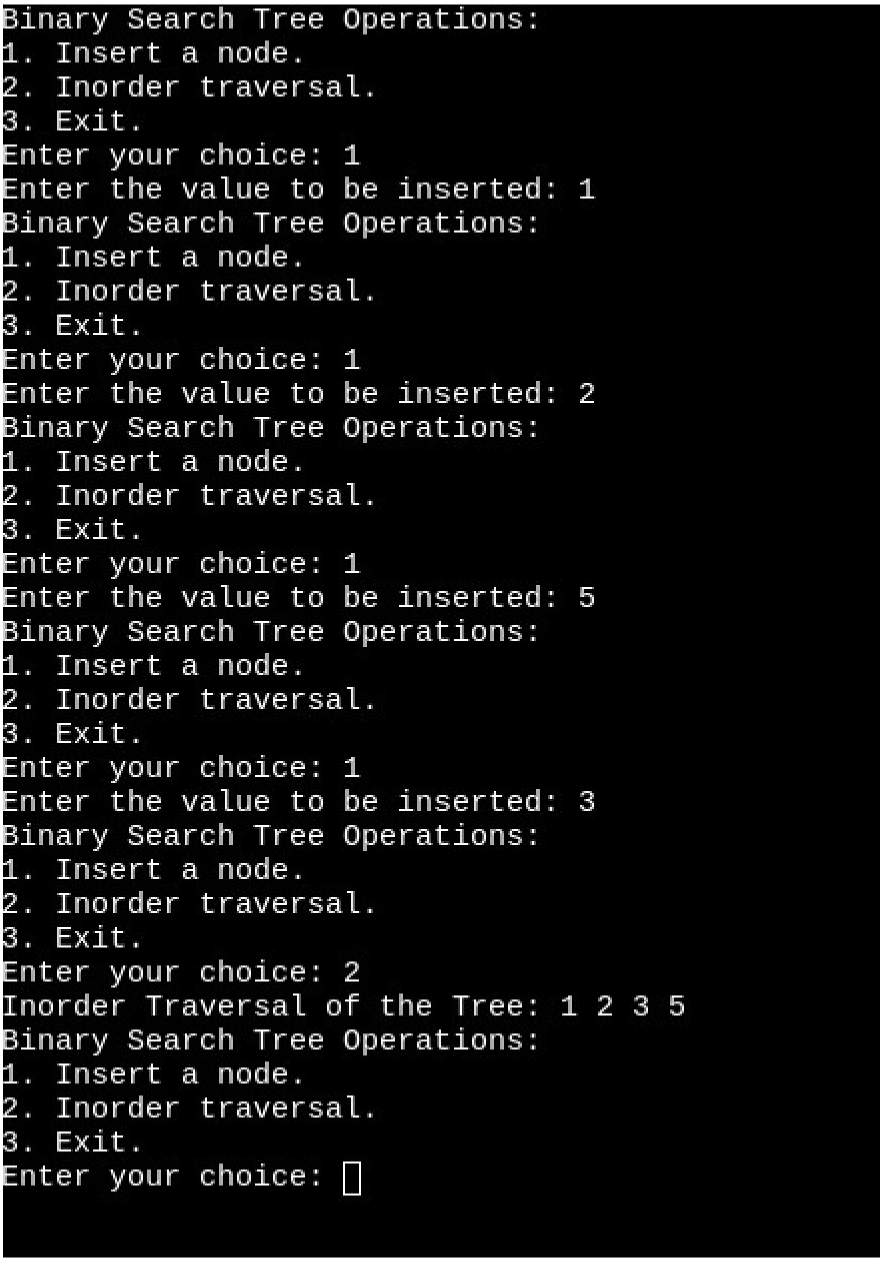
22AIE112 DATA STRUCTURES AND ALGORITHMS
LABSHEET 7
TREE

Date : 30/7/23 Roll No : AM.EN.U4AIE2209

1.

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

struct TreeNode {
    int value;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* createTreeNode(int value) {
    struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->value = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void insertTreeNode(struct TreeNode** root, int value) {
    if (*root == NULL) {
        *root = createTreeNode(value);
        return;
    }

    if (value < (*root)->value) {
        insertTreeNode(&((*root)->left), value);
    } else {
        insertTreeNode(&((*root)->right), value);
    }
}

void inorderTraversal(struct TreeNode* root) {
    if (root == NULL) {
        return;
    }

    inorderTraversal(root->left);
    printf("%d ", root->value);
    inorderTraversal(root->right);
}

int main() {
    struct TreeNode* root = NULL;

    while (1) {
        printf("Binary Search Tree Operations:\n");
        printf("1. Insert a node.\n");
        printf("2. Inorder traversal.\n");
        printf("3. Exit.\n");

        int choice;
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to be inserted: ");
                int value;
                scanf("%d", &value);
                insertTreeNode(&root, value);
                break;
            case 2:
                printf("Inorder Traversal of the Tree: ");
                inorderTraversal(root);
                printf("\n");
                break;

            case 3:
                exit(0);
                break;

            default:
                printf("Invalid choice. \n");
                break;
        }
    }

