```c
// EXP08 circular linked list

#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node* next;
};

//INSERTION OF DATA

//FOR A NEW NODE

struct Node* addToEmpty(struct Node* last, int data)
{
  if (last != NULL)
  return last;
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  last = newNode;
  last->next = last;
  return last;
}
//AT THE BEGINNING

struct Node* addFront(struct Node* last, int data)
{
  if (last == NULL)
  return addToEmpty(last, data);
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = last->next;
  last->next = newNode;
  return last;
}
// AT THE END

struct Node* addEnd(struct Node* last, int data)
{
  if (last == NULL)
  return addToEmpty(last, data);
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = last->next;
  last->next = newNode;
  last = newNode;
  return last;
}
//IN BETWEEN SPECIFIC NODES
```

```
struct Node* addAfter(struct Node* last, int data, int item)
{
  if (last == NULL)
  return NULL;
  struct Node *newNode, *p;
  p = last->next;
  do {
  if (p->data == item)
  {
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = p->next;
    p->next = newNode;
    if (p == last) last = newNode;
    return last;
    }
  p = p->next;
  } while (p != last->next);
  printf("\nThe given node is not present in the list");
  return last;
}

//DELETION
//LIST WITH SINGLE NODE

void deleteNode(struct Node** last, int key)
{
  if ((*last) == NULL)
  return;
  if (((*last)->data == key) && ((*last)->next == (*last)))
  {
  free(*last);
  *last = NULL;
  return;
  }
  struct Node *temp = *last, *d;

//AT THE END
  if ((*last)->data == key)
  {
  while (temp->next != (*last))
  temp = temp->next;
  temp->next = (*last)->next;
  free(*last);
  (*last) = temp;
  }

//A SPECIFIC NODE
  while ((temp->next != *last) && (temp->next->data != key))
  {
  temp = temp->next;
  }
  if (temp -> next -> data == key)
  {
  d = temp -> next;
  temp -> next = d -> next;
  free(d);
  }
}

//TRAVERSE/SEARCH DATA
void traverse(struct Node* last)
{
  struct Node* p;
  if (last == NULL)
  {
  printf("The list is empty");
  return;
  }
  p = last -> next ;
  do {
  printf("%d ", p->data);
  p = p -> next;
  } while (p != last -> next);
}

int main()
{
  struct Node* last = NULL;
  last = addToEmpty(last, 6);
  printf("\n6 added to empty list\n");
  last = addEnd(last, 8);
  printf("8 added at the end of the list");
  last = addFront(last, 2);
  printf("\n2 added to front\n");
  last = addAfter(last, 10, 2);
  printf("6 added between specific node\n\n");
  traverse(last);
  deleteNode(&last, 8);
  printf("\n8 deleted from the list\n");
  printf("\n");
  traverse(last);
  return 0;
}


// EXP09 implement stack using linked list

#include <stdio.h>
```

```
#include <stdlib.h>
typedef struct Node
{
    int data;
    struct Node* next;
}
node;
node* createNode(int data)
{
    node* newNode = (node*)malloc(sizeof(node));
    if (newNode == NULL)
    return NULL;
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
int insertBeforeHead(node** head, int data)
{
    node* newNode = createNode(data);
    if (!newNode)
    return -1;
    if (*head == NULL)
    {
        *head = newNode;
        return 0;
    }
    newNode->next = *head;
    *head = newNode;
    return 0;
}
int deleteHead(node** head)
{
    node* temp = *head;
    *head = (*head)->next;
    free(temp);
    return 0;
}

int isEmpty(node** stack)
{
    return *stack == NULL;
}
void push(node** stack, int data)
{
    if (insertBeforeHead(stack, data))
    {
        printf("Stack Overflow!\n");
    }

}
```

```
int pop(node** stack)
{
    if (isEmpty(stack))
    {
        printf("Stack Underflow\n");
        return -1;
    }
    deleteHead(stack);
}
int peek(node** stack)
{
    if (!isEmpty(stack))
    return (*stack)->data;
    else
    return -1;
}
void printStack(node** stack)
{
    node* temp = *stack;
    while (temp != NULL)
    {
        printf("%d-> ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main()
{
    node* stack = NULL;

    push(&stack, 10);
    push(&stack, 20);
    push(&stack, 30);
    push(&stack, 40);
    push(&stack, 50);

    printf("Stack: ");
    printStack(&stack);

    pop(&stack);
    pop(&stack);

    printf("\nStack: ");
    printStack(&stack);

    return 0;
}
```

```c
// EXP10 implement binary search tree

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

struct BinaryTreeNode
{
        int key;
        struct BinaryTreeNode *left, *right;
};
struct BinaryTreeNode*
newNodeCreate(int value)
{
        struct BinaryTreeNode* temp
                = (struct BinaryTreeNode*)malloc(
                        sizeof(struct BinaryTreeNode));
        temp->key = value;
        temp->left = temp->right = NULL;
        return temp;
}
struct BinaryTreeNode*
searchNode(struct BinaryTreeNode* root, int target)
{
        if (root == NULL || root->key == target)
        {
                return root;
        }
        if (root->key < target)
        {
                return searchNode(root->right, target);
        }
        return searchNode(root->left, target);
}
struct BinaryTreeNode*
insertNode(struct BinaryTreeNode* node, int value)
{
        if (node == NULL)
        {
                return newNodeCreate(value);
        }
        else if (value < node->key)
        {
                node->left = insertNode(node->left, value);
        }
