Arrays – Q1

Write a C function printReverse() that prints an array of integers in reverse order. For example, if ar[5] = {1,2,3,4,5}, then the output 5, 4, 3, 2, 1 will be printed after applying the function printReverse(). The function prototype is given as follows:

void printReverse(int ar[ ], int size);

where *size* indicates the size of the array.

Write two versions of printReverse().

(1) One version (printReverse1) uses index notation
(2) the other version (printReverse2) uses pointer notation for accessing the element of each index location.

In addition, write another C function **reverseAr()** that takes in an array of integers **ar** and a parameter **size** that indicates the size of the array to be processed. The function converts the content in the array in reverse order and passes the array to the calling function via call by reference.

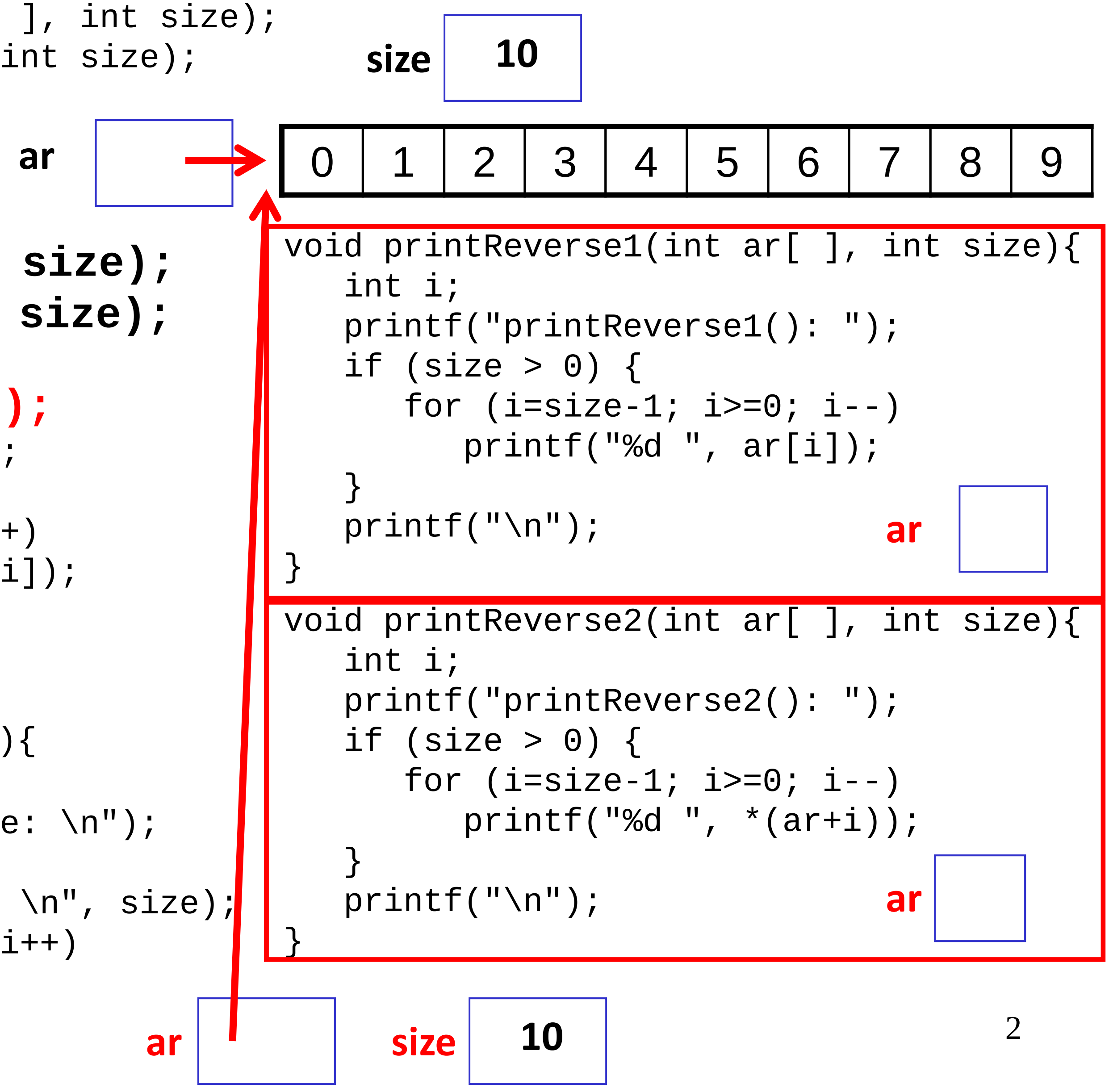
void reverseAr(int ar[ ], int size);

