`$Id: cse112-2020q1-final.mm,v 1.112 2020-03-12 17:09:55-07 - - $`

page 1	page 2	page 3	page 4	Total / 40

Last Name :
First Name :
CruzID : @ucsc.edu

No books ; No calculator ; No computer ; No email ; No internet ; No notes ; No phone. Points will be deducted for messy or unreadable answers. Do your scratch work elsewhere and enter only your final answer into the spaces provided.

OCaml. For all questions asking for code in OCaml, you need not explicitly specify the type of a function's argument or the type of its result. Obviously, OCaml uses type inference.

1. ***Bash.*** Assume the interpreter for the language `nli` has the executable binary in `/opt/local/bin/nli`.

   (a) In order to call it just by using the name `nli`, what file in `$HOME` should you modify ? **[1✔]**

   (b) What command should you add to the file mentioned in (a) ? **[1✔]**

   (c) If you use this program as an interpreter for the program `foo.nli`, what is the first line of this file ? **[1✔]**

   (d) What command must you type to make `foo.nli` an executable file ? **[1✔]**

2. ***Prolog.*** Define the following terms (functions). See the examples for expected results.

   (a) `has/2` takes an atom and a list and succeeds if the atom is somewhere in the list. Use unification for comparison. **[1✔]**

```
?- has(3,[1,2,3,4]).
true.
?- has(9,[1,2,3,4]).
false.
```

   (b) `rev/3` is an accumulator helper for `rev/2`, which reverses a list. The middle argument to `rev/3` is the accumulator. Assume the rule `rev(L,M) :- rev(L,[],M).` **[1✔]**

```
?- rev([1,2,3,4],X).
X = [4, 3, 2, 1].
?- rev([],X).
X = [].
```

   (c) `len/2` computes the length of a list. **[1✔]**

```
?- len([],N).
N = 0.
?- len([1,2,3,4],N).
N = 4.
```

3. ***Perl.*** Write a program that will read files, and at end print out the number of lines, words, and characters in the files. A word is any sequence of characters not containing white space. Use `<>` for input. An example is shown at the left. **[3✔]**

```
bash$ cat /tmp/foo
This is a test
of the word count
program.
Has 4 lines.
bash$ wc.perl /tmp/foo
4 12 55
```

4. ***Prolog.***

   (a) Write a set of facts called **`arrow`** which describe this graph, where **`arrow(a,b)`** means that **a**→**b**. **[2✓]**

$$\begin{tikzcd}
a \arrow[r] & b \arrow[r] \arrow[dl] & d \arrow[d] \\
c \arrow[u] & & e \arrow[ll]
\end{tikzcd}$$

   (b) Given **`ispath`** as defined here, write the relation **`ispath(A,Visited,B)`** which succeeds if there is a path from **`A`** to **`B`** and avoids infinite traversals around a loop. Use **`Visited`** to keep track of the nodes visited. Hint : **`member(X,L)`** succeeds if **`X`** is a member of list **`L`**. **[2✓]**

```
not(X) :- X, !, fail.
not(_).
ispath(A,A).
ispath(A,B) :- ispath(A,[],B).
```

5. Code the function **`find`**. Return the value associated with a given key. Return a special value as indicated below if not found. The first argument is a comparison function to be used in the search. The second argument is the left operand of the comparator, and the keys in the list are used as its right operand.

   (a) ***OCaml.*** Search a list of tuples. Note that the comma creates a tuple and the semi-colon separates elements of a list. **[2✓]**

```
# find (=) 3 [1,2; 3,4; 5,6];;
- : int option = Some 4
# find (=) 9 [0,1; 2,3; 4,5];;
- : int option = None
# find (<) 8 [9,2; 7,3; 5,9];;
- : int option = Some 2
```

   (b) ***Scheme.*** Return **`#f`** if not found. The list argument is a list of lists of length 2. **[2✓]**

```
> (find = 3 '((1 2)(3 4)(5 6)))
4
> (find = 9 '((0 1)(2 3)(4 5)))
#f
> (find < 8 '((2 2)(9 3)(9 9)))
3
```

