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

LAB 01 QUESTIONS

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

- Name: (FILL THIS in)

- NetID: (THE kauf0095 IN kauf0095@umn.edu)

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: OCaml's REPL

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

(A)

~~~

In a Unix terminal, change into the directory `lab01-code' and then

into the subdirectory `repl-v-compile'. This can be done using the

change directory command `cd' as in

,----

| > cd lab01-code

| > cd repl-v-compile

`----

In this directory should be an OCaml source file named

`print\_stuff.ml'. You can verify it is there by using the listing

command `ls' as in

,----

| > ls

| ... print\_stuff.ml ...

`----

Start the OCaml REPL by typing the command `ocaml' which should change

the prompt as follows.

,----

| > ocaml

| OCaml version 4.06.0

|

| #

`----

The version reported may be different by in most cases this will not

matter.

Load the source file `print\_stuff.ml' into the REPL with the `#use'

directive as follows:

,----

| > ocaml

| OCaml version 4.06.0

|

| # #use "print\_stuff.ml";;

| ....

`----

Copy and paste the results of doing this into the space below as your

answer to this problem.

Solution :solution:

--------

**,----**

**| > ocaml**

**| OCaml version 4.06.0**

**|**

**| # #use "print\_stuff.ml";;**

**| val print\_hello : string -> unit = <fun>**

**| val student : string = "Meatbag #43281"**

**| Hello there!**

**| Welcome to OCaml, Meatbag #43281**

**| - : unit = ()**

**| Remember, there is no spoon.**

**| - : unit = ()**

**`----**

(B)

~~~

Top-level statements in OCaml files can define values which are

registered with the REPL when loading files. These will be output

lines starting with `val' such as

,----

| val student : string = "Meatbag #43281"

`----

One can verify that a name is bound to a value by typing the name

followed by two semicolons and then enter as in

,----

| # student;;

| - : string = "Meatbag #43281"

`----

This reports the type of value bound to a name and its value.

What other name is bound in the `print\_stuff.ml' file? Give its name

and type below and describe it to the best of your ability after

examining the source file.

Solution :solution:

--------

**Name `print\_hello' is bound and appears to be a function that takes a**

**string as an argument and does some printing.**

**,----**

**| # print\_hello;;**

**| - : string -> unit = <fun>**

**`----**

(C)

~~~

One can create new name/value bindings in the REPL using `let x =

...;;' and override old bindings as well. Create the following

bindings in the REPL:

- Bind `life' to the integer 42

- Bind `pie' to the floating point number 3.14159

- Bind `student' to your name surrounded by quotes (this overrides the

previous binding for `student'

Show the code you used to create these bindings in the REPL

Solution :solution:

--------

**,----**

**| # let life = 42;;**

**| val life : int = 42**

**| # let pie = 3.14159;;**

**| val pie : float = 3.14159**

**| # let student = "Chris";;**

**| val student : string = "Chris"**

`----

(D)

~~~

Reload the source file by typing

,----

| # #use "print\_stuff.ml";;

`----

Determine what value is associated with names `life', `pie' and

`student'. Have any changed?

Solution :solution:

--------

**,----**

**| # #use "print\_stuff.ml";;**

**| val print\_hello : string -> unit = <fun>**

**| val student : string = "Meatbag #43281"**

**| Hello there!**

**| Welcome to OCaml, Meatbag #43281**

**| - : unit = ()**

**| Remember, there is no spoon.**

**| - : unit = ()**

**| # life;;**

**| - : int = 42**

**| # pie;;**

**| - : float = 3.14159**

**| # student;;**

**| - : string = "Meatbag #43281"**

**`----**

**The value for student is again the one in the file.**

(E)

~~~

In a text editor, alter `print\_stuff.ml' so that it has the bindings

from the previous part as in:

- Bind `life' to the integer 42

- Bind `pie' to the floating point number 3.14159

- Bind `student' to your name surrounded by quotes; alter the existing

binding for this

Make sure to save your changes in the text editor.

QUIT the REPL by issuing the `#quit;;' directive then start it again

and load the altered source file. Show the output below to verify that

the changes you made have taken effect.

Solution :solution:

--------

**,----**

**| # #quit;;**

**| > ocaml**

**| OCaml version 4.06.0**

**|**

**| # #use "print\_stuff.ml";;**

**| val print\_hello : string -> unit = <fun>**

**| val life : int = 42**

**| val pie : float = 3.14159**

**| val student : string = "Chris"**

**| Hello there!**

**| Welcome to OCaml, Chris**

**| - : unit = ()**

**| Remember, there is no spoon.**

**| - : unit = ()**

`----

PROBLEM 2: Compiling OCaml to Byte-Code

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

(A)

~~~

Quit the REPL or open a new terminal and navigate to the

`lab01-code/repl-v-compile' directory.

Once in the `repl-v-compile' directory, issue the following command to

compile the source file `print\_stuff.ml' to byte-code.

,----

| > ocamlc print\_stuff.ml

`----

Verify that there are now a new set of files in the directory by

issuing an `ls' command. Also issue a `file \*' command which will

report the type of all files in the directory.

Copy the results of your commands below.

