**Singly Linked List**

#include<stdio.h>

#include<stdlib.h>

struct Node{

    int data;

    struct Node \*next;

};

struct Node \*head = NULL;

void insertFront(int data){

    struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode -> data = data;

    newNode -> next = head;

    head = newNode;

}

void insertEnd(int data){

    struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode -> data = data;

    newNode -> next = NULL;

    if (head == NULL){

        head = newNode;

    }

    else{

        struct Node \*temp = head;

        while(temp -> next != NULL){

            temp = temp -> next;

        }

        temp -> next = newNode;

    }

}

void insertAfter(int data, int position){

    struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode -> data = data;

    struct Node \*temp = head;

    while(temp -> data != position){

        temp = temp -> next;

    }

    newNode -> next = temp -> next;

    temp -> next = newNode;

}

void deleteFront(){

    if (head == NULL){

        printf("List is empty\n");

    }

    struct Node \*temp = head;

    head = head -> next;

    free(temp);

}

void deleteEnd(){

    if(head == NULL){

        printf("List is empty\n");

    }

    struct Node \*temp = head;

    struct Node \*prev = NULL;

    while(temp -> next != NULL){

        prev = temp;

        temp = temp -> next;

    }

    prev -> next = NULL;

    free(temp);

}

void deleteAfter(int position){

    struct Node \*temp = head;

    struct Node \*prev = NULL;

    while(temp -> data != position){

        prev = temp;

        temp = temp -> next;

    }

    prev -> next = temp -> next;

    free(temp);

}

void display(){

    struct Node \*temp = head;

    while(temp != NULL){

        printf("%d\t", temp -> data);

        temp = temp -> next;

    }

}

int main(){

    int t = -1, value, position;

    while(t != 6){

        printf("\n1. Insert at front\n2. Insert at end\n3. Insert after a node\n4. Delete from front\n5. Delete from end\n6. Delete after a node\n7. Display\n8. Exit\n");

        scanf("%d", &t);

        switch(t){

            case 1:

                printf("Enter data: ");

                scanf("%d", &value);

                insertFront(value);

                break;

            case 2:

                printf("Enter data: ");

                scanf("%d", &value);

                insertEnd(value);

                break;

            case 3:

                printf("Enter data: ");

                scanf("%d", &value);

                printf("Enter position: ");

                scanf("%d", &position);

                insertAfter(value, position);

                break;

            case 4:

                deleteFront();

                break;

            case 5:

                deleteEnd();

                break;

            case 6:

                printf("Enter position: ");

                scanf("%d", &position);

                deleteAfter(position);

                break;

            case 7:

                display();

                break;

            case 8:

                exit(0);

                break;

            default:

                printf("Invalid input\n");

        }

    }

}

**Doubly Linked List**

#include<stdio.h>

#include<stdlib.h>

struct Node{

  int data;

  struct Node \*next;

  struct Node \*prev;

};

struct Node \*head = NULL;

void insertFront(int data){

  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

  newNode -> data = data;

  newNode -> next = head;

  newNode -> prev = NULL;

  if(head != NULL){

    head -> prev = newNode;

  }

  head = newNode;

}

void insertEnd(int data){

  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

  newNode -> data = data;

  newNode -> next = NULL;

  if (head == NULL){

    newNode -> prev = NULL;

    head = newNode;

  }else{

    struct Node \*temp = head;

    while(temp -> next != NULL){

      temp = temp -> next;

    }

    temp -> next = newNode;

    newNode -> prev = temp;

  }

}

void insertAfter(int data, int position){

  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

  newNode -> data = data;

  struct Node \*temp = head;

  while(temp -> data != position){

    temp = temp -> next;

  }

  newNode -> next = temp -> next;

  temp -> next = newNode;

  newNode -> prev = temp;

  if(newNode -> next != NULL){

    newNode -> next -> prev = newNode;

  }

}

void deleteFront(){

  if (head == NULL){

    printf("List is empty\n");

  }

  struct Node \*temp = head;

  head = head -> next;

  head -> prev = NULL;

  free(temp);

}

void deleteEnd(){

  if(head == NULL){

    printf("List is empty\n");

  }

  struct Node \*temp = head;

  struct Node \*prev = NULL;

  while(temp -> next != NULL){

    prev = temp;

    temp = temp -> next;

  }

  prev -> next = NULL;

  free(temp);

}

void deleteAfter(int position){

  struct Node \*temp = head;

  struct Node \*prev = NULL;

  while(temp -> data != position){

    prev = temp;

    temp = temp -> next;