6. ***Scheme.*** Write a function to reverse a list. Assume the argument is a proper list. Your solution must, of course, be tail recursive. Use **`foldl`** if you like, or just code it directly. **[2✓]**

```
> (reverse '(1 2 3 4 5))
(5 4 3 2 1)
> (reverse '())
()
> (reverse '((1 2) (3 4) (5 6)))
((5 6) (3 4) (1 2))
```

7. ***Smalltalk.*** Write code in Smalltalk which will print the numbers 1 through 10000 inclusive, one number per line. **[1✔]**

8. ***Smalltalk.*** Extend class **`Array`** with a unary message **`reverse`**, which reverses an array. The array itself is reversed. Nothing is returned. **[3✔]**

```
st> a := #(1 2 3 4 5) copy.
(1 2 3 4 5 )
st> a reverse.
(5 4 3 2 1 )
st> a := #( #(1 2) #(3 4) #(5 6)) copy.
((1 2 ) (3 4 ) (5 6 ) )
st> a reverse.
((5 6 ) (3 4 ) (1 2 ) )
```

9. ***Scheme.*** Define the function **`evalexpr`** which takes as an argument either a number or an expression. A number is always returned as real. An expression is a list of length 3, where the **`car`** is a binary function, and the **`cdr`** is a list of two expressions. See the Scheme interaction at the left. Note that there is no hash table for functions. The actual function's value is in the list in the function position. Note the quasiquote and unquote. **[2✔]**

```
> (evalexpr `(,+ (,* 2 3) (,* 4 5)))
26.0
> (evalexpr 3)
3.0
> (evalexpr `(,* (,+ 8 3) (,+ (,* 2 9) 6)))
264.0
```

10. ***Smalltalk.*** Extend class **`Array`** with a unary method **`sum`**, which returns the sum of all elements of the array. Assume the array contains numbers. **[2✔]**

```
st> #(1 2 3 4 5) sum.
15
st> #() sum.
0
```

11. ***Scheme.*** Define the function **`map`**. **[2✔]**

```
> (map (lambda (x) (+ 5 x)) '(1 2 3 4))
(6 7 8 9)
> (map (lambda (x) (cons 5 x)) '(1 2 3 4))
((5 . 1) (5 . 2) (5 . 3) (5 . 4))
```

12. ***Ocaml.*** Code the Fibonacci function. Be sure to use tail recursion, and make it run in $O(n)$ time. Do not consider the case where $n < 0$. Assume $n \geq 0$. For reference, the mathematical expression is given here. **[2✔]**
$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$
```
# #use "fib.ml";;
val fib : int -> int = <fun>
# List.map fib [0;1;2;3;4;5;6;7];;
- : int list = [0; 1; 1; 2; 3; 5; 8; 13]
```

13. ***Prolog.*** Write a function **zip**, which will zip two lists into a single list of pairs. If the lists are not of equal length, ignore excess elements of the longer list. **[2✔]**
```
?- zip( [1,2,3,4,5], [4,5,6], X).
X = [pair(1, 4), pair(2, 5), pair(3, 6)] .
?- zip( [1,2,3], [4,5,6], X).
X = [pair(1, 4), pair(2, 5), pair(3, 6)] .
?- zip( [], [1,2,3], X).
X = [].
```

14. ***Ocaml.*** Define the function **merge** which merges two sorted lists according to a predicate and produces a resulting list. Assume the argument list are sorted. **[3✔]**
```
# merge;;
val merge : ('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list
# merge (<) [1;3;5;7;9] [2;4;6;8];;
- : int list = [1; 2; 3; 4; 5; 6; 7; 8; 9]
# merge (>) [9;5;3;1] [77;2;-5];;
- : int list = [77; 9; 5; 3; 2; 1; -5]
# merge (<) [] [1;2;3];;
- : int list = [1; 2; 3]
```

15. ***Perl.*** Write a program which prints out the file size, modification time, and filename for each file mentioned in **@ARGV**. Hints: The result of the **stat** function is an array where **$stat[7]** is the file size and **$stat[9]** is the modification time. To format the time as required, use the following statement. **[3✔]**
```
   $time = strftime "%b %e %H:%S", localtime $stat[9];
-bash-60$ ls.perl *.perl
      84 Nov 12 13:37 count.perl
     240 Nov 16 12:39 euclid.perl
     253 Nov 25 19:03 ls.perl
     110 Dec  5 17:53 range.perl
      91 Mar 14 21:31 wc.perl
```