    return 0;
}
```

Output

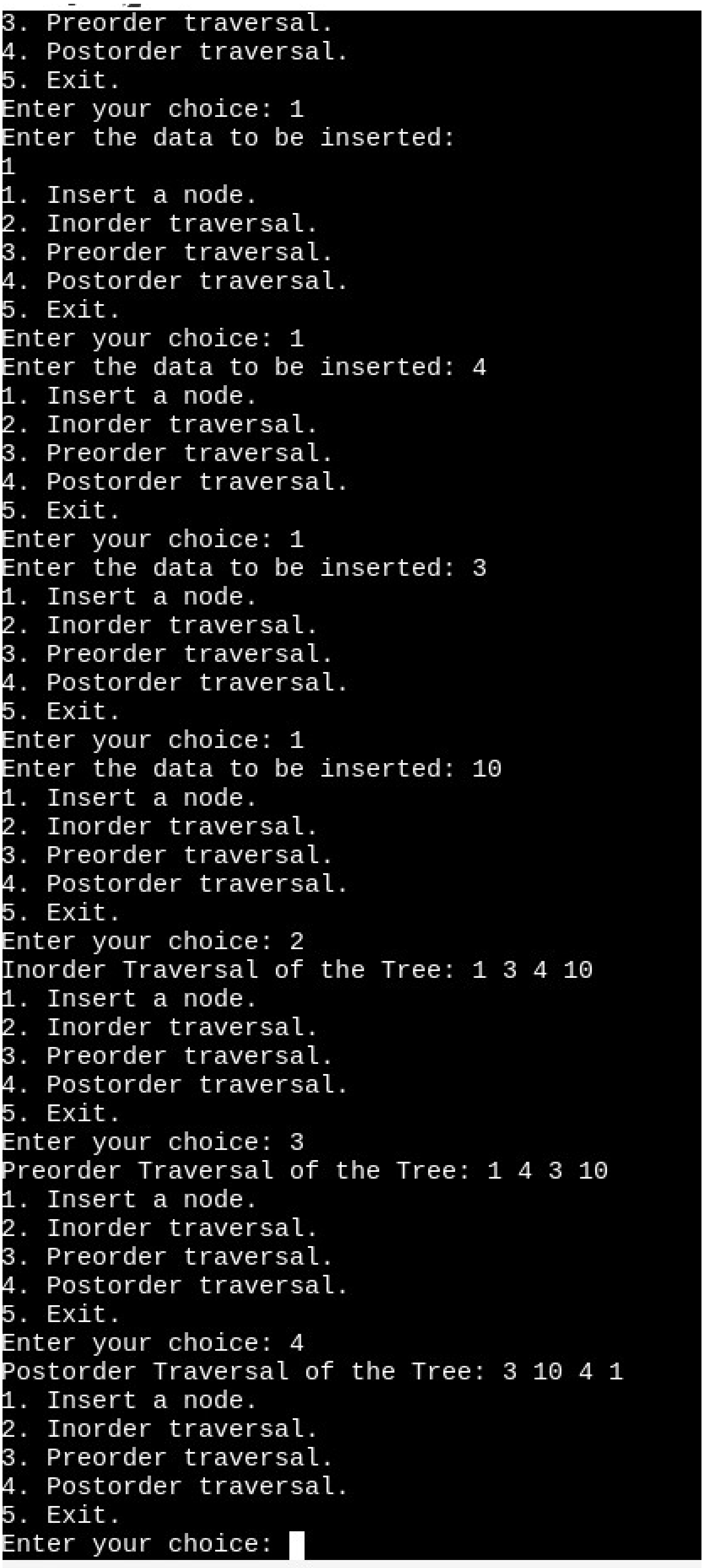


```
Binary Search Tree Operations:
1. Insert a node.
2. Inorder traversal.
3. Exit.
Enter your choice: 1
Enter the value to be inserted: 1
Binary Search Tree Operations:
1. Insert a node.
2. Inorder traversal.
3. Exit.
Enter your choice: 1
Enter the value to be inserted: 2
Binary Search Tree Operations:
1. Insert a node.
2. Inorder traversal.
3. Exit.
Enter your choice: 1
Enter the value to be inserted: 5
Binary Search Tree Operations:
1. Insert a node.
2. Inorder traversal.
3. Exit.
Enter your choice: 1
Enter the value to be inserted: 3
Binary Search Tree Operations:
1. Insert a node.
2. Inorder traversal.
3. Exit.
Enter your choice: 2
Inorder Traversal of the Tree: 1 2 3 5
Binary Search Tree Operations:
1. Insert a node.
2. Inorder traversal.
3. Exit.
Enter your choice: 
```

2.

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

struct node {
   int data;
   struct node* left;
   struct node* right;
};
```

