1. What is printed on each line?

```c
void printMatrix(int **m, int nrows, int ncols)
{
    int i, j;
    for (i = 0; i < ncols; ++i)
    {
        for (j = 0; j < nrows; ++j)
            printf("%d ", m[j][i]);
        printf("\n");
    }
}
```

(a) nothing in the matrix is printed
(b) the rows
(c) the elements
(d) the columns

2. What is printed on each line?

```c
void printMatrix(int **m, int nrows, int ncols)
{
    int i, j;
    for (i = 0; i < nrows; ++i)
    {
        for (j = 0; j < ncols; ++j)
            printf("%d ", m[i][j]);
        printf("\n");
    }
}
```

(a) the columns
(b) the rows
(c) the elements
(d) nothing in the matrix is printed

3. What is printed on each line?

```c
void printMatrix(int **m, int nrows, int ncols)
{
    int i, j;
    for (i = 0; i < nrows; ++i)
    {
        for (j = 0; j < ncols; ++j)
        {
            printf("%d ", m[i][j]);
            printf("\n");
        }
    }
}
```

(a) neither the rows or the columns
(b) the elements
(c) the rows
(d) the columns

Concept: recognizing patterns: counting and accumulating
4. Which pattern does the following function implement?

```c
int g(int **m, int nrows, int ncols)
{
    int i, j, t = 1;
    for (j = 0; j < nrows; ++j)
        for (i = 0; i < ncols; ++i)
            t = t * m[j][i];
    return t;
}
```

(a) the **filtered-counting** pattern  
(b) the **accumulate** pattern  
(c) the **counting** pattern  
(d) the **filtered-accumulate** pattern

5. Which pattern does the following function implement?

```c
int f(int **m, int nrows, int ncols) // m is a matrix
{
    int i, j, s = 0;
    for (i = 0; i < nrows; ++i)
        for (j = 0; j < ncols; ++j)
            if (isEven(m[i][j]))
                s = s + m[i][j];
    return s;
}
```

(a) the **filtered-counting** pattern  
(b) the **counting** pattern  
(c) the **accumulate** pattern  
(d) the **filtered-accumulate** pattern

6. Which pattern does the following function implement?

```c
int g(int **m, int nrows, int ncols) // m is a matrix
{
    int i, j, s = 0;
    for (i = 0; i < nrows; ++i)
        for (j = 0; j < ncols; ++j)
            s = s + 1;
    return s;
}
```

(a) the **filtered-accumulate** pattern  
(b) the **filtered-counting** pattern  
(c) the **accumulate** pattern  
(d) the **counting** pattern

7. Which pattern does the following function implement?

```c
int g(int **m, int nrows, int ncols) // m is a matrix
{
    int i, j, t = 0;
    for (j = 0; j < nrows; ++j)
        for (i = 0; i < ncols; ++i)
            if (X(m[j][i]))
                t = t + 1
    return t;
}
```

(a) the **filtered-counting** pattern  
(b) the **counting** pattern  
(c) the **filtered-accumulate** pattern  
(d) the **accumulate** pattern
8. Which pattern does the following function implement?

```c
int h(int **m, int nrows, int ncols) //m is a matrix
{
    int i, j;
    int **v = createMatrix(nrows, ncols);
    for (j = 0; j < nrows; ++j)
        for (i = 0; i < ncols; ++i)
            v[j][i] = g(m[j][i])
    return v;
}
```

(a) the extreme pattern
(b) the search pattern
(c) the filter pattern
(d) the map pattern
(e) the extreme index pattern

9. Which pattern does the following function implement?

```c
int f(int h, int **m, int nrows, int ncols)
{
    int i, j;
    for (j = 0; j < nrows; ++j)
        for (i = 0; i < ncols; ++i)
            if (m[j][i] == h)
                return 1;
    return 0;
}
```

(a) the extreme pattern
(b) the map pattern
(c) the extreme index pattern
(d) the filter pattern
(e) the search pattern

10. Which pattern does the following function implement?

```c
int f(int h, int **m, int nrows, int ncols)
{
    int i, j, w = 0;
    for (j = 0; j < nrows; ++j)
        for (i = 0; i < ncols; ++i)
            if (m[j][i] == h)
                w = 1;
    return w;
}
```

(a) the search pattern
(b) the filter pattern
(c) the extreme pattern
(d) the extreme index pattern
(e) the map pattern

