COSC 455

Programming Languages: Design and Implementation

Fall 2025

**Lab Exercise #6**

*Whereas some declarative programmers only pay lip service to equational reasoning, users of functional languages exploit them every time they run a compiler, whether they notice it or not.*

-- Philip Wadler

**Name: Blessing Abumere / 20**

**Goals:** The intention of this lab exercise is to introduce you to a pure functional programming language and gain experience in understanding and developing simple SML functions. In doing so, you should develop an initial understanding of a new programming paradigm and better programming skills.

**Language/Compiler:** For this lab exercise, we will use the Standard ML of New Jersey (SML/NJ) compiler for the Standard ML ’97 (<http://www.smlnj.org/sml97.html>) programming language. The SML/NJ compiler is open source and freely available to download and install on various platforms at <http://www.smlnj.org/dist/working/110.72/index.html>, but the easiest (and recommended) way to use it is through the online REPL SOSML IDE at <https://sosml.org/>.

In addition, the Rust programming language (<https://rustlang.org/>) and the Rust Playground <https://play.rust-lang.org/> should be used for the Rust programming problems.

**Deadline:** Submitted via Blackboard by 11:59pm on Friday, October 17, 2025

**Submission:** The submission of this lab exercise to Blackboard should be submitted in a single source file (i.e., \*.doc, \*.odt, \*.txt, etc.) with a naming convention of *FirstNameLastNameLab6*. For example, if your name is Homer Simpson, you would have a file named *HomerSimponLab6.doc*. The SML and Rust code portion of this lab should include the SOSML and Rust Playground links to your developed code, as requested. ***Lab submissions not following this convention may not be graded.***

1. **SML Types and Type Inferencing?** (2 points) Consider the following SML function that calculates the cube of a given parameter:

Within the SOSML interactive environment, enter in this function and test it with the following commands:

* cube 2;
* cube 2.0;
* val x = cube;
* cube x;
* val y = cube;
* y x;

A screenshot of a computer

AI-generated content may be incorrect.

***Does the SML cube function provide the results you expected given its definition? Why or why not? Provide a screenshot of your interaction with the SML environment from above.***

* The cube function surprised me for 2.0 input because I thought 2.0 would work the same way 2 does but it didn’t.

As you have likely noticed, after finishing defining any function the SML interactive environment provides cryptic messages. For example, after defining your cube function the SML interactive environment should have printed:

* val cube = fn : int -> int

***Briefly describe what SML is trying to tell you with these messages.***

I believe SML is trying to tell me the type of parameters it inferred from the way the function is written and what its currently going to return

Consider a Java implementation of a method that cubes a provided parameter:

int cube(int x) {

return x \* x \* x;

}

***When you compare the Java implementation with that of SML, what is the major syntactical difference? How does SML handle this difference? From your experience with SML thus far, what is the fundamental language feature differentiating it from Java?***

The major syntactical difference is java is more verbose and doesn’t infer anything here while SML does. I believe the fundamental language feature is that SML is a functional programming language while java is imperative .

1. **SML List Processing.** (2 points) List processing is a very powerful and important part of many functional languages (e.g., SML, Scheme, Lisp, Scala, Rust). Enter in the following sequence of commands in the SOSML interactive environment:

* val aList = “COSC455”::“is”::“fun!”::[];
* aList;
* hd aList;
* hd aList;
* tl aList;
* tl (tl aList);
* hd (tl aList);
* aList;

***What do the “::”, “hd” and “tl” operators do to a list? Provide a screenshot of your interaction with the SML environment from above.***

:: adds to a list.

Hd gets the single head of a list

Tl gets everything after the head

@ adds two lists together

***A screenshot of a computer

AI-generated content may be incorrect.***

***Develop an SML function, named removeThird, that takes a list as a parameter and returns the same list (i.e., list -> list) with the third element in the list removed. You may assume that the provided parameter always has at least 3 elements.***

Once developed, your removeThird function should be tested and produce the following series of interactions:

* val aList = [“COSC455”, “is”, ”not”, “fun!”];
* removeThird aList;
* val it = = [“COSC455”, “is”, “fun!”]

***Provide a screenshot of your interaction with the SML environment from above.***

A screenshot of a computer

AI-generated content may be incorrect.

1. **SML and Recursion.** (4 points) Recursion is fundamental in functional programming and can be effectively applied in imperative languages. To develop practice in developing recursive solutions and further practice with SML and functional programming languages, develop the following SML functions in SOSML:

* *A function, named pow, that takes two parameters: one of type real and one of type int and returns a real (i.e., real \* int -> real) that is the real parameter raised to the power of the integer parameter. You can assume that the integer parameter is always a positive number.*
* *A function, named* [*collatz*](https://www.youtube.com/watch?v=U7LrFp9RBek)*, tests the* [*collatz conjecture*](http://en.wikipedia.org/wiki/Collatz_conjecture)*. That is, your function should take an integer parameter and returns an integer list (i.e., int -> int list) of the collatz sequence starting with the integer parameter. A* [*collatz sequence*](http://upload.wikimedia.org/wikipedia/commons/1/1c/CollatzFractal.png) *is constructed as follows: given an integer, n, if n is even, divide it by 2; if n is odd, multiple it by 3 and add 1 (i.e., 3n + 1); repeat this process until you reach 1.*

***Provide a share link of your SOSML SML code with these functions here:***

[SML Link 3](https://sosml.org/share/bea9da9fcc1c91cdb493632c0ea7c422d1a70fbbd860494b1bd6506fa631fa03)