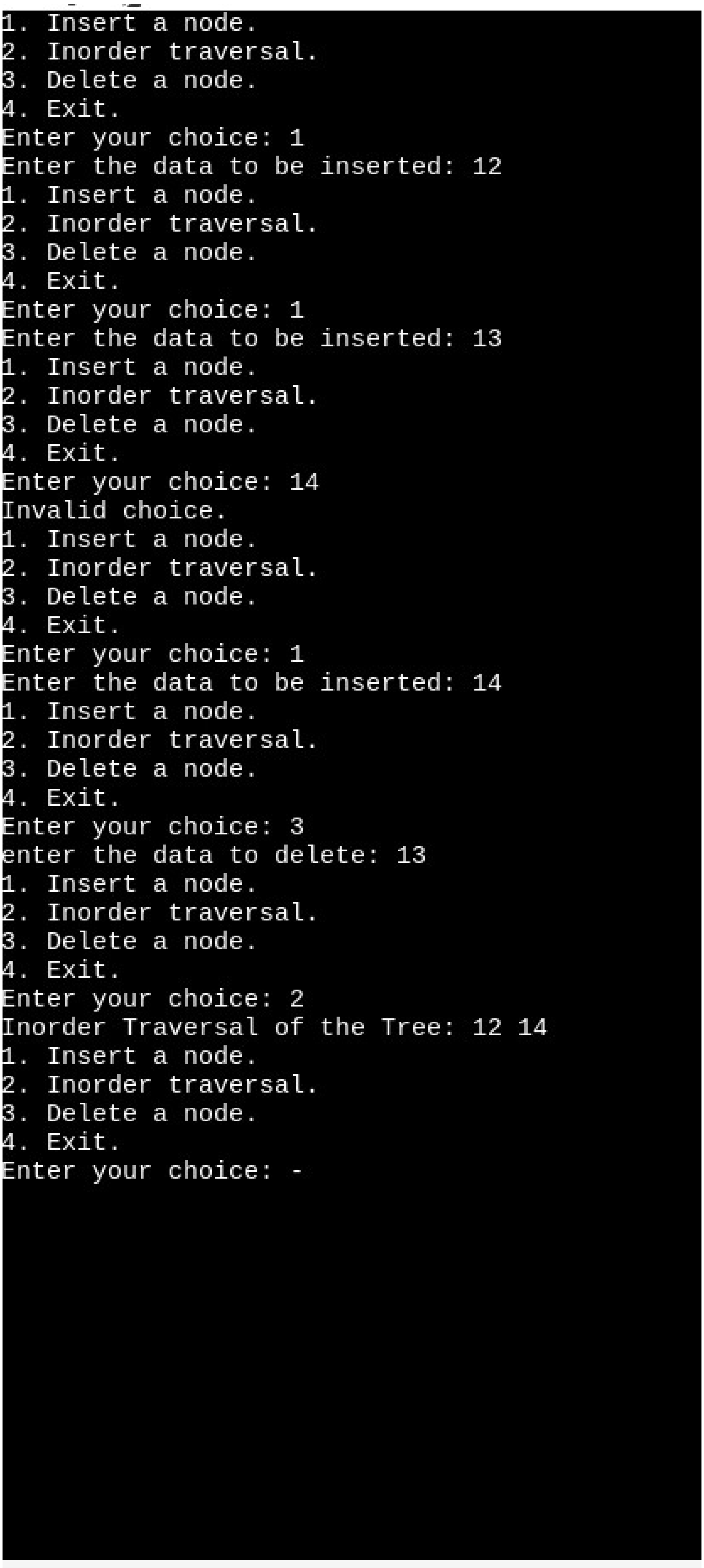
```
struct node* create_node(int data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

void insert_node(struct node** root, int data) {
    if (*root == NULL) {
        *root = create_node(data);
        return;
    }

    if (data < (*root)->data) {
        insert_node(&((*root)->left), data);
    } else {
        insert_node(&((*root)->right), data);
    }
}

void inorder(struct node* root) {
    if (root == NULL) {
        return;
    }

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

void preorder(struct node* root) {
    if (root == NULL) {
        return;
    }

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

void postorder(struct node* root) {
    if (root == NULL) {
        return;
    }

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

int main() {
    struct node* root = NULL;

    while (1) {
        printf("1. Insert a node.\n");
        printf("2. Inorder traversal.\n");
        printf("3. Preorder traversal.\n");
        printf("4. Postorder traversal.\n");
        printf("5. Exit.\n");

        int choice;
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter the data to be inserted: ");
            int data;
            scanf("%d", &data);
            insert_node(&root, data);
            break;

        case 2:
            printf("Inorder Traversal of the Tree: ");
            inorder(root);
            printf("\n");
            break;

        case 3:
            printf("Preorder Traversal of the Tree: ");
            preorder(root);
            printf("\n");
            break;

        case 4:
            printf("Postorder Traversal of the Tree: ");
            postorder(root);
            printf("\n");
            break;

        case 5:
            exit(0);
            break;

        default:
            printf("Invalid choice. \n");
            break;
        }
    }

    return 0;
}
```



```
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 1
Enter the data to be inserted:
1
1. Insert a node.
2. Inorder traversal.
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 1
Enter the data to be inserted: 4
1. Insert a node.
2. Inorder traversal.
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 1
Enter the data to be inserted: 3
1. Insert a node.
2. Inorder traversal.
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 1
Enter the data to be inserted: 10
1. Insert a node.
2. Inorder traversal.
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 2
Inorder Traversal of the Tree: 1 3 4 10
1. Insert a node.
2. Inorder traversal.
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 3
Preorder Traversal of the Tree: 1 4 3 10
1. Insert a node.
2. Inorder traversal.
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 4
Postorder Traversal of the Tree: 3 10 4 1
1. Insert a node.
2. Inorder traversal.
3. Preorder traversal.
4. Postorder traversal.
5. Exit.
Enter your choice: 
```

3.