11. Which pattern does the following function implement?

```c
int * h(int (*isG)(int), int **m, int nrows, int ncols, int *s)
{
    int i, j, count = 0;
    int *v = malloc(sizeof(int)*nrows*ncols);
    //malloc failure check omitted
}
for (i = 0; i < nrows; ++i)
    for (j = 0; j < ncols; ++j)
        if (isG(m[i][j])
            v[count++] = m[i][j];
    v = realloc(sizeof(int)*count);
    *s = count;
    return v;
}

(a) the extreme index pattern
(b) the extreme pattern
(c) the map pattern
(d) the filter pattern
(e) the search pattern

12. Which pattern does the following function implement?

int h(int **m,int nrows,int ncols) //m is a matrix
{
    int i,j,w = m[0][0];
    for (i = 0; i < nrows; ++i)
        for (j = 0; j < ncols; ++j)
            if (m[i][j] < w)
                w = m[i][j];
    return w;
}

(a) the filter pattern
(b) the map pattern
(c) the extreme pattern
(d) the extreme index pattern
(e) the search pattern

13. Which pattern does the following function implement?

int g(int **m,int nrows,int ncols)
{
    int i,j,v = 0,y = 0;
    for (j = 0; j < nrows; ++j)
        for (i = 0; i < ncols; ++i)
            if (m[j][i] > m[v][y])
                {v = j;
                 y = i;
                }
    return m[v][y];
}

(a) the filter pattern
(b) the extreme pattern
(c) the extreme index pattern
(d) the search pattern
(e) the map pattern

14. Which pattern does the following function implement?

int *f(int **m,int nrows,int ncols) //m is a matrix
{
    int i,j;
    int u = 0,x = 0;
    int *a = malloc(sizeof(int)*2);  
    for (j = 0; j < nrows; ++j)
        for (i = 0; i < ncols; ++i)
            if (m[j][i] > m[u][x])
{
    u = j;
    x = i;
}

a[0] = u;
a[1] = x;
return a;
}

(a) the *filter* pattern
(b) the *extreme index* pattern
(c) the *map* pattern
(d) the *extreme* pattern
(e) the *search* pattern

15. Consider a function that takes a matrix, the number of rows, and the number of columns as arguments and sums all of the elements in the matrix. This function implements the:

(a) the *extreme* pattern
(b) the *search* pattern
(c) the *filtered-counting* pattern
(d) the *map* pattern
(e) the *extreme index* pattern
(f) the *filter* pattern
(g) the *filtered-accumulate* pattern
(h) the *accumulate* pattern

16. Consider a function that takes a matrix, the number of rows, and the number of columns as arguments and returns the largest number in the matrix. This function implements the:

(a) the *map* pattern
(b) the *extreme index* pattern
(c) the *filtered-accumulate* pattern
(d) the *filtered-counting* pattern
(e) the *extreme* pattern
(f) the *filter* pattern
(g) the *search* pattern
(h) the *accumulate* pattern

17. Consider a function that takes a matrix, the number of rows, and the number of columns as arguments and returns the row index of the largest number in the matrix. This function implements:

(a) the *search* pattern
(b) the *filtered-counting* pattern
(c) the *map* pattern
(d) the *accumulate* pattern
(e) the *extreme* pattern
(f) the *extreme index* pattern
(g) the *filter* pattern
(h) the *filtered-accumulate* pattern

18. Consider a function that takes a matrix, the number of rows, the number of columns, and a threshold as arguments and, searching the matrix in row order, returns the first number in the matrix that is less than the threshold and NULL otherwise. This function implements a form of:

(a) the *filtered-counting* pattern
(b) the *map* pattern
(c) the *extreme* pattern
(d) the *extreme index* pattern
19. Consider a function that takes a matrix, the number of rows, the number of columns, and a threshold as arguments and, searching the matrix in column order, returns the row and column indices of the first number in the matrix that is greater than the threshold and [-1,-1] otherwise. This function implements a form of:

(a) the accumulate pattern
(b) the filtered-accumulate pattern
(c) the extreme index pattern
(d) the extreme pattern
(e) the search pattern
(f) the map pattern
(g) the filter pattern
(h) the filtered-accumulate pattern

20. Consider a function that takes a matrix, the number of rows, and the number of columns as arguments and, searching the matrix in row order, returns the row and column indices of the first even number in the matrix and [-1,-1] otherwise. This function implements a pattern that would best be termed:

(a) the search-accumulate pattern
(b) the search-value pattern
(c) the search-index pattern
(d) the search-counting pattern

21. Consider a function that takes a matrix, the number of rows, the number of columns, and an item as arguments and returns true if the item is not in the matrix and false otherwise. This function implements the:

(a) the map pattern
(b) the filter pattern
(c) the search pattern
(d) the filtered-accumulate pattern
(e) the filtered-counting pattern
(f) the accumulate pattern
(g) the extreme pattern
(h) the extreme index pattern

Concept: verifying code

22. Consider the problem statement: sum the numbers in rows a to b (inclusive), with −1 < a ≤ b < nrows. Does this function compute the correct result?

