2.[1 pt.] Which one of the following 5 properties of a grammar is *equivalent* to the property of being ***syntactically ambiguous***? Circle the correct answer.

(a) Every sequence of terminals in the language of the grammar has two or more parse trees.
(b) At least one sequence of terminals of the grammar has two or more parse trees.
(c) At least one sequence of terminals of the grammar has exactly two parse trees.
(d) At least one sequence of terminals of the grammar has more than two parse trees.
(e) Every sequence of terminals in the language of the grammar has exactly two parse trees.

[1 pt.] Consider the following grammar: E ::= 5 | E . F
F ::= 5 | F . E

Suppose a professor has asked four students to give a ***leftmost*** *derivation* of 5 . 5 . 5 based on this grammar, and those students have offered the following solutions **S1**, **S2**, **S3**, and **S4**, each of which may be correct or incorrect:

S1: E ⇒ E . F ⇒ 5 . F ⇒ 5 . F . E ⇒ 5 . 5 . E ⇒ 5 . 5 . 5
S2: E ⇒ E . F ⇒ E . F . F ⇒ 5 . F . F ⇒ 5 . 5 . F ⇒ 5 . 5 . 5
S3: E ⇒ E . F . F ⇒ 5 . 5 . F ⇒ 5 . 5 . 5
S4: E ⇒ E . F ⇒ E . 5 ⇒ E . F . 5 ⇒ E . 5 . 5 ⇒ 5 . 5 . 5

Exactly which, if any, of **S1**, **S2**, **S3**, and **S4** are correct solutions to the professor's problem? In other words, exactly which of **S1**, **S2**, **S3**, and **S4** are ***leftmost*** *derivations* of 5 . 5 . 5 based on the above grammar? If none of **S1**, **S2**, **S3**, and **S4** is a correct solution, write **NONE**; otherwise, write down the label (**S1**, **S2**, **S3**, or **S4**) of ***each*** correct solution to the professor's problem.

ANSWER: _____

4.[1 pt.] What is the value of the Lisp expression `(append '(+ 3 4) '(+ 3 4))`? Circle the correct answer: (a) `((+ 3 4) + 3 4)` (b) `(+ 3 4 + 3 4)` (c) `(7 7)`
(d) `(7 + 3 4)` (e) `((+ 3 4) (+ 3 4))`

5.[1 pt.] What is the value of the following Lisp expression?
`(cons (+ 1 2 3 4) (append (list '(+ 5 6)) '(+ 7 8)))`
Circle the answer: (a) `(+ 1 2 3 4 (+ 5 6) 15)` (b) `(10 (+ 5 6) + 7 8)` (c) `(10 (11 + 7 8))`
(d) `(10 + 5 6 (+ 7 8))` (e) `((+ 1 2 3 4) + 5 6 + 7 8)`

6.[2 pts.] Suppose you have ***already*** defined a Lisp function `SQUARE` that takes a number as argument and returns the square of the number. For example: `(square 3.0) => 9.0`

A point (x, y) in the plane can be represented as a 2-element list $(x\ y)$. ***Complete the following definition of a Lisp function*** `DST` ***that takes two such lists*** `p1` ***and*** `p2` ***as arguments and returns the distance between the corresponding points.*** (Recall that the distance between points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is given by $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, and that the built-in function `SQRT` returns the square root of its argument.)

Example: `(dst '(4.0 6.0) '(1.0 2.0))` => $\sqrt{(4.0-1.0)^2+(6.0-2.0)^2} = 5.0$

```
(defun dst (p1 p2)
  (sqrt (+ (square ( - _____ _____ ))
           (square ( - _____ _____ )))))
```

7.(i)[1 pt.] Suppose a Lisp function `F` is defined as follows:

```
(defun f (u x)
  (let* ((x 3)
         (y (+ x u)))
    (+ x y u)))
```

Which one of the following is a ***correct*** statement about the value of the expression `(f 4 4)`?
Circle the correct answer: (a) Its value is 10. (b) Its value is 12. (c) Its value is 13.
(d) Its value is 14. (e) Its value is 15.

