Answers For Question 1

After using the current code use this to update it:

* bool isValid(const int arr[],int length, int pos,…): returns true if pos is within the valid range of array indices, false otherwise. This function must be called in every function that checks the validity of the indices.
* void remove\_element(int arr[],int length, int pos): removes the array element at index pos by shifting up all preceding array elements by one position, see Fig 1.1. The value of the array at index 0 remains unchanged. If “pos” is not a valid array index, print a relevant message and exit the function.

Updated code:

#include <stdio.h>

#include <stdbool.h>

#define SIZE 24

#define nRows 8

#define nCols 3

// PROTOTYPES

bool isValid(const int arr[], int length, int pos);

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j);

void print\_array(int array[], int length);

void print\_matrix(int mat[][nCols], int rows);

void remove\_element(int arr[], int\* length, int pos);

int main() {

int arr[SIZE] = {0}; // Initialize array with zero for testing

int length = SIZE; // Initially, the array length is SIZE

int arr2d[nRows][nCols] = {{0}}; // Initialize matrix with zero for testing

// Call to print 1D array

print\_array(arr, length);

// Remove an element at position 5

remove\_element(arr, &length, 5);

printf("After removing element at index 5:\n");

print\_array(arr, length);

// Call to print 2D matrix

print\_matrix(arr2d, nRows);

return 0;

}

// Function to check if index is valid in a 1D array

bool isValid(const int arr[], int length, int pos) {

return pos >= 0 && pos < length;

}

// Function to check if indices are valid in a 2D matrix

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j) {

return i >= 0 && i < rows && j >= 0 && j < cols;

}

// Function to print all elements of a 1D array

void print\_array(int array[], int length) {

for (int i = 0; i < length; i++) {

printf("array[%d] = %d\n", i, array[i]);

}

}

// Function to print all elements of a 2D matrix

void print\_matrix(int mat[][nCols], int rows) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < nCols; j++) {

printf("mat[%d][%d] = %d\n", i, j, mat[i][j]);

}

puts(""); // Print a newline after each row

}

}

// Function to remove an element from the array and shift elements to the left

void remove\_element(int arr[], int\* length, int pos) {

// Check if the position is valid

if (!isValid(arr, \*length, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the left

for (int i = pos; i < \*length - 1; i++) {

arr[i] = arr[i + 1]; // Move element to the left

}

// Decrease the array's logical length

(\*length)--;

// Optionally, clear the last element to avoid leftover data (not necessary in many cases)

arr[\*length] = 0; // Optional: Clear the last element

}

* void insert\_element(int arr[],int length, int pos, int value): inserts the parameter value at the specified index pos, shifting the original value at pos and all its preceding elements one position down, see Fig 1.2. If the value of pos is not a valid array index, print a relevant message and exit the function:

#include <stdio.h>

#include <stdbool.h>

#define SIZE 24

#define nRows 8

#define nCols 3

// PROTOTYPES

bool isValid(const int arr[], int length, int pos);

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j);

void print\_array(int array[], int length);

void print\_matrix(int mat[][nCols], int rows);

void remove\_element(int arr[], int\* length, int pos);

void insert\_element(int arr[], int\* length, int pos, int value);

int main() {

int arr[SIZE] = {0}; // Initialize array with zero for testing

int length = SIZE; // Initially, the array length is SIZE

int arr2d[nRows][nCols] = {{0}}; // Initialize matrix with zero for testing

// Call to print 1D array

print\_array(arr, length);

// Insert an element at position 5

insert\_element(arr, &length, 5, 99);

printf("After inserting 99 at index 5:\n");

print\_array(arr, length);

// Call to print 2D matrix

print\_matrix(arr2d, nRows);

return 0;

}

// Function to check if index is valid in a 1D array

bool isValid(const int arr[], int length, int pos) {

return pos >= 0 && pos < length;

}

// Function to check if indices are valid in a 2D matrix

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j) {

return i >= 0 && i < rows && j >= 0 && j < cols;

}

// Function to print all elements of a 1D array

void print\_array(int array[], int length) {

for (int i = 0; i < length; i++) {

printf("array[%d] = %d\n", i, array[i]);

}

}

// Function to print all elements of a 2D matrix

void print\_matrix(int mat[][nCols], int rows) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < nCols; j++) {

printf("mat[%d][%d] = %d\n", i, j, mat[i][j]);

}

puts(""); // Print a newline after each row

}

}

// Function to remove an element from the array and shift elements to the left

void remove\_element(int arr[], int\* length, int pos) {

// Check if the position is valid

if (!isValid(arr, \*length, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the left

for (int i = pos; i < \*length - 1; i++) {

arr[i] = arr[i + 1]; // Move element to the left

}

// Decrease the array's logical length

(\*length)--;