```c
int sum(int **m, int nrows, int ncols, int a, int b)
{
    int i,j,total = 0;
    for (j = a; j < b; ++j)
        for (i = 0; i < ncols; ++i)
            total = total + m[j][i];
    return total;
}
```

(a) No, rows are right, columns are wrong
(b) No, rows are wrong, columns are right
(c) Yes
(d) No, rows are wrong, columns are wrong

23. Consider the problem statement: sum the numbers in rows a to b (inclusive), with −1 < a ≤ b < nrows. Does this function compute the correct result?
int sum(int **m, int nrows, int ncols, int a, int b)
{
    int i, j, total = 0;
    for (j = 0; j < nrows; ++j)
        for (i = a; i < b+1; ++i)
            total = total + m[j][i];
    return total;
}

(a) Yes
(b) No, rows are wrong, columns are wrong
(c) No, rows are right, columns are wrong
(d) No, rows are wrong, columns are right

24. Consider the problem statement: sum the numbers in rows a to b (inclusive), with $-1 < a \leq b < nrows$. Does this function compute the correct result?

int sum(int **m, int nrows, int ncols, int a, int b)
{
    int i, j, total = 0;
    for (i = a+1; i < b+1; ++i)
        for (j = 0; j < ncols; ++j)
            total = total + m[i][j];
    return total;
}

(a) No, rows are wrong, columns are wrong
(b) No, rows are wrong, columns are right
(c) No, rows are right, columns are wrong
(d) Yes

25. Consider the problem statement: sum the numbers in rows a to b (inclusive), with $-1 < a \leq b < nrows$. Does this function compute the correct result?

int sum(int **m, int nrows, int ncols, int a, int b)
{
    int i, j, total = 0;
    for (j = a; j < b+1; ++j)
        for (i = 0; i < ncols-1; ++i)
            total = total + m[j][i];
    return total;
}

(a) No, rows are wrong, columns are wrong
(b) No, rows are wrong, columns are right
(c) No, rows are right, columns are wrong
(d) Yes

26. Consider the problem statement: sum the numbers in rows a to b (inclusive), with $-1 < a \leq b < nrows$. Does this function compute the correct result?

int sum(int **m, int nrows, int ncols, int a, int b)
{
    int i, j, total = 0;
    for (j = a; j < b+1; ++j)
        for (i = 1; i < ncols; ++i)
            total = total + m[j][i];
    return total;
}

(a) No, rows are wrong, columns are wrong
(b) No, rows are wrong, columns are right
(c) No, rows are right, columns are wrong
(d) Yes

7
27. Consider the problem statement: **sum the numbers in rows a to b (inclusive)**, with \(-1 < a \leq b < \text{nrows}\). Does this function compute the correct result?

```c
int sum(int **m, int nrows, int ncols, int a, int b)
{
    int i, j, total = 0;
    for (j = 1; j < ncols; ++j)
        for (i = a; i < b+1; ++i)
            total = total + m[i][j];
    return total;
}
```

(a) Yes  
(b) No, rows are wrong, columns are right  
(c) No, rows are wrong, columns are wrong  
(d) No, rows are right, columns are wrong

28. Define a function \(f\) that takes a matrix, the number of rows, and the number of columns as arguments and sums all of the elements in the matrix.

29. Define a function \(g\) that takes a matrix, the number of rows, and the number of columns as arguments and returns the smallest number in the matrix.

30. Define a function \(h\) that takes a matrix, the number of rows, and the number of columns as arguments and returns the row index of the smallest number in the matrix.

31. Define a function \(p\) that takes a matrix, the number of rows, and the number of columns as arguments and returns the largest number in the matrix.

32. Define a function \(q\) that takes a matrix, the number of rows, and the number of columns as arguments and returns the column index of largest number in the matrix.

33. Define a function \(r\) that takes a matrix, the number of rows, the number of columns, and a threshold as arguments and, searching the matrix in column order, returns the first number in the matrix that is greater than the threshold and NULL otherwise.
34. Define a function \( s \) that takes a matrix, the number of rows, the number of columns, and a threshold as arguments and returns the row and column indices of the first number in the matrix that is greater than the threshold and \([-1,-1]\) otherwise.

35. Define a function \( t \) that takes a matrix, the number of rows, the number of columns, and a threshold as arguments and returns the row and column indices of the first number in the matrix that is greater than the threshold and \([-1,-1]\) otherwise.

36. Define a function \( a \) that takes a matrix, the number of rows, the number of columns, and a threshold as arguments and returns the first number in the matrix that is less than the threshold and - otherwise.

37. Define a function \( v \) that takes a matrix, the number of rows, and the number of columns as arguments and returns the first even number in the matrix and -1 otherwise.

38. Define a function \( w \) that takes a matrix, the number of rows, and the number of columns as arguments and returns the row and column indices of the first even number in the matrix and \([-1,-1]\) otherwise.

39. Define a function \( x \) that takes a matrix, the number of rows, the number of columns, and an item as arguments and returns true if the item is in the matrix and false otherwise.

40. Define a function \( y \) that takes a matrix, the number of rows, the number of columns, and an item as arguments and returns true if the item is not in the matrix and false otherwise.

41. Define a function \( z \) that takes a matrix and an item as arguments and returns the row and column indices of item if it is in the matrix and \([-1,-1]\) otherwise.