NAME_____
login:_____
Signature: _____

Computer Science and Engineering 150
Programming Languages for Artificial Intelligence
February 13, 2001
FIRST MIDTERM EXAM
DO NOT TURN THIS PAGE UNTIL YOU ARE TOLD TO START!!!!

- Please DO NOT put your name at the top of each page:
  - This should prevent residual sexism, racism, favoritism lurking in the unconscious of your professor & TA from biasing grading!!
- THE EXAM IS CLOSED BOOK. IN FACT, PLEASE LEAVE YOUR BOOKS AT THE FRONT OF THE CLASS!
- Once the exam has started, SORRY, NO TALKING!!!
- No, you can't even say "later, dude!"
- There are 6 problems: Make sure you have all of them - AFTER I TELL YOU TO START!
- Read each question carefully.
- Remain calm at all times!

Problem	Type	Points	Score
1	True/False	14	
2	Fill in the blanks	18	
3	Multiple Choice	18	
4	Lisp: Threat, or menace?	10	
5	Recursive functions	20	
6	State space search	20	
	Total	100	

True/False

1. (14 pts: +1 for correct, -1 for incorrect, 0 for no answer)
   If you would like to justify an answer, feel free.

_____ In Lisp, defining a structure implicitly defines a set of access functions for that structure.

_____ Dynamic variables are useful for things like exceptions – the most recent activation on the stack of an exception handler would be the one called if the exception handler's name was dynamically bound.

_____ (defun test (x) (test (+ 1 x))) is tail recursive.

_____ (defun test2 (x) (+ 1 (test2 (- x 1)))) is tail recursive.

_____ Iterative Deepening, since it calls Depth-limited depth first search (DLDFS) over and over again, takes at least twice as long as calling DLDFS with the correct depth bound.

_____ Bidirectional search works best in domains with many goal states.

_____ If I wanted to stress my employees, I would ask them to write a program to translate C programs into "pure" lisp code.

_____ Uniform cost search requires the edge costs to be uniform.

_____ Algorithm A = Uniform Cost Search + Greedy Search

_____ Variable capture refers to the environment in a lexical closure: the variables wrapped up in the environment are captured by the closure.

_____ The backquote operator, combined with the comma, makes it easy to write higher order functions.

_____ An A* algorithm that is more informed than another always expands the same or fewer nodes in the search tree.

_____ The requirement on h(n) in A to make it A* is that h(n) must always overestimate the amount of remaining work to be done to get to the goal.

_____ In general, an A* algorithm will expand fewer nodes than Uniform Cost search, but with the downside that it may not find the optimal path.

_____ The Macroexpansions are a post-Industrial post-Gothic performance art band from Ocean Beach.

Fill in the blanks/short answer (18pts)

2(a). (6 pts total) (2 pts) Draw the box notation representation of of the list "(A B)".

(2 pts) Now draw the box notation representation of the cons "(A.B)"

(2 pts) Suppose I do the following:

USER(1): (setf L '(A b))
(A B)
USER(2): (setf (cddr L) L)
(A B A B A B A B A B ...)

Explain what is going on here with a box notation representation of the list L.

(1 pt each blank):

2(b). Every time you do an append operation on two non-empty lists, a new _____ _____is created, and the first list is also _____to avoid unwanted side effects.

2(c). In the proof of A*'s optimality, the key idea was that there is always a node on the _____ list that is on the _____path to the goal.

2(d). The main reason why Iterative Deepening is to be preferred to all other blind search algorithms is its relatively small use of _____.

2(e). Fill in the blanks in the following table:

Criterion	Time	Space	Is it Optimal?	Is it Complete?
Breadth First	b^d	b^d	Yes	Yes
Uniform Cost	b^d	b^d	Yes	Yes
Depth First	b^d			No
Depth-limited(l)	b^l	bl	No	
Iter. Deepening	b^d			Yes
Bidirectional	$b^{d/2}$		Yes	Yes

b = branching factor, d = depth of solution, m = max depth of search tree, l = depth limit

Multiple Choice (18 pts, 2 pts each) (circle the *BEST* answer)

3(a). Lambda exressions are another name for
a) closures b) functions c) macros d) greek words

3(b). The difference between a "lexical closure" and a "dynamic closure" is
a) In the dynamic closure, variables that are free in the closure are gensym'ed.
b) In the dynamic closure, there may be free variables that get their binding from the environment in which the closure is executed: these variables must then be made special or otherwise made dynamic or it will cause an error.
c) In the lexical closure, any free variables must be gensym'ed.
d) one of the above, I'm not sure which.

3(c). In AI, we characterize many "puzzle solving" problems as
a) real world problems
b) Greedy search problems
c) constraint satisfaction problems.
d) search problems

3(d). A lot of Lisp programming on lists involves transforming, filtering, counting and finding. The *templates* for doing these are called
(a) mapping control structures
(b) programming cliches
(c) lisp structures
(d) (a) and (c)

3(e). If I want to make a variable be dynamic in lisp, I have to
(a) declare it "special"
(b) introduce it with a "defvar"
(c) introduce it with a "defparameter"
(d) Any of (a), (b), or (c) will work.