(ii)[1 pt.] Suppose a Lisp function `G` is defined as follows:

```
(defun g (u x)
  (let ((x 3)
        (y (+ x u)))
    (+ x y u)))
```

Which one of the following is a ***correct*** statement about the value of the expression `(g 4 4)`?
Circle the correct answer: (a) Its value is 10. (b) Its value is 12. (c) Its value is 13.
(d) Its value is 14. (e) Its value is 15.

8.[1 pt.] Which one of the Lisp expressions below evaluates to `((A) (A)) ((B) (B)) ((C) (C))`?
Circle the only correct answer:

(a) `(mapcar (lambda (y) (list y y)) '(A B C))`
(b) `(mapcar (lambda (y) (list y y)) '((A) (B) (C)))`
(c) `(mapcar (lambda (y) (cons y y)) '((A) (B) (C)))`
(d) `(mapcar (lambda (y) (append y y)) '((A) (B) (C)))`
(e) `(mapcar (lambda (y) (cons y y)) '(A B C))`

[1 pt.] Which one of the five Lisp expressions below evaluates to ((A A) (B B) (C C))?
Circle the only correct answer:

(a) `(mapcar (lambda (y) (cons y y)) '(A B C))`
(b) `(mapcar (lambda (y) (list y y)) '((A) (B) (C)))`
(c) `(mapcar (lambda (y) (cons y y)) '((A) (B) (C)))`
(d) `(mapcar (lambda (y) (append y y)) '((A) (B) (C)))`
(e) `(mapcar (lambda (y) (append y y)) '(A B C))`

10. [3 pts.] Complete the following definition of a Lisp function PARTITION such that if L is a list of real numbers and P is a real number then (PARTITION L P) returns a list whose CAR is a list of the elements of L that are ***strictly less*** than P, and whose CADR is a list of the other elements of L. Each element of L must appear in the CAR or CADR of (PARTITION L P), and should appear there just as many times as it appears in L. Examples:

```
(partition   ()  5) => (NIL NIL)
(partition  '(0  11  8  4  9  6  11  8  10)  9)  =>  ((0 8 4 6 8)  (11 9 11 10))  ; example A
(partition  '(3  0  11  8  4  9  6  11  8  10)  9)  =>  ((3 0 8 4 6 8)  (11 9 11 10))  ; example B
(partition  '(15  0  11  8  4  9  6  11  8  10)  9)  =>  ((0 8 4 6 8) (15 11 9 11 10))  ; example C

(defun partition (L  P)
  (if (endp L)
      '(() ())
      (let ((X (partition (cdr L) P)))
        (if (< (car L) P)
            (list _____ _____)  ; see examples B & A
            (list _____ _____)  ; see examples C & A
        )
      )
  )
)
```

Hint This problem can be correctly solved by filling in each of the four gaps above with exactly one of the following eight expressions:

```
(car X)          (cdr X)          (cadr X)          (cons (car L) (car X))
(cons (car L) (cdr X))            (cons (car L) (cadr X))
(list (car L) (cdr X))            (append (car L) (cdr X))
```

Of course, some of these eight expressions should *not* be used!

correct) solution to TinyJ Assignment 3 compiles the TinyJ program shown on the next page, and utes the generated code with a debugging stop after execution of 519 TinyJ virtual machine uctions. (The values 1, 10, and 5 were entered by the user at the keyboard during execution.) The **NITSTKFRM** instructions in the generated code (which are at code memory addresses 4, 59, 86, 79, and 250) are as follows:

```
INITSTKFRM 2          86: INITSTKFRM 1          179: INITSTKFRM 1
INITSTKFRM 1         140: INITSTKFRM 1          250: INITSTKFRM 1
```

tructions at code memory addresses 196 through 205 in the generated code are shown ext page. The "dump" produced *after* the debugging stop is shown on p. 6; notice how the truction to be executed:" line of the dump indicates that the address of the *last* instruction executed before the debugging

[illegible]

