Part I: Multiple Choice (30 points)

Answer all of the following questions. READ EACH QUESTION CAREFULLY. Fill the correct bubble on your mark-sense sheet. Each correct question is worth 2 points. Choose the one BEST answer for each question. Assume that all given C code is syntactically correct unless a possibility to the contrary is suggested in the question.

Remember not to devote too much time to any single question, and good luck!

1. How many `*`'s will the following function print, expressed as a rough or approximate function of `n`?

```
void pr(int n) {
   int i, j;

   for (i = 0; i < n; i++)
      for (j = n; j > 0; j = j/2)
         printf("*");
}
```

A. log **n**
B. **n**
C. **n** * log **n**
D. $\mathbf{n}^2$
E. 2^n

2. The string library functions `strcpy` and `strcat` both copy a source string into a portion of a destination array. How are they different?

A. Function **`strcpy`** can write past the bounds of the destination array, while function **`strcat`** cannot.
B. Function **`strcat`** can write past the bounds of the destination array, while function **`strcpy`** cannot.
C. Function **`strcat`** copies the source string into the destination array starting at position 0, while **`strcpy`** copies the source string into the destination array starting at the string terminator character of the first string in the destination array.
D. Function **`strcpy`** copies the source string into the destination array starting at position 0, while **`strcat`** copies the source string into the destination array starting at the string terminator character of the first string in the destination array.
E. Function **`strcpy`** leaves the source string as is, while function **`strcat`** overwrites it.

3. Suppose array X is declared as

```
int X[10];
```

and we want to pass the whole array X to the function print_array defined by

```
void print_array(int ar[], int length) {
  int i;
  for (i = 0; i < length; i = i + 1)
    printf("%d ", ar[i]);
}
```

but we want print_array to print ONLY the first 5 elements.

Which of the following is the correct call?

A. `print_array(X[10], 5);`
B. `print_array(&X, 5);`
C. `print_array(X, 5);`
D. `print_array(X[5], 5);`
E. `print_array(&X, 10);`

4. What happens when a file is opened for reading?

A. The file is copied into an array in memory.
B. The file is copied into a struct in memory.
C. An internal file variable is associated with the external file.
D. An internal array is associated with the external file.
E. The I/O system checks for the **`EOF`** condition.

5. Consider the following fragment of code:

```
nc = 0;
while (fscanf(mydata, "%c", &c) != EOF) {
   nc = nc + 1;
}
```

If `c` is of type `char`, `nc` is of type `int`, and the file specified in the call to `fscanf` has already been opened for reading, what does this code fragment accomplish?

A. It counts the number of characters in the file.
B. It counts the number of **`EOF`** marks in the file.
C. It counts the number of records in the file.
D. It finds the numeric equivalent of the first character in the file.
E. Since the Boolean expression in the while statement is always true, it loops forever.

6. Given the factorial function shown below, suppose that `factorial(25)` has been executing for some time and it finally reaches the base case where t is assigned the value `1`.

```
int factorial (int n) {
  int t;
  if (n <= 1)
    t = 1;
  else
    t = n * factorial(n-1);
  return (t);
}
```

Which of the following statements are true?

A. There are 25 separate instances of **`factorial`** on the stack, each with its own storage for **`n`** and for **`t`**.
B. There are 26 separate instances of **`factorial`** on the stack, each with its own storage for **`n`** and for **`t`**.
C. There are 25 values of **`n`** on the stack, but only one value of **`t`**.
D. There are 25 values of **`t`** on the stack, but only one value of **`n`**.
E. None of the above can be concluded without knowing what function called **`factorial`**.

7. Consider the `is_path` function from lecture as given below:

```
int is_path(char m[MAXX][MAXY], int x, int y)
{
  if (m[x][y] == 'F') {
    return (TRUE);
  } else {
    m[x][y] = 'X';
    return ((legal_mv(m,x+1,y) && is_path(m,x+1,y)) ||
            (legal_mv(m,x-1,y) && is_path(m,x-1,y)) ||
            (legal_mv(m,x,y-1) && is_path(m,x,y-1)) ||
            (legal_mv(m,x,y+1) && is_path(m,x,y+1)));
  }
}
```

Given that there are four recursive calls in `is_path`, what keeps `is_path` from having exponential complexity, i.e. being really, really slow?

(Recall that `legal_mv(m,x,y)` returns true if <`x`, `y`> is a legal move in the maze `m`; otherwise it returns false.)

