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

#define MAX_VERTICES 10 // Maximum number of vertices in the graph

// Structure for the adjacency list node
struct Node {
   int vertex;
   struct Node* next;
};

// Structure for the graph using adjacency list
struct Graph {
   struct Node* adjList[MAX_VERTICES];
   int numVertices;
};

// Function to create a new adjacency list node
struct Node* createNode(int v) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); // Allocate memory for the
new node
   newNode->vertex = v;  // Set the vertex number
   newNode->next = NULL; // Set the next pointer to NULL
   return newNode;       // Return the new node
}

// Function to initialize the graph
void createGraph(struct Graph* graph, int vertices) {
   graph->numVertices = vertices; // Set the number of vertices

   // Initialize all adjacency list heads to NULL
   for (int i = 0; i < vertices; i++) {
      graph->adjList[i] = NULL;
   }
}

// Function to add an edge to the graph (undirected)
void addEdge(struct Graph* graph, int src, int dest) {
   // Add edge from src to dest
   struct Node* newNode = createNode(dest);
   newNode->next = graph->adjList[src]; // Point the new node to the head of the src list
   graph->adjList[src] = newNode;       // Make the new node the head of the src list

   // Add edge from dest to src (since it's an undirected graph)
   newNode = createNode(src);
   newNode->next = graph->adjList[dest];
   graph->adjList[dest] = newNode;
}

// Function to print the adjacency list representation of the graph
void printGraph(struct Graph* graph) {
   for (int i = 0; i < graph->numVertices; i++) {
      struct Node* temp = graph->adjList[i];
      printf("Vertex %d: ", i);
      while (temp != NULL) {
         printf("-> %d ", temp->vertex);
         temp = temp->next;
```

```
        }
        printf("\n");
    }
}

// Simplified BFS function
void BFS(struct Graph* graph, int startVertex) {
    int visited[MAX_VERTICES] = {0}; // Array to track visited vertices
    int queue[MAX_VERTICES];        // Queue for BFS
    int front = 0, rear = 0;        // Front and rear of the queue

    visited[startVertex] = 1;       // Mark the start vertex as visited
    queue[rear++] = startVertex;    // Enqueue the start vertex

    while (front < rear) {          // While the queue is not empty
        int currentVertex = queue[front++]; // Dequeue a vertex
        printf("%d ", currentVertex); // Print the vertex

        struct Node* temp = graph->adjList[currentVertex]; // Get the adjacent vertices
        while (temp) {
            int adjVertex = temp->vertex;
            if (!visited[adjVertex]) {  // If the vertex is not visited
                visited[adjVertex] = 1; // Mark it as visited
                queue[rear++] = adjVertex; // Enqueue it
            }
            temp = temp->next; // Move to the next adjacent vertex
        }
    }
}

// Simplified DFS function using a stack-based approach
void DFS(struct Graph* graph, int startVertex) {
    int visited[MAX_VERTICES] = {0}; // Array to track visited vertices
    int stack[MAX_VERTICES];        // Stack for DFS
    int top = -1;                   // Top of the stack

    stack[++top] = startVertex;     // Push the start vertex onto the stack

    while (top != -1) {             // While the stack is not empty
        int currentVertex = stack[top--]; // Pop a vertex from the stack

        if (!visited[currentVertex]) {  // If the vertex is not visited
            printf("%d ", currentVertex); // Print the vertex
            visited[currentVertex] = 1;  // Mark it as visited
        }

        // Get all adjacent vertices and push them onto the stack if not visited
        struct Node* temp = graph->adjList[currentVertex];
        while (temp) {
            int adjVertex = temp->vertex;
            if (!visited[adjVertex]) {  // If the vertex is not visited
                stack[++top] = adjVertex; // Push it onto the stack
            }
            temp = temp->next; // Move to the next adjacent vertex
        }
    }
}
```

```
// Main function to demonstrate the graph implementation
int main() {
    struct Graph graph;             // Declare a graph variable (no malloc needed)
    createGraph(&graph, 5);         // Create a graph with 5 vertices

    // Add edges to the graph
    addEdge(&graph, 0, 1);
    addEdge(&graph, 0, 2);
    addEdge(&graph, 1, 3);
    addEdge(&graph, 2, 3);


    // Print the adjacency list representation of the graph
    printf("Adjacency List Representation:\n");
    printGraph(&graph);

    // Perform BFS traversal
    printf("\nBFS Traversal starting from vertex 0:\n");
    BFS(&graph, 0);

    // Perform DFS traversal
    printf("\n\nDFS Traversal starting from vertex 0:\n");
    DFS(&graph, 0);

    return 0;
}
```