Lab Extra Credit

Tyler Gillette

Due Tuesday November 20th 6pm

Write the following assignment, then turn it in on BB as a PDF.

Choose some set of questions from Labs 1-8 (they do not have to be questions you missed), and for each question do the following:

1. Rephrase the solution in your own words
    1. for Coding questions, give a brief explanation of what the code is doing
2. List all relevant sections and figures in the book that relate to this question, include page numbers
3. Come up with a reason for why you think we asked the question. Think about what main topics the question was trying to get at.
4. Write your own similar question and provide a solution.

Lab 2

```
% 3. Write a lazy function (section 1.8) that generates the list
%       N | N-1 | N-2 | ... | 1 | 2 | 3 | ...    where N is a positive number
% Hint: you cannot do this with only one argument

% Program code
/*
declare
fun lazy {G2 N C}
  if N==1 then N|{G2 N+1 1}
    elseif C==0 then N|{G2 N-1 0}
  else N|{G2 N+1 1}
  end
  end

*/
```

Here we are creating a lazy function with the "lazy" keyword. We are creating a for loop that checks if N equals 1 if it does then you would append N to the head of a list and then recursively calls the function G2 passing in the N adding 1 and 1 which is the true or false variable I used to tell if it needs to decrement or increment. If N doesn't equal 1 the you would hit the else if clause which checks C to find out if its 0 to decrement or 1 to increment. If C equals 0 then take N and append It to the head of a list and then for the tail recursively call G2 and pass N and decrement it by 1 and the continue to pass in 0 for C. if C doesn't equal 0 then append N to the head of the list and recursively call G2 passing N adding 1 and then pass in 1 so that it continues to increment forever.
Because this is a lazy function it will only compute what is needed. Even though this is a function that would go into an infinite loop. The lazy keyword will prevent it from looping.

Section 1.8 – pg.13-15
Section 2.7 – pg. 100

Lab Extra Credit

Tyler Gillette
Due Tuesday November 20th 6pm

Section 3.6 – pg. 185
Section 3.6.4 -pg. 196
Section 4.5 – pg. 283-309

I think the question was asked because it is a simple way for the student to be able to test out lazy functions. The question specifically asks for a function that would create an infinite loop and its very obvious to the student. This shows a practical use to the student while creating a small challenge as they figure out how to implement the algorithm in the new language oz. The main topics the question was trying to get to was

- How to use oz.
- How lazy functions work
- Why we would use them.

Write a lazy function that recursively adds 5 to a number 10, 15, 20 …

```
declare
fun lazy {Add5 N}
N = N + 5
{Add5 N}
end
```

Lab 2

```
% 4. Write a procedure (proc) that displays ({Browse}) the first N elements of a List
% and run this procedure on the list created in Q3

% Program Code

/*
L = {G2 5 0}

local
  proc {Show N G2}
    if N \= 0 then
   case G2 of H|T then ({Browse H} {Show (N-1) T})
   end
    end
  end
in {Show 9 L}
end
```

Lab Extra Credit

Tyler Gillette
Due Tuesday November 20th 6pm

Create a variable L and call the function from the previous question 3. Now create a local environment and create a proc named Show that takes in two variables one for the number of elements that you want displayed and the function from question 3.
In the proc check if N doesn't equal 0 and if it does not then you would make sure the list isn't empty then browse the head of the list and recursively call the Show function again subtracting 1 from the variable passed in denoting the total number of list elements that you want to display. Also passing in the tail.
This basically takes in a number and list. If the number is not 0 then it browses the head of the list and recursively calls the function again till the number passed in is 0, then it breaks.

Section 2.3.3 – pg. 54
Section 2.4 – pg. 57
Section 2.4 – pg. 59-61
Section 2.4 – pg. 66-67

This is a good question to show if statements, for loops, case statements and Browse. This can get the student comfortable with the language while introducing new syntax with minimal work figuring out the algorithm. This is a good question. This shows how the lazy function will only do what is required and nothing else. This is a great way to view the output of that function to show what is actually happening.

