\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

LAB 08 QUESTIONS

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

- Name: Moti Begna

- NetID: begna002

Answer the questions below according to the lab specification. Write

your answers directly in this text file and submit it to complete the

lab.

PROBLEM 1: Persistent Binary Trees

==================================

(A)

~~~

Examine the provided file `strtree.ml' which implements a few

functions pertaining to a Binary Search Tree on strings. You can see

example uses for these trees in `use\_strtree.ml'. Compile these two

files together and run the resulting executable as in

,----

| > ocamlc strtree.ml use\_strtree.ml

| > ./a.out

| ...

`----

Show the results of the run

**tree1:**

**tree2:**

**0: Mario**

**tree3:**

**1: Luigi**

**0: Mario**

**tree4:**

**1: Luigi**

**0: Mario**

**1: Princess**

**liltree:**

**1: Donkey Kong**

**2: Luigi**

**0: Mario**

**1: Princess**

**2: Toad**

**bigtree:**

**3: Bob-omb**

**2: Bowser**

**4: Bullet Bill**

**3: Buzzy Beetle**

**4: Chain Chomp**

**1: Donkey Kong**

**3: Goomba**

**4: Koopa**

**2: Luigi**

**0: Mario**

**2: Pokey**

**1: Princess**

**3: Thwomp**

**2: Toad**

**3: Wario**

**liltree:**

**1: Donkey Kong**

**2: Luigi**

**0: Mario**

**1: Princess**

**2: Toad**

(B)

~~~

Examine the `strtree.ml' and `use\_strtree.ml' and answer the following

series of short questions.

1. How is the type associated with string trees defined? What OCaml

mechanisms are used?

**strtree implements pattern matching in order to define a segement of a tree as either having a Node or not. If it does have a node, then it uses a record where the node has 3 values: data of type string, left “child” of type strtree (a node) and right “child” of type strtree (another node)**

2. How does one create an empty tree?

**In order to create an empty tree from another file, Strtree.empty (or another alias for Strtree, like ST in the example, must be assigned to a module choosen\_name) is assigned to some given tree.**

**Ex. Let tree1 = ST.Empty**

3. When adding a string to a tree, does it actually change or does

something else happen?

**When adding a string to a tree, a new tree is created with the same values as the previous tree in addition to the new string.**

4. What technique is used in `use\_stretree.ml' to refer to bindings in

the `Strtree' module without writing the entire module name?

**module ST is bound to Strtree in order to shorten the alias**

(C)

~~~

Examine the `Strtree.add' and `Strtree.tree\_string' function and

answer the following questions.

1. How is pattern matching used to decompose the tree structurally?

**Empty and Node(node) values are used to define a segment of a tree. If a Node has children, then it is matched to Node(node), else, it is matched to Empty.**

2. What happens during `add' when a string is not present and the bottom of the tree is reached?

**When a string is not present and the bottom of the tree is reached, a new node is added with the string as its data value, and its left and right values (children) are Empty.**

3. What function is used to compute a "difference" between Node data

and a string being inserted? What is the return value for this

"difference" function?

**String.compare is used to evaluate str and node.data. If str = node.data, 0 is returned. Else if str < node.data, a negative number is returned. Otherwise, a positve number is returned.**

4. During `add', when a left or right branch is "visited", what is

done with the return value of recursive calls to `add'?

**The return value is assigned to either the value for the left or right child of the current node, depending on the compare statement.**

5. What module and type is used in `Strtree.tree\_string' to create a

string representation of the tree? How is it used?

**The Buffer module of type t with function create (Buffer.create) is used in order to create a string representation of a tree. It is used as such…**

**let buf = Buffer.create (number of allocated characters)**

6. How does `Strtree.tree\_string' create different indentation levels

for different nodes during its recursive run?

**Based on the depth of the node, a for loop (from 1 to depth) is used to prepend the string “ “ to the front of the node’s string.**

(D)

~~~

Complete the function `Strtree.getopt'. Use the patterns outlined in

`Strtree.add' to guide your code for `getopt'. Standard

implementations should be 10-15 lines long.

