```c
// 1. Using linked list representation, create a binary search tree for:

// 50, 10, 20, 30, 5, 90, 80, 100, 85
// Write a menu-driven program for the functions to traverse the tree in preorder, postorder
// and inorder traversal.
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   int data;
   struct node *left, *right;
} Node;

Node *root = NULL;

Node *create(int data)
{
   Node *newNode = malloc(sizeof(Node));
   newNode->data = data;
   newNode->right = newNode->left = NULL;
   return newNode;
}

Node *insert(Node *root, int data)
{
   if (!root) return create(data);
   if (root->data >= data)
       root->left = insert(root->left, data);
   else
```

```
        root->right = insert(root->right, data);
    return root;
};

void preorder(Node *root)
{
    if (!root) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void inorder(Node *root)
{
    if (!root) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

void postorder(Node *root)
{
    if (!root) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

int main()
{
    int choice;
```

```
    int arr[] = {50, 10, 20, 30, 5, 90, 80, 100, 85};

    for (int i = 0; i < 9; i++)
    {
        root = insert(root, arr[i]);
    }

    do
    {
        printf("\n-----------------------");
        printf("\n1.Preorder?");
        printf("\n2.Inorder?");
        printf("\n3.Postorder?");
        printf("\n4.Exit?");

        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                preorder(root);
                break;
            case 2:
                inorder(root);
                break;
            case 3:
                postorder(root);
                break;
        }
    } while (choice != 4);
    return 0;
}
```

```c
// 2. In program 1, add the functions for search, insert and
delete an element from the tree.
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   int data;
   struct node *left, *right;
} Node;

Node *root = NULL;

Node *create(int data)
{
   Node *newNode = malloc(sizeof(Node));
   newNode->data = data;
   newNode->right = newNode->left = NULL;
   return newNode;
}

Node *insert(Node *root, int data)
{
   if (!root) return create(data);
   if (root->data >= data)
       root->left = insert(root->left, data);
   else
       root->right = insert(root->right, data);
   return root;
};
```

```
Node *search(Node *root, int key)
{
   if (!root || root->data == key) return root;
   if (root->data > key) return search(root->left, key);
   return search(root->right, key);
}

Node *minNode(Node *root)
{
   if (root->left == NULL) return root;

   return minNode(root->left);
}

Node *deleteNode(Node *root, int key)
{
   if (root == NULL) return root;

   if (key < root->data)
       root->left = deleteNode(root->left, key);
   else if (key > root->data)
       root->right = deleteNode(root->right, key);
   else
   {
       // Node found
       if (root->left == NULL)
       {
           Node *temp = root->right;
           free(root);
           return temp;
       }
       else if (root->right == NULL)
```

```
        {
            Node *temp = root->left;
            free(root);
            return temp;
        }
        // Node with two children
        Node *temp = minNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void preorder(Node *root)
{
    if (!root) return;
    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
}

void inorder(Node *root)
{
    if (!root) return;
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

void postorder(Node *root)
{
    if (!root) return;
```

```
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

int main()
{
    int choice;

    int arr[] = {50, 10, 20, 30, 5, 90, 80, 100, 85};

    for (int i = 0; i < 9; i++)
    {
        root = insert(root, arr[i]);
    }

    do
    {
        printf("\n------------------------");
        printf("\n1.Preorder?");
        printf("\n2.Inorder?");
        printf("\n3.Postorder?");
        printf("\n4.Minimum?");
        printf("\n5.Search?");
        printf("\n6.Delete?");
        printf("\n8.Exit?");

        scanf("%d", &choice);
        Node *node;
        int key;
        switch (choice)
        {
```

```
            case 1:
                preorder(root);
                break;
            case 2:
                inorder(root);
                break;
            case 3:
                postorder(root);
                break;
            case 4:
