

# Project 2

Bharath Karumudi

July 19, 2019

## **Abstract**

This project is to demonstrate the capabilities of functional programming using the tools and techniques -  $\text{\LaTeX}$ ,  $\text{\AcuTeX}$ , emacs and ML. Each chapter documents the given problems with a structure of:

1. Problem Statement
2. Relevant Code
3. Test Cases
4. Execution Transcripts
5. Explanation of results

**Acknowledgments:** Professor Marvine Hamner and Professor Shiu-Kai Chin who taught the Certified Security By Design.

---

# Contents

---

<b>1</b>	<b>Executive Summary</b>	<b>4</b>
<b>2</b>	<b>Exercise 4.6.3</b>	<b>5</b>
2.1	Problem Statement . . . . .	5
2.1.1	4.6.3A . . . . .	5
2.1.2	4.6.3B . . . . .	5
2.1.3	4.6.3C . . . . .	5
2.1.4	4.6.3D . . . . .	5
2.1.5	4.6.3E . . . . .	5
2.1.6	Test Cases . . . . .	6
2.2	Execution Transcripts . . . . .	7
2.2.1	Explanation of Results . . . . .	7
<b>3</b>	<b>Exercise 4.6.4</b>	<b>8</b>
3.1	Problem Statement . . . . .	8
3.2	Relevant Code . . . . .	8
3.3	Test Cases . . . . .	8
3.4	Execution Transcripts . . . . .	8
3.4.1	Explanation of Results . . . . .	8
<b>4</b>	<b>Exercise 5.3.4</b>	<b>9</b>
4.1	Problem Statement . . . . .	9
4.2	Relevant Code . . . . .	9
4.3	Test Cases . . . . .	9
4.4	Execution Transcripts . . . . .	9
4.4.1	Explanation of Results . . . . .	10
<b>5</b>	<b>Exercise 5.3.5</b>	<b>11</b>
5.1	Problem Statement . . . . .	11
5.2	Relevant Code . . . . .	11
5.3	Test Cases . . . . .	11
5.4	Execution Transcripts . . . . .	11
5.4.1	Explanation of Results . . . . .	11
<b>6</b>	<b>Exercise 6.2.1</b>	<b>12</b>
6.1	Problem Statement . . . . .	12
6.1.1	6.2.1.1 . . . . .	12
<b>7</b>	<b>Appendix A: Exercise 4.6.3</b>	<b>13</b>
<b>8</b>	<b>Appendix B: Exercise 4.6.4</b>	<b>16</b>
<b>9</b>	<b>Appendix C: Exercise 5.3.4</b>	<b>17</b>

**10 Appendix D: Exercise 5.3.5****18**

# Executive Summary

---

All the requirements for this project are statisfied.

**Reproducibility in ML and  $\text{\LaTeX}$**

Our ML and  $\text{\LaTeX}$  source files compile with no errors.

# Exercise 4.6.3

---

## 2.1 Problem Statement

In this exercise we define five ML functions using `fun` and `val`.

### 2.1.1 4.6.3A

In this we define a function that takes a 3-tuple of integers  $(x, y, z)$  as input and returns the value corresponding to the sum  $x + y + z$ .

```
val funA1 = (fn (x,y,z) => x+y+z);  
fun funA2 (x,y,z) = x+y+z;
```

### 2.1.2 4.6.3B

In this we define a function that takes two integer inputs  $x$  and  $y$  (where  $x$  is supplied first followed by  $y$ ) and returns the boolean value corresponding to  $x \leq y$ .

```
val funB1 = (fn x => (fn y => x < y));  
fun funB2 x y = x < y;
```

### 2.1.3 4.6.3C

In this we define a function that takes two strings  $s_1$  and  $s_2$  (where  $s_1$  is supplied first followed by  $s_2$ ) and concatenates them, where `^` denotes string concatenation. For example, "Hi" ^ "there" results in the string "Hi there".

```
val funC1 = (fn s1 => (fn s2 => s1 ^ s2));  
fun funC2 s1 s2 = s1 ^ s2;
```

### 2.1.4 4.6.3D

In this we define a function that takes two lists  $list_1$  and  $list_2$  (where  $list_1$  comes first) and appends them, where `@` denotes list append. For example `[true,false] @ [false, false, false]` results in the list `[true,false,false,false,false]`.

```
val funD1 = (fn l1 => (fn l2 => l1@l2));  
fun funD2 l1 l2 = l1@l2;
```

### 2.1.5 4.6.3E

In this we define a function that takes a pair of integers  $(x, y)$  and returns the larger of the two values. You note that the conditional statement `if condition then a else b` returns  $a$  if `condition` is true, otherwise it returns  $b$ .

```

val funE1 = (fn (x,y) => if (x>y) then x else y);
fun funE2 (x,y) = if (x>y) then x else y;

```

### 2.1.6 Test Cases

Below are the test cases to evaluate.