Write a C program to test the functions.

```
Enter array size:
5
Enter 5 array:
1 2 3 6 7
printReverse1(): 7 6 3 2 1
printReverse2(): 7 6 3 2 1
reverseAr(): 7 6 3 2 1
```

```c
#include <stdio.h>
int readArray(int ar[ ]);
void printReverse1(int ar[ ], int size);
void printReverse2(int ar[ ], int size);
void reverseAr(int ar[ ], int size);
int main(){
   int ar[10];
   int size, i;
   size = readArray(ar);
    printReverse1(ar, size);
    printReverse2(ar, size);

   reverseAr(ar, size);
   printf("reverseAr(): ");
   if (size > 0) {
      for (i=0; i<size; i++)
         printf("%d ", ar[i]);
   }
   return 0;
}
int readArray(int ar[ ]){
   int i, size;
   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d array: \n", size);
   for (i=0; i <= size-1; i++)
      scanf("%d", &ar[i]);
   return size;
}
```

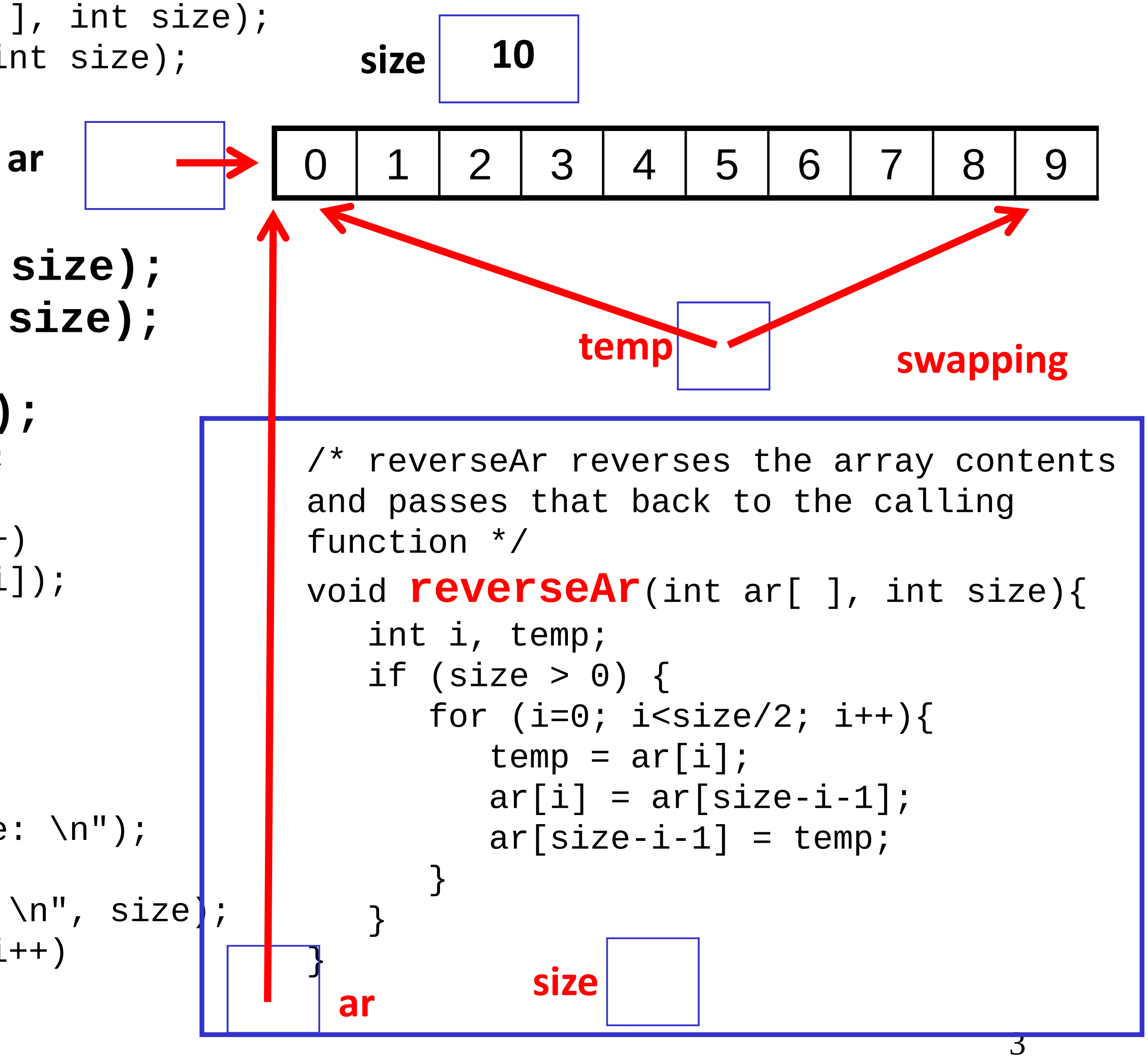


```c
void printReverse1(int ar[ ], int size){
   int i;
   printf("printReverse1(): ");
   if (size > 0) {
      for (i=size-1; i>=0; i--)
         printf("%d ", ar[i]);
   }
   printf("\n");
}
```

```c
void printReverse2(int ar[ ], int size){
   int i;
   printf("printReverse2(): ");
   if (size > 0) {
      for (i=size-1; i>=0; i--)
         printf("%d ", *(ar+i));
   }
   printf("\n");
}
```

```
#include <stdio.h>
int readArray(int ar[ ]);
void printReverse1(int ar[ ], int size);
void printReverse2(int ar[ ], int size);
void reverseAr(int ar[ ], int size);
int main(){
   int ar[10];
   int size, i;
   size = readArray(ar);
    printReverse1(ar, size);
    printReverse2(ar, size);

   reverseAr(ar, size);
   printf("reverseAr(): ");
   if (size > 0) {
      for (i=0; i<size; i++)
         printf("%d ", ar[i]);
   }
   return 0;
}
int readArray(int ar[ ]){
   int i, size;
   printf("Enter array size: \n");
   scanf("%d", &size);
   printf("Enter %d array: \n", size);
   for (i=0; i <= size-1; i++)
      scanf("%d", &ar[i]);
   return size;
}
```



```
/* reverseAr reverses the array contents
and passes that back to the calling
function */
void reverseAr(int ar[ ], int size){
   int i, temp;
   if (size > 0) {
      for (i=0; i<size/2; i++){
         temp = ar[i];
         ar[i] = ar[size-i-1];
         ar[size-i-1] = temp;
      }
   }
}
```

