Assigned Reading for the Final Exam: Read *Sec. 5.2 on pp. 155 – 60 and the discussion of call by name on p. 165 in Sethi*. Also, read *the comments on call / pass by value-result on the next page*. [You should ***already*** understand call / pass by value and call / pass by reference, as those parameter passing modes are used in C++. So most of the material on pp. 155 – 8 should be familiar to you.] There will be a problem on the Final Exam, worth 5 pts., that is similar in nature to the examples below. (The maximum score on the Final Exam will be 40 pts.) For more on call/pass by name, see pp. 455–6 of this book, which can be viewed via these links: **p. 455*** **p. 456**

Example 1 (based on an example in a different programming language in: L. B. Wilson and R. G. Clark, *Comparative Programming Languages*, Addison-Wesley, 3rd Ed., 2001, pp. 137 – 8):

Complete the table below to show the output that is produced when the following program is executed. When completing each row of the table, assume that parameters are passed by the indicated mode.

```
class Example {
  static int e;
  static int a[] = new int[3];

  static void test (int x)
  {
    a[1] = 6;
    e = 2;
    x += 3;
  }

  public static void main(String[] args)
  {
    a[1] = 1; a[2] = 2; e = 1;
    test(a[e]);
    System.out.println(a[1]+" "+a[2]+" "+e);
  }
}
```

Output for each parameter passing mode:

	a[1]	a[2]	e
value:			
reference:			
value-result:			
value-result (Algol W):			
name:			

Example 2 (based on an old exam question):

Complete the table below to show the output that is produced when the following program is executed. When completing each row of the table, assume that parameters are passed by the indicated mode.

```
class FinalExam {
  static int e = 1;
  static int a[] = {0,1,2};

  public static void main(String args[])
  {
    test(a[e], a[e-1]);
    System.out.println(a[0] + " " + a[1]
                    + " " + a[2] + " " + e);
  }

  static void test (int x, int y)
  {
    a[1] = 6;
    e = 2;
    x += 3;
    y--;
    System.out.print(x + " " + y + " ");
  }
}
```

Output for each parameter passing mode:

	x	y	a[0]	a[1]	a[2]	e
value:						
reference:						
value-result:						
value-result (Algol W):						
name:						

SOLUTIONS

Problem 1:

Output for each parameter passing mode:

	a[1]	a[2]	e
value:	6	2	2
reference:	9	2	2
value-result:	4	2	2
value-result (Algol W):	6	4	2
name:	6	5	2

Problem 2:

Output for each parameter passing mode:

	x	y	a[0]	a[1]	a[2]	e
value:	4	-1	0	6	2	2
reference:	9	-1	-1	9	2	2
value-result:	4	-1	-1	4	2	2
value-result (Algol W):	4	-1	0	-1	4	2
name:	5	5	0	5	5	2

*While Algol W is listed on p. 455 as a language that dropped pass by name, name parameters were in fact supported by Algol W, though the best known Algol W compiler discouraged the use of name parameters by issuing a warning (warning 2031 on p. 80 of this manual) if they were used.

Comments on Call / Pass by Value-Result

There are two subtleties relating to call / pass by value-result:

1. If the same variable is passed as two different arguments, then the final value of that argument variable *may depend on the order in which formal parameter values are copied back into the actual argument variables' locations.* As an example, consider a function of the form

```
void p(int a, int b)
{
  a = 4;
  b = 7;
}
```

where the parameters `a` and `b` are passed by value-result. Suppose this function `p()` is called within the function `main()` as follows:

```
p(j,j);
System.out.print(j)
```

Then, when control returns to `main()` from the call `p(j,j)`, the following must happen:

(i) The final value of formal parameter `a` (i.e., 4) is copied into argument variable `j`.

(ii) The final value of formal parameter `b` (i.e., 7) is copied into argument variable `j`.

The definition of call/pass by value-result does ***not*** say which of (i) and (ii) occurs first. If (i) occurs after (ii), then the final value of `j` will be 4, the final value of formal parameter `a`. But if (ii) occurs after (i), then the final value of `j` will be 7, the final value of parameter `b`. [***If you were given the above code and asked to write down the output of*** `System.out.print(j)` ***assuming pass by value-result, then neither "4" nor "7" would be correct—you would be expected to write "4 or 7".***]

2. The discussion of call/pass by value-result on pp. 159–60 of Sethi applies to "standard" pass by value-result. A slightly different version of pass by value-result was used in the language Algol W. In *standard* pass by value-result, when control returns to the caller the final value of each formal parameter is copied into the *location that belonged to the corresponding actual argument variable at the time the call was made* (i.e., into the *location that belonged to the actual argument variable immediately before the called function's body was executed*). But, in ***Algol W style*** pass by value-result, when control returns to the caller the final value of each formal parameter is copied into the *location that belongs to the corresponding actual argument variable at that time* (i.e., into the *location that belongs to the actual argument variable immediately after the called function's body is executed*).*

These two versions of pass by value-result may produce different results if the actual argument is, e.g., an indexed variable `v[`*expr*`]` whose index expression *expr* changes in value during execution of the called function's body. As an example, consider a function

```
void q(int c)
{
  c = 55;
  i = 17;
}
```

where `i` is a global variable and `i`'s parameter `c` is passed by value-result. Suppose this function `q` is called within the function `main` as follows:

```
i = 23;
q(arr[i]);
```

In ***standard*** pass by value-result, when control returns to `main` from the call `q(arr[i])` the final value of `q`'s parameter `c` (i.e., 55) will be copied into `arr[23]` because `i`'*s value was 23 immediately before the body of* `q` *was executed* (and so `arr[i]` was `arr[23]`). However, in ***Algol W style*** pass by value-result the final value of parameter `c` will instead be copied into `arr[17]` because `i`'*s value is 17 immediately after the body of* `q` *is executed* (so that `arr[i]` is `arr[17]`).

*Students interested in learning more may refer to subsections 5.3.2.2 and 7.3.2 either in Part II of N. Wirth and C. A. R. Hoare, A contribution to the development of Algol, *Communications of the ACM*, vol. 9, 1966, 413–32 or in the Language Description section of this manual. (However, no exam question will assume that students have read those subsections.)

The dump on pp. 5 − 7 below was produced when TJasn.TJ compiled the TinyJ program on p. 2 and then executed the generated code with a debugging stop after execution of exactly **23,172** instructions with the following sequence of input values: 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. The code that was generated is shown on pp. 3 − 5. Note that the INITSTKFRM instructions in this code are at code memory addresses 4, 322, 391, and 490.

Examples of Possible Questions Relating to the Dump **(Some explanatory comments are given on p. 8.)**

- **Recommendation**: Work on the ***green* questions** shortly before or soon after doing TJ Assignment 2.
  Hint for question 2: If *x* is an array variable and *x* != null, then *x* stores a pointer to the location *x*`[0]`, and address of *x*`[`*n*`]` = *n* + address of *x*`[0]`. If the array is an array of arrays, then each array element *x*`[`*k*`]` stores a pointer to the location *x*`[`*k*`][0]`, and address of *x*`[`*k*`][`*n*`]` = *n* + address of *x*`[`*k*`][0]`.
  Hint for questions 4 and 5: The FP register—see the "`PC=503 ESP=2 FP= …`" line of the dump—points to the location at offset 0 in the currently executing method activation's stackframe.
  Hint for questions 10, 11, 12, and 15: The ESP register—see the "`PC=503 ESP=2 FP= …`" line of the dump—contains a count of the number of items that are currently on EXPRSTACK. When ESP > 0, `EXPRSTACK[ESP-1]` is the ***top*** item on EXPRSTACK and `EXPRSTACK[0]` is the ***bottom*** item.
- **Recommendation**: Work on the ***red* questions** soon after the last lecture of the course.

1.(a) For each method, say how many locations are allocated to local variables in its stackframe.
ANSWER: `main: 7; readRow: 2`
`transpose: 5; writeOut: 3`

(b) Write down the size of a stackframe of `readRow()`, `transpose()`, and `writeOut()`.
ANSWER: `readRow: 7; transpose: 10;`
`writeOut: 8`

Questions 2 – 9 are about the state of the TinyJ virtual machine at the time of the debugging stop after execution of 23,172 instructions:

2. Consider the static variables mat, count, and tm. What values are stored in the following locations?
(a) `count` ANSWER: `1`
(b) `mat[2][4]` ANSWER: `5`
(c) `tm[0][3][2]` ANSWER: `8`

3. Which method is being executed?
ANSWER: `writeOut`

4. Which data memory locations constitute the stackframe of the executing method?
ANSWER: `addresses 200 through 207`

5. What values are stored in the stackframe locations of the local variables and formal parameters of the executing method?
ANSWER: `i = 742, j = 0, mm = null,`
`rows = -2, cols = 0,`
`matr[ ][ ] = PTR to 10085`

6. Which method called the executing method?
ANSWER: `writeOut`

7. Which method called the caller? ANSWER: `readRow`

8. Which method called the caller's caller? And which method called that method?
ANSWER: `transpose; main`

9. What are the addresses of main's local variables `h` and `j`?
ANSWER: `h's addr is 171; j's is 172`

Now suppose the debugging stop had not occurred.