DATA MEMORY DUMP [illegible]

[illegible] addresses [illegible] top of stack, and [illegible] heap locations:

```
0: 2147428116 = POINTER TO 10004
1: 0 = Ctrl-@
2: 5 = Ctrl-E
3: 2147428113 = POINTER TO 10001
4: 32 = ' '
5: 10 = Ctrl-J
6: 10 = Ctrl-J
7: 109 = 'm'
8: 105 = 'i'
9: 110 = 'n'
10: 32 = ' '
11: 110 = 'n'
12: 117 = 'u'
13: 109 = 'm'
14: 58 = ':'
15: 32 = ' '
16: 66 = 'B'
17: 97 = 'a'
18: 100 = 'd'
19: 32 = ' '
20: 97 = 'a'
21: 110 = 'n'
22: 115 = 's'
23: 119 = 'w'
24: 101 = 'e'
25: 114 = 'r'
26: 33 = '!'
27: 109 = 'm'
28: 97 = 'a'
29: 120 = 'x'
30: 32 = ' '
31: 110 = 'n'
32: 117 = 'u'
33: 109 = 'm'
34: 58 = ':'
35: 32 = ' '
36: 66 = 'B'
37: 97 = 'a'
38: 100 = 'd'
39: 32 = ' '
40: 97 = 'a'
41: 110 = 'n'
42: 115 = 's'
43: 119 = 'w'
44: 101 = 'e'
5: 114 = 'r'
6: 33 = '!'
: 109 = 'm'
: 97 = 'a'
120 = 'x'
32 = ' '
110 = 'n'
117 = 'u'
09 = 'm'
2 = ' '
55: 111 = 'o'
56: 102 = 'f'
57: 32 = ' '
58: 116 = 't'
59: 101 = 'e'
60: 114 = 'r'
61: 109 = 'm'
62: 115 = 's'
63: 58 = ':'
64: 32 = ' '
65: 66 = 'B'
66: 97 = 'a'
67: 100 = 'd'
68: 32 = ' '
69: 97 = 'a'
70: 110 = 'n'
71: 115 = 's'
72: 119 = 'w'
73: 101 = 'e'
74: 114 = 'r'
75: 33 = '!'
76: 78 = 'N'
77: 111 = 'o'
78: 116 = 't'
79: 32 = ' '
80: 101 = 'e'
81: 110 = 'n'
82: 111 = 'o'
83: 117 = 'u'
84: 103 = 'g'
85: 104 = 'h'
86: 32 = ' '
87: 109 = 'm'
88: 101 = 'e'
89: 109 = 'm'
90: 33 = '!'
91: 84 = 'T'
92: 111 = 'o'
93: 111 = 'o'
94: 32 = ' '
95: 109 = 'm'
96: 97 = 'a'
97: 110 = 'n'
98: 121 = 'y'
99: 32 = ' '
100: 116 = 't'
101: 101 = 'e'
102: 114 = 'r'
103: 109 = 'm'
104: 115 = 's'
105: 33 = '!'
106: 49 = '1'
107: 32 = ' '
108: 116 = 't'
109: 111 = 'o'
110: 32 = ' '
111: 113 = 'q'
112: 117 = 'u'
113: 105 = 'i'
114: 116 = 't'
115: 44 = ','
116: 32 = ' '
117: 48 = '0'
118: 32 = ' '
119: 116 = 't'
120: 111 = 'o'
121: 32 = ' '
122: 114 = 'r'
123: 101 = 'e'
124: 116 = 't'
125: 114 = 'r'
126: 121 = 'y'
127: 58 = ':'
128: 32 = ' '
129: 2147438112 = POINTER TO 20000
130: 0 = Ctrl-@
131: 3 = Ctrl-C
132: 3 = Ctrl-C
133: 0 = Ctrl-@
134: 44 = ','
135: 2147418241 = POINTER TO 129
136: 1 = Ctrl-A
137: 10 = Ctrl-J
138: 1 = Ctrl-A
139: 226
140: 2147418247 = POINTER TO 135
141: 2 = Ctrl-B
142: 5 = Ctrl-E
143: 2 = Ctrl-B
144: 239
145: 2147418252 = POINTER TO 140
146: 3 = Ctrl-C
147: 16 = Ctrl-P
148: 3 = Ctrl-C
149: 226
150: 2147418257 = POINTER TO 145
151: 4 = Ctrl-D
10000: 2147428115 = POINTER TO 10003
10001: 1 = Ctrl-A
10002: 10 = Ctrl-J
10003: 2147428121 = POINTER TO 10009
10004: 3 = Ctrl-C
10005: 10 = Ctrl-J
10006: 5 = Ctrl-E
10007: 16 = Ctrl-P
10008: 0 = Ctrl-@

PC= 197   ESP= 2
FP= POINTER TO 150   ASP= POINTER TO 152
HP= POINTER TO 10009   HMAX= POINTER TO 15000

Total number of instructions executed: 519
Last instruction to be executed: 196: PUSHNUM 1

Expression evaluation stack:
EXPRSTACK[1] = 1
EXPRSTACK[0] = 16
```