1. **SML with Pattern Matching, Anonymous and Higher-Order Functions.** (8 points) Pattern matching, anonymous functions, and higher-order functions are key features of many functional programming langauges and, more recently, can be found in many imperative languages (e.g., Java, F#, Rust, Scala, etc.). Develop the following SML functions in SOSML:

* *A function, called twins, that takes a list as an argument and returns a list where every element contains a twin. For example, twin [1,2,3] returns the list [1,1,2,2,3,3] and twin [true, false] returns the list [true, true, false, false]. Your solution must use pattern-matching (i.e., you may not use hd or tl and no if-then-else statements).*
* *A function, named convert, of type bool list -> int list that takes a list of booleans and returns the list of integer equivalents (i.e., 1 for true, 0 for false). For example, if you evaluate convert [true, true, false, true] it would return [1, 1, 0, 1].*
* *A function il2rl of type int list -> real list that takes a list of integers and returns a list of the same numbers converted to type real. For example, if you evaluate il2rl [1, 2, 3] it will return [1.0, 2.0, 3.0]. Your solution must use the map curried function.*
* *A function squarelist of type int list -> int list that takes a list of integers and returns the list of squares of those integers. For example, if you evaluate squarelist [1, 2, 3, 4] it would return [1, 4, 9, 16]. Your solution must use the map curried function.*

***Provide a share link of your SOSML SML code with these functions here:***

[SML share link 4](https://sosml.org/share/9d3b9ed9d986fe991baad2787167df46826e193c9f74ed3718f9834954101c65)

1. **Functional Programming in Rust.** (4 points) Using the Rust code provided in this document as Rust solutions for Questions 3 and 4, run the code in the Rust Playground to verify correctness and then:

* *Add comments to the code to demonstrate your understanding and provide a Rust Playground share link below.* 
  + [Rust Playground](https://play.rust-lang.org/?version=stable&mode=debug&edition=2024&gist=a8c3756d70d7b08f0519777fef97a703)
* *Describe below (including specific code examples from the provided solution) how Rust adopts the functional programming ideas of pattern matching, anonymous functions, and higher-order functions.* 
  + An example of Rust adopting pattern matching would be in the convert function . It matches the boolean variable b with either being true or false and returns 1 if true and 0 for false in the end of the match.
  + An example of rust adopting anonymous functions in this function would be the |&b| being passed into the map function so that it takes b and changes it

    pub fn convert(xs: &[bool]) -> Vec<i32> {

        xs.iter().map(|&b| match b {

            true => 1,

            false => 0,

        }).collect()

    }

* + An example of Rust adopted higher order functions would be in the il2rl function. In the .map() part it takes in a function called to\_real as a parameter and uses it to change values

pub fn il2rl(xs: &[i32]) -> Vec<f64> {

    xs.iter().copied().map(to\_real).collect()

}

* *Using Google, AI, and/or any LLM, research and in your own words describe below how Java (or your favorite, commonly used language) incorporates the functional programming ideas of pattern matching, higher-order functions, and anonymous functions (if at all) into the language.* 
  + My favorite commonly used language, C-Sharp, incorporates pattern matching with its switch statement.
  + C-Sharp incorporates higher-order functions with its built in System.Func class and it allows the creation of delegate type which is pretty much just a pointer to a function but type safe.

System.Func<int,int> cube = x => x\*x\*x;

* + C-Sharps calls anonymous functions lambda expressions and it would be the stuff after “=>” .

// Functional Programming Examples in Rust

// Demonstrating pattern matching, recursion, and map-style higher-order functions.

pub fn pow(base: f64, exp: i32) -> f64 {

match exp {

1 => base,

\_ => base \* pow(base, exp - 1),

}

}

pub fn collatz(n: i32) -> Vec<i32> {

match n {

1 => vec![1],

\_ if n % 2 == 0 => {

let mut seq = vec![n];

seq.extend(collatz(n / 2));

seq

}

\_ => {

let mut seq = vec![n];

seq.extend(collatz(3 \* n + 1));

seq

}

}

}

pub fn twins(xs: &[i32]) -> Vec<i32> {

match xs {

[] => Vec::new(),

[x, rest @ ..] => {

let mut front = vec![\*x, \*x];

front.extend(twins(rest));

front

}

}

}

pub fn convert(xs: &[bool]) -> Vec<i32> {

xs.iter()

.map(|&b| match b {

true => 1,

false => 0,

})

.collect()

}

fn to\_real(x: i32) -> f64 {

x as f64

}

pub fn il2rl(xs: &[i32]) -> Vec<f64> {

xs.iter()

.copied()

.map(to\_real)

.collect()

}

fn square(x: i32) -> i32 {

x \* x

}

pub fn squarelist(xs: &[i32]) -> Vec<i32> {

xs.iter()

.copied()

.map(square)

.collect()

}

fn main() {

// pow

println!("pow(2.0, 3) = {}", pow(2.0, 3)); // 8.0

println!("pow(1.5, 4) = {}", pow(1.5, 4)); // 5.0625

println!();

// collatz

println!("collatz(6) = {:?}", collatz(6)); // [6, 3, 10, 5, 16, 8, 4, 2, 1]

println!();

// twins

println!("twins([1,2,3]) = {:?}", twins(&[1, 2, 3]));

println!("twins([]) = {:?}", twins(&[]));

println!();

// convert

println!("convert([true,true,false,true]) = {:?}", convert(&[true, true, false, true]));

println!();

// il2rl

println!("il2rl([1,2,3]) = {:?}", il2rl(&[1, 2, 3]));

println!();

// squarelist

println!("squarelist([1,2,3,4]) = {:?}", squarelist(&[1, 2, 3, 4]));

}