10. What would be the code memory addresses of the next 10 instructions to be executed (i.e., the 23,173rd through 23,182nd instructions to be executed)?
ANSWER: `503 through 511, then 500.`

11. What output, if any, would be produced by execution of those instructions? ANSWER: `None`

12. Which data memory locations, if any, would be changed in value by execution of the 10 instructions? Name the variable(s) whose values are stored there.
ANSWER: `address 205; i`

13. Write down what the PC, FP, ASP and ESP registers would contain *after execution of the first 3 of the above 10 instructions*, and also write down the value or values of EXPRSTACK[j] for 0 ≤ j < ESP.
ANSWER: `PC=506, FP=PTR TO 204,`
`ASP=PTR TO 208, ESP=1`
`EXPRSTACK[0]=PTR TO 205`

14. When the currently executing method activation RETURNs to its caller, what will PC, FP, and ASP be set to?
ANSWER: `PC=568, FP=PTR TO 196`
`ASP=PTR TO 200`

15. Answer questions 10 – 12 and 13 (regarding the 23,173rd through 23,182nd instructions to be executed) again under the (very unlikely!) assumption that, immediately before executing the 23,173rd instruction, soft errors in one of the computer's memory chips change the LT instruction at code memory address 503 to a GE instruction.
ANSWER: `503 and 504, then 512 - 519.`
`Output: 1 then newline`
`No data memory location's`
`value is changed.`
`After execution of the first`
`3 of the 10 instructions,`
`PC=513, FP=PTR TO 204,`
`ASP=PTR TO 208, ESP=1`
`EXPRSTACK[0]=PTR TO 1.`

```
import java.util.Scanner;

class DumpEx {

  static int mat[ ][ ], count;

  static int tm[ ][ ][ ] = new int[5][ ][ ];


  public static void main (String args[ ])
  {
    int r[ ] = new int[5], c[ ] = new int[5], n = 1;
    int layer = -1;

    while (n == 1) {
      if (layer < 4)
        layer = layer + 1;
      else
        layer = 0;

      Scanner input = new Scanner(System.in);

      System.out.print("Enter number of rows: ");
      r[layer] = input.nextInt();
      System.out.print("Enter number of columns: ");
      c[layer] = input.nextInt();

      mat = new int [r[layer]][ ];
      tm[layer] = mat;

      int i = 0;

      while (i < r[layer]) {
        mat[i] = new int[c[layer]];
        readRow(i + 1, mat[i], c[layer]);
        i = i + 1;
      }

      int h = 0;

      while (h <= layer) {
        System.out.println("Given matrix: ");
        writeOut(r[h], c[h], tm[h]);
        System.out.println("Transposed matrix: ");
        writeOut(c[h], r[h],
                 transpose(tm[h], r[h], c[h]));
        h = h + 1;
      }

      System.out.println("Doubled matrices: ");

      h = 0;
      while (h <= layer) {
        i = 0;
        while (i < r[h]) {
          int j = 0;
          while (j < c[h]) {
            tm[h][i][j] = tm[h][i][j] * 2;
            System.out.print(tm[h][i][j]);
            System.out.print(" ");
            j = j + 1;
          }
          System.out.println();
          i = i + 1;
        }
        h = h + 1;
        System.out.println("\n");
      }

      System.out.print
           ("\n\nType 1 to continue, 0 to quit: ");
      n = input.nextInt();
    }
  }

  static void readRow(int rowNum, int m[ ], int c)
  {
    if (rowNum >= 0) {
      System.out.print("Row ");
      System.out.println(rowNum);
    }
    int i = 0;
    while (i < c) {
      if (rowNum == -1) {
        int mm[ ][ ] = new int[1][ ];
        writeOut(-1, 10, mm);
        i = i + 1;
      }
      else {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter value in column ");
        System.out.print(i+1);
        System.out.print(": ");
        m[i] = input.nextInt();
        i = i + 1;
      }
    }
  }


  static int[ ][ ] transpose(int m[ ][ ], int r,
                                                int c)
  {
    int k,  i,  m1[ ][ ] = new int[c][ ];
    k = 0;
    while (k < c) {
      m1[k] = new int[r];
      k = k + 1;
    }
    i = 0;
    while (i < r) {
      int j = 0;
      while (j < c) {
        m1[j][i] = m[i][j];
        int mm[ ] = new int [1];
        readRow(-1, mm, 10);
        j = j + 1;
      }
      i = i + 1;
    }
    return m1;
  }


  static void writeOut (int rows, int cols,
                                     int matr[ ][ ])
  {
    int i = 0;
    if (rows == -2) {
      while (i < 1000) i = i + 1;
      System.out.println(count);
      count = count+1;
    }
    while (i < rows | rows == -1 & i < cols) {
        int j = 0;
        while (j < cols) {
          if (rows == -1) {
            int mm[ ][ ] = new int[1][ ];
            writeOut(-2,0,mm);
            j = j + 1;
          }
          else {
            System.out.print(matr[i][j]);
            System.out.print(" ");
            j = j + 1;
          }
        }
        if (rows >= 0) System.out.println();
        i = i + 1;
    }
  }
}
```

```
Instructions Generated:

0:      PUSHSTATADDR    2
1:      PUSHNUM         5
2:      HEAPALLOC
3:      SAVETOADDR
4:      INITSTKFRM      7
5:      PUSHLOCADDR     1
6:      PUSHNUM         5
7:      HEAPALLOC
8:      SAVETOADDR
9:      PUSHLOCADDR     2
10:     PUSHNUM         5
11:     HEAPALLOC
12:     SAVETOADDR
13:     PUSHLOCADDR     3
14:     PUSHNUM         1
15:     SAVETOADDR
16:     PUSHLOCADDR     4
17:     PUSHNUM         1
18:     CHANGESIGN
19:     SAVETOADDR
20:     PUSHLOCADDR     3
21:     LOADFROMADDR
22:     PUSHNUM         1
23:     EQ
24:     JUMPONFALSE     321
25:     PUSHLOCADDR     4
26:     LOADFROMADDR
27:     PUSHNUM         4
28:     LT
29:     JUMPONFALSE     37
30:     PUSHLOCADDR     4
31:     PUSHLOCADDR     4
32:     LOADFROMADDR
33:     PUSHNUM         1
34:     ADD
35:     SAVETOADDR
36:     JUMP            40
37:     PUSHLOCADDR     4
38:     PUSHNUM         0
39:     SAVETOADDR
40:     WRITESTRING     3     24
41:     PUSHLOCADDR     1
42:     LOADFROMADDR
43:     PUSHLOCADDR     4
44:     LOADFROMADDR
45:     ADDTOPTR
46:     READINT
47:     SAVETOADDR
48:     WRITESTRING     25    49
49:     PUSHLOCADDR     2
50:     LOADFROMADDR
51:     PUSHLOCADDR     4
