Question 1: Arrays and Pointers

```
#include <stdio.h>
void reverseArray(int *arr, int size) {
    int temp;
    for (int i = 0; i < size / 2; i++) {
        temp = arr[i];
        arr[i] = arr[size - i - 1];
        arr[size - i - 1] = temp;
    }
}
void printArray(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int arr[] = {1, 2, 3, 4, 5, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    reverseArray(arr, size);
    printArray(arr, size);
    return 0;
}
```

Sub-Questions:

1. What is the size of the array `arr` in the `main()` function?
   a) 5
   b) 6
   c) 24
   d) Undefined
2. What does the `reverseArray()` function do?
   a) Reverses the array in-place.
   b) Copies the array to a new array in reverse order.
   c) Prints the array in reverse.
   d) Does nothing.
3. What will be the output of the program?
   a) 6 5 4 3 2 1
   b) 1 2 3 4 5 6

c) 1 6 5 4 3 2
d) Undefined behavior

4. Select the correct syntax to declare a pointer to an integer in the program:

   a) `int ptr*;`

   b) `int *ptr;`

   c) `int &ptr;`

   d) `int ptr[];`

5. What happens if `size` is 0?
   a) Segmentation fault
   b) Prints an empty line
   c) Program crashes
   d) Undefined behavior

Question 2: Dynamic Memory Allocation

```
#include <stdio.h>
#include <stdlib.h>
int* createArray(int size) {
    int *arr = (int*)malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        arr[i] = i + 1;
    }
    return arr;
}
void freeArray(int *arr) {
    free(arr);
}
int main() {
    int size = 5;
    int *arr = createArray(size);
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
```

```
    freeArray(arr);
    return 0;
}
```

Sub-Questions:

1. What does the `createArray()` function return?
   a) A pointer to a dynamically allocated array.
   b) A pointer to a stack-allocated array.
   c) A pointer to a constant array.
   d) Undefined behavior.
2. What happens if `malloc()` fails in `createArray()`?
   a) Returns NULL.
   b) Program crashes.
   c) Returns a pointer to garbage memory.
   d) Undefined behavior.
3. What will be the output of the program?
   a) 1 2 3 4 5
   b) 0 1 2 3 4
   c) Garbage values
   d) Undefined behavior
4. What will happen if `free()` is not called?
   a) Memory leak
   b) Segmentation fault
   c) Double free error
   d) No effect
5. What is the purpose of casting `malloc()`?
   a) To ensure compatibility with C++ compilers.
   b) To prevent type mismatch errors.
   c) It is mandatory in C.
   d) It has no effect.

Question 3: File Handling

```
#include <stdio.h>
int main() {
    FILE *fp = fopen("data.txt", "w+");
    if (!fp) {
        printf("File cannot be opened\n");
        return 1;
    }
```

```
    fprintf(fp, "Hello, World!\nWelcome to File Handling in C.");
    rewind(fp);
    char ch;
    while ((ch = fgetc(fp)) != EOF) {
        putchar(ch);
    }
    fclose(fp);
    return 0;
}
```

Sub-Questions:

1. What does the mode `"w+"` do in `fopen()`?
   a) Opens the file for reading only.
   b) Opens the file for writing and reading, truncating its content.
   c) Opens the file for appending and reading.
   d) None of the above.
2. What is the purpose of the `rewind()` function?
   a) Moves the file pointer to the beginning.
   b) Moves the file pointer to the end.
   c) Closes the file.
   d) Deletes the content of the file.
3. What will the program output?
   a) Hello, World!
   Welcome to File Handling in C.
   b) File cannot be opened
   c) Nothing
   d) Undefined behavior
4. What happens if the file `"data.txt"` does not exist?
   a) It is created.
   b) Program crashes.
   c) Returns NULL.
   d) Undefined behavior.
5. What will happen if `fclose(fp)` is omitted?
   a) File is not closed, and resources are leaked.
   b) Program crashes.
   c) File is automatically closed.
   d) Undefined behavior.

Question 4: Functions and Recursion

```
#include <stdio.h>
int factorial(int n) {
    if (n == 0) return 1;
    return n * factorial(n - 1);
}
int main() {
    int num = 5;
    printf("Factorial of %d is %d\n", num, factorial(num));
    return 0;
}
```

Sub-Questions:

1. What is the base case for the recursive function `factorial()`?
   a) `n == 0`
   b) `n == 1`
   c) `n == -1`
   d) No base case
2. What happens if `num` is negative?
   a) Program crashes.
   b) Undefined behavior.
   c) Stack overflow.
   d) Program does not terminate.
3. What is the time complexity of `factorial()`?
   a) O(1)
   b) O(n)
   c) O(n^2)
   d) O(log n)
4. What is the value of `factorial(5)`?
   a) 120
   b) 60
   c) 24
   d) Undefined
5. What happens if `factorial()` does not have a base case?
   a) Infinite recursion
   b) Program crashes with stack overflow
   c) Both a and b
   d) None of the above