Solution :solution:

--------

**,----**

**| > ocamlc print\_stuff.ml**

**| > ls**

**| a.out print\_stuff.cmi print\_stuff.cmo print\_stuff.ml**

**| > file \***

**| a.out: a /usr/bin/ocamlrun script executable (binary data)**

**| print\_stuff.cmi: OCaml interface file (.cmi) (Version 022)**

**| print\_stuff.cmo: OCaml object file (.cmo) (Version 022)**

**| print\_stuff.ml: ASCII text**

**| >**

`----

(B)

~~~

The traditional default name for compiled programs in Unix is `a.out'

which should be present after compiling. You can run this program by

typing `./a.out' as in

,----

| > ./a.out

`----

The initial "dot slash" means "in the current directory". Without it,

your shell will search for a program named `a.out' in standard

directories like `/usr/bin' and probably not find one.

Show the results of running the `a.out' program below.

Solution :solution:

--------

**,----**

**| > ocamlc print\_stuff.ml**

**| > ./a.out**

**| Hello there!**

**| Welcome to OCaml, Chris**

**| Remember, there is no spoon.**

**`----**

(C)

~~~

The output from the program will be somewhat different from the

results of loading the same file in OCaml's REPL. Describe these

differences and explain why certain lines appear in both while others

appear only in the REPL.

Solution :solution:

--------

**Running the `a.out' program will only show those lines that are**

**printed. These lines also appear when the file is loaded in the REPL**

**but the REPL also shows any top-level bindings that are loaded along**

**with their types and values. The executable does not do this.**

PROBLEM 3: Basic Style of OCaml Programs

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

(A)

~~~

Navigate in the terminal to the `lab01-code/separate-compilation'. To

move up a directory in a Unix terminal, use the command `cd ..' so a

sequence of commands like the following may be useful.

,----

| > pwd # show me what directory I'm in

| /home/kau0095/lab01-code/repl-v-compile # in the problem 1 directory

| > cd .. # change up one directory

| > ls # show what's in this directory

| QUESTIONS.txt repl-v-compile separate-compilation

| > cd separate-compilation # change to the problem 2 directory

| > ls # show what's in this directory

| defs\_and\_main.ml defs\_only.ml main\_only.ml

`----

Examine the source file `defs\_and\_main.ml' in a text editor. This file

represents the style we will follow for self-contained OCaml source

files.

Compile and run this program via the following commands.

,----

| > ocamlc defs\_and\_main.ml

| > ./a.out

| ...

`----

Paste the output you get for the run below.

Solution :solution:

**--------**

**,----**

**| > ocamlc defs\_and\_main.ml**

**| > ./a.out**

**| Bonjour!**

**| Welcome to OCaml, Triangle Lover**

**| A right triangle has sides length 4.0 and 3.0**

**| Its hypotenuse has length 5.0**

**| Au revoir!**

**`----**

(B)

~~~

Describe the basic structure of the `defs\_and\_main.ml' which is broken

into roughly 3 sections. What is in each section?

Solution :solution:

--------

**The top section merely lists and opens any dependencies for the**

**program, in this case only the Printf module.**

**The middle section defines some values and functions like `user' and**

**hypotenuse.**

**The last section is a sort of `main' function which uses the previous**

**functions to actually do something.**

(C)

~~~

Now examine the source file `defs\_only.ml'.

- Describe how it differs from `defs\_and\_main.ml'

- Compile it and run the resulting program. Describe anything odd

about the output

Solution :solution:

--------

**`defs\_only.ml' has only the value and function definitions from the**

**previous file. When compiled and run, it produces no output as no code**

**is executed.**

**,----**

**| > ocamlc defs\_only.ml**

**| > ./a.out**

**| >**

**`----**

(D)

~~~

Examine the source file `main\_only.ml' and note how it differs from

`defs\_and\_main.ml'. Pay particular attention to the first section of

the file which will differ slightly. Attempt to compile this to a

program and show the result.

**Solution :solution:**

**--------**

**`main\_only.ml' has the "open" statements at the top and has add an**

**`open Defs\_only' in it. It also has the last "main" code of**

**`defs\_and\_main.ml'. Attempt to compile it doesn't work and gives the**

**following error:**

**,----**

**| > ocamlc main\_only.ml**

**| File "main\_only.ml", line 1:**

**| Error: Required module `Defs\_only' is unavailable**

**| >**

**`----**

(E)

~~~

Unlike `defs\_and\_main.ml', `main\_only.ml' depends on the contents of

the `defs\_only.ml' file. There are several ways that one can compile

them together. The most straight-forward is to simply give both files

on the command line as shown below.

,----

| > ocamlc defs\_only.ml main\_only.ml

| > ./a.out

| Bonjour!

| Welcome to OCaml, Triangle Lover

| A right triangle has sides length 4.0 and 3.0

| Its hypotenuse has length 5.0

| Au revoir!

`----

Note that order of the files on the command line matters as the

compiler needs to compile `main\_only.ml' after it has produced the

results for `defs\_only.ml' so the following produces an error.

,----

| > ocamlc main\_only.ml defs\_only.ml

| File "defs\_only.ml", line 1:

| Error: Required module `Defs\_only' is unavailable

| >

`----

Finally, any run of the compiler can be instructed to name the output

program something aside from `a.out'. This can be done with the `-o'

option as shown below where the output program is named

`triangle\_greeter'.

,----

| > ocamlc -o triangle\_greeter defs\_only.ml main\_only.ml

| > ./triangle\_greeter

| Bonjour!

| Welcome to OCaml, Triangle Lover

| A right triangle has sides length 4.0 and 3.0

| Its hypotenuse has length 5.0

| Au revoir!

`----

Choose a different output name and compile the files `defs\_only.ml'

and `main\_only.ml' together to produce a program. Show your results

below.

**Solution :solution:**

**--------**

**Any output name is fine; below `finished' was used.**

**,----**

**| > ocamlc -o finished defs\_only.ml main\_only.ml**

**| > ./finished**

**| Bonjour!**

**| Welcome to OCaml, Triangle Lover**

**| A right triangle has sides length 4.0 and 3.0**

**| It's hypotenuse has length 5.0**

**| Au revoir!**

**`----**