- **DS Lab Codetantra Codes**

1. #include<stdio.h>

```
void main() {
int a[20], i, n, key, flag = 0, pos;
printf("Enter value of n : ");
scanf("%d", &n);
// Write code to read array elements
for(i=0; i<n;i++)
{
printf("Enter element for a[%d] : ",i);
scanf("%d", &a[i]);
}
printf("Enter key element : ");
scanf("%d", &key);
// Write code for linear search process
for(pos = 0; pos<n; pos++)
{
if(a[pos]== key)
{
break;
}
}
if (pos<n ) { //Write the condition part
printf("The key element %d is found at the position %d\n", key, pos ); //Complete the statement
} else {
printf("The key element %d is not found in the array\n", key ); //Complete the statement
}
```

```
}
```

- #include<stdio.h>

```
void main() {
int a[20], i, j, n, key, flag = 0, low, high, mid, temp;
printf("Enter value of n : ");
scanf("%d", &n);

for(i = 0; i < n; i++){
printf("Enter element for a[%d] : ", i);
scanf("%d", &a[i]);
}

printf("Enter key element : ");
scanf("%d", &key);

// Write the code to sort the elements using any sorting technique
for (int i = 0; i < n-1; i++){
for(int j = 0; j < n - i - 1; j++){
if(a[j] > a[j + 1]){
temp = a[j];
a[j] = a[j + 1];
a[j + 1] = temp;
}
}
}
printf("After sorting the elements in the array are\n");
// Write the code to display the elements
for (int i = 0;i < n; i++){
printf("Value of a[%d] = %d\n", i, a[i]);
```

```
}
low = 0; // Complete the statement
high = n - 1; // Complete the statement
int position = -1;
while(low <= high){
mid = low + (high - low) / 2;
if (a[mid] == key){
position = mid;
}
```

2.
```
#include<stdio.h>
#include<string.h>
void main() {
char a[20][20];
int i, n, j;
char temp[20];
printf("Enter value of n : ");
scanf("%d", &n);
//Write the code to read n strings.
for(i=0; i<n;i++){
printf("Enter string for a[%d] : ", i);
scanf("%s", a[i]);
}
printf("Before sorting the strings in the array are\n");
//Write the code to print the strings before sorting
for(i=0; i <n;i++){
printf("String at a[%d] = %s\n", i,a[i]);
}
for(i=0;i<n-1;i++){
```

```
for(j=0;j<n-i-1;j++){
if(strcmp(a[j],a[j+1])>0){
strcpy(temp,a[j]);
strcpy(a[j],a[j+1]);
strcpy(a[j+1],temp);
}
}
}
//Write the code to sort the strings using bubble sort technique
printf("After sorting the strings in the array are\n");
for(i=0;i<n;i++){
printf("String at a[%d] = %s\n", i ,a[i]);
}
}
```

- #include<stdio.h>

```
void main() {
int a[20], i, n, j, large, index;
printf("Enter value of n : ");
scanf("%d", &n);
// Write the code to read an array elements
int temp;
for (i=0;i<n;i++)
{
printf("Enter element for a[%d] : ",i);
scanf("%d",&a[i]);
}
printf("Before sorting the elements in the array are\n");
// Write the code to print the given array elements before sorting
```

```
for (i=0;i<n;i++)
{
printf("Value of a[%d] = %d\n",i,a[i]);
}
// Write the code for selection sort largest element method
for (i=n-1;i>0;i--)
{
index=0;
for (j=1;j<=i;j++)
{
if (a[j]>a[index])
{
index = j;
}
}
temp=a[index];
a[index] = a[i];
a[i] = temp;
}
printf("After sorting the elements in the array are\n");
// Write the code to print the given array elements after sorting
}
```