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

struct node {
    int data;
    struct node* left;
    struct node* right;
};

struct node* create_node(int data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

void insert_node(struct node** root, int data) {
    if (*root == NULL) {
        *root = create_node(data);
        return;
    }

    if (data < (*root)->data) {
        insert_node(&((*root)->left), data);
    } else {
        insert_node(&((*root)->right), data);
    }
}

void inorder(struct node* root) {
    if (root == NULL) {
        return;
    }

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

struct node* find_min_node(struct node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}

struct node* delete_node(struct node* root, int data) {
    if (root == NULL) {
        return root;
    }

    if (data < root->data) {
        root->left = delete_node(root->left, data);
    } else if (data > root->data) {
        root->right = delete_node(root->right, data);
    } else {
        if (root->left == NULL) {
            struct node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct node* temp = root->left;
            free(root);
            return temp;
        }
        struct node* temp = find_min_node(root->right);
        root->data = temp->data;
        root->right = delete_node(root->right, temp->data);
    }

    return root;
}

int main() {
    struct node* root = NULL;

    while (1) {
        printf("1. Insert a node.\n");
        printf("2. Inorder traversal.\n");
        printf("3. Delete a node.\n");
        printf("4. Exit.\n");

        int choice;
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter the data to be inserted: ");
            int data;
            scanf("%d", &data);
            insert_node(&root, data);
            break;

        case 2:
            printf("Inorder Traversal of the Tree: ");
            inorder(root);
            printf("\n");
            break;
        case 3:
            printf("enter the data to delete: ");
            int val;
            scanf("%d", &val);
            delete_node(root, val);
            break;

        case 4:
            exit(0);
            break;

        default:
            printf("Invalid choice. \n");
            break;
        }
    }