- How to use procedures.
- How to use Browse.
- Using local instances.
- Using case statements.

Write a function that calls the lazy function below and browses the elements that have been created.

```
declare
fun lazy {Add5 N}
N = N + 5
  {Add5 N}
  {Browse N}
end

declare
local
  proc {Show N Add5}
    if N \= 0 then
   case Add5 of H|T then ({Browse H} {Show (N-1) T})
   end
    end
  end
in {Show 9 L}
end
```

Lab Extra Credit

Tyler Gillette
Due Tuesday November 20th 6pm

```
Lab 2
% 7. Define functions {Accumulate N} and {Unaccumulate N} such that the output of
% {Browse {Accumulate 5}} {Browse {Accumulate 100}} {Browse {Unaccumulate 45}}
% is 5, 105, and 60. This will be implemeted using memory cells (section 1.12).

% Program Code

declare
C={NewCell 0}
fun {Accumulate N}
  C:=@C+N
end

fun {Unaccumulate N}
  C:=@C-N
end

{Browse {Accumulate 5}}
{Browse {Unaccumulate 5}}
{Browse @C}
```

Here we are creating a new cell C and initializing it at 0. Then we are creating a function called Accumulate and it takes in a number N. The new Cell at C we are getting the value stored in C adding N to it.

The second function called Unaccumulated takes in a number N and subtracts the value passed in using N from the value held in the cell.

Section 1.12 – pg. 18-19
Section 1.14 – pg. 20-21

This question was asked because it introduces memory cells and allows the student to see how they work in a simple function. The concept and implementation is simple and allows the student to focus on what the memory cell does and less on how to make the function.

- Memory cells
- Functions

Create a function that adds 10 to a memory cell every time it called.

```
declare
C={NewCell 0}
fun {Plus10 N}
  C:=@C+N+10
```

```
end
```

Lab3

Lab 3

1. If a function body has an 'if' statement with a missing 'else' clause, then an exception raised if the 'if' condition is false. Explain why this behavior is correct. This situation does not occur for procedures. Explain why not.

A function must return a single value, whereas a procedure does not return any value, unless using the $. If a function has a missing else clause, then there is a chance, if the else is entered, that the function does not return any value, which breaks the rule that functions return values. This behavior is okay for procedures, because they have no return.

A function must return a value and cannot, not return anything. Whereas a Procedure can return nothing at all or it could return a value. If an "if" statement and its missing the else it could break the function and not return anything.

Functions:
Section 1.3 – pg. 4-6
Section 1.5 – pg. 9-11
Section 1.9 – pg. 15-16
Section 1.14 – pg. 20
Section 2.1 – pg. 40-41
Section 2.3 – pg. 55

Procedures:
Section 2.3.3 – pg. 53-54
Section 2.3.4 – pg. 54-55
Section 2.4 – pg. 57-66

This question was asked because it shows the differences between functions and procedures. It shows why you might use a function over a procedure or vice-versa. The question makes you really think about why you might need a function that doesn't return anything. Such as to call multiple functions from one procedure.

Lab Extra Credit

Tyler Gillette
Due Tuesday November 20th 6pm

What would be a case when it would be better to use a procedure rather than a function? Give an example.

Section 2.3 – pg. 55

"Procedures are more appropriate than objects because they are simpler. Ob- jects are actually quite complicated, as Chapter 7 explains. Procedures are more appropriate than functions because they do not necessarily define entities that behave like mathematical functions.[6] For example, we define both components and objects as abstractions based on procedures. In addition, procedures are flexible because they do not make any assumptions about the number of inputs and outputs. A function always has exactly one output. A procedure can have any number of inputs and outputs, including zero."

```
local U=1 V=2 in {Try}

   proc {$}
    thread

{Try proc {$} U=V end
proc {$} {Browse bing} end}

end end

proc {$} {Browse bong} end} end
```

we could use procedures when we don't need to return anything and just do some function or equation.