- #include<stdio.h>

```
void main() {
int a[20], i, n, j, temp, pos;
printf("Enter value of n : ");
scanf("%d", &n);
// write the for loop to read array elements
```

```
for(i=0;i<n;i++)
{
printf("Enter element for a[%d] : ",i);
scanf("%d",&a[i]);
}
printf("Before sorting the elements in the array are\n");
// write the for loop to display array elements before sorting
for(i=0;i<n;i++)
{
printf("Value of a[%d] = %d\n",i,a[i]);
}
for(i=1;i<n;i++){
temp = a[i];
pos=i;
for(j=i-1;j>=0 && a[j]>temp;j--)
{
a[j+1]=a[j];
pos=j;
}
a[pos]=temp;
}
printf("After sorting the elements in the array are\n");
// write the code to sort elements
// write the for loop to display array elements after sorting
for(i=0;i<n;i++)
{
printf("Value of a[%d] = %d\n",i,a[i]);
}
```

3. void display(int arr[15], int n) {

```
int i;
for(i=0;i<n;i++)
printf("%d ",arr[i]);
printf("\n");
}
void merge(int arr[15], int low, int mid, int high) {
int i=low,h=low,j=mid+1,k,temp[15];
while(h<=mid && j<=high){
if(arr[h]<=arr[j]){
temp[i]=arr[h];
h++;
}else{
temp[i]=arr[j];
j++;
}
i++;
}
if(h>mid){
for(k=j;k<=high;k++){
temp[i]=arr[k];
i++;
}
}else{
for(k=h;k<=mid;k++){
temp[i]=arr[k];
i++;
}
}
for(k=0;k<=high;k++)
```

```
arr[k]=temp[k];
}

void splitAndMerge(int arr[15], int low, int high) {
if(low<high){
int mid=(low+high)/2;
}
```

4.
```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
struct Node* next;
};

struct LinkedList {
struct Node* head;
int size;
};

struct Node* createNode(int data) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
if (newNode == NULL) {
printf("Memory allocation failed\n");
exit(1);
}
newNode->data = data;
newNode->next = NULL;
```

```
return newNode;
}

struct LinkedList* initializeList() {
struct LinkedList* list = (struct LinkedList*)malloc(sizeof(struct LinkedList));
if (list == NULL) {
printf("Memory allocation failed\n");
exit(1);
}
list->head = NULL;
list->size = 0;
return list;
}

void insert(struct LinkedList* list, int data, int position) {
// write your code here for insertion
if (position < 0 || position > list->size) {
printf("Invalid position\n");
return;
}

struct Node* newNode = createNode(data);
if (position == 0) {
newNode->next = list->head;
list->head = newNode;
} else {
struct Node* temp = list->head;
for (int i = 0; i < position - 1; i++) {
temp = temp->next;
```

- typedef struct node{

```
int data;
struct node* next;
} Node;
typedef Node* NODE;
NODE addNode(NODE head,int data){
NODE newNode;
newNode=(NODE)malloc(sizeof(Node));
newNode->data=data;
newNode->next=NULL;
if(head==NULL){
head=newNode;
}else{
NODE temp=head;
while(temp->next!=NULL){
temp=temp->next;
}
temp->next=newNode;
}
return head;
NODE temp=head;
while(temp!=NULL){
printf("%d --> ",temp->data);
temp=temp->next;
}
printf("NULL\n");
}
void displayList(NODE head){
NODE temp=head;
while(temp!=NULL){
```

```
printf("%d --> ",temp->data);
temp=temp->next;
}
printf("NULL\n");
}
NODE concatenate(NODE list1, NODE list2){
}
```

5.
```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
int data;
struct Node* next;
}Node;
typedef Node* NODE;
NODE createNode(int data){
NODE newNode = (NODE)malloc(sizeof(struct Node));
newNode -> data = data;
newNode -> next = newNode;
return newNode;
}
NODE insertAtPositionInCLL(NODE first,int pos,int data)
{
NODE newNode = (NODE)malloc(sizeof(NODE));
newNode -> data = data;
newNode -> next = NULL;
if(first == NULL){
if(pos==1){
newNode->next = newNode;
```

```
return newNode;
}else{
printf("Position not found\n");
free(newNode);
return first;
}
}
// write your code here
int count = 0;
NODE temp = first;
do{
count++;
temp=temp->next;
}while(temp!=first);
if(pos<1 || pos>count+1){
```

