's the program output:



```

C:\Windows\system32\cmd.exe

windchills:
[10.3; 6.617183865; 4.265757549; 2.502420764; 1.077668233; -0.1247976142;
-1.169195601; -2.094951123; -2.928105347; -3.686808801; -4.384245093;
-5.030308427; -5.632624341; -6.197200865; -6.728860391; -7.231535603;
-7.708477954; -8.162408101; -8.59562673; -9.010097712]

Press any key to continue . . .

```

The output is a list of 20 windchills: W=1, W=2, W=3, ..., W=20. For example, for T=10 and W=4, the windchill is roughly 2.5 degrees F. For W=10, the windchill is roughly -9.0 degrees F.

To help you get started, here's a near-complete program skeleton. You'll need to complete the **windchill** function at the top, in particular to compute the value for WC, the windchill, based on the equation:

```

#light

let windchill W =
  let T = 10.0
  let WC = 35.7 + ...
  WC

[<EntryPoint>]
let main argv =
  let WS = [1.0 .. 20.0]
  let windchills = List.map windchill WS

  printfn ""
  printfn "Windchills:"
  printfn "%A" windchills
  printfn ""

  0 // return an integer exit code

```

Use the library function `System.Math.Pow(W, 0.16)` to compute  $W^{0.16}$ . Notice in the main program that we do *not* use a counter-based loop to compute the windchills for W = 1, 2, 3, ..., 20. Instead, we create a wind

speed list **WS** that contains the values for W, and then we **map** the *windchill* function across the wind speed list *WS*. The map function collects the 20 results into a list, which we then output with a single call to `printfn "%A"`.

## Assignment, Part 2

Okay, now let's modify the program to input the temperature from the user, and output the windchills based on that temperature T; assume the user will enter a valid temperature in degrees F. Let's change the skeleton, demonstrating how to pass multiple values to your windchill function, as well as input from the keyboard. The changes are shown in a slightly larger **boldface** font:

`#light`

```
let windchill T W =  
    let WC = 35.7 + ...  
    WC
```

```
[<EntryPoint>]
```

```
let main argv =
```

```
    printf "Please enter temperature (degrees F): "
```

```
    let input = System.Console.ReadLine()
```

```
    let T = System.Convert.ToDouble(input)
```

```
    let WS = [1.0 .. 20.0]
```

```
    let windchills = List.map windchill WS
```

```
    printfn ""
```

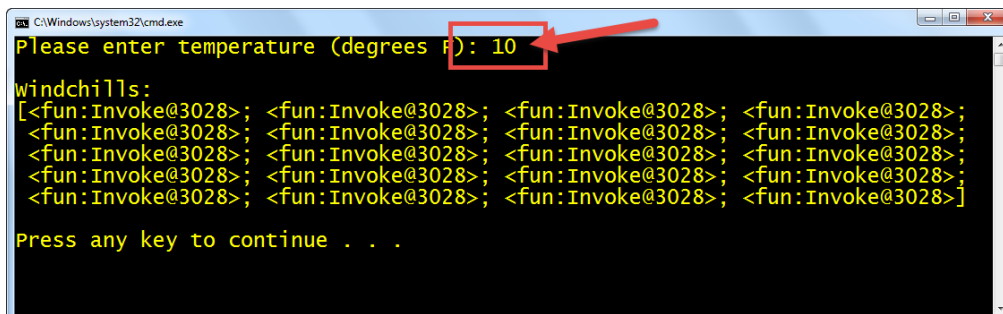
```
    printfn "Windchills:"
```

```
    printfn "%A" windchills
```

```
    printfn ""
```

```
    0 // return an integer exit code
```

Make those changes to your program, and run (Ctrl+F5). Enter a wind speed of 10:



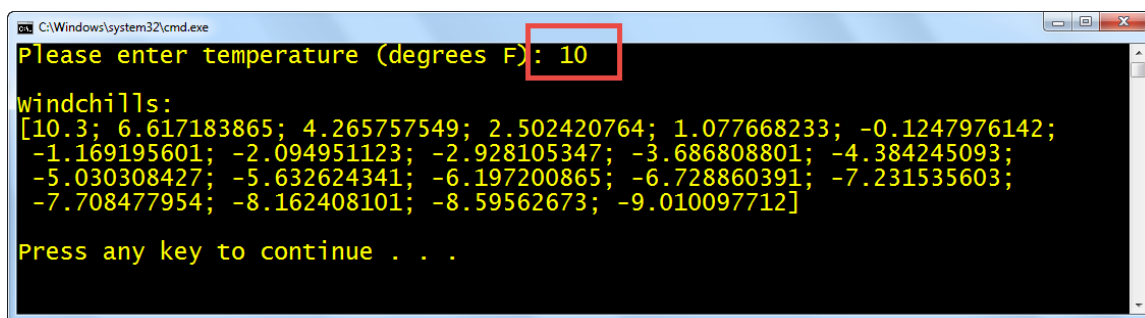
```
C:\Windows\system32\cmd.exe  
Please enter temperature (degrees F): 10  
Windchills:  
[<fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>;  
<fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>;  
<fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>;  
<fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>;  
<fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>; <fun:Invoke@3028>]  
Press any key to continue . . .
```

Oops, that's not good. The **windchill** function is being called, but it's not computing a result. What happened? If we stop and reflect for a second, notice that the windchill function takes 2 parameters: T and W. However, **map** only supplies one: W, an element from the wind speed list WS. So windchill is not getting the 2 parameters that it needs, and therefore fails to compute a result. [ *We'll explain later what it's actually doing.* ]

The solution is to supply the second, missing argument to the windchill function. How to do that? Provide **map** with a *lambda expression* that takes an element W from the list, but then calls windchill with the two parameters T and W:

```
let windchills = List.map (fun W -> windchill T W) WS
```

How does the lambda expression gain access to the variable T? As in C++, the compiler closes over the environment and grabs a copy of T. Run, and the program should behave correctly as before:



```
C:\Windows\system32\cmd.exe
Please enter temperature (degrees F): 10
windchills:
[10.3; 6.617183865; 4.265757549; 2.502420764; 1.077668233; -0.1247976142;
-1.169195601; -2.094951123; -2.928105347; -3.686808801; -4.384245093;
-5.030308427; -5.632624341; -6.197200865; -6.728860391; -7.231535603;
-7.708477954; -8.162408101; -8.59562673; -9.010097712]
Press any key to continue . . .
```

Welcome to functional programming!

## Electronic Submission

Using Blackboard ( <http://uic.blackboard.com/> ), submit your “Program.fs” source code file under the assignment “HW4”. Attach the file, do not copy-paste. If you worked with <http://ideone.com>, copy-paste your code into a text file, and attach the text file. The top of your main source file should contain a header comment like the following:

```
//
// F# program to compute windchills.
//
// <<YOUR NAME HERE>>
// U. of Illinois, Chicago
// CS341, Fall 2015
// Homework 4
//
```

Your program should be readable with proper indentation and naming conventions; commenting is expected

where appropriate. You may submit as many times as you want before the due date, but we grade the last version submitted. This implies that if you submit a version before the due date and then another after the due date, we will grade the version submitted after the due date — we will *\*not\** grade both and then give you the better grade. We grade the last one submitted. In general, do not submit after the due date unless you had a non-working program before the due date.

## Policy

Late work *\*is\** accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. your “iClicker teammates” or Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *\*cannot\** work in teams, you cannot work side-by-side, you cannot submit someone else’s work (partial or complete) as your own. The University’s policy is described here:

<http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf> .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else’s iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at <http://www.uic.edu/depts/dos/studentconductprocess.shtml> .