52:     LOADFROMADDR
53:     ADDTOPTR
54:     READINT
55:     SAVETOADDR
56:     PUSHSTATADDR    0
57:     PUSHLOCADDR     1
58:     LOADFROMADDR
59:     PUSHLOCADDR     4
60:     LOADFROMADDR
61:     ADDTOPTR
62:     LOADFROMADDR
63:     HEAPALLOC
64:     SAVETOADDR
65:     PUSHSTATADDR    2
66:     LOADFROMADDR
67:     PUSHLOCADDR     4
68:     LOADFROMADDR
69:     ADDTOPTR
70:     PUSHSTATADDR    0
71:     LOADFROMADDR
72:     SAVETOADDR
73:     PUSHLOCADDR     5
74:     PUSHNUM         0
75:     SAVETOADDR
76:     PUSHLOCADDR     5
77:     LOADFROMADDR
78:     PUSHLOCADDR     1
79:     LOADFROMADDR
80:     PUSHLOCADDR     4
81:     LOADFROMADDR
82:     ADDTOPTR
83:     LOADFROMADDR
84:     LT
85:     JUMPONFALSE     127
86:     PUSHSTATADDR    0
87:     LOADFROMADDR
88:     PUSHLOCADDR     5
89:     LOADFROMADDR
90:     ADDTOPTR
91:     PUSHLOCADDR     2
92:     LOADFROMADDR
93:     PUSHLOCADDR     4
94:     LOADFROMADDR
95:     ADDTOPTR
96:     LOADFROMADDR
97:     HEAPALLOC
98:     SAVETOADDR
99:     PUSHLOCADDR     5
100:    LOADFROMADDR
101:    PUSHNUM         1
102:    ADD
103:    PASSPARAM
104:    PUSHSTATADDR    0
105:    LOADFROMADDR
106:    PUSHLOCADDR     5
107:    LOADFROMADDR
108:    ADDTOPTR
109:    LOADFROMADDR
110:    PASSPARAM
111:    PUSHLOCADDR     2
112:    LOADFROMADDR
113:    PUSHLOCADDR     4
114:    LOADFROMADDR
115:    ADDTOPTR
116:    LOADFROMADDR
117:    PASSPARAM
118:    CALLSTATMETHOD  322
119:    NOP
120:    PUSHLOCADDR     5
121:    PUSHLOCADDR     5
122:    LOADFROMADDR
123:    PUSHNUM         1
124:    ADD
125:    SAVETOADDR
126:    JUMP            76
127:    PUSHLOCADDR     6
128:    PUSHNUM         0
129:    SAVETOADDR
130:    PUSHLOCADDR     6
131:    LOADFROMADDR
132:    PUSHLOCADDR     4
133:    LOADFROMADDR
134:    LE
135:    JUMPONFALSE     209
136:    WRITESTRING     50    63
137:    WRITELNOP
138:    PUSHLOCADDR     1
139:    LOADFROMADDR
140:    PUSHLOCADDR     6
141:    LOADFROMADDR
142:    ADDTOPTR
143:    LOADFROMADDR
144:    PASSPARAM
145:    PUSHLOCADDR     2
146:    LOADFROMADDR
147:    PUSHLOCADDR     6
148:    LOADFROMADDR
149:    ADDTOPTR
150:    LOADFROMADDR
151:    PASSPARAM
152:    PUSHSTATADDR    2
153:    LOADFROMADDR
154:    PUSHLOCADDR     6
155:    LOADFROMADDR
156:    ADDTOPTR
157:    LOADFROMADDR
158:    PASSPARAM
159:    CALLSTATMETHOD  490
160:    NOP
161:    WRITESTRING     64    82
162:    WRITELNOP
163:    PUSHLOCADDR     2
164:    LOADFROMADDR
165:    PUSHLOCADDR     6
166:    LOADFROMADDR
167:    ADDTOPTR
168:    LOADFROMADDR
169:    PASSPARAM
170:    PUSHLOCADDR     1
171:    LOADFROMADDR
172:    PUSHLOCADDR     6
173:    LOADFROMADDR
174:    ADDTOPTR
175:    LOADFROMADDR
176:    PASSPARAM
177:    PUSHSTATADDR    2
178:    LOADFROMADDR
179:    PUSHLOCADDR     6
180:    LOADFROMADDR
181:    ADDTOPTR
182:    LOADFROMADDR
183:    PASSPARAM
184:    PUSHLOCADDR     1
185:    LOADFROMADDR
186:    PUSHLOCADDR     6
187:    LOADFROMADDR
188:    ADDTOPTR
189:    LOADFROMADDR
190:    PASSPARAM
191:    PUSHLOCADDR     2
192:    LOADFROMADDR
193:    PUSHLOCADDR     6
194:    LOADFROMADDR
195:    ADDTOPTR
196:    LOADFROMADDR
197:    PASSPARAM
198:    CALLSTATMETHOD  391
199:    PASSPARAM
200:    CALLSTATMETHOD  490
201:    NOP
202:    PUSHLOCADDR     6
203:    PUSHLOCADDR     6
204:    LOADFROMADDR
205:    PUSHNUM         1
206:    ADD
207:    SAVETOADDR
208:    JUMP            130
209:    WRITESTRING     83    100
210:    WRITELNOP
211:    PUSHLOCADDR     6
212:    PUSHNUM         0
213:    SAVETOADDR
214:    PUSHLOCADDR     6
215:    LOADFROMADDR
216:    PUSHLOCADDR     4
217:    LOADFROMADDR
218:    LE
219:    JUMPONFALSE     316
220:    PUSHLOCADDR     5
221:    PUSHNUM         0
222:    SAVETOADDR
```

```
223:    PUSHLOCADDR     5
224:    LOADFROMADDR
225:    PUSHLOCADDR     1
226:    LOADFROMADDR
227:    PUSHLOCADDR     6
228:    LOADFROMADDR
229:    ADDTOPTR
230:    LOADFROMADDR
231:    LT
232:    JUMPONFALSE     307
233:    PUSHLOCADDR     7
234:    PUSHNUM         0
235:    SAVETOADDR
236:    PUSHLOCADDR     7
237:    LOADFROMADDR
238:    PUSHLOCADDR     2
239:    LOADFROMADDR
240:    PUSHLOCADDR     6
241:    LOADFROMADDR
242:    ADDTOPTR
243:    LOADFROMADDR
244:    LT
245:    JUMPONFALSE     299
246:    PUSHSTATADDR    2
247:    LOADFROMADDR
248:    PUSHLOCADDR     6
249:    LOADFROMADDR
250:    ADDTOPTR
251:    LOADFROMADDR
252:    PUSHLOCADDR     5
253:    LOADFROMADDR
254:    ADDTOPTR
255:    LOADFROMADDR
256:    PUSHLOCADDR     7
257:    LOADFROMADDR
258:    ADDTOPTR
259:    PUSHSTATADDR    2
260:    LOADFROMADDR
261:    PUSHLOCADDR     6
262:    LOADFROMADDR
263:    ADDTOPTR
264:    LOADFROMADDR
265:    PUSHLOCADDR     5
266:    LOADFROMADDR
267:    ADDTOPTR
268:    LOADFROMADDR
269:    PUSHLOCADDR     7
270:    LOADFROMADDR
271:    ADDTOPTR
272:    LOADFROMADDR
273:    PUSHNUM         2
274:    MUL
275:    SAVETOADDR
276:    PUSHSTATADDR    2
277:    LOADFROMADDR
278:    PUSHLOCADDR     6
279:    LOADFROMADDR
280:    ADDTOPTR
281:    LOADFROMADDR
282:    PUSHLOCADDR     5
283:    LOADFROMADDR
284:    ADDTOPTR
285:    LOADFROMADDR
286:    PUSHLOCADDR     7
287:    LOADFROMADDR
288:    ADDTOPTR
289:    LOADFROMADDR
290:    WRITEINT
291:    WRITESTRING     101   101
292:    PUSHLOCADDR     7
293:    PUSHLOCADDR     7
294:    LOADFROMADDR
295:    PUSHNUM         1
296:    ADD
297:    SAVETOADDR
298:    JUMP            236
299:    WRITELNOP
300:    PUSHLOCADDR     5
301:    PUSHLOCADDR     5
302:    LOADFROMADDR
303:    PUSHNUM         1
304:    ADD
305:    SAVETOADDR
306:    JUMP            223
307:    PUSHLOCADDR     6
308:    PUSHLOCADDR     6
309:    LOADFROMADDR
310:    PUSHNUM         1
311:    ADD
312:    SAVETOADDR
313:    WRITESTRING     102   102
314:    WRITELNOP
315:    JUMP            214
316:    WRITESTRING     103   135
317:    PUSHLOCADDR     3
318:    READINT
319:    SAVETOADDR
320:    JUMP            20
321:    STOP
322:    INITSTKFRM      2
323:    PUSHLOCADDR     -4
324:    LOADFROMADDR
325:    PUSHNUM         0
326:    GE