3(f). The garbage collector in Lisp
a) Runs so efficiently it is rarely noticed.
b) removes comments from programs before they are run, as these are "garbage" to the interpreter
c) collects cons cells from the heap that are no longer being used.
d) Has gone on strike many times, creating problems in computers in New York City.

3(g). In "pure lisp", where there are no assignment statements, variables are bound to values through
a) "let" expressions
b) "setf" expressions
c) being on a parameter list of a function
d) (a) and (b)
e) (c) and (d)
f) (a) and (c)
CONTINUED NEXT PAGE

3(h) A heuristic is

a) a function that represents domain specific knowledge

b) a static evaluation function that always finds the right answer when incorporated into algorithm A

c) a rule that works for almost any problem.

d) a member of a techno band from Philadelphia

3(i). For some combinatorial search problems, such as the Traveling Salesperson Problem,

(a) IDA* is the best approach.

(b) the heuristic may need to be inadmissible.

(c) an admissible heuristic is best.

(d) all of the above.

(e) (b) and (c) only.

Lisp: Threat, or Menace? (10 points)

4. (10 points, 2 points each) Suppose we type the following into the LISP interpreter:

(setf bob '(bob bob))
(setf fred 'bob)

Write the result of evaluating each of the following (as if we typed them into the LISP interpreter next). [HINT: These are ALL legal!] [HINT HINT: Just remember, every "eval" simply evaluates the value of its argument]

a. (car bob)
b. (eval (car bob))
c. (eval (car (eval (car bob))))
d. (let ((bob fred)) bob)
e. (let ((bob fred)) (declare (special bob)) bob)

Recursive functions (20 points)

5.(14 points) This problem involves writing Lisp code recursively (what else is there?).

(a) (15 points) Write two functions, each taking two arguments. One is name BEFORE and one is named AFTER. The first argument is an atom and the second is a list in both cases. You may assume that the atom occurs in the list. BEFORE Returns a list containing the part of the list before the first occurrence of the atom; AFTER returns the part of the list coming after the first occurrence of the atom.

Here are some examples:

USER(23): (BEFORE 'here '(thats neither here nor there))
(thats neither)

USER(24): (AFTER 'here '(thats neither here nor there))
(nor there)

You may assume that the arguments will match these specifications. NOTE: There are NON-recursive ways to write these functions using the built-in functions of Lisp (e.g., MEMBER). THAT'S NOT WHAT WE WANT YOU TO DO.

(b) (5 points) If you haven't already, re-write BEFORE so that it is tail-recursive, using an optional argument or an auxiliary function. [HINT: Careful! Make sure your result is in the correct order!!!] If your BEFORE is already tail- recursive, go on to the next question.

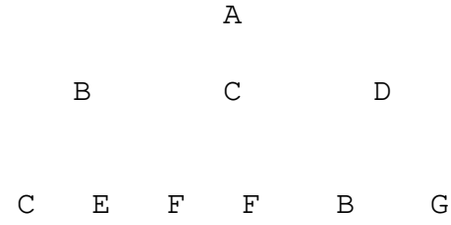
State space search (20 points)

6. You are part of a design team for an autonomous robot named Max. Max is to be used in time critical applications (pizza delivery), and thus he has to move as efficiently as possible. It is your job to design his navigation plan. The prototype moves along a track with a limited number of routes from one destination to another. You decide to use space state search. To test your approach, you are given a sample environment, consisting of "nodes" connected by the track that Max moves along. The nodes are labeled A through G, and each track has an associated cost based on how long it takes Max to traverse its length. The nodes, and their associated connecting nodes and costs are as follows:

From:	To:
A	B with cost 50
	C with cost 40
	D with cost 40
B	C with cost 40
	E with cost 40
	F with cost 50
C	F with cost 30
	B with cost 40
D	G with cost 100
E	G with cost 50
F	G with cost 30

Note that the tracks are one-way.

Given that Max wants to go from A to G, using as little time as possible, you first attempt to use Breadth First Search to generate a minimal solution. The following tree represents this effort:



Nodes in this tree have been fully expanded (although their successors are not shown since they are not fully expanded). This discovers a path to G of length 140, which is not optimal, and took a good deal of effort to produce.

(a) (2 pts) What is the "bug" in your Breadth First Search Algorithm, compared to the way you were shown in class?

(b) (2 pts) Which blind search technique should you have used in this space?

(c) (4 pts) Now try DFS on this space, and see if it provides a better solution. Assume that the open list that is generated by DFS follows the same order as presented in the above table. For example, the next successors of A are (in order) B then C then D. Write the tree for DFS as above, and write the length of the solution (if any).

(d) (2 pts) What would happen if we used DFS on this space, but there was no link from C to F. (i.e. C only leads to B with cost 40). How could we avoid this problem?

(e) (10 pts) The demo is due tomorrow, and your boss threatens to fire you. To placate her, you decide to try a heuristic search. A reasonable heuristic for this might be the straight-line distance from each node to G. This is given in the following table for your heuristic function, h(n):

Node	h(n) <- absolute distance from n to G
====	====
A	100
B	60
C	50
D	80
E	40
F	25
G	0

Now using f(n) = g(n) + h(n), complete the following heuristic search tree, labelling values for h(n), g(n), and f(n).

```
                    A: g=0, h=100, f=100

B: g=50             C: g=40             D: g=40
   h=60                h=50                h=80
   f=110               f=90                f=120
```