  }

  prev -> next = temp -> next;

  temp -> next -> prev = prev;

  free(temp);

}

void display(){

  struct Node \*temp = head;

  while(temp != NULL){

    printf("%d\t", temp -> data);

    temp = temp -> next;

  }

}

int main(){

    int t = -1, value, position;

    while(t != 6){

        printf("\n1. Insert at front\n2. Insert at end\n3. Insert after a node\n4. Delete from front\n5. Delete from end\n6. Delete after a node\n7. Display\n8. Exit\n");

        scanf("%d", &t);

        switch(t){

            case 1:

                printf("Enter data: ");

                scanf("%d", &value);

                insertFront(value);

                break;

            case 2:

                printf("Enter data: ");

                scanf("%d", &value);

                insertEnd(value);

                break;

            case 3:

                printf("Enter data: ");

                scanf("%d", &value);

                printf("Enter position: ");

                scanf("%d", &position);

                insertAfter(value, position);

                break;

            case 4:

                deleteFront();

                break;

            case 5:

                deleteEnd();

                break;

            case 6:

                printf("Enter position: ");

                scanf("%d", &position);

                deleteAfter(position);

                break;

            case 7:

                display();

                break;

            case 8:

                exit(0);

                break;

            default:

                printf("Invalid input\n");

        }

    }

}

**Circular Linked List**

#include<stdio.h>

#include<stdlib.h>

struct Node{

  int data;

  struct Node \*next;

};

struct Node \*head = NULL;

void insertFront(int data){

  struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

  newNode -> data = data;

  newNode -> next = head;

  head = newNode;

}

void insertEnd(int data){

  struct Node \*newNode = (struct Node\*)malloc(sizeof(struct Node));

  newNode -> data = data;

  newNode -> next = NULL;

  if (head == NULL){

    head = newNode;

    newNode -> next = head;

  }

  else{

    struct Node \*temp = head;

    while(temp -> next != NULL){

      temp = temp -> next;

    }

    temp -> next = newNode;

  }

}

void deleteFront(){

  if (head == NULL){

    printf("List is empty\n");

    return;

    }

    struct Node \*temp = head;

    if (head->next == head){

      head = NULL;

      free(temp);

    }

    struct Node \*last = head;

    while (last->next != head) {

      last = last->next;

      }

    last->next = head->next;

    head = last->next;

    free(temp);

}

void display(){

  struct Node \*temp = head;

  if (head == NULL){

    printf("List is empty\n");

    return;

  }

  do{

    printf("%d\t", temp -> data);

    temp = temp -> next;

  }while(temp != head);

}

int main(){

  int t = -1, value;

  while(t != 4){

    printf("\n1. Insert at front\n2. Insert at end\n3. Delete from front\n4. Display\n5. Exit\n");

    scanf("%d", &t);

    switch(t){

      case 1:

        printf("Enter data: ");

        scanf("%d", &value);

        insertFront(value);

        break;

      case 2:

        printf("Enter data: ");

        scanf("%d", &value);

        insertEnd(value);

        break;

      case 3:

        deleteFront();

        break;

      case 4:

        display();

        break;

      case 5:

        exit(0);

      default:

        printf("Invalid choice\n");

    }

  }

}

**Circular Doubly Linked List**

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node \*next;

    struct Node \*prev;

};

struct Node \*head = NULL;

void insertFront(int data) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = head;

    newNode->prev = NULL;

    if (head != NULL) {

        head->prev = newNode;

    }

    head = newNode;

    if (head->next == NULL) {

        head->next = head;

        head->prev = head;

    }

}

void insertEnd(int data) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = newNode;

    newNode->prev = newNode;

    if (head == NULL) {

        head = newNode;

    } else {

        struct Node \*tail = head->prev;

        tail->next = newNode;

        newNode->prev = tail;

        newNode->next = head;

        head->prev = newNode;

    }

}

void deleteFront() {

    if (head == NULL) {

        printf("List is empty\n");

        return;

    }

    if (head->next == head) {

        free(head);

        head = NULL;

    } else {

        struct Node \*temp = head;

        struct Node \*last = head->prev;

        head = head->next;

        head->prev = last;

        last->next = head;

        free(temp);

    }

}

void deleteEnd() {

    if (head == NULL) {

        printf("List is empty\n");

        return;

    }

    if (head->next == head) {

        free(head);

        head = NULL;

