MONTANA STATE UNIVERSITY

PROGRAMMING LANGUAGES

Final

SPRING, 2018

Name: _____

Student Id: _____

Question	Points	Score
1	30	
2	8	
3	15	
4	10	
5	15	
6	10	
7	12	
Total:	100	

1. (30 points) Mark each question as either True (T) or False (F).

   (a) ◯ T ◯ F A higher order function is one that accepts functions as input or returns a function as output.

   (b) ◯ T ◯ F Scheme's syntax avoids the dangling-else problem by having every if-then-else enclosed in its own set of parentheses.

   (c) ◯ T ◯ F In Scheme, any value that is not considered as False is interpreted as True.

   (d) ◯ T ◯ F Eager evaluation is a technique that can make it easy to avoid unnecessary computation.

   (e) ◯ T ◯ F Scheme uses backtracking in order to search for a solution to a function call.

   (f) ◯ T ◯ F Both Scheme and Python use dynamic typing.

   (g) ◯ T ◯ F A declarative language like Prolog allows us to describe how we solve a problem rather than what the answer will look like.

   (h) ◯ T ◯ F Prolog is a statically typed language.

   (i) ◯ T ◯ F Prolog's functionality is based on lambda calculus.

   (j) ◯ T ◯ F The initial rise in popularity of Object Oriented langugages was related to the sales of home personal computes and the need for GUIs.

   (k) ◯ T ◯ F In the model-view-controller design pattern, the controller implements the event handling.

   (l) ◯ T ◯ F Java's automatic garbage collection makes heap overflows impossible, unlike C++ where objects had to be explicitly deconstructed.

   (m) ◯ T ◯ F When a thread is waiting for a condition that will never happen, it is said to be deadlocked.

   (n) ◯ T ◯ F Concurrent programs provide performance improvements even on single processor machines.

2. (8 points) Consider the four programming paradigms and the languages we covered this semester:

   1. Imperative, Go
   2. Object-Oriented, Java
   3. Functional, ML
   4. Logical, Prolog

   For each of the following problems, argue which paradigm is best suited to solve using the language listed above. Make sure you justify your choice. You will answer each paradigm/language pair only once. *Hint: this is not a subjective question. If in doubt, pick the choice you think the professor would choose.*

   (a) A program to implement the AI of a character in a virtual world.

   (b) A graphical user interface for architecture design software

   (c) A program to handle dormitory room assignments for a large university given a set of room preferences

   (d) A program that crawls through web page meta data to construct search engine information

(a)	argument	(i)	event-driven program	(q)	parameter
(b)	activation record	(j)	functor	(r)	parallel program
(c)	atomic proposition	(k)	garbage	(s)	race condition
(d)	backtracking	(l)	horn clause	(t)	resolution
(e)	concurrent program	(m)	lambda expression	(u)	run-time stack
(f)	currying	(n)	lazy evaluation	(v)	semaphore
(g)	deadlock	(o)	model-view-controller	(w)	thread
(h)	eager evaluation	(p)	monitor	(x)	unification

3. (15 points) Match the term (a – x) to the definition. Not all terms will be used.

term	definition
	an integer variable and an associated thread queue
	memory model for keeping track of function calls and returns
	block of heap memory that cannot be accessed by the program
	contains a head and a list of predicates
	a program designed to have two or more execution contexts

Match the term (a – x) to the definition (*continued*)

term	definition
	a program unit that can be executed concurrently
	concept particularly useful in event-driven GUI programs
	a function that is not bound to an identifier
	a program designed to have multiple threads active on different processors
	encapsulate a shared variable with operations signal and wait
	a variable identifier that appears in a function header
	a declarative sentence that cannot be broken into simpler sentences
	algorithm for abandoning the failed part of an attempted solution
	translation of multiple argument function to an equivalent series of single argument functions

4. (10 points) Consider this code snippet in some C-like language. Answer the questions below. Assume indexing starts at 0.

```
// global variables
int a = 1;
int b = 2;
void foo(int[] list, int i, int j) {
    i = i + 1;
    j = j + 1;
    list[i] = list[i] + j;
    print(a);
    print(b);
    print(list);
}
void main() {
    int x[5] = {5, 9, 7, 1, 5};
    foo(x, a, b);
    print(a);
    print(b);
    print(x);
}
```

(a) In the above code, circle all function **parameters**.

(b) In the above code, draw a rectangle around all function **arguments**.

(c) Which values are printed if the language passes by value?

(d) Which values are printed if the language passes by value-result?

(e) Which values are printed if the language passes by reference?

5. (15 points) The following questions deal with functional programming in teh Scheme language. You may use any of the primitive Scheme functions, such as `if, cond, null?, equal?, eq?, eqv?, car, cdr and cons, append, list?,` and `member?`

   (a) Define a Scheme function, `odds`, that takes a list and returns every other element, starting with the first.

   Example use:

```
> (odds '(a b c d e f g))
(a c e g)
```

```
; Write your function here
(define (odds lst)









); end define
```

   (b) Define a Scheme function, `evens`, that takes a list and returns the elements in even positions, starting with the second. You may use your function odds, if useful.

   Example use:

```
> (evens '(a b c d e f g))
(b d f)
```

```
; Write your function here
(define (evens l)









); end define
```

(c) Consider these functions. You should assume `odds` and `evens` function correctly even if you did not complete parts (a) and (b).

```
(define (plusOne x)
     (+ 1 x))


(define (plusTwo x)
     (+ 2 x))


(define (mystery lst)
     (if (null? lst)
     '()
     (append (map plusTwo (odds lst))
          (map plusOne (evens lst))
     )
))
```

What is the output of this line of Scheme code?

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

6. (10 points) What do the following Scheme expressions return?

   (a) `(car '(1 (2) 3 4 (5 6 (7 8)) 9))`

   (b) `(cdr '(1 (2) 3 4 (5 6 (7 8)) 9))`

   (c) `(caadr '(1 (2) 3 4 (5 6 (7 8)) 9))`

   (d) `(cddddr '(1 (2) 3 4 (5 6 (7 8)) 9))`

   (e) `(caar (cddddr '(1 ((2) 4) 3 4 (5 6 (7 8)) 9)))`

   (f) `(caddar (cddddr '(1 ((2) 4) 3 4 (5 6 (7 8)) 9)))`

7. (12 points) Complete the following Prolog definitions.

Some Hints:

- Underscore (_) is an anonymous name (i.e., don't care).
- `[A | B]` splits a list into head A and tail B
- `[A, B | C]` takes first two elements A and B, and sets the tail C
- Bang (!) is the cut operator. E.g., `someRule(A, B) :- !.` stops the recursion.

(a) Define `last_but_one/2` function that finds the 2nd-to-last element of a list.

```
% last_but_one(X, L) :- X is the last but one element of the list L
```

(b) Define `get_length/2` function that finds the number of elements of a list.

```
% get_length(L, N) :- the list L contains N elements
```

(c) Define `reverse_list/2` function that reverses a list. You may find a helper function to be useful.

```
% reverse_list(L1, L2) :- L2 is the list obtained from L1 by reversing
```

(d) Define `is_palindrome/1` function that determines if a list is a palindrome.

```
% is_palindrome(L) :- L is a palindrome list
```

THIS PAGE INTENTIONALLY LEFT BLANK