Quiz 4 Prep Answers

March 18, 2025

1 Answers

1.1 Using Struct

1.1.1 Student Struct Definition

```
struct Student {
    char name[100];
    int age;
    float GPA;
};
```

1.1.2 Student Initialization

```
// Method 1: One-liner
struct Student student1 = {"John Smith", 20, 3.0};

// Method 2: Dot notation and strcpy
#include <string.h>

struct Student student2;
strcpy(student2.name, "John Smith");
student2.age = 20;
student2.GPA = 3.0;
```

1.1.3 Function to Print Student

```
void printStudent(struct Student s) {
    printf("Name: %s\n", s.name);
    printf("Age: %d\n", s.age);
    printf("GPA: %.2f\n", s.GPA);
}

// Test function
```

```c
int main() {
    struct Student student;
    strcpy(student.name, "John Smith");
    student.age = 20;
    student.GPA = 3.0;

    printStudent(student);

    return 0;
}
```

1.2 Function Returning Pointer

1.2.1 Allocate Integer Array

```c
int* allocateIntArray(int n) {
    int* arr = (int*)malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    return arr;
}

// Example usage
int main() {
    int size;
    printf("Enter the size of the array: ");
    scanf("%d", &size);

    int* myArr = allocateIntArray(size);

    // Use the array...
    ...

    // Free when done
    free(myArr);

    return 0;
}
```

1.2.2 Pointer to Largest Element

```c
int* findLargest(int* arr, int size) {
    if (size <= 0 || arr == NULL) {
        return NULL;
    }

    int* largest = &arr[0];

    for (int i = 1; i < size; i++) {
        if (arr[i] > *largest) {
            largest = &arr[i];
        }
    }

    return largest;
}

// Example usage
int main() {
    int array[] = {1, 2, 7, 3, 4, 5};
    int size = sizeof(array) / sizeof(int);

    int* largest = findLargest(array, size);

    if (largest != NULL) {
        printf("The largest element is: %d\n", *largest);
    }

    return 0;
}
```

1.3 2D Memory Allocation

1.3.1 Allocate and Initialize 2D Array

```c
int** allocate2DArray(int m, int n) {
    // Allocate array of pointers
    int** arr = (int**)malloc(m * sizeof(int*));

    if (arr == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    // Allocate each row
    for (int i = 0; i < m; i++) {
        arr[i] = (int*)malloc(n * sizeof(int));
```

```
        if (arr[i] == NULL) {
            printf("Memory allocation failed\n");
            // Free previously allocated memory
            for (int j = 0; j < i; j++) {
                free(arr[j]);
            }
            free(arr);
            exit(1);
        }

        // Initialize all elements to 5
        for (int j = 0; j < n; j++) {
            arr[i][j] = 5;
        }
    }

    return arr;
}

// Example usage
int main() {
    int rows, cols;
    printf("Enter number of rows: ");
    scanf("%d", &rows);
    printf("Enter number of columns: ");
    scanf("%d", &cols);

    int** matrix = allocate2DArray(rows, cols);

    // Use the matrix...

    // Free the allocated memory
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);

    return 0;
}
```

1.3.2 malloc vs calloc

Since we are overwriting the values to 5, calloc does not provide any benefit. Thus, we can use malloc. (Performance boost is likely negligible here, but good to consider use cases of these functions; don't need to zero our array if we're immediately overwriting).

1.4 3D Memory Allocation

1.4.1 Allocate 3D Array

```
int*** allocate3DArray(int x, int y, int z) {
    // Allocate array of pointers to pointers
    int*** arr = (int***)malloc(x * sizeof(int**));

    if (arr == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }

    // Allocate array of pointers for each page
    for (int i = 0; i < x; i++) {
        arr[i] = (int**)malloc(y * sizeof(int*));

        if (arr[i] == NULL) {
            printf("Memory allocation failed\n");
            // Free previously allocated memory
            for (int j = 0; j < i; j++) {
                for (int k = 0; k < y; k++) {
                    free(arr[j][k]);
                }
                free(arr[j]);
            }
            free(arr);
            exit(1);
        }

        // Allocate each row
        for (int j = 0; j < y; j++) {
            arr[i][j] = (int*)malloc(z * sizeof(int));

            if (arr[i][j] == NULL) {
                printf("Memory allocation failed\n");
                // Free previously allocated memory
                for (int k = 0; k < j; k++) {
                    free(arr[i][k]);
                }
                for (int k = 0; k < i; k++) {
                    for (int l = 0; l < y; l++) {
                        free(arr[k][l]);
                    }
                    free(arr[k]);
                }
                free(arr[i]);
                free(arr);
```

```
                exit(1);
            }
        }
    }

    return arr;
}
```

1.4.2 Print 3D Array

```
void print3DArray(int*** arr, int x, int y, int z) {
    for (int i = 0; i < x; i++) {
        for (int j = 0; j < y; j++) {
            for (int k = 0; k < z; k++) {
                printf("%d ", arr[i][j][k]);
            }
            printf("\n");  // New line after z elements
        }
    }
}

// Example usage
int main() {
    int x, y, z;
    printf("Enter dimensions (x y z): ");
    scanf("%d %d %d", &x, &y, &z);

    int*** array3D = allocate3DArray(x, y, z);

    // Initialize with some values
    for (int i = 0; i < x; i++) {
        for (int j = 0; j < y; j++) {
            for (int k = 0; k < z; k++) {
                array3D[i][j][k] = i + j + k;
            }
        }
    }

    // Print the array
    print3DArray(array3D, x, y, z);

    // Free the allocated memory
    for (int i = 0; i < x; i++) {
        for (int j = 0; j < y; j++) {
            free(array3D[i][j]);
        }
        free(array3D[i]);
```

```
    }
    free(array3D);

    return 0;
}
```