[...]RTANT: Read the remarks at the bottom of p. 4 before you answer the following questions.

[...] For each of the methods `outputSeq()`, `readMinMax()`, `readArrSiz()`, and `exam()` [...] program on the previous page, say how many locations (if any) are allocated in its stackframes [...] ***[...]l variables declared within the method's body.***

[...]ERS: `outputSeq()`:____ `readMinMax()`:____ `readArrSiz()`:____ `exam()`:____

(b)**[1 pt.]** Write down the size of a stackframe of `outputSeq()`, `readMinMax()`, `readArrSiz()` and `exam()`.

ANSWERS: `outputSeq()`:____ `readMinMax()`:____ `readArrSiz()`:____ `exam()`:____

***Questions* (c) – (i) *ask you about the state of the virtual machine at the time of the debugging stop* after execution of 519 instructions:**

(c)**[0.5 pt.]** What is the data memory ***address*** of `results[3]`? (`results` is a static variable declared on the 3rd non-blank line of the program on p. 5.) **ANSWER:** _____

(d)**[0.5 pt.]** Name the method whose code is being executed. **ANSWER:** _____

(e)**[0.5 pt.]** What is the data memory ***address*** of offset 0 in the currently executing method activation's stackframe? **ANSWER:** _____

(f)**[1 pt.]** What are the ***addresses*** of the data memory locations that constitute the stackframe of the currently executing method activation? **ANSWER:** _____

(g)**[1.5 pts.]** What values are stored in the locations of the formal parameter(s) and local variable(s) of the executing method activation? For ***each*** parameter and ***each*** local variable, write down its ***name*** followed by the ***value*** stored in its location:

ANSWER: _____

h)**[0.5 pt.]** Name the method that called the executing method. **ANSWER:** _____

)**[1 pt.]** What are the ***addresses*** of the data memory locations that constitute the stackframe of the caller? (This must be a stackframe of the method named in (h).)

ANSWER: _____

v suppose the debugging stop had *not* occurred after execution of 519 instructions.

.5 pt.] Write down the ***code memory addresses*** of the ***three*** instructions that would have been ecuted next, in the order in which those instructions would have been executed. (In other words te down the ***code memory addresses*** of the instructions that would have been the 520th, 521st nd instructions to be executed, in that order.)

ANSWERS: _____ _____

t.] ***Immediately after*** execution of the three instructions you tified in (j), what would **ASP** and **ESP** have contained? **ANSWERS:** **ASP:** _____ **ESP:** _____

document that was provided to you earlier this semester:

PUSHSTATADDR *a*	Pushes a pointer to the data memory location whose address is *a*.
PUSHLOCADDR *s*	Pushes a pointer to the data memory location that is at offset *s* in the currently executing method activation's stackframe.
WRITEINT	Pops an item *i*, which is assumed to be an integer. Writes the integer *i* to the screen.

You are reminded that:

(i) The array `EXPRSTACK[]` represents the expression evaluation stack.