A. Any instance of **`is_path`** only makes one of the four recursive calls.
B. No instance of **`is_path`** moves both horizontally and vertically.
C. **`is_path`** knows where the finish square is and always heads in that direction.
D. **`is_path`** always moves up before moving down.
E. **`is_path`** marks the squares it has been to and only visits each one once.

8. How many steps would it take for the binary search algorithm presented in lecture to fail to find the value 80 in the following array? (Count one step each time it has to compare an array element value to 80.)

2 8 15 30 40 90 100

A. 1
B. 2
C. 3
D. 4
E. 7

9. Consider the following statements about selection sort operating on an array of n elements.

1. After there are 3 elements in the sorted part, it must search through n-3 elements in the unsorted part.

2. Each time it finds the minimum value in the unsorted part, it must search for the correct place to put it in the sorted part.

3. On any given iteration, it may move up to n-1 elements in the array.

4. The time it takes to execute is proportional to n^2.

Which statements are true?

A. Only 1.
B. Only 2
C. 1 and 3
D. 1 and 4
E. Only 4

10. Given the following program, what will be the final contents of the array `strbuffer`?

```
#include <string.h>
int main(void)
{
  char strbuffer[8] = "bbbbbbb";
  char str1[] = "odegaard";

  str1[3] = '\0';
  strcpy(strbuffer, str1);

  return (0);
}
```

A. `'o' 'd' 'e' 'g'  'a' 'a' 'r' 'd'`
B. `'o' 'd' 'e' '\0' 'b' 'b' 'b' '\0'`
C. `'o' 'd' 'e' '\0' 'a' 'a' 'r' 'd'`
D. `'o' 'd' 'e' 'b'  'b' 'b' 'b' '\0'`
E. `'o' 'd' 'e' 'b'  'b' 'b' 'b' 'b'`

11. Consider the following structure and function definitions:

```
typedef struct {
  double real, imag;
} complex;

void f(complex *cp) {
  cp->real = 0.0;
}
```

Which of the following choices describes all the statements that are equivalent to the assignment `cp->real = 0.0;`?

A. `*cp.real = 0.0;`
B. `(*cp).real = 0.0;`
C. `*(cp.real) = 0.0;`
D. both (a) and (b)
E. both (a) and (c)

12. What output is produced by the following C program?

```
#include <stdio.h>

typedef struct {
   int x, y;
} pair;

void fn(int a[ ], pair p) {
   a[0] = p.x;
   p.y = a[1];
}

int main(void) {
   int ay[2] = {1, 2};
   pair pr = {3, 4};
   fn(ay, pr);
   printf("%d %d %d %d\n", ay[0], ay[1], pr.x, pr.y);
   return(0);
}
```

A. `1 2 3 4`
B. `1 2 3 2`
C. `3 2 3 4`
D. `3 2 3 2`
E. None of the above

13. What output is produced by the following C program?

```
#include <stdio.h>

int r(int k) {
   if (k > 10)
      return (k-1);
   else
      return (k * r(k+4));
}

int main(void) {
   int a;
   a = r(3);
   printf("%d\n", a);
}
```

A. 21
B. 70
C. 77
D. 210
E. 231

14. The following code that uses the GP142 package was found in a recycle bin. What does it do? (You should assume that all variables have suitable declarations, etc. The question deals with the GP142 event model, not trivial programming details.)

```
...
n = 0;
quit = FALSE;
while (!quit) {
   event = GP142_await_event(&mouse_x, &mouse_y, &key_pressed);
   switch(event) {
   case GP142_QUIT:
      quit = TRUE;
      break;

   case GP142_MOUSE:
      if (n >= 10) {
         zap_bugs();
         n = 0;
      }
      break;

   case GP142_KBD:
      break;

   case GP142_periodic:
      n++;
      break;

   default:
      break;
   }
}
```

A. Nothing happens until the user selects quit from the menu, then the program stops.

B. Function **`zap_bugs()`** is called after every 10 timer events. The program stops when quit is selected from the menu.

C. Function **`zap_bugs()`** is called every time the mouse button is pressed. The program stops when quit is selected from the menu.

D. Function **`zap_bugs()`** is called the first time the mouse button is pressed. The function **`zap_bugs()`** is then called again after every 10 timer events. The program stops when quit is selected from the menu.