Arrays – Q2

Write the code for the following functions:

void swap2Rows(int M[SIZE][SIZE], int r1, int r2);

/* the function swaps the row r1 with the row r2 */

void swap2Cols(int M[SIZE][SIZE], int c1, int c2);

/* the function swaps the column c1 with the column c2 */

Write a C program to test the above functions. In addition, your program should print the resultant matrix after each operation. You may assume that the input matrix is a 3x3 matrix when testing the functions.

```
Enter the matrix row by row: (i.e. row
0, 1, 2)
5 10 15
15 20 25
25 30 35

Enter two rows for swapping:
1 2
The new array is:
   5   10   15
  25   30   35
  15   20   25

Enter two columns for swapping:
1 2
The new array is:
   5   15   10
  25   35   30
  15   25   20
```

```c
#include <stdio.h>
#define SIZE 3
void swap2Rows(int ar[SIZE][SIZE], int r1, int r2);
void swap2Cols(int ar[SIZE][SIZE], int c1, int c2);
void display(int ar[SIZE][SIZE]);
int main(){
   int array[SIZE][SIZE];
   int row1, row2, col1, col2;
   int i,j;

   printf("Enter the matrix (3x3) row by row: \n");
   for (i=0; i<SIZE; i++)
      for (j=0; j<SIZE; j++)
         scanf("%d", &array[i][j]);

   printf("Enter two rows for swapping: \n");
   scanf("%d %d", &row1, &row2);
   swap2Rows(array, row1, row2);
   printf("The new array is: \n");
   display(array);

   printf("Enter two columns for swapping: \n");
   scanf("%d %d", &col1, &col2);
   swap2Cols(array, col1, col2);
   printf("The new array is: \n");
   display(array);
   return 0;
}
```

```
Enter the matrix (3x3) row by row:
5 10 15
15 20 25
25 30 35

Enter two rows for swapping:
1 2
The new array is:
   5   10   15
  25   30   35
  15   20   25
Enter two columns for swapping:
1 2
The new array is:
   5   15   10
  25   35   30
  15   25   20
```

```c
void  display(int  M[SIZE][SIZE])
{
   int l,m;
   for (l = 0; l < 3; l++) {
      for (m = 0; m < 3; m++)
         printf("%d ", M[l][m]);
      printf("\n");
   }
}
```