    } else {

        struct Node \*temp = head->prev;

        struct Node \*prev = temp->prev;

        prev->next = head;

        head->prev = prev;

        free(temp);

    }

}

void display() {

    struct Node \*temp = head;

    if (head == NULL) {

        printf("List is empty\n");

        return;

    }

    do {

        printf("%d\t", temp->data);

        temp = temp->next;

    } while (temp != head);

    printf("\n");

}

int main() {

    int t = -1, value;

    while (t != 6) {

        printf("\n1. Insert at front\n2. Insert at end\n3. Delete from front\n4. Delete from end\n5. Display\n6. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &t);

        switch (t) {

            case 1:

                printf("Enter the value to insert: ");

                scanf("%d", &value);

                insertFront(value);

                break;

            case 2:

                printf("Enter the value to insert: ");

                scanf("%d", &value);

                insertEnd(value);

                break;

            case 3:

                deleteFront();

                break;

            case 4:

                deleteEnd();

                break;

            case 5:

                display();

                break;

            case 6:

                exit(0);

            default:

                printf("Invalid choice\n");

        }

    }

    return 0;

}

**BST**

#include <stdio.h>

#include <stdlib.h>

struct node {

    int data;

    struct node \*left;

    struct node \*right;

};

void create\_tree(struct node \*\*tree) {

    \*tree = NULL;

}

struct node \*insert\_element(struct node \*tree, int data) {

    struct node \*ptr = (struct node \*)malloc(sizeof(struct node));

    ptr->data = data;

    ptr->left = NULL;

    ptr->right = NULL;

    if (tree == NULL) {

        return ptr;

    } else {

        struct node \*parentptr = NULL;

        struct node \*nodeptr = tree;

        while (nodeptr != NULL) {

            parentptr = nodeptr;

            if (data < nodeptr->data) {

                nodeptr = nodeptr->left;

            } else if (data > nodeptr->data) {

                nodeptr = nodeptr->right;

            } else {

                printf("The value %d already exists in the tree. Please enter a different value.\n", data);

                free(ptr);

                return tree;

            }

        }

        if (data < parentptr->data) {

            parentptr->left = ptr;

        } else {

            parentptr->right = ptr;

        }

    }

    return tree;

}

void preorderTraversal(struct node \*tree) {

    if (tree != NULL) {

        printf("%d ", tree->data);

        preorderTraversal(tree->left);

        preorderTraversal(tree->right);

    }

}

void inorderTraversal(struct node \*tree) {

    if (tree != NULL) {

        inorderTraversal(tree->left);

        printf("%d ", tree->data);

        inorderTraversal(tree->right);

    }

}

void postorderTraversal(struct node \*tree) {

    if (tree != NULL) {

        postorderTraversal(tree->left);

        postorderTraversal(tree->right);

        printf("%d ", tree->data);

    }

}

struct node \*del\_element(struct node \*tree, int val) {

    struct node \*ptr, \*parent, \*cur, \*suc, \*psuc;

    if (tree == NULL) {

        printf("\nThe tree is empty.\n");

        return tree;

    }

    parent = NULL;

    cur = tree;

    while (cur != NULL && val != cur->data) {

        parent = cur;

        cur = (val < cur->data) ? cur->left : cur->right;

    }

    if (cur == NULL) {

        printf("\nThe value %d to be deleted is not present in the tree.\n", val);

        return tree;

    }

    if (cur->left == NULL) {

        ptr = cur->right;

    } else if (cur->right == NULL) {

        ptr = cur->left;

    } else {

        psuc = cur;

        suc = cur->right;

        while (suc->left != NULL) {

            psuc = suc;

            suc = suc->left;

        }

        if (psuc != cur) {

            psuc->left = suc->right;

        } else {

            psuc->right = suc->right;

        }

        suc->left = cur->left;

        suc->right = cur->right;

        ptr = suc;

    }

    if (parent == NULL) {

        return ptr;

    } else if (parent->left == cur) {

        parent->left = ptr;

    } else {

        parent->right = ptr;

    }

    free(cur);

    return tree;

}

struct node \*search\_element(struct node \*tree, int data) {

    if (tree == NULL) {

        return NULL;

    }

    if (tree->data == data) {

        return tree;

    } else if (data < tree->data) {

        return search\_element(tree->left, data);

    } else {

        return search\_element(tree->right, data);

