1. .

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

/* Define the Student struct */
struct Student {
    char name[50];
    int age;
    float gpa;
};

/* Function to print a student's details */
void printStudent(struct Student s) {
    printf("Name: %s, Age: %d, GPA: %.2f\n", s.name, s.age, s.gpa);
}

/* Function to update a student's GPA */
void updateGPA(struct Student *s, float newGPA) {
    s->gpa = newGPA;
}

int main() {
    printf("(a) Define and Initialize a Struct\n");
    struct Student johnDoe = {"John Doe", 20, 3.5};
    printStudent(johnDoe);

    printf("\n(b) Accessing Struct Members using Pointers\n");
    struct Student student;
    struct Student *ptr = &student;
    /* Manually copy the string as strcpy is not allowed */
    const char *name = "Alice";
    int i = 0;
    while (name[i] != '\0') {
        ptr->name[i] = name[i];
        i++;
    }
    ptr->name[i] = '\0'; // Null-terminate the string
    ptr->age = 22;
    ptr->gpa = 3.9;
    printStudent(*ptr);

    /* Use the pointer to change the gpa of the student */
    updateGPA(ptr, 3.8);
    printStudent(*ptr);

    printf("\n(c) Dynamic Memory Allocation for Structs\n");
    struct Student *dynamicStudent = (struct Student *)malloc(sizeof(struct Student));
    if (dynamicStudent != NULL) {
        /* Initialize the dynamic student */
        const char *dynamicName = "Bob";
        i = 0;
        while (dynamicName[i] != '\0') {
            dynamicStudent->name[i] = dynamicName[i];
            i++;
        }
        dynamicStudent->name[i] = '\0'; // Null-terminate the string
        dynamicStudent->age = 24;
        dynamicStudent->gpa = 3.7;
        printStudent(*dynamicStudent);

        /* Free the allocated memory */
        free(dynamicStudent);
    } else {
        printf("Memory allocation failed\n");
    }

    printf("\n(d) Array of Structs\n");
    struct Student students[5] = {
```

```
    struct Student students[5] = {
        {"Student1", 20, 3.1},
        {"Student2", 21, 3.2},
        {"Student3", 22, 3.3},
        {"Student4", 23, 3.4},
        {"Student5", 24, 3.5}
    };
    for (i = 0; i < 5; i++) {
        printStudent(students[i]);
    }

    printf("\n(e) Passing Structs to Functions\n");
    struct Student charlie = {"Charlie", 25, 3.6};
    printStudent(charlie); /* Print initial details */

    return 0;
}
```

Output

```
(a) Define and Initialize a Struct
Name: John Doe, Age: 20, GPA: 3.50

(b) Accessing Struct Members using Pointers
Name: Alice, Age: 22, GPA: 3.90
Name: Alice, Age: 22, GPA: 3.80

(c) Dynamic Memory Allocation for Structs
Name: Bob, Age: 24, GPA: 3.70

(d) Array of Structs
Name: Student1, Age: 20, GPA: 3.10
Name: Student2, Age: 21, GPA: 3.20
Name: Student3, Age: 22, GPA: 3.30
Name: Student4, Age: 23, GPA: 3.40
Name: Student5, Age: 24, GPA: 3.50

(e) Passing Structs to Functions
Name: Charlie, Age: 25, GPA: 3.60


...Program finished with exit code 0
Press ENTER to exit console.
```

2. .

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

/* (a) Define the Matrix struct */
typedef struct {
    int rows, cols;
    double *data;
} Matrix;

/* Function prototypes */
void enterMatrixData(Matrix *m);
void printMatrix(const Matrix *m);
void multiply(const Matrix *a, const Matrix *b, Matrix *result);

int main() {
    Matrix a, b, result;

    /* (c) User Input for matrix dimensions */
    printf("Enter the dimensions of the first matrix (rows columns): ");
    scanf("%d %d", &a.rows, &a.cols);
    printf("Enter the dimensions of the second matrix (rows columns): ");
    scanf("%d %d", &b.rows, &b.cols);

    /* Validate dimensions for multiplication */
    if (a.cols != b.rows) {
        printf("Error: The number of columns in the first matrix must be equal to the number of rows in the second matrix.\n");
        return 1;
    }

    /* Dynamically allocate memory for the matrices */
    a.data = (double *)malloc(a.rows * a.cols * sizeof(double));
    b.data = (double *)malloc(b.rows * b.cols * sizeof(double));
    result.data = (double *)malloc(a.rows * b.cols * sizeof(double));
    result.rows = a.rows;
    result.cols = b.cols;

    /* (c) User Input for matrix elements */
    printf("Enter the elements of the first matrix:\n");
    enterMatrixData(&a);
    printf("Enter the elements of the second matrix:\n");
    enterMatrixData(&b);

    /* (d) Matrix Multiplication */
    multiply(&a, &b, &result);

    /* Display the resulting matrix */
    printf("The resulting matrix is:\n");
    printMatrix(&result);

    /* (e) Memory Management */
    free(a.data);
    free(b.data);
    free(result.data);

    return 0;
}

/* Function to enter matrix data */
void enterMatrixData(Matrix *m) {
    for (int i = 0; i < m->rows; i++) {
        for (int j = 0; j < m->cols; j++) {
            scanf("%lf", &m->data[i * m->cols + j]);
```

```c
            scanf("%lf", &m->data[i * m->cols + j]);
        }
    }
}

/* Function to print matrix */
void printMatrix(const Matrix *m) {
    for (int i = 0; i < m->rows; i++) {
        for (int j = 0; j < m->cols; j++) {
            printf("%.2f ", m->data[i * m->cols + j]);
        }
        printf("\n");
    }
}

/* (b) Implement the 'multiply' Function */
void multiply(const Matrix *a, const Matrix *b, Matrix *result) {
    for (int i = 0; i < a->rows; i++) {
        for (int j = 0; j < b->cols; j++) {
            double sum = 0;
            for (int k = 0; k < a->cols; k++) {
                sum += a->data[i * a->cols + k] * b->data[k * b->cols + j];
            }
            result->data[i * result->cols + j] = sum;
        }
    }
}
```

Output

```
Enter the dimensions of the first matrix (rows columns): 2 2
Enter the dimensions of the second matrix (rows columns): 2 2
Enter the elements of the first matrix:
1 2 3 4
Enter the elements of the second matrix:
1 2 3 4
The resulting matrix is:
7.00 10.00
15.00 22.00


...Program finished with exit code 0
Press ENTER to exit console.
```