327:    JUMPONFALSE     333
328:    WRITESTRING     136   139
329:    PUSHLOCADDR     -4
330:    LOADFROMADDR
331:    WRITEINT
332:    WRITELNOP
333:    PUSHLOCADDR     1
334:    PUSHNUM         0
335:    SAVETOADDR
336:    PUSHLOCADDR     1
337:    LOADFROMADDR
338:    PUSHLOCADDR     -2
339:    LOADFROMADDR
340:    LT
341:    JUMPONFALSE     390
342:    PUSHLOCADDR     -4
343:    LOADFROMADDR
344:    PUSHNUM         1
345:    CHANGESIGN
346:    EQ
347:    JUMPONFALSE     369
348:    PUSHLOCADDR     2
349:    PUSHNUM         1
350:    HEAPALLOC
351:    SAVETOADDR
352:    PUSHNUM         1
353:    CHANGESIGN
354:    PASSPARAM
355:    PUSHNUM         10
356:    PASSPARAM
357:    PUSHLOCADDR     2
358:    LOADFROMADDR
359:    PASSPARAM
360:    CALLSTATMETHOD  490
361:    NOP
362:    PUSHLOCADDR     1
363:    PUSHLOCADDR     1
364:    LOADFROMADDR
365:    PUSHNUM         1
366:    ADD
367:    SAVETOADDR
368:    JUMP            389
369:    WRITESTRING     140   161
370:    PUSHLOCADDR     1
371:    LOADFROMADDR
372:    PUSHNUM         1
373:    ADD
374:    WRITEINT
375:    WRITESTRING     162   163
376:    PUSHLOCADDR     -3
377:    LOADFROMADDR
378:    PUSHLOCADDR     1
379:    LOADFROMADDR
380:    ADDTOPTR
381:    READINT
382:    SAVETOADDR
383:    PUSHLOCADDR     1
384:    PUSHLOCADDR     1
385:    LOADFROMADDR
386:    PUSHNUM         1
387:    ADD
388:    SAVETOADDR
389:    JUMP            336
390:    RETURN          3
391:    INITSTKFRM      5
392:    PUSHLOCADDR     3
393:    PUSHLOCADDR     -2
394:    LOADFROMADDR
395:    HEAPALLOC
396:    SAVETOADDR
397:    PUSHLOCADDR     1
398:    PUSHNUM         0
399:    SAVETOADDR
400:    PUSHLOCADDR     1
401:    LOADFROMADDR
402:    PUSHLOCADDR     -2
403:    LOADFROMADDR
404:    LT
405:    JUMPONFALSE     422
406:    PUSHLOCADDR     3
407:    LOADFROMADDR
408:    PUSHLOCADDR     1
409:    LOADFROMADDR
410:    ADDTOPTR
411:    PUSHLOCADDR     -3
412:    LOADFROMADDR
413:    HEAPALLOC
414:    SAVETOADDR
415:    PUSHLOCADDR     1
416:    PUSHLOCADDR     1
417:    LOADFROMADDR
418:    PUSHNUM         1
419:    ADD
420:    SAVETOADDR
421:    JUMP            400
422:    PUSHLOCADDR     2
423:    PUSHNUM         0
424:    SAVETOADDR
425:    PUSHLOCADDR     2
426:    LOADFROMADDR
427:    PUSHLOCADDR     -3
428:    LOADFROMADDR
429:    LT
430:    JUMPONFALSE     487
431:    PUSHLOCADDR     4
432:    PUSHNUM         0
433:    SAVETOADDR
434:    PUSHLOCADDR     4
435:    LOADFROMADDR
436:    PUSHLOCADDR     -2
437:    LOADFROMADDR
438:    LT
439:    JUMPONFALSE     480
440:    PUSHLOCADDR     3
441:    LOADFROMADDR
442:    PUSHLOCADDR     4
443:    LOADFROMADDR
444:    ADDTOPTR
445:    LOADFROMADDR
446:    PUSHLOCADDR     2
447:    LOADFROMADDR
```

```
448:    ADDTOPTR
449:    PUSHLOCADDR       -4
450:    LOADFROMADDR
451:    PUSHLOCADDR       2
452:    LOADFROMADDR
453:    ADDTOPTR
454:    LOADFROMADDR
455:    PUSHLOCADDR       4
456:    LOADFROMADDR
457:    ADDTOPTR
458:    LOADFROMADDR
459:    SAVETOADDR
460:    PUSHLOCADDR       5
461:    PUSHNUM           1
462:    HEAPALLOC
463:    SAVETOADDR
464:    PUSHNUM           1
465:    CHANGESIGN
466:    PASSPARAM
467:    PUSHLOCADDR       5
468:    LOADFROMADDR
469:    PASSPARAM
470:    PUSHNUM           10
471:    PASSPARAM
472:    CALLSTATMETHOD    322
473:    PUSHLOCADDR       4
474:    PUSHLOCADDR       4
475:    LOADFROMADDR
476:    PUSHNUM           1
477:    ADD
478:    SAVETOADDR
479:    JUMP              434
480:    PUSHLOCADDR       2
481:    PUSHLOCADDR       2
482:    LOADFROMADDR
483:    PUSHNUM           1
484:    ADD
485:    SAVETOADDR
486:    JUMP              425
487:    PUSHLOCADDR       3
488:    LOADFROMADDR
489:    RETURN            3
490:    INITSTKFRM        3
491:    PUSHLOCADDR       1
492:    PUSHNUM           0
493:    SAVETOADDR
494:    PUSHLOCADDR       -4
495:    LOADFROMADDR
496:    PUSHNUM           2
497:    CHANGESIGN
498:    EQ
499:    JUMPONFALSE       522
500:    PUSHLOCADDR       1
501:    LOADFROMADDR
502:    PUSHNUM           1000
503:    LT
504:    JUMPONFALSE       512
505:    PUSHLOCADDR       1
506:    PUSHLOCADDR       1
507:    LOADFROMADDR
508:    PUSHNUM           1
509:    ADD
510:    SAVETOADDR
511:    JUMP              500
512:    PUSHSTATADDR      1
513:    LOADFROMADDR
514:    WRITEINT
515:    WRITELNOP
516:    PUSHSTATADDR      1
517:    PUSHSTATADDR      1
518:    LOADFROMADDR
519:    PUSHNUM           1
520:    ADD
521:    SAVETOADDR
522:    PUSHLOCADDR       1
523:    LOADFROMADDR
524:    PUSHLOCADDR       -4
525:    LOADFROMADDR
526:    LT
527:    PUSHLOCADDR       -4
528:    LOADFROMADDR
529:    PUSHNUM           1
530:    CHANGESIGN
531:    EQ
532:    PUSHLOCADDR       1
533:    LOADFROMADDR
534:    PUSHLOCADDR       -3
535:    LOADFROMADDR
536:    LT
537:    AND
538:    OR
539:    JUMPONFALSE       607
540:    PUSHLOCADDR       2
541:    PUSHNUM           0
542:    SAVETOADDR
543:    PUSHLOCADDR       2
544:    LOADFROMADDR
545:    PUSHLOCADDR       -3
546:    LOADFROMADDR
547:    LT
548:    JUMPONFALSE       594
549:    PUSHLOCADDR       -4
550:    LOADFROMADDR
551:    PUSHNUM           1
552:    CHANGESIGN
553:    EQ
554:    JUMPONFALSE       575
555:    PUSHLOCADDR       3
556:    PUSHNUM           1
557:    HEAPALLOC
558:    SAVETOADDR
559:    PUSHNUM           2
560:    CHANGESIGN
561:    PASSPARAM
562:    PUSHNUM           0
563:    PASSPARAM
564:    PUSHLOCADDR       3
565:    LOADFROMADDR
566:    PASSPARAM
567:    CALLSTATMETHOD    490
568:    PUSHLOCADDR       2
569:    PUSHLOCADDR       2
570:    LOADFROMADDR
571:    PUSHNUM           1
572:    ADD
573:    SAVETOADDR
574:    JUMP              593
575:    PUSHLOCADDR       -2
576:    LOADFROMADDR
577:    PUSHLOCADDR       1
578:    LOADFROMADDR
579:    ADDTOPTR
580:    LOADFROMADDR
581:    PUSHLOCADDR       2
582:    LOADFROMADDR
583:    ADDTOPTR
584:    LOADFROMADDR
585:    WRITEINT
586:    WRITESTRING       164   164
587:    PUSHLOCADDR       2
588:    PUSHLOCADDR       2
589:    LOADFROMADDR
590:    PUSHNUM           1