Note: For the specified rows or columns, just perform a swapping operation for each element of the rows and columns

```c
void swap2Rows(int M[SIZE][SIZE], int r1, int r2)
/* swaps row M[r1] with row M[r2] */
{
   int temp;
   int n;

   for(n = 0; n < SIZE; n++) {
      temp = M[r1][n] ;
      M[r1][n] = M[r2][n];
      M[r2][n] = temp;
   }
}
```

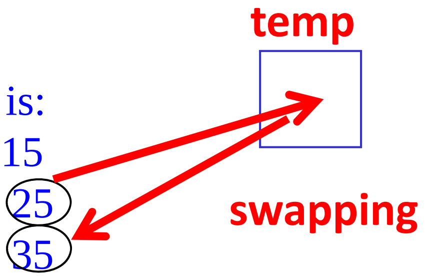
```c
void swap2Cols(int M[SIZE][SIZE], int c1, int c2)
/* swaps column  M[][c1] with  column  M[][c2] */
{
   int temp;
   int n;

   for(n = 0; n < SIZE; n++) {
      temp = M[n][c1] ;
      M[n][c1] = M[n][c2];
      M[n][c2] = temp;
   }
}
```

```
Enter the matrix (3x3) row by row:
5 10 15
15 20 25
25 30 35
The array is:
  5  10  15
 15  20  25
 25  30  35
Enter two rows for swapping:

1 2
The new array is:
  5  10  15
 25  30  35
 15  20  25
Enter two columns for swapping:

1 2
The new array is:
  5  15  10
 25  35  30
 15  25  20
```



Arrays – Q3

A square matrix (2-dimensional array of equal dimensions) can be reduced to upper-triangular form by setting each diagonal element to the sum of the original elements in that column and setting to 0s all the elements below the diagonal. For example, the 4-by-4 matrix:

4 3 8 6
9 0 6 5
5 1 2 4
9 8 3 7

would thus be reduced to

27 3 8 6
0 9 6 5
0 0 5 4
0 0 0 7

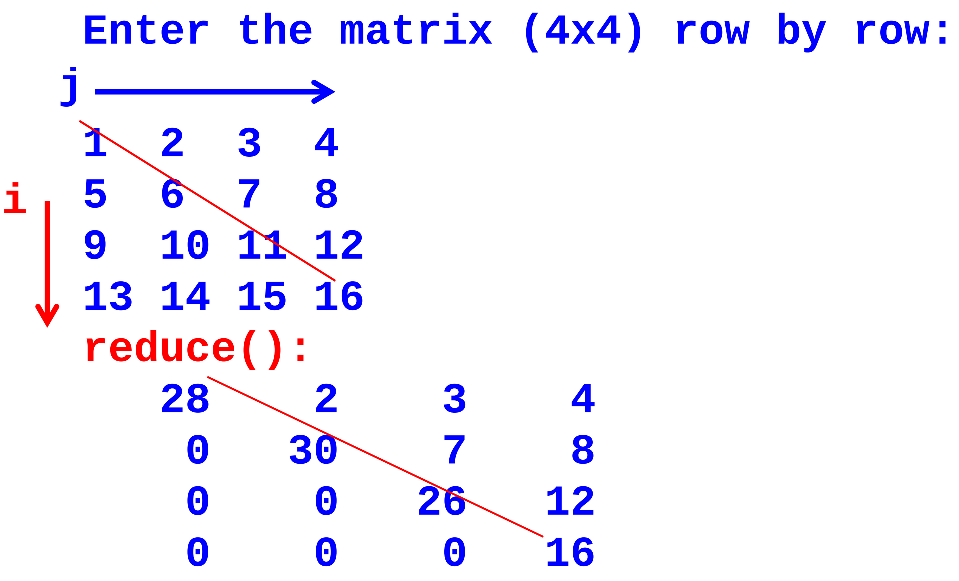
```
Enter the matrix (4x4):
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
reduceMatrix():
   28    2    3    4
    0   30    7    8
    0    0   26   12
    0    0    0   16
```

Write a function to reduce a 4-by-4 matrix. The prototype of the function is:

```
void reduceMatrix(int matrix[4][4]);
```

