COMP 212 : Functional Programming, Spring 2022

Homework 01

Name: _____

Wes Email: _____

Question	Points	Score
1	5	
2	14	
3	15	
4	6	
5	19	
Total:	59	

If possible, please type/write your answers on this sheet and upload a copy of the PDF to your google drive handin folder. Otherwise, please write the answers in some sort of word processor and upload a PDF. Please name the file `hw01-written.pdf`.

1. **Collaboration Policy**

Read the collaboration policy on the course website. Then, for each of the following situations, decide whether or not the students' actions are permitted by the policy. Explain your answers.

(1) (a) Ted and Sam are discussing a regular problem over Zoom. Meanwhile, Ted is writing up his solution to that problem, and hands in what he writes during the call.

Solution:

(1) (b) Lizzie and Max eat lunch (at noon) while talking about a regular homework problem, and by the end of lunch, they have covered their napkins with notes and solutions. They throw out all of the napkins and go to class from 1pm-5pm. Then, each individually writes up their solution.

Solution:

(1) (c) Robby and Emily write out a solution to a regular homework problem on a whiteboard. Then, they run home and each student types up the solution.

Solution:

(1) (d) Ben shares his screen with Justin during a Zoom meeting to show him how to set

up Atom, and Ben's solution to a homework problem is open in Atom at the time.

Solution:

(1) (e) George asks Caroline for help with a challenge problem.

Solution:

2. **Type Checking and Evaluation**

In this section we will explore the step-by-step reasoning of type checking and evaluation to better understand when an SML expression is well-typed, what its type is, how it will evaluate, and what its value will be. We will also do some basic analysis of the number of steps in the evaluation of an expression.

We will start with an example. Consider the expression `intToString 7` and assume that we know `intToString : int -> string`. To determine the type of this expression, we first note that the expression `7` has the type `int`. Now, using this and the fact that `intToString` has the type `int -> string` we conclude that the application of these two expressions has the type `string` since the first expression has a function type and the type of the second expression matches the type of the argument for this function.

(2) (a) Determine the type of the expression:

```
(intToString 3) ^ (intToString 6)
```

Describe your reasoning in the same manner as the example.

Solution:

(2) (b) Explain why the expression, `intToString "1"`, is not well-typed.

Solution:

(2) (c) We will now look at an example of reasoning about evaluation. Consider the expression `(intToString 7) ^ "1"` and assume that we know the application,

`(intToString 7)`, evaluates to the value `"7"`. Note that the value `"1"` evaluates to itself. Using these two facts, we conclude that the whole expression evaluates to `"71"` since the `^` operator evaluates its two subexpressions and then evaluates to the concatenation of the two strings that result from these evaluations.

Determine the value that results from the evaluation of the expression:

```
intToString (4 + 4)
```

Describe your reasoning in the same manner as the example.

Solution:

(d) Recall, from Lecture 2, the following categories of expressions, each of which is strictly larger than the next:

- syntactically correct expressions
- well-typed expressions
- valuable expressions
- values

For each of the following expressions, state the **most specific** set that the expression belongs to. Briefly explain why your answer is correct.

(2) i. `f (5 div (1 - 1))` where
`fun f (x : int) : int = 47`

Solution:

(2) ii. `6`

Solution:

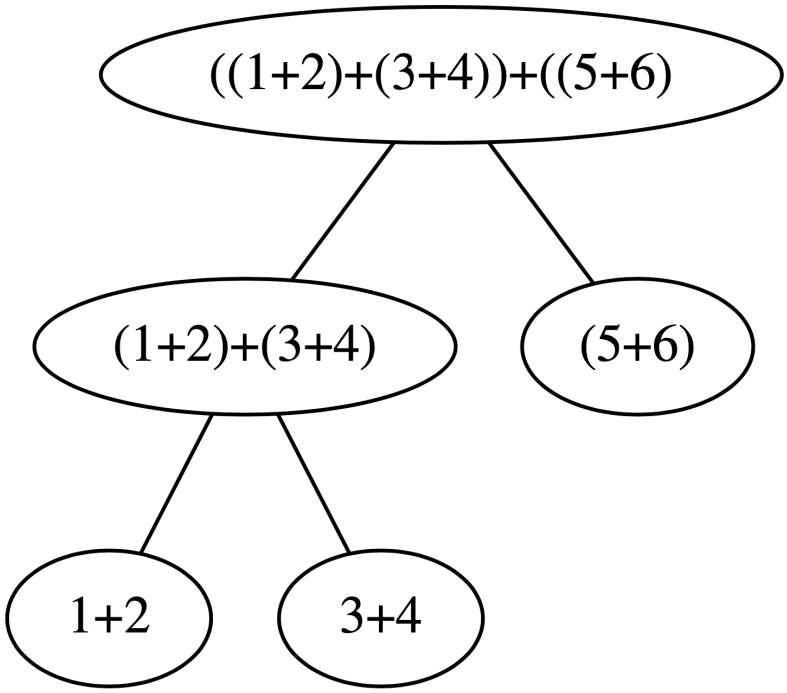
(2) iii. `1+1`

Solution:

(2) iv. `(intToString 5) + 1`

Solution:

3. **Parallel Computing** We can draw an expression, such as `((1 + 2) + (3 + 4)) + (5 + 6)`, as a tree, such as



Each circle is a *node* of the tree.[1] The *work*, W, of the expression tree is the total number of nodes, while the *span*, S, is the number of nodes along the longest path from the top to the bottom.