591:    ADD
592:    SAVETOADDR
593:    JUMP              543
594:    PUSHLOCADDR       -4
595:    LOADFROMADDR
596:    PUSHNUM           0
597:    GE
598:    JUMPONFALSE       600
599:    WRITELNOP
600:    PUSHLOCADDR       1
601:    PUSHLOCADDR       1
602:    LOADFROMADDR
603:    PUSHNUM           1
604:    ADD
605:    SAVETOADDR
606:    JUMP              522
607:    RETURN            3
```

```
****** Debugging Stop ******

Data memory dump

  Data memory--addresses 0 to top of
    stack, and allocated heap locations:
  0: 2147428131 = PTR TO 10019
  1: 1 = Ctrl-A
  2: 2147428113 = PTR TO 10001
  3: 69 = 'E'
  4: 110 = 'n'
  5: 116 = 't'
  6: 101 = 'e'
  7: 114 = 'r'
  8: 32 = ' '
  9: 110 = 'n'
  10: 117 = 'u'
  11: 109 = 'm'
  12: 98 = 'b'
  13: 101 = 'e'
  14: 114 = 'r'
  15: 32 = ' '
  16: 111 = 'o'
  17: 102 = 'f'
  18: 32 = ' '
  19: 114 = 'r'
  20: 111 = 'o'
  21: 119 = 'w'
  22: 115 = 's'
  23: 58 = ':'
  24: 32 = ' '
  25: 69 = 'E'
  26: 110 = 'n'
  27: 116 = 't'
  28: 101 = 'e'
  29: 114 = 'r'
```

```
30: 32 = ' '
31: 110 = 'n'
32: 117 = 'u'
33: 109 = 'm'
34: 98 = 'b'
35: 101 = 'e'
36: 114 = 'r'
37: 32 = ' '
38: 111 = 'o'
39: 102 = 'f'
40: 32 = ' '
41: 99 = 'c'
42: 111 = 'o'
43: 108 = 'l'
44: 117 = 'u'
45: 109 = 'm'
46: 110 = 'n'
47: 115 = 's'
48: 58 = ':'
49: 32 = ' '
50: 71 = 'G'
51: 105 = 'i'
52: 118 = 'v'
53: 101 = 'e'
54: 110 = 'n'
55: 32 = ' '
56: 109 = 'm'
57: 97 = 'a'
58: 116 = 't'
59: 114 = 'r'
60: 105 = 'i'
61: 120 = 'x'
62: 58 = ':'
63: 32 = ' '
64: 84 = 'T'
65: 114 = 'r'
66: 97 = 'a'
67: 110 = 'n'
68: 115 = 's'
69: 112 = 'p'
70: 111 = 'o'
71: 115 = 's'
72: 101 = 'e'
73: 100 = 'd'
74: 32 = ' '
75: 109 = 'm'
76: 97 = 'a'
77: 116 = 't'
78: 114 = 'r'
79: 105 = 'i'
80: 120 = 'x'
81: 58 = ':'
82: 32 = ' '
83: 68 = 'D'
84: 111 = 'o'
85: 117 = 'u'
86: 98 = 'b'
87: 108 = 'l'
88: 101 = 'e'
89: 100 = 'd'
90: 32 = ' '
91: 109 = 'm'
92: 97 = 'a'
93: 116 = 't'
94: 114 = 'r'
95: 105 = 'i'
96: 99 = 'c'
97: 101 = 'e'
98: 115 = 's'
99: 58 = ':'
100: 32 = ' '
101: 32 = ' '
102: 10 = Ctrl-J
103: 10 = Ctrl-J
104: 10 = Ctrl-J
105: 84 = 'T'
106: 121 = 'y'
107: 112 = 'p'
108: 101 = 'e'
109: 32 = ' '
110: 49 = '1'
111: 32 = ' '
112: 116 = 't'
113: 111 = 'o'
114: 32 = ' '
115: 99 = 'c'
116: 111 = 'o'
117: 110 = 'n'
118: 116 = 't'
119: 105 = 'i'
120: 110 = 'n'
121: 117 = 'u'
122: 101 = 'e'
123: 44 = ','
124: 32 = ' '
125: 48 = '0'
126: 32 = ' '
127: 116 = 't'
128: 111 = 'o'
129: 32 = ' '
130: 113 = 'q'
131: 117 = 'u'
132: 105 = 'i'
133: 116 = 't'
134: 58 = ':'
135: 32 = ' '
136: 82 = 'R'
137: 111 = 'o'
138: 119 = 'w'
139: 32 = ' '
140: 69 = 'E'
141: 110 = 'n'
142: 116 = 't'
143: 101 = 'e'
144: 114 = 'r'
145: 32 = ' '
146: 118 = 'v'
147: 97 = 'a'
148: 108 = 'l'
149: 117 = 'u'
150: 101 = 'e'
151: 32 = ' '
152: 105 = 'i'
153: 110 = 'n'
154: 32 = ' '
155: 99 = 'c'
156: 111 = 'o'
157: 108 = 'l'
158: 117 = 'u'
159: 109 = 'm'
160: 110 = 'n'
161: 32 = ' '
162: 58 = ':'
163: 32 = ' '
164: 32 = ' '
165: 2147438112 = PTR TO 20000
166: 2147428119 = PTR TO 10007
167: 2147428125 = PTR TO 10013
168: 1 = Ctrl-A
169: 0 = Ctrl-@
170: 4 = Ctrl-D
171: 0 = Ctrl-@
172: 0 = Ctrl-@
173: 5 = Ctrl-E
174: 4 = Ctrl-D
175: 2147428131 = PTR TO 10019
176: 4 = Ctrl-D
177: 5 = Ctrl-E
178: 199
179: 2147418277 = PTR TO 165
```

```
180: 5 = Ctrl-E
181: 0 = Ctrl-@
182: 2147428160 = PTR TO 10048
183: 0 = Ctrl-@
184: 2147428191 = PTR TO 10079
185: -1
186: 2147428191 = PTR TO 10079
187: 10 = Ctrl-J
188: 473
189: 2147418291 = PTR TO 179
190: 0 = Ctrl-@
191: 2147428193 = PTR TO 10081
192: -1
193: 10 = Ctrl-J
194: 2147428193 = PTR TO 10081
195: 361
196: 2147418301 = PTR TO 189
197: 0 = Ctrl-@
198: 1 = Ctrl-A
199: 2147428197 = PTR TO 10085
200: -2
201: 0 = Ctrl-@
202: 2147428197 = PTR TO 10085
203: 568
204: 2147418308 = PTR TO 196
205: 742
206: 0 = Ctrl-@
207: 0 = Ctrl-@
10000: 2147428118 = PTR TO 10006
10001: 2147428131 = PTR TO 10019
10002: 0 = Ctrl-@
10003: 0 = Ctrl-@
10004: 0 = Ctrl-@
10005: 0 = Ctrl-@
10006: 2147428124 = PTR TO 10012
10007: 4 = Ctrl-D
10008: 0 = Ctrl-@
10009: 0 = Ctrl-@
10010: 0 = Ctrl-@
10011: 0 = Ctrl-@
10012: 2147428130 = PTR TO 10018
10013: 5 = Ctrl-E
10014: 0 = Ctrl-@
10015: 0 = Ctrl-@
10016: 0 = Ctrl-@
10017: 0 = Ctrl-@
10018: 2147428135 = PTR TO 10023
10019: 2147428136 = PTR TO 10024
10020: 2147428142 = PTR TO 10030
10021: 2147428148 = PTR TO 10036
10022: 2147428154 = PTR TO 10042
10023: 2147428141 = PTR TO 10029
10024: 1 = Ctrl-A
10025: 2 = Ctrl-B
10026: 3 = Ctrl-C
10027: 4 = Ctrl-D
10028: 5 = Ctrl-E
10029: 2147428147 = PTR TO 10035
10030: 6 = Ctrl-F
10031: 7 = Ctrl-G
10032: 8 = Ctrl-H
10033: 9 = Ctrl-I
10034: 0 = Ctrl-@
10035: 2147428153 = PTR TO 10041
10036: 1 = Ctrl-A
10037: 2 = Ctrl-B
10038: 3 = Ctrl-C
10039: 4 = Ctrl-D
10040: 5 = Ctrl-E
10041: 2147428159 = PTR TO 10047
10042: 6 = Ctrl-F
10043: 7 = Ctrl-G
10044: 8 = Ctrl-H
10045: 9 = Ctrl-I
10046: 0 = Ctrl-@
10047: 2147428165 = PTR TO 10053
10048: 2147428166 = PTR TO 10054
10049: 2147428171 = PTR TO 10059
10050: 2147428176 = PTR TO 10064
10051: 2147428181 = PTR TO 10069
10052: 2147428186 = PTR TO 10074
10053: 2147428170 = PTR TO 10058
10054: 1 = Ctrl-A
10055: 0 = Ctrl-@
10056: 0 = Ctrl-@
10057: 0 = Ctrl-@
10058: 2147428175 = PTR TO 10063
10059: 0 = Ctrl-@
10060: 0 = Ctrl-@
10061: 0 = Ctrl-@
10062: 0 = Ctrl-@
10063: 2147428180 = PTR TO 10068
10064: 0 = Ctrl-@
10065: 0 = Ctrl-@
10066: 0 = Ctrl-@
10067: 0 = Ctrl-@
10068: 2147428185 = PTR TO 10073
10069: 0 = Ctrl-@
10070: 0 = Ctrl-@
10071: 0 = Ctrl-@
10072: 0 = Ctrl-@
10073: 2147428190 = PTR TO 10078
10074: 0 = Ctrl-@
10075: 0 = Ctrl-@
10076: 0 = Ctrl-@
10077: 0 = Ctrl-@
10078: 2147428192 = PTR TO 10080
10079: 0 = Ctrl-@
10080: 2147428194 = PTR TO 10082
10081: 0 = Ctrl-@
10082: 2147428196 = PTR TO 10084
10083: 0 = Ctrl-@
10084: 2147428198 = PTR TO 10086
10085: 0 = Ctrl-@

PC=503  ESP=2  FP= PTR TO 204  ASP= PTR TO 208
HP= PTR TO 10086  HMAX= PTR TO 15000

Total number of instructions executed: 23172
Last instruction to be executed: 502:    PUSHNUM           1000

Expression evaluation stack:
EXPRSTACK[1]: 1000
EXPRSTACK[0]: 742
```