(ii) At any time during execution of virtual machine code, `EXPRSTACK[0]`, ..., `EXPRSTACK[ESP-1]` are the items that are on the expression evaluation stack, and `EXPRSTACK[ESP-1]` is the *topmost* item on the stack.

(iii) A pointer to the data memory location whose address is *a* is represented by *a* + `POINTERTAG`.

(iv) The operand of any `OneOperandInstruction` is represented by its `operand` instance variable.

(v) `FP` represents the TinyJ virtual machine's frame pointer register (which contains a pointer to offset 0 in the stackframe of the currently executing method activation).

You may assume that each of the three classes below is in a file that begins with the following 3 lines:

```
package TJasn.virtualMachine;
import static TJasn.virtualMachine.CodeInterpreter.*;
import TJasn.TJ;
```

```
public class WRITEINTinstr extends ZeroOperandInstruction {

  void execute()
  {




  }
```

```
public class PUSHSTATADDRinstr extends OneOperandInstruction {

  void execute()
  {




  }
```

```
public class PUSHLOCADDRinstr extends OneOperandInstruction {

  void execute()
  {
```

```
3  class Shape {
4    public:
5              void drawShapeWithBorder() { drawBorder(); drawShape(); }
6              void drawBorder(){ cout << "stub B " ; }
7      virtual void drawShape() { cout << "stub S " ;}
8  };
9
10 class Circle : public Shape {
11   public:  void drawShape() { cout << "stub C ";}
12 };
13
14 class Triangle : public Shape {
15   public:  void drawShape() { cout << "stub T ";}
16 };
17
18 int main()
19 {
20   Shape *p;
21   char x;
22
23   cin >> x;
24   switch (x) {
25      case 'T':  p = new Triangle;  break;
26      case 'C':  p = new Circle;  break;
27      default :  p = new Shape;
28   }
29   p->drawShapeWithBorder();  p->drawShape();
30 }
```

(i)[**1 pt.**] Suppose the character 'T' is read on line 23, so that line 25 is executed. In this case, happens when line 29 is executed? [Circle the one correct answer.]

(a) `stub B stub S stub S` is output. (b) `stub B stub T stub S` is output

(c) `stub B stub S stub T` is output. (d) `stub B stub T stub T` is output

(e) An error occurs when line 29 is executed because of the word `virtual` on line 7.

(ii)[**1 pt.**] Suppose the character 'C' is read on line 23, so that line 26 is executed. In this case happens when line 29 is executed?

(a) `stub B stub S stub S` is output. (b) `stub B stub C stub S` is output

(c) `stub B stub S stub C` is output. (d) `stub B stub C stub C` is output

(e) An error occurs when line 29 is executed because of the word `virtual` on line 7.

i)[**1 pt.**] Suppose we ***delete*** the word `virtual` on line 7 and execute the changed program character 'T' is read on line 23, what happens when line 29 is executed?

(a) `stub B stub S stub S` is output. (b) `stub B stub T stub S` is output

(c) `stub B stub S stub T` is output. (d) `stub B stub T stub T` is output

line 29 is executed because the word `virtual` is missing

14. Study the following program and then complete the table below the program to show program's output would be if the parameters **x** and **y** of **test()** are passed by valu value-result, by Algol W-style value-result, by reference, or by name. (Assume that is written in a language whose semantics is the same as that of Java except possibly parameter passing mode that is used.)

```
class Fall22_8pm {
    static int b[] = new int[2],   i = 1;

    public static void main(String args[])
    {
        b[0] = 4;  b[1] = 2;  test(b[i], b[1]);
        System.out.println(b[0] + " " + b[1] + " " + i);
    }

    static void test (int x, int y)
    {
        x = y + 6;
        y = b[0] * b[1];
        i = 0;                    " "
        System.out.print(x + " " + y + " ");
    }
}
```

Output for each parameter passing mode:

	x	y	b[0]	b[1]	i
value:					
value-result:					
value-result (Algol W):					
eference:					
ame:					