    return 0;
}
```

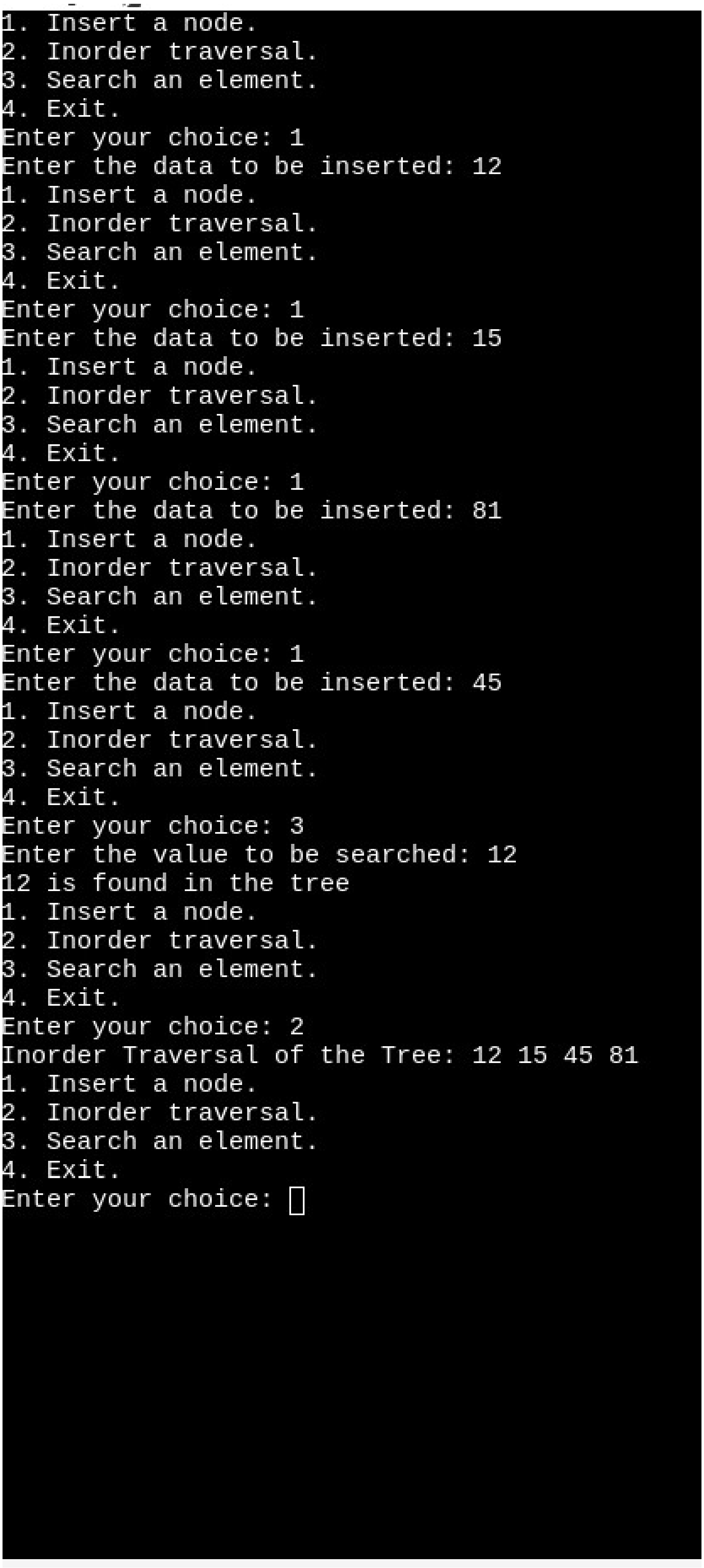


```
1. Insert a node.
2. Inorder traversal.
3. Delete a node.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 12
1. Insert a node.
2. Inorder traversal.
3. Delete a node.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 13
1. Insert a node.
2. Inorder traversal.
3. Delete a node.
4. Exit.
Enter your choice: 14
Invalid choice.
1. Insert a node.
2. Inorder traversal.
3. Delete a node.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 14
1. Insert a node.
2. Inorder traversal.
3. Delete a node.
4. Exit.
Enter your choice: 3
enter the data to delete: 13
1. Insert a node.
2. Inorder traversal.
3. Delete a node.
4. Exit.
Enter your choice: 2
Inorder Traversal of the Tree: 12 14
1. Insert a node.
2. Inorder traversal.
3. Delete a node.
4. Exit.
Enter your choice: -
```

4.

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

struct node {
    int data;
    struct node* left;
    struct node* right;
};

struct node* create_node(int data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;

    return new_node;
}

void insert_node(struct node** root, int data) {
    if (*root == NULL) {
        *root = create_node(data);
        return;
    }

    if (data < (*root)->data) {
        insert_node(&((*root)->left), data);
    } else {
        insert_node(&((*root)->right), data);
    }
}

void inorder(struct node* root) {
    if (root ==  NULL) {
        return;
    }

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

struct node* search_val(struct node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }

    if (data < root->data) {
        return search_val(root->left, data);
    } else {
        return search_val(root->right, data);
    }
}

int main() {
    struct node* root = NULL;

    while (1) {
        printf("1. Insert a node.\n");
        printf("2. Inorder traversal.\n");
        printf("3. Search an element.\n");
        printf("4. Exit.\n");

        int choice;
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the data to be inserted: ");
                int data;
                scanf("%d", &data);
                insert_node(&root, data);
                break;

            case 2:
                printf("Inorder Traversal of the Tree: ");
                inorder(root);
                printf("\n");
                break;

            case 3:
                printf("Enter the value to be searched: ");
                int val;
                scanf("%d", &val);
                struct node* result = search_val(root, val);
                if (result != NULL) {
                    printf("%d is found in the tree\n", val);
                } else {
                    printf("%d is not found in the tree\n", val);
                }
                break;

            case 4:
                exit(0);
                break;

            default:
                printf("Enter a valid choice!!!\n");
                break;
        }
    }

    return 0;
```

```
}
```

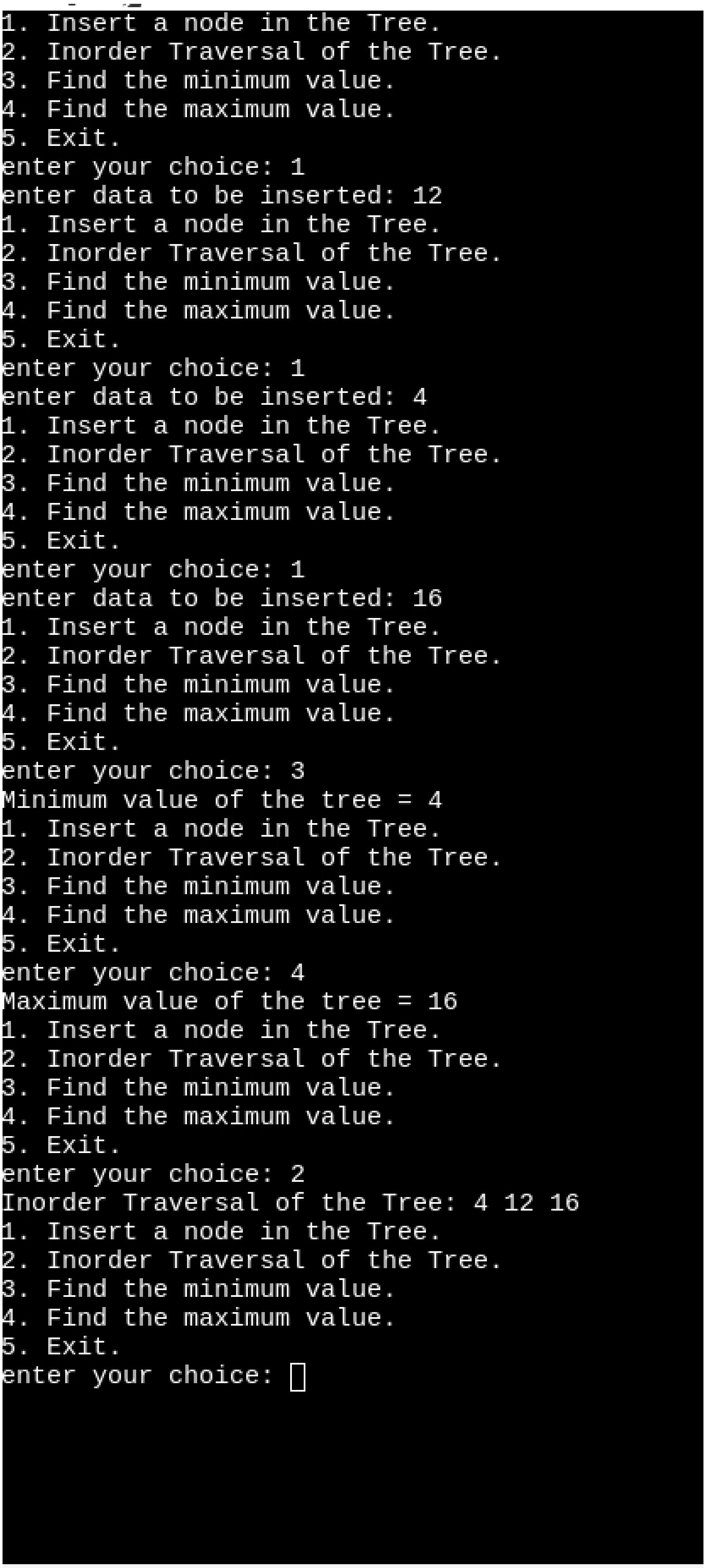


```
1. Insert a node.
2. Inorder traversal.
3. Search an element.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 12
1. Insert a node.
2. Inorder traversal.
3. Search an element.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 15
1. Insert a node.
2. Inorder traversal.
3. Search an element.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 81
1. Insert a node.
2. Inorder traversal.
3. Search an element.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 45
1. Insert a node.
2. Inorder traversal.
3. Search an element.
4. Exit.
Enter your choice: 3
Enter the value to be searched: 12
12 is found in the tree
1. Insert a node.
2. Inorder traversal.
3. Search an element.
4. Exit.
Enter your choice: 2
Inorder Traversal of the Tree: 12 15 45 81
1. Insert a node.
2. Inorder traversal.
3. Search an element.
4. Exit.
Enter your choice: 
```

