Week-8

(a) To Implement Inorder, preorder, postorder Traversal

Code:
```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *left, *right;
};

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

struct Node * insert (struct Node *root, int value) {
    if (root == NULL) return createNode(value);
    if (value < root -> data) root -> left = insert(root -> left, value);
    else if (value > root -> data) root -> right = insert(root -> right, value);
    return root;
}

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);
}
```

```c
void postorder(struct node *root)
    if(root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->data);
}

void display(struct node *root){
    printf("BST elements (inorder): ");
    inorder(root);
    printf("\n");
}

int main() {
    struct node *root = NULL;
    int choice, value;
    while(1) {
        printf("\n--- Binary Search Tree Menu ---");
        printf("1. Insert into BST\n");
        printf("2. Inorder Traversal\n");
        printf("3. Preorder Traversal\n");
        printf("4. Postorder Traversal\n");
        printf("5. Display BST\n");
        printf("6. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;
            case 2:
                printf("Inorder traversal: ");
                inorder(root);
                printf("\n");
                break;
```

```c
            case 3:
                printf("Preorder traversal: ");
                preorder(root);
                printf("\n");
                break;
            case 4:
                printf("Postorder traversal: ");
                postorder(root);
                printf("\n");
                break;
            case 5:
                display(root);
                break;
            case 6:
                exit(0);
            }
                printf("Continue ...");
        }
}
```

o/p
```
--- Binary Search tree Menu ---
1.  Insert into BST
2.  Inorder Traversal
3.  Preorder Traversal
4.  Postorder Traversal
5.  Display BST
6.  Exit
Enter choice: 1
Enter value to insert: 23

Enter choice: 1
Enter value to insert: 45

Enter choice: 1
Enter value to insert: 567

Enter choice: 2
Inorder Traversal: 23 45 567

Enter choice: 4
Postorder Traversal: 567 45 23

Enter choice: 3
preorder Traversal: 23 45 567
```

Enter choice : 5

Nbr (element) : 22   45   567

Enter choice : 6

    Exiting. . . .

Leet code : Linked list cycle

class   Solution {

public :

    bool has_cycle (ListNode * head) {
        if (! head || !head → next) return false;

        ListNode * slow = head
        ListNode ** fast = head
        while (fast && fast → next) {
            slow = slow → next
            fast = fast → nen → next;
                            if (slow == fast) return true;

        };

    . return false;

    }
}

ϕ1 22/2/25

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

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

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

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
    } else if (data > root->data) {
        root->right = insert(root->right, data);
    }
    return root;
}

void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
```

```c
34          inorderTraversal(root->right);
35      }
36  }
37
38  void preorderTraversal(struct Node* root) {
39      if (root != NULL) {
40          printf("%d ", root->data);
41          preorderTraversal(root->left);
42          preorderTraversal(root->right);
43      }
44  }
45
46  void postorderTraversal(struct Node* root) {
47      if (root != NULL) {
48          postorderTraversal(root->left);
49          postorderTraversal(root->right);
50          printf("%d ", root->data);
51      }
52  }
53
54  void displayBST(struct Node* root) {
55      if (root == NULL) {
56          printf("BST is empty.\n");
57          return;
58      }
59      printf("Inorder Traversal: ");
60      inorderTraversal(root);
61      printf("\n");
62      printf("Preorder Traversal: ");
63      preorderTraversal(root);
64      printf("\n");
```

```c
        printf("Postorder Traversal: ");
        postorderTraversal(root);
        printf("\n");
}

int main() {
    struct Node* root = NULL;
    int choice, data;

    while (1) {
        printf("\nBinary Search Tree Operations:\n");
        printf("1. Insert a node\n");
        printf("2. Inorder Traversal\n");
        printf("3. Preorder Traversal\n");
        printf("4. Postorder Traversal\n");
        printf("5. Display BST\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &data);
                root = insert(root, data);
                printf("Node inserted successfully.\n");
                break;
            case 2:
                printf("Inorder Traversal: ");
                inorderTraversal(root);
                printf("\n");
                break;
```

```
97              case 3:
98                  printf("Preorder Traversal: ");
99                  preorderTraversal(root);
00                  printf("\n");
01                  break;
02              case 4:
03                  printf("Postorder Traversal: ");
04                  postorderTraversal(root);
05                  printf("\n");
06                  break;
07              case 5:
08                  displayBST(root);
09                  break;
10              case 6:
11                  printf("Exiting...\n");
12                  exit(0);
13              default:
14                  printf("Invalid choice. Please try again.\n");
15          }
16      }
17
18      return 0;
19  }
20
```

```
Enter your choice: 1
Enter data to insert: 56
Node inserted successfully.

Binary Search Tree Operations:
1. Insert a node
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 1
Enter data to insert: 40
Node inserted successfully.

Binary Search Tree Operations:
1. Insert a node
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 5
Inorder Traversal: 3 10 26 40 56
Preorder Traversal: 10 3 26 56 40
Postorder Traversal: 3 40 56 26 10

Binary Search Tree Operations:
1. Insert a node
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 6
Exiting...
```

```
Binary Search Tree Operations:
1. Insert a node
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 1
Enter data to insert: 10
Node inserted successfully.

Binary Search Tree Operations:
1. Insert a node
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 1
Enter data to insert: 26
Node inserted successfully.

Binary Search Tree Operations:
1. Insert a node
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
5. Display BST
6. Exit
Enter your choice: 1
Enter data to insert: 3
Node inserted successfully.

Binary Search Tree Operations:
1. Insert a node
2. Inorder Traversal
3. Preorder Traversal
4. Postorder Traversal
```