DATA STRUCTURE

DAY-5

1.Binary tree

Program:

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct Node {
   int data;
   struct Node* left;
   struct Node* right;
} Node;
 Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;
   return newNode;
}
 Node* insert(Node* root, int data) {
   if (root == NULL) {
      return createNode(data);
   }
    Node* queue[100];
```

```
    int front = 0, rear = 0;
    queue[rear++] = root;

    while (front < rear) {
        Node* current = queue[front++];
        if (current->left == NULL) {
            current->left = createNode(data);
            return root;
        } else {
            queue[rear++] = current->left;
        }
        if (current->right == NULL) {
            current->right = createNode(data);
            return root;
        } else {
            queue[rear++] = current->right;
        }
    }
    return root;
}
 void inOrderTraversal(Node* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
 void preOrderTraversal(Node* root) {
```

```
    if (root != NULL) {
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}
 void postOrderTraversal(Node* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}
 void freeTree(Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
int main() {
    Node* root = NULL;
       root = insert(root, 1);
    insert(root, 2);
    insert(root, 3);
    insert(root, 4);
    insert(root, 5);
    insert(root, 6);
```

```
    insert(root, 7);
     printf("In-order traversal: ");
    inOrderTraversal(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preOrderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
    postOrderTraversal(root);
    printf("\n");
     freeTree(root);
    return 0;
}
```

Output:

```
In-order traversal: 4 2 5 1 6 3 7
Pre-order traversal: 1 2 4 5 3 6 7
Post-order traversal: 4 5 2 6 7 3 1
```

2.Binary search tree

Program:

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;
 Node* createNode(int data) {
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
Node* insert(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;
}
 Node* search(Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return search(root->left, data);
    } else {
```

```
        return search(root->right, data);
    }
}
 Node* findMin(Node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}
 Node* deleteNode(Node* root, int data) {
    if (root == NULL) {
        return NULL;
    }
    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        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* temp = findMin(root->right);
```

```
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
void inOrderTraversal(Node* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}

int main() {
    Node* root = NULL;
     root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);
     printf("In-order traversal: ");
    inOrderTraversal(root);
    printf("\n");
     int value = 60;
    Node* searchResult = search(root, value);
    if (searchResult != NULL) {
```

```
    printf("Value %d found in the BST.\n", value);
  } else {
    printf("Value %d not found in the BST.\n", value);
  }
   root = deleteNode(root, 20);
  printf("In-order traversal after deleting 20: ");
  inOrderTraversal(root);
  printf("\n");
  return 0;
}
```

Output:

```
In-order traversal: 20 30 40 50 60 70 80
Value 60 found in the BST.
In-order traversal after deleting 20: 30 40 50 60 70 80
```

3.Binary tree Traversal

Program:

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct Node {
  int data;
  struct Node* left;
  struct Node* right;
} Node;
 Node* createNode(int data) {
  Node* newNode = (Node*)malloc(sizeof(Node));
  if (newNode == NULL) {
    printf("Memory allocation failed\n");
```

```
        exit(1);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
 void inOrderTraversal(Node* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
void preOrderTraversal(Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}
 void postOrderTraversal(Node* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}
```

```
int main() {
    Node* root = createNode(1);
  root->left = createNode(2);
  root->right = createNode(3);
  root->left->left = createNode(4);
  root->left->right = createNode(5);
  root->right->left = createNode(6);
  root->right->right = createNode(7);
   printf("In-order traversal: ");
  inOrderTraversal(root);
  printf("\n");
  printf("Pre-order traversal: ");
  preOrderTraversal(root);
  printf("\n");
  printf("Post-order traversal: ");
  postOrderTraversal(root);
  printf("\n");
  return 0;
}
```

Output:

In-order traversal: 4 2 5 1 6 3 7

Pre-order traversal: 1 2 4 5 3 6 7

Post-order traversal: 4 5 2 6 7 3 1