Comments on the Answers

1(a) The answers are deduced from the operands of the methods' INITSTKFRM instructions at code memory addresses 4, 322, 391, and 490. [It is also possible to work out the answers from the local variable declarations in each method. In main(), for example, the local variables r, c, n, and layer are given the stackframe offsets 1, 2, 3, and 4; i is given offset 5; h is given offset 6; and j is given offset 7. Note that the scopes of local variable declarations need to be taken into account. Thus if we add a declaration of a local variable hh inside the block of the while (h <= layer) { ... } loop that follows the declaration of h, then both hh and j will be given the offset 7 because the scopes of the declarations of hh and j will not overlap.]

(b) For any method other than main():

stackframe size = no. of parameters + 2 + no. of locations allocated to local vars.

The 2 extra locations are for the dynamic link (at offset 0) and the return address (at offset -1).
For main():

stackframe size = 1 + no. of locations allocated to local vars.

In TinyJ, main() is not called by another method and its stackframe has no return address. The INITSTKFRM instruction always allocates a location (offset 0) for a dynamic link, but in the case of main() that location serves no purpose and always points to the illegal data memory address 20000. (The highest legal data memory address is 19999; moreover, data memory addresses 10000 - 19999 are reserved for use as heap memory.)

2. mat's address is 0, count's address is 1, and tm's address is 2. (b) and (c) are intended to test your understanding of arrays. (c) is solved as follows: tm's address is 2. That location points to tm[0], so tm[0]'s addr is 10001. That location points to tm[0][0], so tm[0][0]'s addr is 10019, and hence tm[0][3]'s addr is 10022. That location points to tm[0][3][0], so tm[0][3][0]'s addr is 10042, and hence tm[0][3][2]'s addr is 10044. That location contains the answer, 8.

3. From the addresses of the INITSTKFRM instructions, we see that main's code is at 4 - 321, readRow's code is at 322 - 390, transpose's code is at 391 - 489, writeOut's code is at 490 - 607. The last instruction to be executed was at 502 (as stated on the 5th-last line of the dump). This is within writeOut's code.

4. We see from FP that offset 0 of the stackframe is at 204. The beginning and end of the stackframe can be deduced from this and the answers to 1(a) and (b) for writeOut.

5. The answers are deduced from the stackframe offsets of the parameters and variables, and the fact that offset 0 is at 204. [In fact the variables j and mm are not in scope in the "while (i < 1000)" loop that is being executed at this time. So the values stored in the locations of j and mm are just "garbage" values!]

6. Return addr (at offset -1, addr 203) is 568. This is within writeOut's code.

7,8. The dynamic link in the stackframe of the currently executing method points to addr 196. That location points to 189. That location points to 179. That location points to 165. Thus 196. 189, 179, and 165 are the addresses of the offset 0 locations in the stackframes of the caller, the caller's caller, the caller's caller's caller, and the caller's caller's caller's caller. The return addresses stored in the first three of these stackframes (at addresses 195, 188, and 178) are 361, 473, and 199, which are instructions in the code of readRow, transpose, and main, respectively.

Note: Another way to tell that the caller's caller's caller's caller is main is to observe that offset 0 in its stackframe (addr 165) points to the illegal data memory address 20000--see the above comment on question 1(b).

9. Offset 0 in main's frame is at addr 165 (see comments on questions 7,8). h's stackframe offset is 6 and j's is 7.

10. PC contains 503, so 503: LT is the first of the 10 instructions. We see from the last few lines of the dump (on p. 7) that at this time ESP = 2, EXPRSTACK[0] = 742, and EXPRSTACK[1] = 1000. Thus 1000 is on top of EXPRSTACK and 742 is the second item from the top. Since 742 < 1000, execution of LT replaces these two integers with the value 1 (which represents true), so the JUMPONFALSE at 504 does not jump after popping off this value.

11. Only WRITEINT, WRITESTRING, and WRITELNOP produce output.

12. Data memory is changed only by SAVETOADDR, PASSPARAM, CALLSTATMETHOD, INITSTKFRM, and HEAPALLOC. The only one of these that is executed here is SAVETOADDR (at 510). When this is executed, the pointer that is second from top on EXPRSTACK was put there by 505: PUSHLOCADDR 1. This refers to offset 1 in the currently executing method's stackframe, which is the location of i and has address 205 (since offset 0 has address 204).

[Note: HEAPALLOC changes data memory only because it sets the location that immediately precedes the block of heap memory it allocates to point to the location that immediately follows the block. This allows allocated blocks of heap memory that have become inaccessible to be deallocated by the garbage collector, and makes it possible to check at runtime that every array index is less than the length of the array.]

13,14. Questions like these are intended to test your understanding of what specific machine instructions do to the TinyJ virtual machine. Here the instructions you are being tested on are LT, JUMPONFALSE, PUSHLOCADDR, and RETURN.

The dump below was produced when TJasn.TJ compiled the TinyJ program on p. 2 and executed the generated code with a debugging stop after execution of **1,209,788** instructions. The sequence of input values was 4, 3, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2. The INITSTKFRM instructions in the generated code are:

```
4: INITSTKFRM 7    339: INITSTKFRM 4    408: INITSTKFRM 6    509: INITSTKFRM 5
```

The instructions at addresses 351 – 407 in the generated code are shown on page 3.

Some Examples of Possible Questions Relating to the Dump

- **Recommendation**: Work on the ***green*** **questions** shortly before or soon after doing TJ Assignment 2.
  Hint for question 2: If *x* is an array variable and *x* != null, then *x* stores a pointer to the location *x*[0], and address of *x*[*n*] = *n* + address of *x*[0]. If the array is an array of arrays, then each array element *x*[*k*] stores a pointer to the location *x*[*k*][0], and address of *x*[*k*][*n*] = *n* + address of *x*[*k*][0].
  Hint for questions 4 and 5: The FP register—see the "PC=383 ESP=3 FP= …" line of the dump—points to the location at offset 0 in the currently executing method activation's stackframe.
  Hint for questions 9–11, 14, and 15: The ESP register—see the "PC=383 ESP=3 FP= …" line of the dump—contains a count of the number of items that are currently on EXPRSTACK. When ESP > 0, EXPRSTACK[ESP-1] is the ***top*** item on EXPRSTACK and EXPRSTACK[0] is the ***bottom*** item.
- **Recommendation**: Work on the ***red*** **questions** soon after the last lecture of the course.

1.(a) For each method, say how many locations are allocated to local variables in its stackframes.

ANSWER:
```
main: 7; readRow: 4
transpose: 6; writeOut: 5
```

(b) Write down the size of a stackframe of `readRow()`, `transpose()`, and `writeOut()`.

ANSWER:
```
readRow: 10; transpose: 13;
writeOut: 10
```

Questions 2 – 8 are about the state of the TinyJ virtual machine at the time of the debugging stop after execution of 1,209,788 instructions:

2. Consider the static variables mat, count, and tm. What values are stored in the following locations?

(a) `count` ANSWER: 100

(b) `mat[3][2]` ANSWER: 2

(c) `tm[0][1][2]` ANSWER: 6

3. Which method is being executed? ANSWER: readRow

4. Which data memory locations constitute the stackframe of the executing method?

ANSWER: addresses 170 through 179

5. What values are stored in the stackframe locations of the formal parameters and first two local variables of the executing method?

ANSWER:
```
rowNum = -1
m = PTR TO 10059
c = 10   d = 6
i = 0    mm = PTR TO 10061
```

6. Which method called the executing method?

ANSWER: transpose

7. Which method called the caller? ANSWER: main

8. What are the addresses of main's local variables `layer` and `hhhhh`?

ANSWER:
```
layer's addr is 151;
hhhhh's addr is 154
```

Next, suppose the debugging stop had not occurred.

9. What would be the code memory addresses of the next 10 instructions to be executed (i.e., the 1,209,789th through 1,209,798th instructions to be executed)?

ANSWER: 383-5, then 406, then 353-8

10. What output, if any, would be produced by execution of these 10 instructions? ANSWER: None

11. Which data memory locations, if any, would be changed in value by execution of the 10 instructions? Name the variable(s) stored there and say what its/their value(s) is/are after execution of the 10 instructions.

ANSWER: address 176, i, 1

12. Write down what the PC, FP, ASP and ESP registers would contain after execution of the first 3 of the above 10 instructions.

ANSWER:
```
PC=406, FP=PTR TO 175,
ASP=PTR TO 180, ESP=0
```

13. When the currently executing method activation RETURNs to its caller, what will PC, FP, and ASP be set to?

ANSWER:
```
PC=492, FP=PTR TO 163
ASP=PTR TO 170
```

14. What is the code memory address of the next instruction to be executed after the execution of the 10 instructions listed in your answer to question 9?