Regardless of the number of processors used to evaluate a compound expression in parallel, the number of steps required to calculate the final result must be at least as great as the span because evaluating one node requires first evaluating the node below it (a *data dependency*). Also, if each of P processors performs one evaluation step in parallel during each *time cycle*, it would require at least W/P (rounded up) time cycles to perform all W operations. Thus, you can't do better than W/P and you can't do better than S. Putting these together, this means you cannot run a calculation in time less than $\max(W/P, S)$, so we say this is a *lower bound* on the time. However, not all calculations can be done in exactly this lower bound. But, it turns out that you can run a calculation in time *proportional to* this lower bound, which is known as Brent's Principle:

Theorem 1 (Brent's Principle)**.** *If an expression, e has work W and span S, then evaluating e on a P-processor machine takes time proportional to* $\max(W/P, S)$.

(4) (a) What are the work W and span S for the above tree in Figure 3?

[1] There should be an additional row with the numeral arguments of each bottommost operation, but these are not counted by the work and span, so we leave them out.

Solution:

When S is greater than W/P, a calculation is *limited by the span*—the running time cannot be less than the span. Conversely, when W/P is greater than S, the running time for a calculation is *limited by the work*—there is enough work to do that W/P, rather than S, is the overall limiting factor.

(2) (b) Suppose you have $P = 1$ processor. Is the calculation limited by the work or the span (i.e., which is bigger, W/P or S?)? Explain.

Solution:

(2) (c) Suppose you have $P = 2$ processors. Is the calculation limited by the work or by the span? Does having 2 processors speed up the calculation relative to having 1 processor? Explain.

Solution:

(2) (d) Suppose you have $P = 3$ processors. Is the calculation limited by the work or the span? Does having 3 processors speed up the calculation relative to having 2 processors?

Solution:

(5) (e) In the first lecture, "we" acted out the process of counting the number of students in the class who took 211 in C and analyzed the work and span of this process. Describe a different real-life process in which parallelism occurs, and analyze its work and span.

Solution:

4. **Interpreting Error Messages**

   Download the file `hw01.sml` from the assignments page.

   You can evaluate the SML declarations in this file using the command

   ```
   use "hw01.sml";
   ```

   at the SML REPL prompt (start smlnj in the same directory as where you saved the file). Unfortunately, the file has some errors that must be corrected. The next tasks will guide you through the process of correcting these errors. (If you fix the errors differently than I had in mind, you might find more or fewer errors than the boxes below, which is fine — please describe things in whichever boxes make sense for the way you did it.)

(2) (a) What is the first error message do you see when you evaluate the unmodified `hw01.sml` file? What caused this error and how can it be fixed?[2]

[2] *Hint:* Compare the syntax of `double` and `intToString`. What is different?

Solution:

Correct this one error in the `hw01.sml` file and load it again.

(2) (b) Now what is the first of the remaining errors? What caused this error?

Solution:

Again, correct just this one error in the `hw01.sml` file and reload it.

(2) (c) What is the final error message? What does this error message mean? How can you fix it?

Solution:

When you correct this final error and evaluate the file there should be no more error messages.

5. **Writing Functions**

For this problem, you should code your answer in `hw01.sml`*, and hand it in to your handin directory on google drive. See the directions on the Resources tab of the course web site for installing VSCode or Atom or Sublime Text, if you didn't install an editor in lab already. The workflow for this problem is to write the code in a text editor, and then in the terminal in SMLNJ* `use` *the file like you did in the final task in lab to load it and run your code.*

(7) (a) Recall the representation of rectangles from Thursday's class: a rectangle is a pair of points, representing its lower left and upper right corners; a point is a pair of integers, representing its x and y coordinates. Design a function that translates a rectangle vertically upwards by a given amount, producing a new rectangle.

The amount should be an input to the function, not a fixed number. **Hint:** Think of a function that takes in two inputs as a function that takes a pair as input — in this sense, the `area` example from class is a two-input function (it takes two points), and also a four-input function (it takes four numbers).

(3) (b) For this problem, we say that a medical record consists of a person's name, birth date, and medical history. Design an SML type representing medical records.

(9) (c) Write a function `update` that takes a medical record and a new note about their care and produces their updated medical record.

For both functions, be sure to follow the design recipe:

1. Write the function's name and type.
2. Write a purpose statement that clearly explains what the function does.
3. Write at least one example of how the function should behave.
4. Write the code.
5. Turn your example into a test case.

The purpose and examples should be in a comment above the function definition. Comments in SML start with `(*` and end with `*)`. These are called "block comments", because there is a symbol to start and end them; there is no "line comment" (ends at the end of the line) syntax in SML.

For example, here is a sample problem and what the whole design recipe should look like.

Problem: Design a function that creates a square, i.e. a rectangle with both sides the same length.

Answer:

```
(* Purpose: Given an integer n, create a rectangle with both sides of
length n.  Since the problem doesn't specify where the rectangle should
be located, this function puts the lower-left corner at (0,0).
```

```
Examples:
make_square 4 should be ((0,0),(4,4))
make_square 6 should be ((0,0),(6,6))
*)

fun make_square(n : int) : rect = ((0,0),(n,n))

(* I tested this interactively by typing the following into SMLNJ:
- make_square 4;
val it = ((0,0),(4,4))
- make_square 6;
val it = ((0,0),(6,6))
*)
```