

Gabbi Forsythe

Dr. Rybarczyk

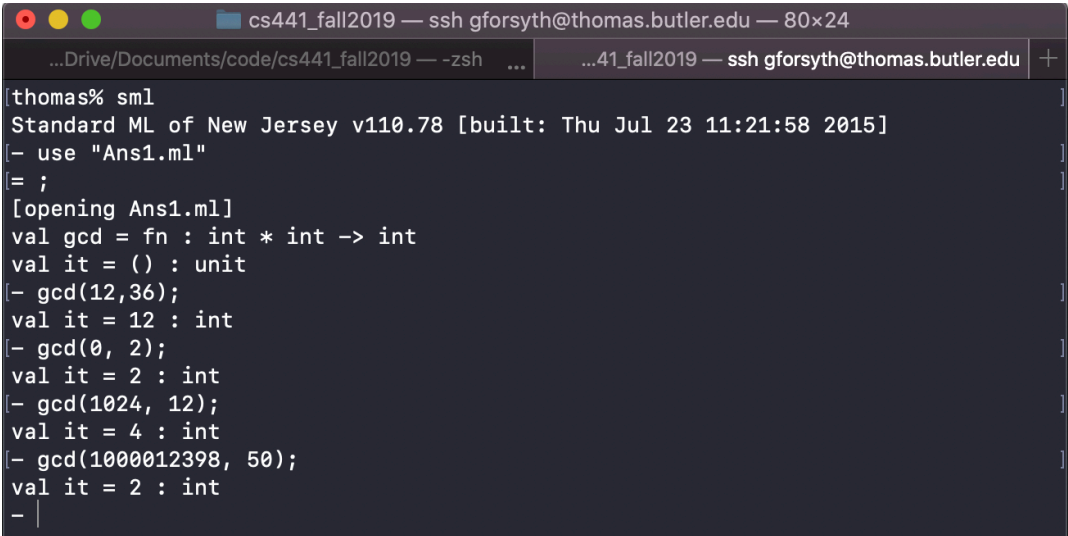
CS441

13 December 2019

Assignment #6 Report: SML

#1 – TESTING AND IMPLEMENTATION (Ans1.ml)

In order to calculate the Greatest Common Divisor (GCD), I first had to remember what the GCD is and the best way to calculate it. So, obviously I googled it and read the Wikipedia article about the GCD. It was on this page where I came across, Euclid's algorithm of subtracting the numbers to find the GCD. It seemed simple enough if it was done recursively so I decided to do it this way. The numbers are subtracted from one another until they are equivalent to each other, that number is the GCD. The program does not handle negative numbers, but according to the specifications it is not supposed to. So, I decided not to handle negative values in my program. Below are the tests that were run:



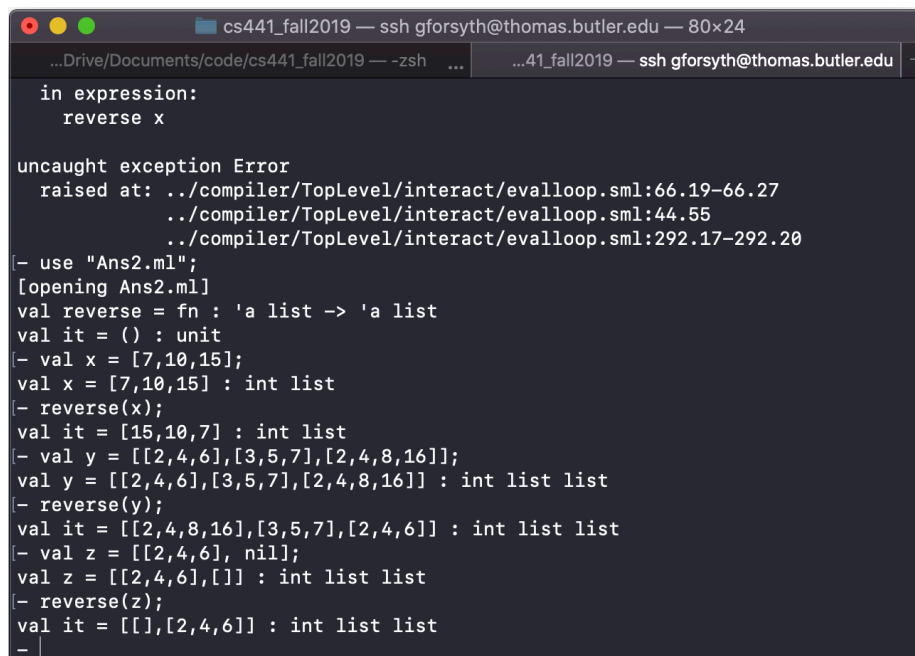
```
cs441_fall2019 — ssh gforsyth@thomas.butler.edu — 80x24
...Drive/Documents/code/cs441_fall2019 — zsh ...
...41_fall2019 — ssh gforsyth@thomas.butler.edu +

[thomas% sml
Standard ML of New Jersey v110.78 [built: Thu Jul 23 11:21:58 2015]
[- use "Ans1.ml"
= ;
[opening Ans1.ml]
val gcd = fn : int * int -> int
val it = () : unit
[- gcd(12,36);
val it = 12 : int
[- gcd(0, 2);
val it = 2 : int
[- gcd(1024, 12);
val it = 4 : int
[- gcd(1000012398, 50);
val it = 2 : int
- |
```