- #include <stdio.h>

```
#include <stdlib.h>
typedef struct Node {
int data;
struct Node* next;
} Node;

typedef Node* NODE;


NODE insertEnd(NODE head, int data) {
NODE newNode = (NODE)malloc(sizeof(Node));
newNode->data = data;
```

```
newNode->next = newNode;



if (head == NULL)
return newNode;

NODE temp = head;
while (temp->next != head)
// write your code here
temp = temp->next;

temp->next = newNode;
newNode->next = head;
return head;
}



void displayList(NODE head) {
if (head == NULL) {
printf("Circular Linked List is empty\n");
return;
```

6.
```
#include <stdio.h>
#include <stdlib.h>

// write your code here
struct Node {
int data;
struct Node* prev;
struct Node* next;
```

```
};

struct Node* head = NULL;

void insert(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
struct Node* temp = head;
newNode->data = value;
newNode->next = NULL;
newNode->prev = NULL;

if (head == NULL) {
head = newNode;
} else {
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
newNode->prev = temp;
}
}

void remov(int value) {
struct Node* temp = head;

if (head == NULL) {
printf("%d not found\n", value);
```

- #include <stdio.h>

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

// write your code here
struct Node{
int data;
struct Node* prev;
struct Node* next;
};
struct DoublyLinkedList{
struct Node*head;
struct Node*tail;
};
void append(struct DoublyLinkedList* list,int data){
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = data;
newNode->next = NULL;
newNode->prev = list->tail;
if(list->tail==NULL){
list->head=newNode;
}else{
list->tail->next=newNode;
}
list->tail = newNode;
}
void displayList(struct DoublyLinkedList* list){
struct Node* temp = list->head;
if(temp==NULL){
printf("List is empty\n");
```

```
return;
}while(temp!=NULL){
printf("%d <-> ",temp->data);
temp = temp->next;
}
printf("NULL\n");
```

7. #include <stdio.h>

```
}

// Inorder Traversal using recursion
void inorder(struct node *root) {
if (root != NULL) {
// Traverse left
inorder(root->left);
// Traverse root
printf("%d ", root->key);
// Traverse right
inorder(root->right);
}
}

void preorder(struct node *root) {
if (root != NULL) {
// Traverse left
printf("%d ", root->key);
preorder(root->left);
// Traverse right
preorder(root->right);
```

```
}
}
// Find the inorder successor
struct node *minValueNode(struct node *node) {
struct node *current = node;
// Find the leftmost leaf
while (current && current->left != NULL)
current = current->left;
return current;
}

// Insert a node in BST
struct node *insert(struct node *node, int key) {
// write your code here to perform insertion operation
if(node==NULL)
return newNode(key);
if(key<node -> key)
node->left = insert(node -> left,key);
else if(key>node -> key)
node -> right = insert(node -> right , key);
return node;



}

// Deleting a node
struct node *deleteNode(struct node *root, int key) {

// write your code here to perform deletion operation
if (root == NULL)
```

return root ;

if(key<root -> key)

root -> left = deleteNode (root -> left , key);

else if(key>root -> key)

- #include <stdio.h>

inorderInBST(root->left);

printf("%d ", root->data);

inorderInBST(root->right);

}

}

// Function to perform pre-order traversal

void preorderInBST(BSTNODE root) {

if (root != NULL) {

printf("%d ", root->data);

preorderInBST(root->left);

preorderInBST(root->right);

}

}

void postorderInBST(BSTNODE root) {

if (root != NULL) {

postorderInBST(root->left);

postorderInBST(root->right);

printf("%d ", root->data);

}

}

// Function to search for an element in BST

BSTNODE searchNodeInBST(BSTNODE root, int ele) {