// Optionally, clear the last element to avoid leftover data (not necessary in many cases)

arr[\*length] = 0; // Optional: Clear the last element

}

// Function to insert an element at a specific position in the array

void insert\_element(int arr[], int\* length, int pos, int value) {

// Check if the position is valid and within bounds

if (!isValid(arr, \*length + 1, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the right to make space for the new value

for (int i = \*length; i > pos; i--) {

arr[i] = arr[i - 1]; // Move element to the right

}

// Insert the new value at the specified position

arr[pos] = value;

// Increase the logical length of the array

(\*length)++;

}

* void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]) implements the two requirements:

1. If the length of arr is not equal “nRows\*nCols”, print a relevant error message and exit the function.
2. Otherwise, copy the elements of the 1-D array arr into arr2d, filling it column by column.

Code:

#include <stdio.h>

#include <stdbool.h>

#define SIZE 24

#define nRows 8

#define nCols 3

// PROTOTYPES

bool isValid(const int arr[], int length, int pos);

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j);

void print\_array(int array[], int length);

void print\_matrix(int mat[][nCols], int rows);

void remove\_element(int arr[], int\* length, int pos);

void insert\_element(int arr[], int\* length, int pos, int value);

void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]);

int main() {

int arr[SIZE] = {0}; // Initialize array with zero for testing

int length = SIZE; // Initially, the array length is SIZE

int arr2d[nRows][nCols] = {{0}}; // Initialize matrix with zero for testing

// Call to print 1D array

print\_array(arr, length);

// Insert an element at position 5

insert\_element(arr, &length, 5, 99);

printf("After inserting 99 at index 5:\n");

print\_array(arr, length);

// Call to reshape 1D array to 2D matrix

reshape(arr, length, nRows, nCols, arr2d);

// Call to print 2D matrix

print\_matrix(arr2d, nRows);

return 0;

}

// Function to check if index is valid in a 1D array

bool isValid(const int arr[], int length, int pos) {

return pos >= 0 && pos < length;

}

// Function to check if indices are valid in a 2D matrix

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j) {

return i >= 0 && i < rows && j >= 0 && j < cols;

}

// Function to print all elements of a 1D array

void print\_array(int array[], int length) {

for (int i = 0; i < length; i++) {

printf("array[%d] = %d\n", i, array[i]);

}

}

// Function to print all elements of a 2D matrix

void print\_matrix(int mat[][nCols], int rows) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < nCols; j++) {

printf("mat[%d][%d] = %d\n", i, j, mat[i][j]);

}

puts(""); // Print a newline after each row

}

}

// Function to remove an element from the array and shift elements to the left

void remove\_element(int arr[], int\* length, int pos) {

// Check if the position is valid

if (!isValid(arr, \*length, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the left

for (int i = pos; i < \*length - 1; i++) {

arr[i] = arr[i + 1]; // Move element to the left

}

// Decrease the array's logical length

(\*length)--;

// Optionally, clear the last element to avoid leftover data (not necessary in many cases)

arr[\*length] = 0; // Optional: Clear the last element

}

// Function to insert an element at a specific position in the array

void insert\_element(int arr[], int\* length, int pos, int value) {

// Check if the position is valid and within bounds

if (!isValid(arr, \*length + 1, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the right to make space for the new value

for (int i = \*length; i > pos; i--) {

arr[i] = arr[i - 1]; // Move element to the right

}

// Insert the new value at the specified position

arr[pos] = value;

// Increase the logical length of the array

(\*length)++;

}

// Function to reshape the 1D array into a 2D matrix

void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]) {

// Check if the total number of elements in arr matches the number of elements in the 2D array

if (length != nRows \* nCols) {

printf("Error: The number of elements in the array does not match nRows \* nCols\n");

return; // Exit the function if dimensions do not match

}

// Copy elements from the 1D array into the 2D array, filling column by column

int index = 0; // Index for the 1D array

for (int j = 0; j < nCols; j++) { // Iterate over columns

for (int i = 0; i < nRows; i++) { // Iterate over rows

arr2d[i][j] = arr[index++]; // Assign value from 1D array to 2D array

}

}

printf("Reshaping complete. Elements copied into the 2D matrix.\n");

}

* void trans\_matrix(int nRows, int nCols, const int mat[nRows][nCols], int mat[nCols][ nRows]): generates mat\_transp, the transpose of the input matrix mat.