Next complete the related `Strtree.contains' which should be a

one-liner which uses `getopt'.

Demonstrate your functions work in a REPL or through modification of

`use\_strtree.ml'.

Paste your completed code and demo below.

**let rec getopt tree str =**

**match tree with**

**| Empty ->**

**None**

**| Node(node) ->**

**let diff = String.compare str node.data in**

**if diff = 0 then**

**Some str**

**else if diff < 0 then**

**getopt node.left str**

**else**

**getopt node.right str**

**;;**

**let contains tree str =**

**let opt = getopt tree str in**

**match opt with**

**| None ->**

**false**

**| Some a ->**

**true**

**;;**

**# let tree3 = add tree2 "Wario";;**

**val tree3 : strtree =**

**Node**

**{data = "Mario"; left = Empty;**

**right = Node {data = "Wario"; left = Empty; right = Empty}}**

**# getopt tree3 "Mario";;**

**- : String.t option = Some "Mario"**

**# getopt tree3 "Princess";;**

**- : String.t option = None**

**# contains tree3 "Mario";;**

**- : bool = true**

**# contains tree3 "Toad";;**

**- : bool = false**

PROBLEM 2: Module Signatures

============================

Examine the file `sigdemo.ml'. This file declares several nested

modules using `module/struct/end' syntax. Some of these modules also

have explicit \*signatures\* specified using `sig/end' syntax.

Signatures are to modules as types are to values: signatures describe

the contents of the module while types describe the contents of

values.

(A) Default Signatures

~~~~~~~~~~~~~~~~~~~~~~

Open a REPL and issue a `#use' directive to bring in the contents of

`sigdemo.ml' as in

,----

| > ocaml

| OCaml version 4.07.0

|

| # #use "sigdemo.ml";;

| ...

`----

The signatures of each of the modules in the file will be shown though

short module signatures may be on a single line making them harder to

read.

- Is there any difference between the signatures for `All' and

`AllSig'?

- What is the default signature for a module without an explicit

signature?

**All values and functions are publicly accessible in both functions, so besides syntactical differences in the layout of the modules, the signitures are the same. The default signature for a module with no explicit signature is all of the values and functions defined in the module.**

(B)

~~~

Compare the signatures of `All' and `Func1Onlyly' and describe any

differences. Note that the printing of the signature of `Func1Onlyly'

may appear on a single line rather than spread across several lines.

Attempt to access the bindings for `val1,val2,func1' in `All, AllSig,

Func1Onlyly'. Describe your success/failures in accessing

values/functions from the various modules below. Relate this to the

purpose of a module signature.

**Modules like Func1Only show how when you give a signature to only a one value/function within a module, then only that value/function is accessible outside of the module. If you attempt to call a value/function that doesn’t have a signature, then and Unbound Value Error will occur. For example, while you can call any value/function in All and AllSig, you can only call the function func1 in Func1Only.**

Additional Info

~~~~~~~~~~~~~~~

Named Signatures

----------------

Signatures can be named via the `module/type' syntax such as the

definition `ONLY2S'. This can be can shorten code which uses the

signature several times such as with `A2SModule' and `Another2S'.

Module Aliasing and Signatures

------------------------------

Existing modules can be aliased with a new, restricted signature such

as is done with with the module `RestrictAll' that aliases `All' but

makes use of a new signature, `ONLY2S'.

Interface Files: Signatures for Source Code Modules

---------------------------------------------------

Source level modules have signatures as well. By default, all

top-level bindings in the source file are public which may be

undesirable. For a source module file `src.ml' the corresponding

\*interface file\* `src.mli' will specify its signature of publicly

available bindings. An example is given with `counter.ml' and

`counter.mli' in which binding `the\_count' is present in `counter.ml'

but not in `counter.mli'. This means other modules cannot access

`the\_count'. You can try this by compiling with

,----

| > ocamlc counter.mli counter.ml use\_counter.ml

| > ./a.out

| ...

`----

and playing with the application. Then, in `use\_counter.ml', try to

modify `Counter.the\_count' directly and recompile to see an error.