Question 5: Image Processing (LodePNG)

```
#include "lodepng.h"

#include <stdio.h>

int main() {

    const char *filename = "image.png";

    unsigned char *image;

    unsigned width, height;

    unsigned error = lodepng_decode32_file(&image, &width, &height,
filename);

    if (error) {

        printf("Error: %s\n", lodepng_error_text(error));

        return 1;

    }

    printf("Image loaded: %ux%u\n", width, height);

    free(image);

    return 0;

}
```

Sub-Questions:

1. What does `lodepng_decode32_file()` do?
   a) Loads a 32-bit PNG image.
   b) Converts an image to 32-bit.
   c) Creates a new PNG image.
   d) Compresses an image.

2. What happens if `image.png` does not exist?
   a) Returns an error code.
   b) Crashes the program.
   c) Creates a new image.
   d) Undefined behavior.
3. What will the program output if the image is 800x600?
   a) Error: File not found
   b) Image loaded: 800x600
   c) Image loaded: 600x800
   d) Undefined behavior
4. What is the purpose of `free(image)`?
   a) Deallocate memory used by the image.
   b) Clears the image content.
   c) Saves the image.
   d) None of the above.
5. What happens if `lodepng.h` is not included?
   a) Compilation error
   b) Program runs normally
   c) Runtime error
   d) None of the above

Question 6: String Compression

```
#include <stdio.h>
#include <string.h>

void compressString(char *str, char *result) {
    int count = 1;
    int j = 0;
    for (int i = 1; i <= strlen(str); i++) {
        if (str[i] == str[i - 1]) {
            count++;
        } else {
            result[j++] = str[i - 1];
            if (count > 1) {
                j += sprintf(&result[j], "%d", count);
            }
            count = 1;
        }
```

```
    }
    result[j] = '\0';
}

int main() {
    char str[100], result[200];
    printf("Enter a string: ");
    scanf("%s", str);

    compressString(str, result);

    printf("Compressed string: %s\n", result);
    return 0;
}
```

Sub-questions:

1. What is the time complexity of the `compressString()` function?
   a) O(n)
   b) O(n^2)
   c) O(log n)
   d) O(n log n)
2. What happens if the input string is empty?
   a) The program crashes.
   b) The output will be an empty string.
   c) It results in a segmentation fault.
   d) The output will be "0".
3. Select the correct syntax to declare the `compressString` function.
   a) `void compressString(char str[], char result[]);`
   b) `void compressString(char *str, char *result);`
   c) `char compressString(char *str, char *result);`
   d) `char *compressString(char str[], char result[]);`
4. If the input string is "aaabbc", what will the output be?
   a) "a3b2c"
   b) "3a2b1c"
   c) "a3b2c1"
   d) "aabbc"
5. Which standard library function is used to append a character count in the code?
   a) `sprintf()`
   b) `strcat()`

c) `strcpy()`
d) `snprintf()`

Question 7: Matrix Multiplication

```
#include <stdio.h>
#include <stdlib.h>

void inputMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("Enter element [%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
        }
    }
}

void multiplyMatrices(int **A, int **B, int **C, int r1, int c1, int
c2) {
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c2; j++) {
            C[i][j] = 0;
            for (int k = 0; k < c1; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

void displayMatrix(int **matrix, int rows, int cols) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main() {
```

```
    int r1, c1, r2, c2;
    printf("Enter rows and columns for the first matrix: ");
    scanf("%d%d", &r1, &c1);
    printf("Enter rows and columns for the second matrix: ");
    scanf("%d%d", &r2, &c2);

    if (c1 != r2) {
        printf("Matrix multiplication not possible.\n");
        return 0;
    }

    int **A = (int **)malloc(r1 * sizeof(int *));
    int **B = (int **)malloc(r2 * sizeof(int *));
    int **C = (int **)malloc(r1 * sizeof(int *));
    for (int i = 0; i < r1; i++) A[i] = (int *)malloc(c1 *
sizeof(int));
    for (int i = 0; i < r2; i++) B[i] = (int *)malloc(c2 *
sizeof(int));
    for (int i = 0; i < r1; i++) C[i] = (int *)malloc(c2 *
sizeof(int));

    printf("Enter elements for the first matrix:\n");
    inputMatrix(A, r1, c1);

    printf("Enter elements for the second matrix:\n");
    inputMatrix(B, r2, c2);

    multiplyMatrices(A, B, C, r1, c1, c2);

    printf("Resultant Matrix:\n");
    displayMatrix(C, r1, c2);

    for (int i = 0; i < r1; i++) free(A[i]);
    for (int i = 0; i < r2; i++) free(B[i]);
    for (int i = 0; i < r1; i++) free(C[i]);
    free(A);
    free(B);
    free(C);
```

```
    return 0;
}
```