---

```

(*****)
(* Part A *)
(*****)

val testListA = [(1,2,3),(4,5,6),(7,8,9)]

val outputsA = map funA2 testListA

val testResultA = test463A funA1 funA2 testListA

(*****)
(* Part B *)
(*****)

val testListB = [(0,0),(1,2),(4,3)]

val outputsB = map (f2P funB1) testListB

val testResultB = test463B funB1 funB2 testListB

(*****)
(* Part C *)
(*****)

val testListC = [("Hi","_there!"),("Oh","_no!"),("What","_the_...")]

val outputsC = map (f2P funC1) testListC

val testResultC = test463B funC1 funC2 testListC

(*****)
(* Part D *)
(*****)

val testListD1 = [([0,1],[2,3,4]),([],[0,1])]
val testListD2 = [([true,true],[])]

val outputsD1 = map (f2P funD1) testListD1
val outputsD2 = map (f2P funD2) testListD2

val testResultD1 = test463B funD1 funD2 testListD1
val testResultD2 = test463B funD1 funD2 testListD2

(*****)

```

---



```
(* Part E *)
(*****)
```

```
val testListE = [(2,1),(5,5),(5,10)]
```

```
val sampleResultE = map funE1 testListE
```

```
val testResultE = test463A funE1 funE2 testListE
```

## 2.2 Execution Transcripts

<pre>----- HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]  For introductory HOL help, type: help "hol"; To exit type &lt;Control&gt;-D ----- &gt; &gt; &gt; &gt; # # # # # val test463A = fn: ('a -&gt; 'b) -&gt; ('a -&gt; 'b) -&gt; 'a list -&gt; bool &gt; &gt; # # # # # val f2P = fn: ('a -&gt; 'b -&gt; 'c) -&gt; 'a * 'b -&gt; 'c val test463B = fn:   ('a -&gt; 'b -&gt; 'c) -&gt; ('a -&gt; 'b -&gt; 'c) -&gt; ('a * 'b) list -&gt; bool &gt; *** Emacs/HOL command completed ***  &gt; val funA1 = fn: int * int * int -&gt; int &gt; val funA2 = fn: int * int * int -&gt; int &gt; # # # # # val outputsA = [6, 15, 24]: int list val testListA = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]: (int * int * int) list val testResultA = true: bool &gt; val funB1 = fn: int -&gt; int -&gt; bool &gt; val funB2 = fn: int -&gt; int -&gt; bool &gt; &gt; # # # # # val outputsB = [false, true, false]: bool list val testListB = [(0, 0), (1, 2), (4, 3)]: (int * int) list val testResultB = true: bool &gt; val funC1 = fn: string -&gt; string -&gt; string &gt; val funC2 = fn: string -&gt; string -&gt; string &gt; &gt; # # # # # val outputsC = ["Hi there!", "Oh no!", "What the ..."]: string list val testListC = [("Hi", " there!"), ("Oh ", "no!"), ("What", " the ...")]:   (string * string) list val testResultC = true: bool &gt; val funD1 = fn: 'a list -&gt; 'a list -&gt; 'a list &gt; val funD2 = fn: 'a list -&gt; 'a list -&gt; 'a list &gt; # # # # # val outputsD1 = [[0, 1, 2, 3, 4], [0, 1]]: int list list val outputsD2 = [[true, true]]: bool list list val testListD1 = [[([0, 1], [2, 3, 4]), ([], [0, 1])]:   (int list * int list) list val testListD2 = [[([true, true], [])]: (bool list * 'a list) list val testResultD1 = true: bool val testResultD2 = true: bool &gt; val funE1 = fn: int * int -&gt; int &gt; val funE2 = fn: int * int -&gt; int &gt; # # # # # val sampleResultE = [2, 5, 10]: int list val testListE = [(2, 1), (5, 5), (5, 10)]: (int * int) list val testResultE = true: bool &gt;</pre>	1
--	---

### 2.2.1 Explanation of Results

All the results in the test cases shows they are passed against the given test function.

# Exercise 4.6.4

---

## 3.1 Problem Statement

In this exercise we need to solve the list concatenation as stated below:

In ML, define a function `listSquares` that when applied to the empty list of integers returns the empty list, and when applied to a non-empty list of integers returns a list where each element is squared. For example, `listSquares [2,3,4]` returns `[4,9,16]`. Define the function using a `let` expression in ML. A function that takes two lists `list 1` and `list 2` (where `list 1` comes first) and appends them, where `'@'` denotes list append. For example `[true,false] @ [false, false, false]` results in the list `[true,false,false,false,false]`.