E. Function **`zap_bugs()`** is called when the mouse button is pressed, provided there have been at least 10 timer events since the program started or since the last time the button was pressed. The program stops when quit is selected from the menu.

15. Which of the following will be the certain result of failing to fill in properly your name, student ID, section number, and exam version on your Scantron answer sheet?

A. A score of 0 will be recorded for the multiple choice portion of the final exam, regardless of how many questions you answer correctly.
B. Your grade in the course will be lower than it might otherwise be since a 0 will be recorded for the multiple choice portion of the final exam.
C. The grade you get for the multiple choice portion will rhyme well with the name of the Roman emperor Nero. (Hint: Starts with a Z.)
D. You will need to do exceptionally well on the programming portion of this exam to help offset the 0 that you will earn for the multiple choice portion.
E. all of the above

Part II: Programming Questions (29 points)

16. (11 points total) Given the following type definition to represent the coordinates of a 3D point (x,y,z):

```
typedef struct {
  double x, y, z;
} point3d;
```

A. (5 points) Write a function

```
point3d add3d(point3d p1, point3d p2)
```

that takes two points (`p1` and `p2`) as its input parameters and returns a third point that is the vector sum of `p1` and p2 as its return value. Note that in mathematics, the vector sum of (x1, y1, z1) and (x2, y2, z2) is (x1+x2, y1+y2, z1+z2).

(This question continues on the next page.)

B. (6 points) Use your function `add3d` to write another function:

```
void translate(point3d old[ ], point3d new[ ],
               int npts, point3d trans)
```

This function is given arrays `old` and `new` that each hold `npts` `point3d`s (i.e. they are both arrays of structs) and a single `point3d` called `trans`. It should take each `point3d` in array `old`, add the `point3d` `trans` to it, and put the result in the corresponding element of array `new`. For example, if the first element of `old` contains the point (6, 3, -9) and `trans` has coordinates (1, -1, -3), then the first element of `new` should become (7, 2, -12). The same point `trans` is added to each element of `old` to produce the elements of `new`. (In computer graphics and computer vision, we call this the translation of a point set; it allows us to move an object in 3D space.)

17. (18 points total) Many engineering and scientific applications represent data as a 2-dimensional grid of values, say temperatures at points on a flat plate or brightness of pixels (dots) in a video image. A common algorithm on these grids is to smooth the data by updating the value of each point to be the average of its old value and the points immediately surrounding it. For example, suppose we had the following 3x3 grid of values for a point and its immediate neighbors:

5.0 5.0 9.0
5.0 6.0 9.0
7.0 8.0 9.0

The smoothing algorithm would update the point in the middle (old value 6.0) to have a new value of 7.0, which is the average of all 9 numbers in this 3x3 part of the grid. If the point were on the edge of the grid, the average would include only that point and its immediate neighbors that actually exist (i.e. are on the grid).

For this problem, complete the definitions of functions `avg` (in Part A) and `smooth` (in Part B) such that they replace each number in the grid with the average of that number and its immediate neighbors. Notice that, as in Homework 5, you might not be able to change an element of the grid as soon as you've calculated its new value, because the old value might still be needed to compute the new average values for its neighbors.

Hint: Be careful not to reference values off the edge of the grid. It might be useful to define another function that, given a row and column number, returns true or false depending on whether those numbers specify a a location inside or outside the grid.

Hint: Keep it simple. Take a minute to think about the problem and sketch a solution before you write the detailed code.

You may define additional functions at the end of the problem if you want to.

(Parts A and B are on the following pages. Do not write your answers on this page.)

```
/* number of rows and columns in the grid */
/* (Use these names, not the actual numbers.) */
#define NROWS  42
#define NCOLS  17
```

A. (10 points) Complete the following function so it yields the average of the 3x3 grid of array elements centered at row `r`, column `c`. If `grid[r][c]` is on the edge of the grid, the average should include only those array elements that actually exist.

```
/* = average of 3x3 grid centered at grid[r][c] */
double avg(double grid[NROWS][NCOLS], int r, int c)
```

B. (8 points) Complete the definition of the following function so it updates every grid element with the average of that element and its immediate neighbors. Use function `avg` from Part A to calculate the new value for each element.

```
/* smooth the values in grid g by replacing each element */
/* with the average of its old value and the values of its */
/* immediate neighbors */
void smooth(double grid[NROWS][NCOLS])
```