    }

}

int main() {

    struct node \*tree = NULL;

    int choice, val, n;

    while (1) {

        printf("\n1. Create a binary search tree\n");

        printf("2. Insert element\n");

        printf("3. Delete element\n");

        printf("4. Preorder traversal\n");

        printf("5. Inorder traversal\n");

        printf("6. Postorder traversal\n");

        printf("7. Search element\n");

        printf("8. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("\nEnter the number of elements you want to insert in the binary search tree: ");

                scanf("%d", &n);

                for (int i = 0; i < n; i++) {

                    printf("Enter the value of the element you want to insert: ");

                    scanf("%d", &val);

                    tree = insert\_element(tree, val);

                }

                break;

            case 2:

                printf("Enter the value of the element you want to insert: ");

                scanf("%d", &val);

                tree = insert\_element(tree, val);

                break;

            case 3:

                printf("Enter the value of the element you want to delete: ");

                scanf("%d", &val);

                tree = del\_element(tree, val);

                break;

            case 4:

                printf("The elements in the tree (Preorder): ");

                preorderTraversal(tree);

                printf("\n");

                break;

            case 5:

                printf("The elements in the tree (Inorder): ");

                inorderTraversal(tree);

                printf("\n");

                break;

            case 6:

                printf("The elements in the tree (Postorder): ");

                postorderTraversal(tree);

                printf("\n");

                break;

            case 7:

                printf("Enter the value of the element you want to search: ");

                scanf("%d", &val);

                if (search\_element(tree, val) != NULL) {

                    printf("The element %d is present in the tree.\n", val);

                } else {

                    printf("The element %d is not present in the tree.\n");

                }

                break;

            case 8:

                printf("\nExiting the program.\n");

                exit(0);

                break;

            default:

                printf("\nInvalid choice! Please try again.\n");

                break;

        }

    }

    return 0;

}

**Without recursion, without user input**

#include <stdio.h>

#include <stdlib.h>

struct TreeNode {

    int value;

    struct TreeNode\* left;

    struct TreeNode\* right;

};

struct TreeNode\* newNode(int value) {

    struct TreeNode\* node = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

    node->value = value;

    node->left = NULL;

    node->right = NULL;

    return node;

}

void inorderTraversal(struct TreeNode\* root) {

    struct TreeNode\* stack[100];

    int top = -1;

    struct TreeNode\* current = root;

    while (current != NULL || top >= 0) {

        while (current != NULL) {

            stack[++top] = current;

            current = current->left;

        }

        current = stack[top--];

        printf("%d ", current->value);

        current = current->right;

    }

}

void preorderTraversal(struct TreeNode\* root) {

    if (root == NULL)

        return;

    struct TreeNode\* stack[100];

    int top = -1;

    stack[++top] = root;

    while (top >= 0) {

        struct TreeNode\* current = stack[top--];

        printf("%d ", current->value);

        if (current->right != NULL)

            stack[++top] = current->right;

        if (current->left != NULL)

            stack[++top] = current->left;

    }

}

void postorderTraversal(struct TreeNode\* root) {

    if (root == NULL)

        return;

    struct TreeNode\* stack1[100];

    struct TreeNode\* stack2[100];

    int top1 = -1, top2 = -1;

    stack1[++top1] = root;

    while (top1 >= 0) {

        struct TreeNode\* current = stack1[top1--];

        stack2[++top2] = current;

        if (current->left != NULL)

            stack1[++top1] = current->left;

        if (current->right != NULL)

            stack1[++top1] = current->right;

    }

    while (top2 >= 0) {

        struct TreeNode\* current = stack2[top2--];

        printf("%d ", current->value);

    }

}

int main() {

    struct TreeNode\* root = newNode(10);

    root->left = newNode(20);

    root->right = newNode(30);

    root->left->left = newNode(40);

    root->left->right = newNode(50);

    root->right->right = newNode(60);

    printf("Inorder Traversal: ");

    inorderTraversal(root);

    printf("\n");

    printf("Preorder Traversal: ");

    preorderTraversal(root);

    printf("\n");

    printf("Postorder Traversal: ");

    postorderTraversal(root);

    printf("\n");

    return 0;

}

**With user input**

#include <stdio.h>

#include <stdlib.h>

struct TreeNode {

    int value;

    struct TreeNode\* left;

    struct TreeNode\* right;

};

struct TreeNode\* newNode(int value) {

    struct TreeNode\* node = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

    node->value = value;

    node->left = NULL;

    node->right = NULL;

    return node;

}

struct TreeNode\* insert\_element(struct TreeNode\* root, int value) {

    if (root == NULL) {

        return newNode(value);