ANSWER: 359

15. What will be on top of EXPRSTACK after execution of the instruction in your answer to question 14?

ANSWER: PTR TO 170

```
import java.util.Scanner;

class DumpEx2 {

  static int tm[][][] = new int[5][][];
  static int mat[][], count;
  static Scanner input = new Scanner(System.in);

  public static void main (String args[])
  {
    int r[] = new int[5], c[] = new int[5], n = 1;

    int layer = -1;

    while (n == 1) {
      if (layer < 4)
        layer = layer + 1;
      else
        layer = 0;

      System.out.print("Enter number of rows: ");
      r[layer] = input.nextInt();
      System.out.print("Enter number of columns: ");
      c[layer] = input.nextInt();

      mat = new int [r[layer]][ ];
      tm[layer] = mat;

      int i = 0;

      while (i < r[layer]) {
        int iiiii = i;
        mat[i] = new int[c[layer]];
        readRow(i + 1, mat[i], c[layer], iiiii);
        i = i + 1;
      }

      int h = 0;

      while (h <= layer) {
        int hhhhh = h*2;
        System.out.println("Given matrix: ");
        writeOut(r[h], c[h], tm[h]);
        System.out.println("Transposed matrix: ");
        writeOut(c[h], r[h],
           transpose(tm[h], r[h], c[h], hhhhh, hhhhh));
        h = h + 1;
      }
      h = 0;
      while (h <= layer) {
        i = 0;
        while (i < r[h]) {
          int j = 0;
          while (j < c[h]) {
            tm[h][i][j] = tm[h][i][j] * 2;
            System.out.print(tm[h][i][j]);
            System.out.print(" ");
            j = j + 1;
          }
          System.out.println();
          i = i + 1;
        }
        h = h + 1;
        System.out.println("\n");
      }

      int jjjjj;

      System.out.print
        ("\n\nType 1 to continue, 0 to quit: ");
      n = input.nextInt();
    }
  }

  static void readRow (int rowNum, int m[],  int c, int d)
  {
    if (rowNum >= 0) {
      System.out.print("Row ");
      System.out.println(rowNum);
    }
    int i = 0;
    while (i < c) {
      if (rowNum == -1) {
        int mm[][] = new int[1][];
        writeOut(-1, 10, mm);
        i = i + 1;
      }
      else {
        int p, q, r;
        System.out.print("Enter value in column ");
        System.out.print(i+1);
        System.out.print(": ");
        m[i] = input.nextInt();
        i = i + 1;
      }
    }
  }

  static int[][] transpose(int m[][], int r,
                           int c, int p, int q)
  {
    int temp, k, i, m1[][] = new int[c][];
    k = 0;
    while (k < c) {
      m1[k] = new int[r];
      k = k + 1;
    }
    i = 0;
    while (i < r) {
      int j = 0;
      while (j < c) {
        m1[j][i] = m[i][j];
        int mm[ ] = new int [1];
        readRow(-1, mm, 10, 6);
        j = j + 1;
      }
      i = i + 1;
    }
    return m1;
  }

  static void writeOut (int rows, int cols, int matrix[][])
  {
    int i = 0, tmp, tmp1;
    if (rows == -2) {
      while (i < 1000) i = i + 1;
      System.out.println(count);
      count = count+1;
    }
    while (i < rows | rows == -1 & i < cols) {
        int j = 0;
        while (j < cols) {
          if (rows == -1) {
            int mm[][] = new int[1][];
            writeOut(-2,0,mm);
            j = j + 1;
          }
          else {
            System.out.print(matrix[i][j]);
            System.out.print(" ");
            j = j + 1;
          }
        }
        if (rows >= 0) System.out.println();
        j = 1;
        i = i + j;
    }
  }
}
```

```
351:    PUSHNUM        0
352:    SAVETOADDR
353:    PUSHLOCADDR    1
354:    LOADFROMADDR
355:    PUSHLOCADDR    -3
356:    LOADFROMADDR
357:    LT
358:    JUMPONFALSE    407
359:    PUSHLOCADDR    -5
360:    LOADFROMADDR
361:    PUSHNUM        1
362:    CHANGESIGN
363:    EQ
364:    JUMPONFALSE    386
365:    PUSHLOCADDR    2
366:    PUSHNUM        1
367:    HEAPALLOC
368:    SAVETOADDR
369:    PUSHNUM        1
370:    CHANGESIGN
371:    PASSPARAM
372:    PUSHNUM        10
373:    PASSPARAM
374:    PUSHLOCADDR    2
375:    LOADFROMADDR
376:    PASSPARAM
377:    CALLSTATMETHOD 509
378:    NOP
379:    PUSHLOCADDR    1
380:    PUSHLOCADDR    1
381:    LOADFROMADDR
382:    PUSHNUM        1
383:    ADD
384:    SAVETOADDR
385:    JUMP           406
386:    WRITESTRING    122  143
387:    PUSHLOCADDR    1
388:    LOADFROMADDR
389:    PUSHNUM        1
390:    ADD
391:    WRITEINT
392:    WRITESTRING    144  145
393:    PUSHLOCADDR    -4
394:    LOADFROMADDR
395:    PUSHLOCADDR    1
396:    LOADFROMADDR
397:    ADDTOPTR
398:    READINT
399:    SAVETOADDR
400:    PUSHLOCADDR    1
401:    PUSHLOCADDR    1
402:    LOADFROMADDR
403:    PUSHNUM        1
404:    ADD
405:    SAVETOADDR
406:    JUMP           353
407:    RETURN         4


Data memory dump
  Data memory--addresses 0 to
  top of stack, and allocated
  heap locations:

  0: PTR TO 10001
  1: PTR TO 10019
  2: 100 = 'd'
  3: 69 = 'E'
  4: 110 = 'n'
  5: 116 = 't'
  6: 101 = 'e'
  7: 114 = 'r'
  8: 32 = ' '
  9: 110 = 'n'
 10: 117 = 'u'
 11: 109 = 'm'
 12: 98 = 'b'
 13: 101 = 'e'
 14: 114 = 'r'
 15: 32 = ' '
 16: 111 = 'o'
 17: 102 = 'f'
 18: 32 = ' '
 19: 114 = 'r'
 20: 111 = 'o'
 21: 119 = 'w'
 22: 115 = 's'
 23: 58 = ':'
 24: 32 = ' '
 25: 69 = 'E'
 26: 110 = 'n'
 27: 116 = 't'
 28: 101 = 'e'
 29: 114 = 'r'
 30: 32 = ' '
 31: 110 = 'n'
 32: 117 = 'u'
 33: 109 = 'm'
 34: 98 = 'b'
 35: 101 = 'e'
 36: 114 = 'r'
 37: 32 = ' '
 38: 111 = 'o'
 39: 102 = 'f'
 40: 32 = ' '
 41: 99 = 'c'
 42: 111 = 'o'
 43: 108 = 'l'
 44: 117 = 'u'
 45: 109 = 'm'
 46: 110 = 'n'
 47: 115 = 's'
 48: 58 = ':'
 49: 32 = ' '
 50: 71 = 'G'
 51: 105 = 'i'
 52: 118 = 'v'
 53: 101 = 'e'
 54: 110 = 'n'
 55: 32 = ' '
 56: 109 = 'm'
 57: 97 = 'a'
 58: 116 = 't'
 59: 114 = 'r'
 60: 105 = 'i'
 61: 120 = 'x'
 62: 58 = ':'
 63: 32 = ' '
 64: 84 = 'T'
 65: 114 = 'r'
 66: 97 = 'a'
 67: 110 = 'n'
 68: 115 = 's'
 69: 112 = 'p'
 70: 111 = 'o'
 71: 115 = 's'
 72: 101 = 'e'
 73: 100 = 'd'
 74: 32 = ' '
 75: 109 = 'm'
 76: 97 = 'a'
 77: 116 = 't'
 78: 114 = 'r'
 79: 105 = 'i'
 80: 120 = 'x'
 81: 58 = ':'
 82: 32 = ' '
 83: 32 = ' '
 84: 10 = Ctrl-J
 85: 10 = Ctrl-J
 86: 10 = Ctrl-J
 87: 84 = 'T'
 88: 121 = 'y'
 89: 112 = 'p'
 90: 101 = 'e'
 91: 32 = ' '
 92: 49 = '1'
 93: 32 = ' '
 94: 116 = 't'
 95: 111 = 'o'
 96: 32 = ' '
 97: 99 = 'c'
 98: 111 = 'o'
 99: 110 = 'n'
100: 116 = 't'
101: 105 = 'i'
102: 110 = 'n'
103: 117 = 'u'
104: 101 = 'e'
105: 44 = ','