## 3.2 Relevant Code

```
fun listSquares list =  
let  
  fun squareNum x = x*x  
  in  
    map squareNum list  
  end;
```

## 3.3 Test Cases

The required test cases are:

```
val testList = [1,2,3,4,5]
```

## 3.4 Execution Transcripts

```
-----  
HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]  
  
For introductory HOL help, type: help "hol";  
To exit type <Control>-D  
-----  
> > > # # # # val listSquares = fn: int list -> int list  
> val testList = [1, 2, 3, 4, 5]: int list  
> val testResults = [1, 4, 9, 16, 25]: int list  
>
```

1

### 3.4.1 Explanation of Results

The above transcript shows the given tests has been passed.

## Exercise 5.3.4

### 4.1 Problem Statement

In this exercise we need to define a function `Filter` in ML, whose behavior is identical to `filter`. Note: you cannot use `filter` in the definition of `Filter`. However, you can adapt the definition of `filter` and use it in your definition. Show test cases of your function returning the expected results by comparing the outputs of both `Filter` and `filter`.

### 4.2 Relevant Code

```
fun Filter l list=
let
  fun fnA l [] = []
    | fnA l xs = map l xs

  fun fnB [] fail = []
    | fnB fail [] = []
    | fnB (b::bs) (x::xs) = if b then x::(fnB bs xs) else fnB bs xs
in
  fnB (fnA l list) list
end;
```

### 4.3 Test Cases

The required test cases are:

```
val testResults = Filter (fn x => x < 5) [1,2,3,4,5,6,7,8,9]
val testResults2 = Filter (fn x => x < 5) [4,6]
```

### 4.4 Execution Transcripts

1

```
-----
HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

For introductory HOL help, type: help "hol";
To exit type <Control>-D
-----
> > > ##### val Filter = fn: ('a -> bool) -> 'a list -> 'a list
> > val testResults = [1, 2, 3, 4]: int list
> val testResults2 = [4]: int list
>
```

#### **4.4.1 Explanation of Results**

The above transcript shows the given tests has been passed.

## Exercise 5.3.5

### 5.1 Problem Statement

In this exercise we need to define a ML function `addPairsGreaterThan n list`, whose behavior is defined as follows: (1) given an integer `n`, and (2) given a list of pairs of integers `list`, `addPairsGreaterThan n list` will return a list of integers where each element is the sum of integer pairs in `list` where both elements of the pairs are greater than `n`.

### 5.2 Relevant Code

```
filter ;

fun addPairsGreaterThan n list =
let
fun sumList [] = []
  | sumList ((x,y) :: xs) = (x+y) :: (sumList xs)

fun fil n (x,y) = (x>n andalso y>n)
in
sumList (filter (fil n) list)
end;
```

### 5.3 Test Cases

The required test cases are:

```
addPairsGreaterThan 0 [(0,1),(2,0),(2,3),(4,5)];
```

### 5.4 Execution Transcripts

1

```
-----
HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

For introductory HOL help, type: help "hol";
To exit type <Control>-D
-----
> > > val it = fn: ('a -> bool) -> 'a list -> 'a list
> ##### val addPairsGreaterThan = fn: int -> (int * int) list -> int list
> > val it = [5, 9]: int list
>
```

#### 5.4.1 Explanation of Results

The above transcript shows the given tests has been passed.

## Exercise 6.2.1

---

### 6.1 Problem Statement

For the given sub-problems show the HOL equivalent code.

#### 6.1.1 6.2.1.1

Enter the HOL equivalent of  $P(x)$ . Show what HOL returns. What are the types of  $x$ ,  $y$ ,  $P$ , and  $Q$ ?

---

# Appendix A: Exercise 4.6.3

---

The following code is from the file ex-4-6-3Tests.sml.

```
(*****)  
(* Exercise 4.6.3 *)  
(* Author: Shiu-Kai Chin *)  
(* Modified — Added function code: Bharath Karumudi *)  
(* Date: Jul 19, 2019 *)  
(*****)  
  
(*****)  
(* Test functions you will need. *)  
(* *)  
(* *)  
(*****)  
  
fun test463A f1 f2 inList =  
  let  
    val list1 = map f1 inList  
    val list2 = map f2 inList  
  in  
    foldr  
      (fn (x,y) => (x andalso y))  
      true  
      (ListPair.map (fn (x,y) => x = y) (list1 , list2))  
  end;  
  
fun f2P f (x,y) = f x y  
  
fun test463B f1 f2 inList =  
  let  
    val list1 = map (f2P f1) inList  
    val list2 = map (f2P f2) inList  
  in  
    foldr  
      (fn (x,y) => (x andalso y))  
      true  
      (ListPair.map (fn (x,y) => x = y) (list1 , list2))  
  end;
```