        else if (value > node->key)
        {
                node->right = insertNode(node->right, value);
        }
        return node;
```

```
}
void postOrder(struct BinaryTreeNode* root)
{
        if (root != NULL)
        {
                postOrder(root->left);
                postOrder(root->right);
                printf(" %d ", root->key);
        }
}
void inOrder(struct BinaryTreeNode* root)
{
        if (root != NULL)
        {
                inOrder(root->left);
                printf(" %d ", root->key);
                inOrder(root->right);
        }
}
void preOrder(struct BinaryTreeNode* root)
{
        if (root != NULL)
        {
                printf(" %d ", root->key);
                preOrder(root->left);
                preOrder(root->right);
        }
}
struct BinaryTreeNode*
findMin(struct BinaryTreeNode* root)
{
        if (root == NULL)
        {
                return NULL;
        }
        else if (root->left != NULL)
        {
                return findMin(root->left);
        }
        return root;
}
struct BinaryTreeNode*
delete (struct BinaryTreeNode* root, int x)
{
        if (root == NULL)
                return NULL;
        if (x > root->key)
        {
                root->right = delete (root->right, x);
        }
        else if (x < root->key)
        {
                root->left = delete (root->left, x);
        }
        else
        {
                if (root->left == NULL && root->right == NULL)
                {
                        free(root);
                        return NULL;
                }
                else if (root->left == NULL || root->right == NULL)
                {
                        struct BinaryTreeNode* temp;
                        if (root->left == NULL)
                        {
                                temp = root->right;
                        }
                        else
                        {
                                temp = root->left;
                        }
                        free(root);
                        return temp;
                }
                else
                {
                        struct BinaryTreeNode* temp     = findMin(root->right);
                        root->key = temp->key;
                        root->right = delete (root->right, temp->key);
                }
        }
        return root;
}
int main()
{
        struct BinaryTreeNode* root = NULL;

        root = insertNode(root, 50);
        insertNode(root, 30);
        insertNode(root, 20);
        insertNode(root, 40);
        insertNode(root, 70);
        insertNode(root, 60);
        insertNode(root, 80);

        if (searchNode(root, 60) != NULL)
        {
           printf("60 found");
        }
        else
        {
          printf("60 not found");
        }

        printf("\n");
        postOrder(root);
        printf("\n");
        preOrder(root);
        printf("\n");
        inOrder(root);
        printf("\nDELETE NODE\n");

        struct BinaryTreeNode* temp = delete (root, 70);
        printf("After Delete: \n");
        inOrder(root);
        return 0;
}

//EXP11 - ii) BFS

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

#define MAX 100
void bfs(int adj[MAX][MAX], int V, int s)
{
    int q[MAX], front = 0, rear = 0;
    bool visited[MAX] = { false };
    visited[s] = true;
    q[rear++] = s;
    while (front < rear)
    {
     int curr = q[front++];
     printf("%d ", curr);

        for (int i = 0; i < V; i++)
        {
         if (adj[curr][i] == 1 && !visited[i])
          {
            visited[i] = true;
            q[rear++] = i;
          }
        }
    }
}
void addEdge(int adj[MAX][MAX], int u, int v)
{
    adj[u][v] = 1;
    adj[v][u] = 1;
}
int main()
{
    int V = 5;
    int adj[MAX][MAX] = {0};

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

    printf("BFS starting from 0:\n");
    bfs(adj, V, 0);

    return 0;
}


//EXP11  i)DFS

#include <stdio.h>
#include <stdlib.h>
struct Node
{
    int dest;
    struct Node* next;
};
struct AdjList
{
    struct Node* head;
};
struct Node* createNode(int dest)
{
 struct Node* newNode =(struct Node*)malloc(sizeof(struct Node));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}
void DFSRec(struct AdjList adj[], int visited[], int s)
{
    visited[s] = 1;
    printf("%d ", s);
    struct Node* current = adj[s].head;
    while (current != NULL)
    {
        int dest = current->dest;
        if (!visited[dest])
        {
          DFSRec(adj, visited, dest);
        }
        current = current->next;
    }
}
void DFS(struct AdjList adj[], int V, int s)
{
    int visited[5] = {0};
    DFSRec(adj, visited, s);
}
void addEdge(struct AdjList adj[], int s, int t)
{
    struct Node* newNode = createNode(t);
    newNode->next = adj[s].head;
    adj[s].head = newNode;
    newNode = createNode(s);
    newNode->next = adj[t].head;
    adj[t].head = newNode;
}
int main()
{
    int V = 5;
    struct AdjList adj[V];
    for (int i = 0; i < V; i++)
    {
        adj[i].head = NULL;
    }
    int E = 5;
    int edges[][2] = {{1, 2}, {1, 0}, {2, 0}, {2, 3}, {2, 4}};
    for (int i = 0; i < E; i++)
    {
        addEdge(adj, edges[i][0], edges[i][1]);
    }
    int source = 1;
    printf("DFS from source: %d\n", source);
    DFS(adj, V, source);
    return 0;
}
```