Code:

#include <stdio.h>

#include <stdbool.h>

#define SIZE 24

#define nRows 8

#define nCols 3

// PROTOTYPES

bool isValid(const int arr[], int length, int pos);

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j);

void print\_array(int array[], int length);

void print\_matrix(int mat[][nCols], int rows);

void remove\_element(int arr[], int\* length, int pos);

void insert\_element(int arr[], int\* length, int pos, int value);

void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]);

void trans\_matrix(int nRows, int nCols, const int mat[nRows][nCols], int mat\_transp[nCols][nRows]);

int main() {

int arr[SIZE] = {0}; // Initialize array with zero for testing

int length = SIZE; // Initially, the array length is SIZE

int arr2d[nRows][nCols] = {{0}}; // Initialize matrix with zero for testing

int mat\_transp[nCols][nRows] = {{0}}; // Matrix for storing the transposed result

// Call to print 1D array

print\_array(arr, length);

// Insert an element at position 5

insert\_element(arr, &length, 5, 99);

printf("After inserting 99 at index 5:\n");

print\_array(arr, length);

// Call to reshape 1D array to 2D matrix

reshape(arr, length, nRows, nCols, arr2d);

// Print the original matrix

printf("Original Matrix:\n");

print\_matrix(arr2d, nRows);

// Transpose the matrix

trans\_matrix(nRows, nCols, arr2d, mat\_transp);

// Print the transposed matrix

printf("Transposed Matrix:\n");

print\_matrix(mat\_transp, nCols);

return 0;

}

// Function to check if index is valid in a 1D array

bool isValid(const int arr[], int length, int pos) {

return pos >= 0 && pos < length;

}

// Function to check if indices are valid in a 2D matrix

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j) {

return i >= 0 && i < rows && j >= 0 && j < cols;

}

// Function to print all elements of a 1D array

void print\_array(int array[], int length) {

for (int i = 0; i < length; i++) {

printf("array[%d] = %d\n", i, array[i]);

}

}

// Function to print all elements of a 2D matrix

void print\_matrix(int mat[][nCols], int rows) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < nCols; j++) {

printf("mat[%d][%d] = %d\n", i, j, mat[i][j]);

}

puts(""); // Print a newline after each row

}

}

// Function to remove an element from the array and shift elements to the left

void remove\_element(int arr[], int\* length, int pos) {

// Check if the position is valid

if (!isValid(arr, \*length, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the left

for (int i = pos; i < \*length - 1; i++) {

arr[i] = arr[i + 1]; // Move element to the left

}

// Decrease the array's logical length

(\*length)--;

// Optionally, clear the last element to avoid leftover data (not necessary in many cases)

arr[\*length] = 0; // Optional: Clear the last element

}

// Function to insert an element at a specific position in the array

void insert\_element(int arr[], int\* length, int pos, int value) {

// Check if the position is valid and within bounds

if (!isValid(arr, \*length + 1, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the right to make space for the new value

for (int i = \*length; i > pos; i--) {

arr[i] = arr[i - 1]; // Move element to the right

}

// Insert the new value at the specified position

arr[pos] = value;

// Increase the logical length of the array

(\*length)++;

}

// Function to reshape the 1D array into a 2D matrix

void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]) {

// Check if the total number of elements in arr matches the number of elements in the 2D array

if (length != nRows \* nCols) {

printf("Error: The number of elements in the array does not match nRows \* nCols\n");

return; // Exit the function if dimensions do not match

}

// Copy elements from the 1D array into the 2D array, filling column by column

int index = 0; // Index for the 1D array

for (int j = 0; j < nCols; j++) { // Iterate over columns

for (int i = 0; i < nRows; i++) { // Iterate over rows

arr2d[i][j] = arr[index++]; // Assign value from 1D array to 2D array

}

}

printf("Reshaping complete. Elements copied into the 2D matrix.\n");

}

// Function to transpose the input matrix mat and store it in mat\_transp

void trans\_matrix(int nRows, int nCols, const int mat[nRows][nCols], int mat\_transp[nCols][nRows]) {

// Transpose the matrix by swapping rows and columns

for (int i = 0; i < nRows; i++) {

for (int j = 0; j < nCols; j++) {

mat\_transp[j][i] = mat[i][j]; // Swap rows and columns

* bool found\_duplicate(int arr[],int length, ….): returns true if there is at least a duplicate values in arr; otherwise returns false.