---

```

(*****)
(* Part A *)
(*****)

(* ===== *)

(* function A1, A2 *)

val funA1 = (fn (x,y,z) => x+y+z);
fun funA2 (x,y,z) = x+y+z;

(* ===== *)

val testListA = [(1,2,3),(4,5,6),(7,8,9)]

val outputsA = map funA2 testListA

val testResultA = test463A funA1 funA2 testListA

(*****)
(* Part B *)
(*****)

(* ===== *)
(* function B1, B2 *)

val funB1 = (fn x => (fn y => x < y));
fun funB2 x y = x < y;

(* ===== *)

val testListB = [(0,0),(1,2),(4,3)]

val outputsB = map (f2P funB1) testListB

val testResultB = test463B funB1 funB2 testListB

(*****)
(* Part C *)
(*****)

(* ===== *)
(* function C1, C2 *)

val funC1 = (fn s1 => (fn s2 => s1 ^ s2));
fun funC2 s1 s2 = s1 ^ s2;

(* ===== *)

val testListC = [("Hi", "there!"), ("Oh", "no!"), ("What", "the...")]

val outputsC = map (f2P funC1) testListC

```

---



---

```
val testResultC = test463B funC1 funC2 testListC
```

```
(*****)
(* Part D *)
(*****)
```

```
(* ===== *)
(* function D1, D2 *)
```

```
val funD1 = (fn l1 => (fn l2 => l1@l2));
fun funD2 l1 l2 = l1@l2;
```

```
(* ===== *)
```

```
val testListD1 = [([0,1],[2,3,4]),([],[0,1])]
val testListD2 = [([true,true],[])]
```

```
val outputsD1 = map (f2P funD1) testListD1
val outputsD2 = map (f2P funD2) testListD2
```

```
val testResultD1 = test463B funD1 funD2 testListD1
val testResultD2 = test463B funD1 funD2 testListD2
```

```
(*****)
(* Part E *)
(*****)
```

```
(* ===== *)
(*function E1, E2 *)
```

```
val funE1 = (fn (x,y) => if (x>y) then x else y);
fun funE2 (x,y) = if (x>y) then x else y;
```

```
(* ===== *)
```

```
val testListE = [(2,1),(5,5),(5,10)]
```

```
val sampleResultE = map funE1 testListE
```

```
val testResultE = test463A funE1 funE2 testListE
```

## Appendix B: Exercise 4.6.4

---

The following code is from the file ex-4-6-4Tests.sml.

```
(*****)  
(* Exercise 4.6.4 *)  
(* Author: Shiu-Kai Chin *)  
(* Modified — Added function code: Bharath Karumudi *)  
(* Date: Jul 19 2019 *)  
(*****)  
  
(* ===== *)  
(* function listSquares *)  
  
fun listSquares list =  
  let  
    fun squareNum x = x*x  
    in  
      map squareNum list  
    end;  
  
(* ===== *)  
  
val testList = [1,2,3,4,5]  
  
val testResults = listSquares testList
```

## Appendix C: Exercise 5.3.4

The following code is from the file ex-5-3-4Tests.sml.

```
(***** *)
(* Exercise 5.3.4 *)
(* Author: Shiu-Kai Chin *)
(* Modified - Added function code: Bharath Karumudi *)
(* Date: 20 September 2015 *)
(***** *)

(* ===== *)
(* function Filter *)

fun Filter l list=
let

fun fnA l []=[]
  | fnA l xs=map l xs

fun fnB [] fail=[]
  | fnB fail []=[]
  | fnB (b::bs) (x::xs)=if b then x::(fnB bs xs) else fnB bs xs
in
fnB (fnA l list) list
end;

(* ===== *)

val testResults = Filter (fn x => x < 5) [1,2,3,4,5,6,7,8,9]

(* specified test cases *)
val testResults2 = Filter (fn x => x < 5)[4,6]
```

## Appendix D: Exercise 5.3.5

The following code is from the file ex-5-3-5Tests.sml.

```
(***** *)
(* Exercise 5.3.5 *)
(* Author: Shiu-Kai Chin *)
(* Date: 20 September 2015 *)
(***** *)

(* ===== *)
(* function addPairsGreaterThan *)

filter ;

fun addPairsGreaterThan n list =
let
fun sumList [] = []
  | sumList ((x,y) :: xs) = (x+y) :: (sumList xs)

fun fil n (x,y) = (x>n andalso y>n)
in
sumList (filter (fil n) list)
end;

(* ===== *)

addPairsGreaterThan 0 [(0,1),(2,0),(2,3),(4,5)];
```