                node = minNode(root);
                if (node)
                    printf("Minimum value is %d", node->data);
                else
                    printf("Value not found!");
                break;
            case 5:

                printf("\nKey? ");
                scanf("%d", &key);
                node = search(root, key);
                if (node)
                    printf("Value found!");
                else
                    printf("Value not found!");
                break;
            case 6:

                printf("\nValue to delete: ");
                scanf("%d", &key);
                node = deleteNode(root, key);
                if (node)
```

```
                    printf("Node deleted");
                else
                    printf("Node not found!");
                break;
        }
    } while (choice != 8);
    return 0;
}

// 3. Using the tree created in program 1, write a function to
give the level for the user given
// element. (assuming root of the tree is at level 0)
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int data;
    struct node *left, *right;
} Node;

Node *root = NULL;

Node *create(int data)
{
    Node *newNode = malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

Node *insert(Node *root, int data)
```

```
{
    if (!root) return create(data);
    if (data <= root->data)
        root->left = insert(root->left, data);
    else
        root->right = insert(root->right, data);
    return root;
}

int findLevel(Node *root, int key, int level)
{
    if (!root) return -1;
    if (root->data == key) return level;
    if (key < root->data)
        return findLevel(root->left, key, level + 1);
    else
        return findLevel(root->right, key, level + 1);
}

int main()
{
    int arr[] = {50, 10, 20, 30, 5, 90, 80, 100, 85};
    for (int i = 0; i < 9; i++) root = insert(root, arr[i]);

    int key;
    printf("Enter element to find its level: ");
    scanf("%d", &key);

    int level = findLevel(root, key, 0);
    if (level != -1)
        printf("Element %d is at level %d\n", key, level);
    else
```

```
        printf("Element not found in BST\n");

    return 0;
}

// 4. Write a program to find the largest element in BST created
in program-1.
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int data;
    struct node *left, *right;
} Node;

Node *root = NULL;

Node *create(int data)
{
    Node *newNode = malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

Node *insert(Node *root, int data)
{
    if (!root) return create(data);
    if (data <= root->data)
        root->left = insert(root->left, data);
    else
```

```
        root->right = insert(root->right, data);
    return root;
}

int findLargest(Node *root)
{
    if (!root) return -1;
    while (root->right != NULL) root = root->right;
    return root->data;
}

int main()
{
    int arr[] = {50, 10, 20, 30, 5, 90, 80, 100, 85};
    for (int i = 0; i < 9; i++) root = insert(root, arr[i]);

    int largest = findLargest(root);
    printf("Largest element in BST is: %d\n", largest);

    return 0;
}

// 5. Create Heap tree for

// 50, 10, 20, 30, 5, 90, 80, 100, 85
// Perform the traversing on the Heap Tree.

#include <stdio.h>

#define MAX 100
int heap[MAX];
int size = 0;
```

```
void swap(int *a, int *b)
{
   int t = *a;
   *a = *b;
   *b = t;
}

void insertHeap(int value)
{
   heap[++size] = value;
   int i = size;
   while (i > 1 && heap[i / 2] < heap[i])
   {
       swap(&heap[i], &heap[i / 2]);
       i = i / 2;
   }
}

void inorderHeap(int i)
{
   if (i <= size)
   {
       inorderHeap(2 * i);
       printf("%d ", heap[i]);
       inorderHeap(2 * i + 1);
   }
}

void preorderHeap(int i)
{
   if (i <= size)
```

```
    {
        printf("%d ", heap[i]);
        preorderHeap(2 * i);
        preorderHeap(2 * i + 1);
    }
}

void postorderHeap(int i)
{
    if (i <= size)
    {
        postorderHeap(2 * i);
        postorderHeap(2 * i + 1);
        printf("%d ", heap[i]);
    }
}

int main()
{
    int arr[] = {50, 10, 20, 30, 5, 90, 80, 100, 85};
    int n = sizeof(arr) / sizeof(arr[0]);

    for (int i = 0; i < n; i++) insertHeap(arr[i]);

    printf("Inorder traversal of Heap: ");
    inorderHeap(1);
    printf("\n");

    printf("Preorder traversal of Heap: ");
    preorderHeap(1);
    printf("\n");
```

```
    printf("Postorder traversal of Heap: ");
    postorderHeap(1);
    printf("\n");

    return 0;
}
```