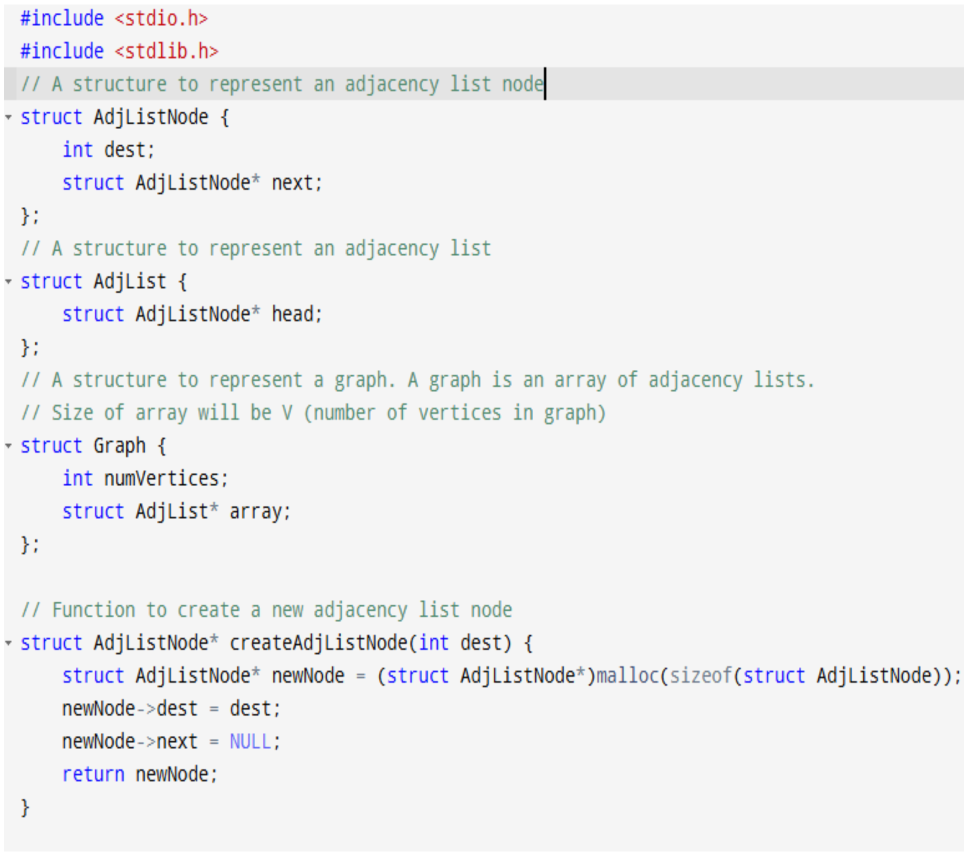
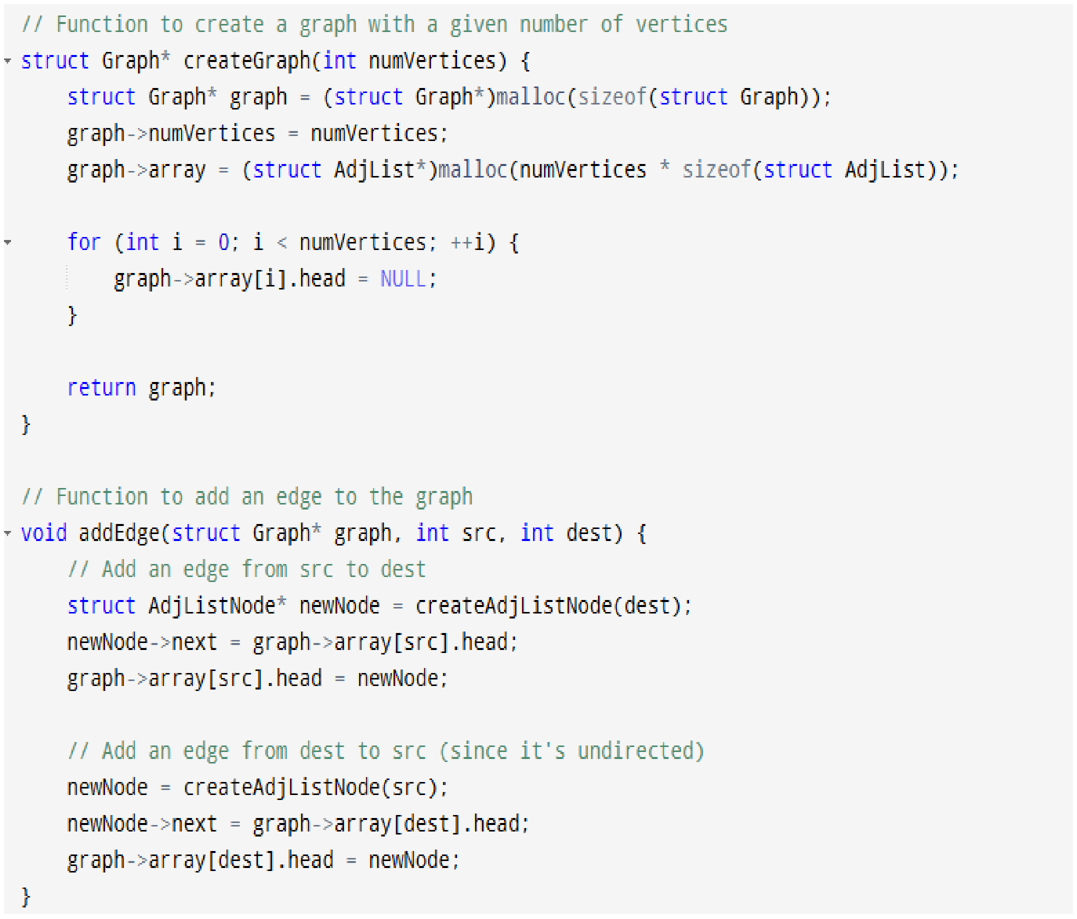
Develop a program to insert and delete nodes in a graph using an adjacency list representation, allowing for evaluation of graph structure modifications.