Code:

#include <stdio.h>

#include <stdbool.h>

#define SIZE 24

#define nRows 8

#define nCols 3

// PROTOTYPES

bool isValid(const int arr[], int length, int pos);

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j);

void print\_array(int array[], int length);

void print\_matrix(int mat[][nCols], int rows);

void remove\_element(int arr[], int\* length, int pos);

void insert\_element(int arr[], int\* length, int pos, int value);

void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]);

void trans\_matrix(int nRows, int nCols, const int mat[nRows][nCols], int mat\_transp[nCols][nRows]);

bool found\_duplicate(int arr[], int length);

int main() {

int arr[SIZE] = {0}; // Initialize array with zero for testing

int length = SIZE; // Initially, the array length is SIZE

int arr2d[nRows][nCols] = {{0}}; // Initialize matrix with zero for testing

int mat\_transp[nCols][nRows] = {{0}}; // Matrix for storing the transposed result

// Call to print 1D array

print\_array(arr, length);

// Insert an element at position 5

insert\_element(arr, &length, 5, 99);

printf("After inserting 99 at index 5:\n");

print\_array(arr, length);

// Call to reshape 1D array to 2D matrix

reshape(arr, length, nRows, nCols, arr2d);

// Print the original matrix

printf("Original Matrix:\n");

print\_matrix(arr2d, nRows);

// Transpose the matrix

trans\_matrix(nRows, nCols, arr2d, mat\_transp);

// Print the transposed matrix

printf("Transposed Matrix:\n");

print\_matrix(mat\_transp, nCols);

// Check for duplicates in the array

if (found\_duplicate(arr, length)) {

printf("Duplicates found in the array.\n");

} else {

printf("No duplicates found in the array.\n");

}

return 0;

}

// Function to check if index is valid in a 1D array

bool isValid(const int arr[], int length, int pos) {

return pos >= 0 && pos < length;

}

// Function to check if indices are valid in a 2D matrix

bool isValidMatrix(int mat[][nCols], int rows, int cols, int i, int j) {

return i >= 0 && i < rows && j >= 0 && j < cols;

}

// Function to print all elements of a 1D array

void print\_array(int array[], int length) {

for (int i = 0; i < length; i++) {

printf("array[%d] = %d\n", i, array[i]);

}

}

// Function to print all elements of a 2D matrix

void print\_matrix(int mat[][nCols], int rows) {

for (int i = 0; i < rows; i++) {

for (int j = 0; j < nCols; j++) {

printf("mat[%d][%d] = %d\n", i, j, mat[i][j]);

}

puts(""); // Print a newline after each row

}

}

// Function to remove an element from the array and shift elements to the left

void remove\_element(int arr[], int\* length, int pos) {

// Check if the position is valid

if (!isValid(arr, \*length, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the left

for (int i = pos; i < \*length - 1; i++) {

arr[i] = arr[i + 1]; // Move element to the left

}

// Decrease the array's logical length

(\*length)--;

// Optionally, clear the last element to avoid leftover data (not necessary in many cases)

arr[\*length] = 0; // Optional: Clear the last element

}

// Function to insert an element at a specific position in the array

void insert\_element(int arr[], int\* length, int pos, int value) {

// Check if the position is valid and within bounds

if (!isValid(arr, \*length + 1, pos)) {

printf("Error: Invalid index %d\n", pos);

return; // Exit the function if the index is not valid

}

// Shift elements to the right to make space for the new value

for (int i = \*length; i > pos; i--) {

arr[i] = arr[i - 1]; // Move element to the right

}

// Insert the new value at the specified position

arr[pos] = value;

// Increase the logical length of the array

(\*length)++;

}

// Function to reshape the 1D array into a 2D matrix

void reshape(const int arr[], int length, int nRows, int nCols, int arr2d[nRows][nCols]) {

// Check if the total number of elements in arr matches the number of elements in the 2D array

if (length != nRows \* nCols) {

printf("Error: The number of elements in the array does not match nRows \* nCols\n");

return; // Exit the function if dimensions do not match

}

// Copy elements from the 1D array into the 2D array, filling column by column

int index = 0; // Index for the 1D array

for (int j = 0; j < nCols; j++) { // Iterate over columns

for (int i = 0; i < nRows; i++) { // Iterate over rows

arr2d[i][j] = arr[index++]; // Assign value from 1D array to 2D array

}

}

printf("Reshaping complete. Elements copied into the 2D matrix.\n");

}

// Function to transpose the input matrix mat and store it in mat\_transp

void trans\_matrix(int nRows, int nCols, const int mat[nRows][nCols], int mat\_transp[nCols][nRows]) {

// Transpose the matrix by swapping rows and columns

for (int i = 0; i < nRows; i++) {

for (int j = 0; j < nCols; j++) {

mat\_transp[j][i] = mat[i][j]; // Swap rows and columns

}

}

printf("Matrix transposition complete.\n");

}

// Function to check if there are duplicate values in the array

bool found\_duplicate(int arr[], int length) {

for (int i = 0; i < length - 1; i++) { // Outer loop, from the first element to the second-last element

for (int j = i + 1; j < length; j++) { // Inner loop, comparing with the subsequent elements

if (arr[i] == arr[j]) { // If duplicate found

return true;

}

}

}

return false; // No duplicates found after checking all element