106: 32 = ' '
107: 48 = '0'
108: 32 = ' '
109: 116 = 't'
110: 111 = 'o'
111: 32 = ' '
112: 113 = 'q'
113: 117 = 'u'
114: 105 = 'i'
115: 116 = 't'
116: 58 = ':'
117: 32 = ' '
118: 82 = 'R'
119: 111 = 'o'
120: 119 = 'w'
121: 32 = ' '
122: 69 = 'E'
123: 110 = 'n'
124: 116 = 't'
125: 101 = 'e'
126: 114 = 'r'
127: 32 = ' '
128: 118 = 'v'
129: 97 = 'a'
130: 108 = 'l'
131: 117 = 'u'
132: 101 = 'e'
133: 32 = ' '
134: 105 = 'i'
135: 110 = 'n'
136: 32 = ' '
137: 99 = 'c'
138: 111 = 'o'
139: 108 = 'l'
140: 117 = 'u'
141: 109 = 'm'
142: 110 = 'n'
143: 32 = ' '
144: 58 = ':'
145: 32 = ' '
146: 32 = ' '
147: PTR TO 20000
148: PTR TO 10007
149: PTR TO 10013
150: 1 = Ctrl-A
151: 0 = Ctrl-@
152: 4 = Ctrl-D
153: 0 = Ctrl-@
154: 0 = Ctrl-@
155: 3 = Ctrl-C
156: 4 = Ctrl-D
157: PTR TO 10019
158: 4 = Ctrl-D
159: 3 = Ctrl-C
160: 0 = Ctrl-@
161: 0 = Ctrl-@
162: 218
163: PTR TO 147
164: 0 = Ctrl-@
165: 3 = Ctrl-C
166: 0 = Ctrl-@
167: PTR TO 10040
168: 0 = Ctrl-@
169: PTR TO 10059
170: -1
171: PTR TO 10059
172: 10 = Ctrl-J
173: 6 = Ctrl-F
174: 492
175: PTR TO 163
176: 0 = Ctrl-@
177: PTR TO 10061
178: 0 = Ctrl-@
179: 0 = Ctrl-@
10000: PTR TO 10006
10001: PTR TO 10019
10002: 0 = Ctrl-@
10003: 0 = Ctrl-@
10004: 0 = Ctrl-@
10005: 0 = Ctrl-@
10006: PTR TO 10012
10007: 4 = Ctrl-D
10008: 0 = Ctrl-@
10009: 0 = Ctrl-@
10010: 0 = Ctrl-@
10011: 0 = Ctrl-@
10012: PTR TO 10018
10013: 3 = Ctrl-C
10014: 0 = Ctrl-@
10015: 0 = Ctrl-@
10016: 0 = Ctrl-@
10017: 0 = Ctrl-@
10018: PTR TO 10023
10019: PTR TO 10024
10020: PTR TO 10028
10021: PTR TO 10032
10022: PTR TO 10036
10023: PTR TO 10027
10024: 1 = Ctrl-A
10025: 2 = Ctrl-B
10026: 3 = Ctrl-C
10027: PTR TO 10031
10028: 4 = Ctrl-D
10029: 5 = Ctrl-E
10030: 6 = Ctrl-F
10031: PTR TO 10035
10032: 7 = Ctrl-G
10033: 8 = Ctrl-H
10034: 9 = Ctrl-I
10035: PTR TO 10039
10036: 0 = Ctrl-@
10037: 1 = Ctrl-A
10038: 2 = Ctrl-B
10039: PTR TO 10043
10040: PTR TO 10044
10041: PTR TO 10049
10042: PTR TO 10054
10043: PTR TO 10048
10044: 1 = Ctrl-A
10045: 0 = Ctrl-@
10046: 0 = Ctrl-@
10047: 0 = Ctrl-@
10048: PTR TO 10053
10049: 0 = Ctrl-@
10050: 0 = Ctrl-@
10051: 0 = Ctrl-@
```

```
10052: 0 = Ctrl-@
10053: PTR TO 10058
10054: 0 = Ctrl-@
10055: 0 = Ctrl-@
10056: 0 = Ctrl-@
10057: 0 = Ctrl-@
10058: PTR TO 10060
10059: 0 = Ctrl-@
10060: PTR TO 10062
10061: 0 = Ctrl-@
10062: PTR TO 10064
10063: 0 = Ctrl-@
10064: PTR TO 10066
10065: 0 = Ctrl-@
10066: PTR TO 10068
10067: 0 = Ctrl-@
10068: PTR TO 10070
10069: 0 = Ctrl-@
10070: PTR TO 10072
10071: 0 = Ctrl-@
10072: PTR TO 10074
10073: 0 = Ctrl-@
10074: PTR TO 10076
10075: 0 = Ctrl-@
10076: PTR TO 10078
10077: 0 = Ctrl-@
10078: PTR TO 10080
10079: 0 = Ctrl-@
10080: PTR TO 10082
10081: 0 = Ctrl-@
10082: PTR TO 10084
10083: 0 = Ctrl-@
10084: PTR TO 10086
10085: 0 = Ctrl-@
10086: PTR TO 10088
10087: 0 = Ctrl-@
10088: PTR TO 10090
10089: 0 = Ctrl-@
10090: PTR TO 10092
10091: 0 = Ctrl-@
10092: PTR TO 10094
10093: 0 = Ctrl-@
10094: PTR TO 10096
10095: 0 = Ctrl-@
10096: PTR TO 10098
10097: 0 = Ctrl-@
10098: PTR TO 10100
10099: 0 = Ctrl-@
10100: PTR TO 10102
10101: 0 = Ctrl-@
10102: PTR TO 10104
10103: 0 = Ctrl-@
10104: PTR TO 10106
10105: 0 = Ctrl-@
10106: PTR TO 10108
10107: 0 = Ctrl-@
10108: PTR TO 10110
10109: 0 = Ctrl-@
10110: PTR TO 10112
10111: 0 = Ctrl-@
10112: PTR TO 10114
10113: 0 = Ctrl-@
10114: PTR TO 10116
10115: 0 = Ctrl-@
10116: PTR TO 10118
10117: 0 = Ctrl-@
10118: PTR TO 10120
10119: 0 = Ctrl-@
10120: PTR TO 10122
10121: 0 = Ctrl-@
10122: PTR TO 10124
10123: 0 = Ctrl-@
10124: PTR TO 10126
10125: 0 = Ctrl-@
10126: PTR TO 10128
10127: 0 = Ctrl-@
10128: PTR TO 10130
10129: 0 = Ctrl-@
10130: PTR TO 10132
10131: 0 = Ctrl-@
10132: PTR TO 10134
10133: 0 = Ctrl-@
10134: PTR TO 10136
10135: 0 = Ctrl-@
10136: PTR TO 10138
10137: 0 = Ctrl-@
10138: PTR TO 10140
10139: 0 = Ctrl-@
10140: PTR TO 10142
10141: 0 = Ctrl-@
10142: PTR TO 10144
10143: 0 = Ctrl-@
10144: PTR TO 10146
10145: 0 = Ctrl-@
10146: PTR TO 10148
10147: 0 = Ctrl-@
10148: PTR TO 10150
10149: 0 = Ctrl-@
10150: PTR TO 10152
10151: 0 = Ctrl-@
10152: PTR TO 10154
10153: 0 = Ctrl-@
10154: PTR TO 10156
10155: 0 = Ctrl-@
10156: PTR TO 10158
10157: 0 = Ctrl-@
10158: PTR TO 10160
10159: 0 = Ctrl-@
10160: PTR TO 10162
10161: 0 = Ctrl-@
10162: PTR TO 10164
10163: 0 = Ctrl-@
10164: PTR TO 10166
10165: 0 = Ctrl-@
10166: PTR TO 10168
10167: 0 = Ctrl-@
10168: PTR TO 10170
10169: 0 = Ctrl-@
10170: PTR TO 10172
10171: 0 = Ctrl-@
10172: PTR TO 10174
10173: 0 = Ctrl-@
10174: PTR TO 10176
10175: 0 = Ctrl-@
10176: PTR TO 10178
10177: 0 = Ctrl-@
10178: PTR TO 10180
10179: 0 = Ctrl-@
10180: PTR TO 10182
10181: 0 = Ctrl-@
10182: PTR TO 10184
10183: 0 = Ctrl-@
10184: PTR TO 10186
10185: 0 = Ctrl-@
10186: PTR TO 10188
10187: 0 = Ctrl-@
10188: PTR TO 10190
10189: 0 = Ctrl-@
10190: PTR TO 10192
10191: 0 = Ctrl-@
10192: PTR TO 10194
10193: 0 = Ctrl-@
10194: PTR TO 10196
10195: 0 = Ctrl-@
10196: PTR TO 10198
10197: 0 = Ctrl-@
10198: PTR TO 10200
10199: 0 = Ctrl-@
10200: PTR TO 10202
10201: 0 = Ctrl-@
10202: PTR TO 10204
10203: 0 = Ctrl-@
10204: PTR TO 10206
10205: 0 = Ctrl-@
10206: PTR TO 10208
10207: 0 = Ctrl-@
10208: PTR TO 10210
10209: 0 = Ctrl-@
10210: PTR TO 10212
10211: 0 = Ctrl-@
10212: PTR TO 10214
10213: 0 = Ctrl-@
10214: PTR TO 10216
10215: 0 = Ctrl-@
10216: PTR TO 10218
10217: 0 = Ctrl-@
10218: PTR TO 10220
10219: 0 = Ctrl-@
10220: PTR TO 10222
10221: 0 = Ctrl-@
10222: PTR TO 10224
10223: 0 = Ctrl-@
10224: PTR TO 10226
10225: 0 = Ctrl-@
10226: PTR TO 10228
10227: 0 = Ctrl-@
10228: PTR TO 10230
10229: 0 = Ctrl-@
10230: PTR TO 10232
10231: 0 = Ctrl-@
10232: PTR TO 10234
10233: 0 = Ctrl-@
10234: PTR TO 10236
10235: 0 = Ctrl-@
10236: PTR TO 10238
10237: 0 = Ctrl-@
10238: PTR TO 10240
10239: 0 = Ctrl-@
10240: PTR TO 10242
10241: 0 = Ctrl-@
10242: PTR TO 10244
10243: 0 = Ctrl-@
10244: PTR TO 10246
10245: 0 = Ctrl-@
10246: PTR TO 10248
10247: 0 = Ctrl-@
10248: PTR TO 10250
10249: 0 = Ctrl-@
10250: PTR TO 10252
10251: 0 = Ctrl-@
10252: PTR TO 10254
10253: 0 = Ctrl-@
10254: PTR TO 10256
10255: 0 = Ctrl-@
10256: PTR TO 10258
10257: 0 = Ctrl-@
10258: PTR TO 10260
10259: 0 = Ctrl-@
10260: PTR TO 10262
10261: 0 = Ctrl-@

PC=383  ESP=3  FP= PTR TO 175  ASP= PTR TO 180
HP= PTR TO 10262  HMAX= PTR TO 15000

Total number of instructions executed: 1209788
Last instruction to be executed: 382: PUSHNUM 1

Expression evaluation stack:
EXPRSTACK[2]: 1
EXPRSTACK[1]: 0
EXPRSTACK[0]: PTR TO 176
```