Sub-questions:

1. What will the program do if the number of columns in the first matrix is not equal to the number of rows in the second matrix?
   a) It will display an error message and exit.
   b) It will multiply matrices incorrectly.
   c) It will still attempt to multiply and crash.
   d) It will display the resultant matrix as zeros.
2. What is the role of the function `multiplyMatrices()`?
   a) To perform matrix addition.
   b) To multiply two matrices and store the result in a third matrix.
   c) To allocate memory dynamically.
   d) To print the resultant matrix.
3. Select the correct syntax for allocating memory for the matrix dynamically.
   a) `int **A = malloc(r1 * c1 * sizeof(int));`
   b) `int *A = malloc(r1 * sizeof(int));`
   c) `int **A = malloc(r1 * sizeof(int *));`
   d) `int A = malloc(r1 * sizeof(int *));`
4. What is the time complexity of the matrix multiplication algorithm used here?
   a) O(n)
   b) O(n^2)
   c) O(n^3)
   d) O(log n)
5. Which of the following correctly frees the allocated memory for a 2D matrix?
   a) `free(A);`
   b) `for (int i = 0; i < r1; i++) free(A[i]); free(A);`
   c) `free(A[0]);`
   d) `for (int i = 0; i < r1; i++) free(A);`

Question 8: File Copying

```
#include <stdio.h>
#include <stdlib.h>

void copyFile(const char *source, const char *destination) {
    FILE *src = fopen(source, "r");
```

```
    FILE *dest = fopen(destination, "w");

    if (src == NULL || dest == NULL) {
        printf("Error opening file.\n");
        exit(1);
    }

    char ch;
    while ((ch = fgetc(src)) != EOF) {
        fputc(ch, dest);
    }

    fclose(src);
    fclose(dest);
}

int main() {
    char source[100], destination[100];
    printf("Enter the source file name: ");
    scanf("%s", source);
    printf("Enter the destination file name: ");
    scanf("%s", destination);

    copyFile(source, destination);

    printf("File copied successfully.\n");
    return 0;
}
```

Sub-questions:

1. What happens if the source file does not exist?
   a) The program terminates with an error.
   b) The destination file is created empty.
   c) A runtime error occurs.
   d) The program continues without copying.
2. Which function is used to read characters from the source file?
   a) `fgetc()`
   b) `fgets()`

c) `getc()`
d) `fread()`

3. What is the purpose of `fputc(ch, dest);`?
   a) Reads a character from the source file.
   b) Writes a character to the destination file.
   c) Appends a character to the destination file.
   d) Copies the EOF character.
4. Select the correct syntax to open the file in write mode.
   a) `FILE *fp = open("filename", "write");`
   b) `FILE *fp = fopen("filename", "w");`
   c) `FILE *fp = fopen("filename", "rw");`
   d) `FILE *fp = fileopen("filename", "w");`
5. What happens to the destination file if it already exists?
   a) It is appended.
   b) It is overwritten.
   c) An error occurs.
   d) Nothing changes.

Question 9: Dynamic Memory for a 2D Array

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int rows, cols;
    printf("Enter rows and columns: ");
    scanf("%d%d", &rows, &cols);

    int **arr = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        arr[i] = (int *)malloc(cols * sizeof(int));
    }

    printf("Enter elements:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &arr[i][j]);
        }
    }
```

```
    int sum = 0;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum += arr[i][j];
        }
    }

    printf("Sum of all elements: %d\n", sum);

    for (int i = 0; i < rows; i++) {
        free(arr[i]);
    }
    free(arr);

    return 0;
}
```

Sub-questions:

1. What does the program compute?
   a) The sum of all elements.
   b) The product of all elements.
   c) The maximum value.
   d) The row-wise sums.
2. How is memory allocated for the 2D array?
   a) Using `malloc` for both rows and columns.
   b) Using a single `malloc` call.
   c) Using `calloc`.
   d) Statically.
3. What happens if `free(arr)` is not called?
   a) Memory leak.
   b) Program crash.
   c) Undefined behavior.
   d) Compiler error.
4. Select the correct syntax for allocating memory for a single row.
   a) `arr[i] = malloc(sizeof(int *));`
   b) `arr[i] = malloc(cols * sizeof(int));`
   c) `arr[i] = (int *)calloc(cols);`
   d) `arr[i] = malloc(cols);`

What is the output if the input is:

2 2

1 2

3 4

5. a) 6
   b) 10
   c) 12
   d) 16