5.

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

struct node{
   int data;
   struct node* left;
   struct node* right;
};

struct node* create_node(int data){
   struct node* new_node = (struct node*)malloc(sizeof(struct node));
   new_node->data = data;
   new_node->left = NULL;
   new_node->right = NULL;

   return new_node;
}

void insert_node(struct node **root, int data){
   if(*root == NULL){
      *root = create_node(data);
      return;
   }

   if((*root)->data > data){
      insert_node(&((*root)->left), data);
   } else{
      insert_node(&((*root)->right), data);
   }
}

void inorder(struct node *root){
    if(root == NULL){
        return;
    }

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

int find_min(struct node* root){
    if(root == NULL){
        printf("The Tree is empty!!");
        return -1;
    }

    while(root->left != NULL){
        root = root->left;
    }

    return root->data;
}

int find_max(struct node *root){
    if(root == NULL){
        printf("Tree is empty!!");
        return -1;
    }

    while(root->right != NULL){
        root = root->right;
    }

    return root->data;
}
```

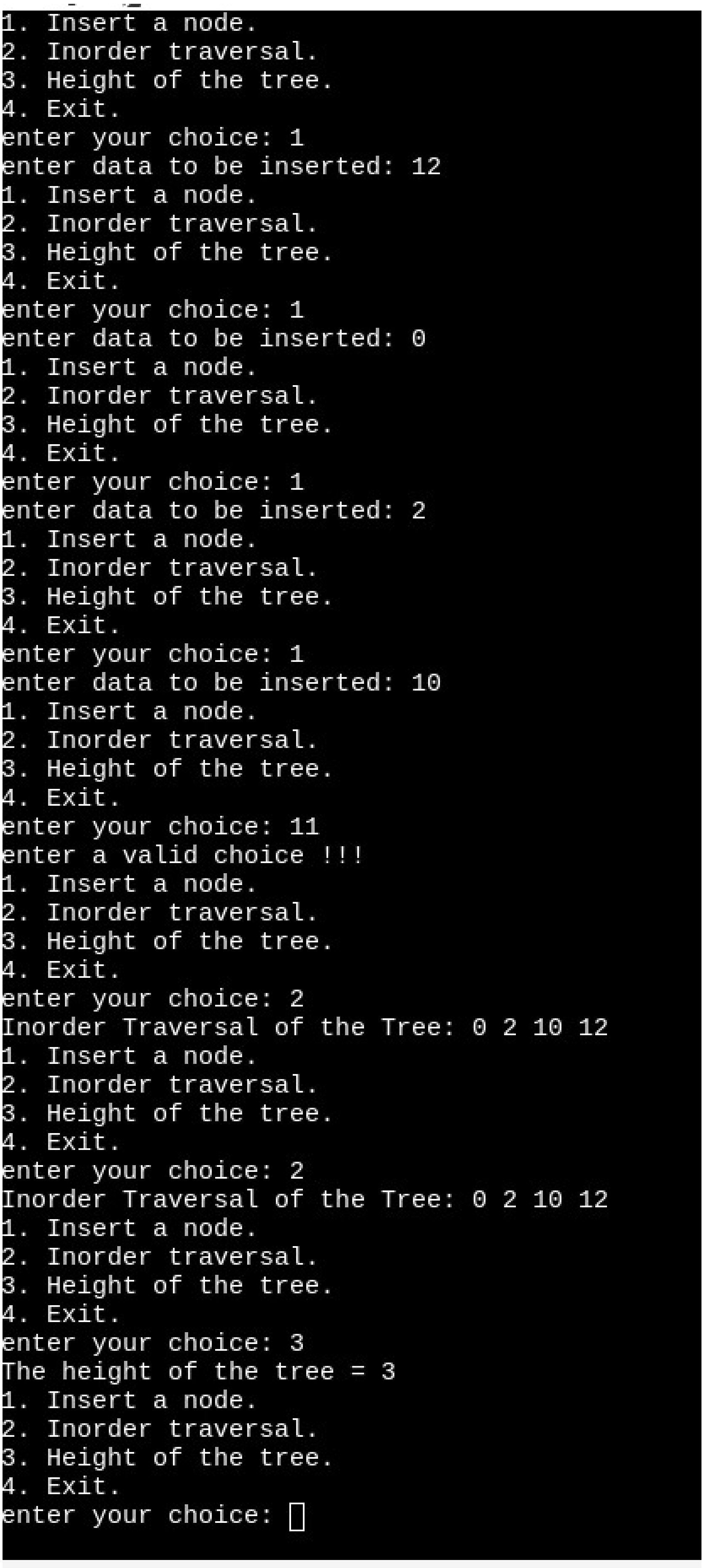
```
int main(){
    struct node *root = NULL;