Photo 1: Testing of Ans1.ml

#2 – TESTING AND IMPLEMENTATION (Ans2.ml)

When I tried to do this the first time, I did not know what I was doing. So, I gave up and moved on to the third part. After finishing the insertion sort portion, I realized how simple this part actually is, especially if recursion is involved. Also, it helped that I knew the proper syntax for lists and recursion from writing the insertion sort algorithm, so that all made this one much easier. This basically takes the original list and takes the elements off of the front (using `::`) and puts them in the back (using `@`). For testing, I used the same test cases from the assignment document, these test cases included normal lists of integers, lists of lists, and list containing empty lists, to they covered much of the possible input. Below are the tests that were run:



```

cs441_fall2019 — ssh gforsyth@thomas.butler.edu — 80x24
...Drive/Documents/code/cs441_fall2019 — -zsh ...
...41_fall2019 — ssh gforsyth@thomas.butler.edu +

in expression:
  reverse x

uncaught exception Error
  raised at: ../compiler/TopLevel/interact/evalloop.sml:66.19-66.27
             ../compiler/TopLevel/interact/evalloop.sml:44.55
             ../compiler/TopLevel/interact/evalloop.sml:292.17-292.20
[- use "Ans2.ml";
[opening Ans2.ml]
val reverse = fn : 'a list -> 'a list
val it = () : unit
[- val x = [7,10,15];
val x = [7,10,15] : int list
[- reverse(x);
val it = [15,10,7] : int list
[- val y = [[2,4,6],[3,5,7],[2,4,8,16]];
val y = [[2,4,6],[3,5,7],[2,4,8,16]] : int list list
[- reverse(y);
val it = [[2,4,8,16],[3,5,7],[2,4,6]] : int list list
[- val z = [[2,4,6], nil];
val z = [[2,4,6], []] : int list list
[- reverse(z);
val it = [[], [2,4,6]] : int list list
-

```

Photo 2: Testing of Ans2.ml

#3 – TESTING AND IMPLEMENTATION (Ans3.ml)

This implementation of insertion sort was very similar to the implementation that I did in Prolog for Assignment #5. I used a helper function called `insert` and then the main function

`sort`, and both functions are basically just ML versions of the Prolog implementation. The most difficult part was trying to figure out the proper syntax for recursive functions and figuring out how to properly call functions within other functions (ex. line 24 in `Ans3.ml`). As far as testing goes, I put in a random list with some duplicate values and then called the function and received the expected output. I also tested the program with negative numbers (which shouldn't have any effect, it was more of a "just in case" type of thing), and the output received was what I expected.

Below are the test runs:

```

Last login: Thu Dec 12 15:12:42 2019 from 10.7.15.110
thomas% cd public_html
thomas% cd cs441_fall2019/
thomas% cd a6
thomas% sml
Standard ML of New Jersey v110.78 [built: Thu Jul 23 11:21:58 2015]
|- use "Ans3.ml";
[opening Ans3.ml]
val insert = fn : int -> int list -> int list
val sort = fn : int list -> int list
val it = () : unit
|- val x = [2,3,5,2,6,4,10];
val x = [2,3,5,2,6,4,10] : int list
|- sort(x);
val it = [2,2,3,4,5,6,10] : int list
|- val y = [2,3,6,4,~3,3,~9,0];
val y = [2,3,6,4,~3,3,~9,0] : int list
|- sort(y);
val it = [~9,~3,0,2,3,3,4,6] : int list
-
```

Photo 3: Testing of `Ans3.ml`

FINAL THOUGHTS ON SML

Overall, SML shared a lot of similarities with imperative languages, such as Java and C++. Things like recursion and if-else statements made it feel like I was using Java or C++. Everything in SML has a type which means it is important to know what type each function and variable is, because the values aren't coerced like in C++ or Java. For me the hardest part about

SML is all about math, and I don't like math, therefore, I didn't like using SML. Another thing I noticed was that when I use languages like Java or C++, I avoid recursion at all costs. However, when I used SML the language made more sense to me when I used recursion. Maybe now that I have used recursion so much in SML, I will be more inclined to use it in other programming languages. Overall, even though I don't like math, I thought using SML was a good experience and challenged my programming skills.