```
if (root == NULL) {
return NULL;
}
if (root->data == ele) {
return root;
}
if (ele < root->data) {
return searchNodeInBST(root->left, ele);
} else {
return searchNodeInBST(root->right, ele);
}
}
void displayMenu() {
printf("\n1.Insert\n");
printf("2.Inorder Traversal\n");
printf("3.Preorder Traversal\n");
printf("4.Postorder Traversal\n");
printf("5.Search an element\n");
printf("6.Exit\n");

}
```

8.
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
typedef struct node {
struct node *next;
int vertex;
} *GNODE;
GNODE graph[MAX] = {NULL};
```

```
void print(int *n);
void insertVertex(int *n);
void insertEdge(int *n);
void deleteVertex(int *n);
void deleteEdge(int *n);

void insertVertex(int *n){
(*n)++;
graph[*n] = NULL;
printf("Vertex inserted: %d\n",*n);}
void insertEdge(int *n){
int src, dest;
printf("Enter the source vertex of the edge : ");
scanf("%d",&src);
printf("Enter the destination vertex of the edge : ");
scanf("%d",&dest);
if(src>*n || dest>*n || src<=0 || dest <=0){
printf("Invalid vertex number.\n");
return;
}
GNODE newNode = (GNODE)malloc(sizeof(struct node));
newNode -> vertex = dest;
newNode -> next = NULL;
if (graph[src]==NULL){
graph[src] = newNode;
}else{
GNODE p = graph[src];
```

9. #include<stdio.h>

```
}
for(i = 1; i<= E; i++){
printf("Enter source: ");
scanf("%d", &s);
printf("Enter destination: ");
scanf("%d", &d);
q = (GNODE)malloc(sizeof(struct node));
q->vertex = d;
q->next = NULL;
if(graph[s] == NULL)
graph[s] = q;
else {
p = graph[s];
while(p->next != NULL)
p = p->next;
p->next = q;
}
}
for(i = 0; i < n; i++)
visited[i] = 0;

printf("Enter Start Vertex for DFS: ");
scanf("%d", &v);
printf("DFS of graph: \n");
DFS(v);
printf("\n");
}
```

- struct node {

```
scanf("%d",&n);
printf("Enter no of edges: ");
scanf("%d",&E);
for(i=1;i<=E;i++) {
printf("Enter source: ");
scanf("%d",&s);
printf("Enter destination: ");
scanf("%d",&d);
q=(GNODE)malloc(sizeof(struct node));
q->vertex=d;
q->next=NULL;
if(graph[s]==NULL) {
graph[s]=q;
} else {
p=graph[s];
while(p->next!=NULL)
p=p->next;
p->next=q;
}
}
for(i=1;i<=n;i++)
visited[i]=0;
printf("Enter Start Vertex for BFS: ");
scanf("%d", &v);
printf("BFS of graph: \n");
BFS(v);
printf("\n");
}
```

10. #include<stdio.h>

```
#include<conio.h>

// complete the missing code..
#define MAX 20
#define INF 999
int cost[MAX][MAX],n,e,i,j,s,d,w;

void prims() {
int selected[MAX] = {0},no_of_edges=0;
int x,y,min,total_cost=0;

selected[1] = 1;

while(no_of_edges < n-1){
min = INF;
for(i=1;i<=n ;i++){
if(selected[i]){
for(j=1;j<=n;j++){
if(!selected[j] && cost[i][j] <min){
min = cost[i][j];
x=i;
y=j;
}
}
}
}
printf("Edge cost from %d to %d : %d\n",x,y,cost[x][y]);
total_cost += cost[x][y];
selected[y] =1;
```

```
no_of_edges++ ;
}
printf("Minimum cost of spanning tree = %d\n",total_cost);
}

void main() {
}
```