    while(1){
        printf("1. Insert a node in the Tree. \n");
        printf("2. Inorder Traversal of the Tree. \n");
        printf("3. Find the minimum value. \n");
        printf("4. Find the maximum value. \n");
        printf("5. Exit. \n");

        int choice;
        printf("enter your choice: ");
        scanf("%d", &choice);

        switch(choice){
            case 1:
                printf("enter data to be inserted: ");
                int data;
                scanf("%d", &data);
                insert_node(&root, data);
                break;

            case 2:
                printf("Inorder Traversal of the Tree: ");
                inorder(root);
                printf("\n");
                break;

            case 3:
                {
                    int min_val = find_min(root);
                    if(min_val != -1){
                        printf("Minimum value of the tree = %d\n",
min_val);
                    }
                }
                break;

            case 4:
                {
                    int max_val = find_max(root);
                    if(max_val != -1){
                        printf("Maximum value of the tree = %d\n",
max_val);
                    }
                }
                break;

            case  5:
                exit(0);
                break;

            default:
                printf("enter a valid choice!!!\n");
                break;
        }
    }
}
```



```
1. Insert a node in the Tree.
2. Inorder Traversal of the Tree.
3. Find the minimum value.
4. Find the maximum value.
5. Exit.
enter your choice: 1
enter data to be inserted: 12
1. Insert a node in the Tree.
2. Inorder Traversal of the Tree.
3. Find the minimum value.
4. Find the maximum value.
5. Exit.
enter your choice: 1
enter data to be inserted: 4
1. Insert a node in the Tree.
2. Inorder Traversal of the Tree.
3. Find the minimum value.
4. Find the maximum value.
5. Exit.
enter your choice: 1
enter data to be inserted: 16
1. Insert a node in the Tree.
2. Inorder Traversal of the Tree.
3. Find the minimum value.
4. Find the maximum value.
5. Exit.
enter your choice: 3
Minimum value of the tree = 4
1. Insert a node in the Tree.
2. Inorder Traversal of the Tree.
3. Find the minimum value.
4. Find the maximum value.
5. Exit.
enter your choice: 4
Maximum value of the tree = 16
1. Insert a node in the Tree.
2. Inorder Traversal of the Tree.
3. Find the minimum value.
4. Find the maximum value.
5. Exit.
enter your choice: 2
Inorder Traversal of the Tree: 4 12 16
1. Insert a node in the Tree.
2. Inorder Traversal of the Tree.
3. Find the minimum value.
4. Find the maximum value.
5. Exit.
enter your choice: 
```

6.

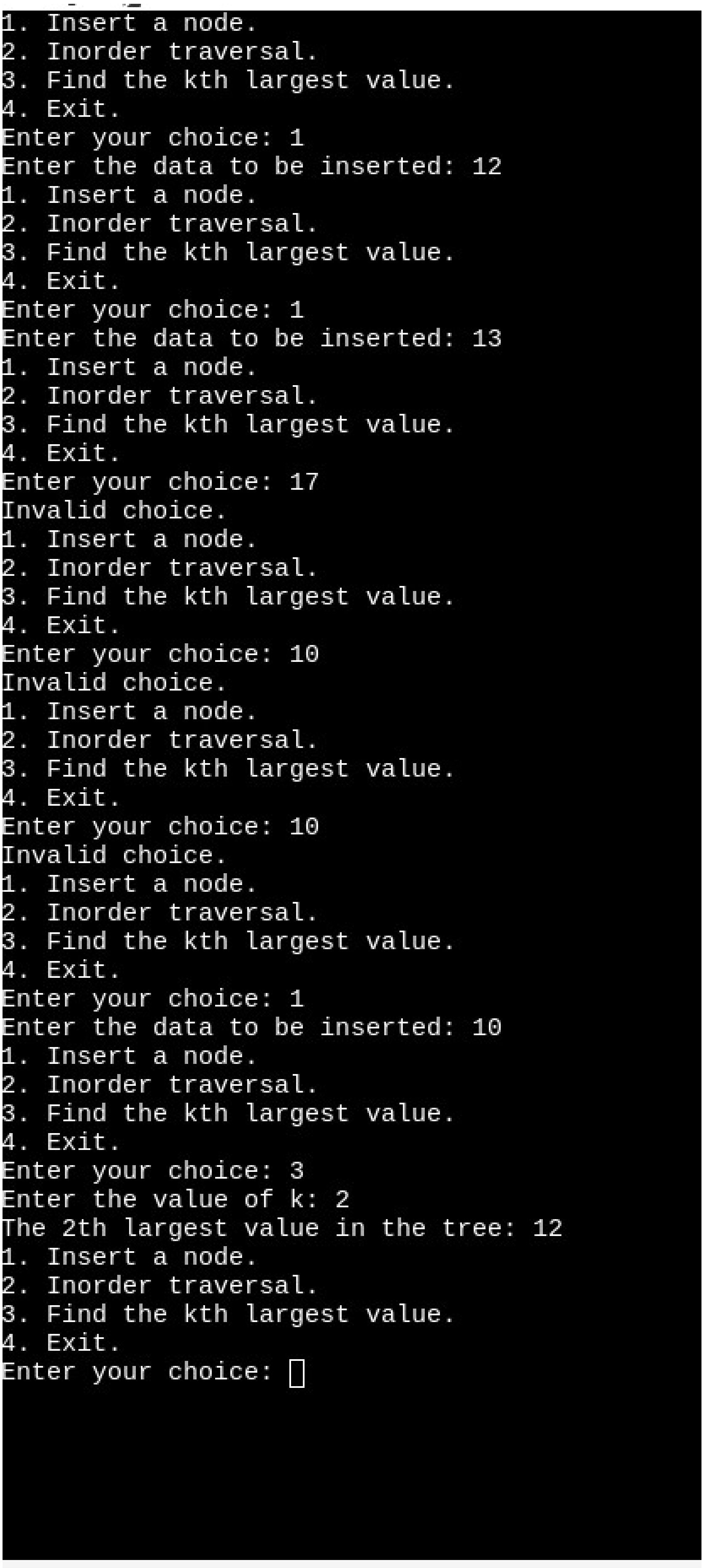
```
#include <stdio.h>
#include <stdlib.h>