    }

    if (value < root->value) {

        root->left = insert\_element(root->left, value);

    } else if (value > root->value) {

        root->right = insert\_element(root->right, value);

    }

    return root;

}

void inorderTraversal(struct TreeNode\* root) {

    struct TreeNode\* stack[100];

    int top = -1;

    struct TreeNode\* current = root;

    while (current != NULL || top >= 0) {

        while (current != NULL) {

            stack[++top] = current;

            current = current->left;

        }

        current = stack[top--];

        printf("%d ", current->value);

        current = current->right;

    }

}

void preorderTraversal(struct TreeNode\* root) {

    if (root == NULL)

        return;

    struct TreeNode\* stack[100];

    int top = -1;

    stack[++top] = root;

    while (top >= 0) {

        struct TreeNode\* current = stack[top--];

        printf("%d ", current->value);

        if (current->right != NULL)

            stack[++top] = current->right;

        if (current->left != NULL)

            stack[++top] = current->left;

    }

}

void postorderTraversal(struct TreeNode\* root) {

    if (root == NULL)

        return;

    struct TreeNode\* stack1[100];

    struct TreeNode\* stack2[100];

    int top1 = -1, top2 = -1;

    stack1[++top1] = root;

    while (top1 >= 0) {

        struct TreeNode\* current = stack1[top1--];

        stack2[++top2] = current;

        if (current->left != NULL)

            stack1[++top1] = current->left;

        if (current->right != NULL)

            stack1[++top1] = current->right;

    }

    while (top2 >= 0) {

        struct TreeNode\* current = stack2[top2--];

        printf("%d ", current->value);

    }

}

int main() {

    struct TreeNode\* root = NULL;

    int choice, value, n;

    while (1) {

        printf("\n1. Insert element\n");

        printf("2. Inorder traversal\n");

        printf("3. Preorder traversal\n");

        printf("4. Postorder traversal\n");

        printf("5. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter the number of elements to insert: ");

                scanf("%d", &n);

                for (int i = 0; i < n; i++) {

                    printf("Enter value: ");

                    scanf("%d", &value);

                    root = insert\_element(root, value);

                }

                break;

            case 2:

                printf("Inorder Traversal: ");

                inorderTraversal(root);

                printf("\n");

                break;

            case 3:

                printf("Preorder Traversal: ");

                preorderTraversal(root);

                printf("\n");

                break;

            case 4:

                printf("Postorder Traversal: ");

                postorderTraversal(root);

                printf("\n");

                break;

            case 5:

                printf("Exiting the program.\n");

                exit(0);

            default:

                printf("Invalid choice! Please try again.\n");

        }

    }

    return 0;

}

**Binary Sort**

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

void swap(int \*a, int \*b){

  int temp = \*a;

  \*a = \*b;

  \*b = temp;

}

void bubbleSort(int arr[], int n){

  bool swapped;

  for(int i = 0; i < n - 1; i++){

    swapped = false;

    for(int j = 0; j < n - i - 1; j++){

      if(arr[j] > arr[j + 1]){

        swap(&arr[j], &arr[j + 1]);

        swapped = true;

      }

    }

    if(!swapped){

      break;

    }

  }

}

void printArray(int arr[], int n){

  for(int i = 0; i < n; i++){

    printf("%d\t", arr[i]);

  }

  printf("\n");

}

int main(){

  int n;

  printf("Enter the number of elements: ");

  scanf("%d", &n);

  int arr[n];

  for(int i = 0; i < n; i++){

    printf("Enter integer %d: ", i + 1);

    scanf("%d", &arr[i]);

  }

  bubbleSort(arr, n);

  printf("Sorted array: ");

  printArray(arr, n);

  return 0;

}

**Shell Sort**

#include <stdio.h>

void shellSort(int arr[], int n) {

    for (int gap = n / 2; gap > 0; gap /= 2) {

        for (int i = gap; i < n; i++) {

            int temp = arr[i];

            int j;

            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {

                arr[j] = arr[j - gap];

            }

            arr[j] = temp;

        }

    }

}

void printArray(int arr[], int n) {

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

int main() {

    int n;

    printf("Enter the number of elements: ");

    scanf("%d", &n);

    int arr[n];

    for (int i = 0; i < n; i++) {

      printf("Enter integer %d: ", i + 1);

      scanf("%d", &arr[i]);

    }

    shellSort(arr, n);

    printf("Sorted array: ");

    printArray(arr, n);

    return 0;

}