Write a C program to test the function.

```
#include <stdio.h>
void readMatrix(int M[4][4]);
void reduceMatrix(int matrix[4][4]);
void display(int M[4][4]);
int main()
{
   int A[4][4];
   readMatrix(A);
   reduceMatrix(A);
   printf("reduceMatrix(): \n");
   display(A);
   return 0;
}
void display(int M[4][4])
{
   int l,m;
   for (l = 0; l < 4; l++) {
      for (m = 0; m < 4; m++)
         printf("%d ", M[l][m]);
      printf("\n");
   }
   printf("\n");
}
void readMatrix(int M[4][4])
{
   int i, j;
   printf("Enter the matrix (4x4): \n");
   for (i=0; i<4; i++)
      for (j=0; j<4; j++)
         scanf("%d", &M[i][j]);
}
```



```
Enter the matrix (4x4) row by row:
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 16
reduce():
   28    2    3    4
    0   30    7    8
    0    0   26   12
    0    0    0   16
```

```
void reduceMatrix(int matrix[4][4])
{
   int i, j, sum; // i for row, j for column
   /* for each column */
   for (j = 0; j < 4; j++){
      sum = 0;
      // process row below the column
      for (i = j+1; i<4; i++){
         sum += matrix[i][j];
         matrix[i][j] = 0;
      }
      matrix[j][j] += sum;
  }
}
```

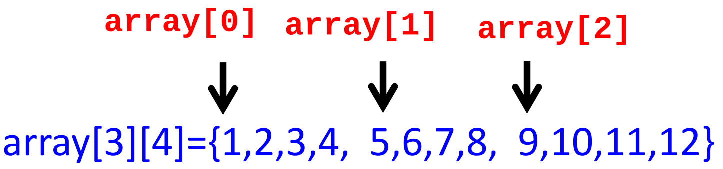
Arrays – Q4

Q: Explain how the addition of 1 to every element of the two dimensional array ‘array’ is done in the following program.

```
#include  <stdio.h>
void  add1(int ar[], int size);
int main(){
    int  array[3][4];
    int h,k;
    for (h = 0; h < 3; h++)
       for (k = 0; k < 4; k++)
          scanf(“%d”, &array[h][k]);

    for (h = 0; h < 3; h++) /* line a */
       add1(array[h], 4);

    for (h = 0; h < 3; h++) {
       for (k = 0; k < 4; k++)
          printf(“%10d”, array[h][k]);
          putchar(‘\n’);
    }
    return 0;
}
void add1(int ar[], int size)
{
    int k;
    for (k = 0; k < size; k++)
       ar[k]++;
}
```



array[0] array[1] array[2]

array[3][4]={1,2,3,4, 5,6,7,8, 9,10,11,12}

Output:

```
2     3     4     5
6     7     8     9
10    11    12    13
```

The function add1() have two parameters. The first one is an array address and the second one is the size of the array. So the function adds 1 to every element of the one dimensional array.

When the function is called in the for statement at line a by

add1(array[h], 4);

array[h] is an one dimensional array of 4 integers. It is the (h+1)th row of the two dimensional array ‘array’. In fact, array[h] is the address of the first element of the (h+1)th row. So every function call works on one row of the two dimensional array.

Arrays – Q4

```
#include  <stdio.h>
void  add1(int ar[], int size);
int main(){
    int  array[3][4];
    int h,k;
    for (h = 0; h < 3; h++)
       for (k = 0; k < 4; k++)
          scanf(“%d”, &array[h][k]);

    //for (h = 0; h < 3; h++)   /* line a */
    // add1(array[h], 4);
    add1(array[0], 3*4);

    for (h = 0; h < 3; h++) {
       for (k = 0; k < 4; k++)
          printf(“%10d”, array[h][k]);
          putchar(‘\n’);
    }
    return 0;
}
void add1(int ar[], int size)
{
    int k;
    for (k = 0; k < size; k++)
       ar[k]++;
}
```

Q: What if the for statement at ‘line a’ is replaced by this statement: add1(array[0], 3 * 4);

array[0]
↓
array[3][4]={1,2,3,4,5,6,7,8,9,10,11,12}

Output:

```
2     3     4     5
6     7     8     9
10    11    12    13
```

When the for statement at line a is replaced by add1(array[0], 3*4), it is passing the address of the first element of the first row to add1() and telling the function that the array size is 12. So add1() works on an one dimensional array starting at array[0] and with 12 elements.