struct node {
    int data;
    struct node* left;
    struct node* right;
};

struct node* create_node(int data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;

    return new_node;
}

void insert_node(struct node** root, int data) {
    if (*root == NULL) {
        *root = create_node(data);
        return;
    }

    if (data < (*root)->data) {
        insert_node(&((*root)->left), data);
    } else {
        insert_node(&((*root)->right), data);
    }
}

void inorder(struct node* root) {
    if (root == NULL) {
        return;
    }

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

int tree_height(struct node* root) {
    if (root == NULL) {
        return -1;
    }

    int left_height = tree_height(root->left);
    int right_height = tree_height(root->right);

    return (left_height > right_height) ? left_height + 1 :
right_height + 1;
}

int main() {
    struct node* root = NULL;

    while (1) {
        printf("1. Insert a node. \n");
        printf("2. Inorder traversal. \n");
        printf("3. Height of the tree. \n");
        printf("4. Exit. \n");

        int choice;
        printf("enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("enter data to be inserted: ");
                int data;
                scanf("%d", &data);
                insert_node(&root, data);
                break;

            case 2:
                printf("Inorder Traversal of the Tree: ");
                inorder(root);
                printf("\n");
                break;

            case 3:
                {
                    int height = tree_height(root);
                    printf("The height of the tree = %d\n", height);
                }
                break;

            case 4:
                exit(0);
                break;

            default:
                printf("enter a valid choice !!!\n");
                break;
        }
    }
    return 0;
}
```



```
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 1
enter data to be inserted: 12
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 1
enter data to be inserted: 0
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 1
enter data to be inserted: 2
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 1
enter data to be inserted: 10
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 11
enter a valid choice !!!
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 2
Inorder Traversal of the Tree: 0 2 10 12
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 2
Inorder Traversal of the Tree: 0 2 10 12
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 3
The height of the tree = 3
1. Insert a node.
2. Inorder traversal.
3. Height of the tree.
4. Exit.
enter your choice: 
```

7.

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

struct node {
    int data;
    struct node* left;
    struct node* right;
};

struct node* create_node(int data) {
    struct node* new_node = (struct node*)malloc(sizeof(struct
node));
    new_node->data = data;
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

void insert_node(struct node** root, int data) {
    if (*root == NULL) {
        *root = create_node(data);
        return;
    }

    if (data < (*root)->data) {
        insert_node(&((*root)->left), data);
    } else {
        insert_node(&((*root)->right), data);
    }
}

void inorder(struct node* root) {
    if (root == NULL) {
        return;
    }

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

void reverse_inorder(struct node* root, int k, int* count, int*
result) {
    if (root == NULL) {
        return;
    }

    reverse_inorder(root->right, k, count, result);

    (*count)++;
    if (*count == k) {
        *result = root->data;
        return;
    }

    reverse_inorder(root->left, k, count, result);
}

int find_kth_largeval(struct node* root, int k) {
    int count = 0;
    int result = -1; // This will hold the kth largest value

    reverse_inorder(root, k, &count, &result);

    return result;
}

int main() {
    struct node* root = NULL;

    while (1) {
        printf("1. Insert a node.\n");
        printf("2. Inorder traversal.\n");
        printf("3. Find the kth largest value. \n");
        printf("4. Exit.\n");

        int choice;
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the data to be inserted: ");
                int data;
                scanf("%d", &data);
                insert_node(&root, data);
                break;

            case 2:
                printf("Inorder Traversal of the Tree: ");
                inorder(root);
                printf("\n");
                break;

            case 3:
                {
                    int k;
                    printf("Enter the value of k: ");
                    scanf("%d", &k);
                    int kth_largest = find_kth_largeval(root, k);
                    printf("The %dth largest value in the tree: %d\n", 
k, kth_largest);
                }
            break;

            case 4:
                exit(0);
                break;

            default:
                printf("Invalid choice. \n");
                break;
            }
    }

    return 0;
}
```



```
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 12
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 13
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 17
Invalid choice.
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 10
Invalid choice.
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 10
Invalid choice.
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 1
Enter the data to be inserted: 10
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 3
Enter the value of k: 2
The 2th largest value in the tree: 12
1. Insert a node.
2. Inorder traversal.
3. Find the kth largest value.
4. Exit.
Enter your choice: 
```