```c
#include <stdio.h>
#include <stdlib.h>
// A structure to represent an adjacency list node
struct AdjListNode {
    int dest;
    struct AdjListNode* next;
};
// A structure to represent an adjacency list
struct AdjList {
    struct AdjListNode* head;
};
// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph {
    int numVertices;
    struct AdjList* array;
};

// Function to create a new adjacency list node
struct AdjListNode* createAdjListNode(int dest) {
    struct AdjListNode* newNode = (struct AdjListNode*)malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}
```



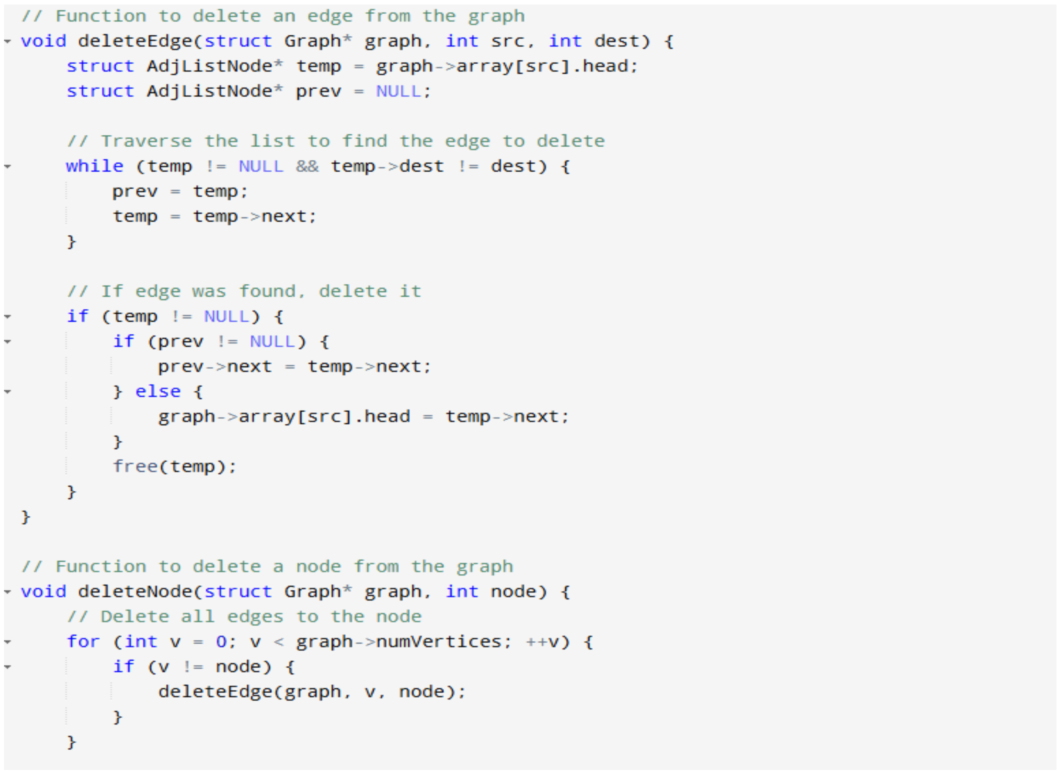
```c
// Function to create a graph with a given number of vertices
struct Graph* createGraph(int numVertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numVertices = numVertices;
    graph->array = (struct AdjList*)malloc(numVertices * sizeof(struct AdjList));

    for (int i = 0; i < numVertices; ++i) {
        graph->array[i].head = NULL;
    }

    return graph;
}

// Function to add an edge to the graph
void addEdge(struct Graph* graph, int src, int dest) {
    // Add an edge from src to dest
    struct AdjListNode* newNode = createAdjListNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;

    // Add an edge from dest to src (since it's undirected)
    newNode = createAdjListNode(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}
```



```c
// Function to delete an edge from the graph
void deleteEdge(struct Graph* graph, int src, int dest) {
    struct AdjListNode* temp = graph->array[src].head;
    struct AdjListNode* prev = NULL;

    // Traverse the list to find the edge to delete
    while (temp != NULL && temp->dest != dest) {
        prev = temp;
        temp = temp->next;
    }

    // If edge was found, delete it
    if (temp != NULL) {
        if (prev != NULL) {
            prev->next = temp->next;
        } else {
            graph->array[src].head = temp->next;
        }
        free(temp);
    }
}

// Function to delete a node from the graph
void deleteNode(struct Graph* graph, int node) {
    // Delete all edges to the node
    for (int v = 0; v < graph->numVertices; ++v) {
        if (v != node) {
            deleteEdge(graph, v, node);
        }
    }
```

```c
int main() {
    int numVertices = 5;
    struct Graph* graph = createGraph(numVertices);

    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    printf("Graph before deletion:\n");
    printGraph(graph);

    deleteNode(graph, 3);

    printf("\nGraph after deleting node 3:\n");
    printGraph(graph);

    return 0;
```