Lab 3

2. Using the following:
(1)-ifXthenS1elseS2end
(2) - case X of Lab(F1: X1 ... Fn: Xn) then S1 else S2 end (a) Define (1) in terms of the 'case' statement.

```
case X of true then S1 else S2 end
```

We would need to create a case statement that checks the like situation of X if the situation matches then we would execute S1 else we execute S2 and then exit.

Section 1 – pg. 8

Lab Extra Credit

Tyler Gillette

Due Tuesday November 20th 6pm

This shows how you can use pattern matching in functions to get the desired outcome similar to if statements or loops but more dynamic.

Create a case statement that would take the head of a list and count how many items are in the list.

Case L of H|T then C+1 {Case T - 1}

Lab 3

2(b) Define (2) in terms of the 'if' statement, using the operations Label, Arity, and '.' (feature selection). Note - Don't forget to make assignment before S1. You should use ... when ranging from F1 to Fn.

```
if {Label X} == Lab then

if {Arity X} == local X1 in

     local X2 in
       X1 = X.F1
       X2 = X.F2
       S1

end end

else S2

 end
else S2
end

[F1 F2 .. Fn] then ...
...
```

the first if statement checks that the function {Label X} which takes in one variable equals "Lab" if it does then we are going to enter the second if statement which calls the function {Arity X} which takes in a variable and checks that the Arity equals X1 in the local definition. The local definition defines X1 as X.F1 which is going to check the first element of the key value pair X. X2is equal to X.F2 which is the second value of X. and returns S1 if these two if statements return true, else return S2.

Lab Extra Credit

Tyler Gillette

Due Tuesday November 20th 6pm

Arity:

Section 2.3.5 - pg. 56

Section 2.4 - pg. 69

Label:

Section 2.1 - pg. 38

Section 2.3.3 - pg. 53

Section 2.3 - pg. 55

'.'

Section 2.3 - pg. 55

Section 2.3.5 - pg. 56-57

This question introduces the student to Arity, label and '.'.

This also gives the student the chance to use each of them and see how they work and how they could use them.

I honestly don't know how to make a question that includes transforming the case statement into an if statement and also using Arity, Label and '.'. this is a well thought out question that includes many different elements and challenges.

Lab 3

2(c) Rewrite the following 'case' statement using 'if' statements

```
declare L
L = lab(f1:5 f2:7 f3:'jim')

case L of lab(f1:X f2:Y f3:Z) then case L.f1 of 5 then

      {Browse Y}
   else
      {Browse a}
   end
else
```

```
    {Browse b}

end

if {Label L} == lab then
if {Arity L} == [f1 f3 f3] then

local X Y Z in
X = L.f1 Y = L.f2 Z = L.f3
if L.f1 == 5 then {Browse Y} else {Browse a} end

end
else {Browse b} end

else {Browse b} end
```

First we need to check if the function {Label L} which takes in a variable if its equal to "lab" if it is then check if the function {Arity L} which takes in a variable checks that it equals f1 and f3 and f3. Otherwise you will Browse b.

Arity:

Section 2.3.5 – pg. 56

Section 2.4 – pg. 69

Label:

Section 2.1 – pg. 38

Section 2.3.3 – pg. 53

Section 2.3 – pg. 55

'.'

Section 2.3 – pg. 55

Section 2.3.5 – pg. 56-57

Records:

Section 1.14 – pg. 20

Lab Extra Credit

Tyler Gillette

Due Tuesday November 20th 6pm

Section 2.1 – pg. 37-38

This question helps to cement the idea of interchanging between functions and procedures. It's helpful to see how they compare and contrast and how they can both be used.

Convert the following if statement to a case statement.

Again, this is a really good question because there are so many things included in this, not just the conversion from case statement to an if statement. It also includes the Arity, Label and '.'. which are used later on in the book to further define new ideas.