DST

DIVISION METHOD

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

#define H_SIZE 10
int Hash[H_SIZE]={0};
int value,key;

void Insert()
{
	printf("\nEnter value:");
	scanf("%d",&value);
	key=(value%H_SIZE);
	if(Hash[key]==0)
		Hash[key]=value;
	else
		printf("\nCollision Occured..");
}

void Delete()
{
	printf("\nEnter value:");
	scanf("%d",&value);
	key=(value%H_SIZE);
	printf("\nDeleted Value = %d",Hash[key]);
	Hash[key]=0;
}

void Search()
```

```
{
        printf("\nEnter value to Search:");
        scanf("%d",&value);
        key=value%H_SIZE;
                if(value==Hash[key])
                        printf("\nElement Found..");
                else
                        printf("\nElement Not Found..");

}

void Display()
{
int i;
printf("\nHash Table:");
        for(i=0;i<H_SIZE;i++)
        if(Hash[i]== 0)
                printf("\nHash[%d] =  ",i);
        else
                printf("\nHash[%d] = %d",i,Hash[i]);
}

int main()
{
        int ch;
printf("\nHash operations Using Division Method");
do
{
        printf("\n1. Insert");
        printf("\n2. Delete");
        printf("\n3. Search");
        printf("\n4. Display");
        printf("\n0. Exit");
        printf("\nEnter Ur Choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                        Insert();
                        break;
                case 2:
                        Delete();
                        break;
                case 3:
                        Search();
                        break;
                case 4:
                        Display();
                        break;
                case 0:
                        exit(0);
                default:
                        printf("\nInvalid Option...");
                        break;
        }
}while(ch!=0);
return 0;
}
```

MID SQUARE METHOD

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

#define H_SIZE 100
int Hash[H_SIZE]={0};
int value,key;
void calculate()
{
        int count=0;
        int n,m;n=value*value;
        m=n;
        while(n!=0)
        {
                count++;
                n=n/10;
        }
        if(count%2==1)
        {
                key=m/(pow(10,count/2));
                key=key%10;
        }
        else
        {
                key=m/(pow(10,(count/2)-1));
                key=key%100;
        }
}
void Insert()
```

```
{

        printf("\nEnter value:");
        scanf("%d",&value);
        calculate();
        if(Hash[key]==0)
                Hash[key]=value;
        else
                printf("\nCollision Occured..");
}

void Delete()
{
        printf("\nEnter value:");
        scanf("%d",&value);
        calculate();
        printf("\nDeleted Value = %d",Hash[key]);
        Hash[key]=0;
}

void Search()
{
        printf("\nEnter value to Search:");
        scanf("%d",&value);
        calculate();
                if(Hash[key]==value)
                        printf("\nElement Found..");
                else
                        printf("\nElement Not Found..");
```

```
}

void Display()
{
int i;
printf("\nHash Table:");
        for(i=0;i<H_SIZE;i++)
        if(Hash[i]!= 0)
                printf("\nHash[%d] = %d",i,Hash[i]);
}

int main()
{
        int ch;
printf("\nHash operations Using MidSquare Method");
do
{
        printf("\n1. Insert");
        printf("\n2. Delete");
        printf("\n3. Search");
        printf("\n4. Display");
        printf("\n0. Exit");
        printf("\nEnter Ur Choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:
                        Insert();
                        break;
                case 2:
```

```
                Delete();
                break;
            case 3:
                Search();
                break;
            case 4:
                Display();
                break;
            case 0:
                exit(0);
            default:
                printf("\nInvalid Option...");
                break;
        }
}while(ch!=0);
return 0;
}
```

LINEAR PROBING

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

#define H_SIZE 10
int Hash[H_SIZE]={0};
int value,key;

void Insert()
{
        int i,x;
        printf("\nEnter value:");
        scanf("%d",&value);
        key=(value%H_SIZE);
        if(Hash[key]==0)
                Hash[key]=value;
        else
        {
                for(i=1;i<H_SIZE;i++)
                {
                        x=(key+i)%H_SIZE;
                        if(Hash[x]==0)
                        {
                                Hash[x]=value;
                                break;
                        }
                }
        }
                //printf("\nCollision Occured..");
}
```

```
void Delete()
{
int i,x;
          printf("\nEnter value:");
          scanf("%d",&value);
          key=(value%H_SIZE);
          if(Hash[key]==value)
          {
          printf("\nDeleted Value = %d",Hash[key]);
          Hash[key]=0;
          }
          else
          {
                    for(i=1;i<H_SIZE;i++)
                    {
                              x=(key+i)%H_SIZE;
                              if(Hash[x]==value)
                              {
                                        printf("\nDeleted Value = %d",Hash[x]);
                                        Hash[x]=0;
                                        break;
                              }
                    }
          }
}

void Search()
{
int flag=0,i,x;
```

```
        printf("\nEnter value to Search:");
        scanf("%d",&value);
        key=value%H_SIZE;
                if(value==Hash[key])
                        flag=1;
                else
                {
                for(i=1;i<H_SIZE;i++)
                {
                        x=(key+i)%H_SIZE;
                        if(Hash[x]==value)
                        {
                                flag=1;
                                break;
                        }
                }
                }
                if(flag==1)
                        printf("\nElement Found..");
                else
                        printf("\nElement Not Found..");
}

void Display()
{
int i;
printf("\nHash Table:");
        for(i=0;i<H_SIZE;i++)
        if(Hash[i]== 0)
                printf("\nHash[%d] =  ",i);
```

```
		else
			printf("\nHash[%d] = %d",i,Hash[i]);
	}

	int main()
	{
		int ch;
	printf("\nHash operations Using Division Method");
	do
	{
		printf("\n1. Insert");
		printf("\n2. Delete");
		printf("\n3. Search");
		printf("\n4. Display");
		printf("\n0. Exit");
		printf("\nEnter Ur Choice:");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				Insert();
				break;
			case 2:
				Delete();
				break;
			case 3:
				Search();
				break;
			case 4:
				Display();
```

```
                        break;
                case 0:
                        exit(0);
                default:
                        printf("\nInvalid Option...");
                        break;
        }
}while(ch!=0);
return 0;
}
```

QUADRATIC PROBING

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

#define H_SIZE 10
int Hash[H_SIZE]={0};
int value,key;

void Insert()
{
        int i,x;
        printf("\nEnter value:");
        scanf("%d",&value);
        key=(value%H_SIZE);
        if(Hash[key]==0)
                Hash[key]=value;
        else
        {
                for(i=1;i<H_SIZE;i++)
                {
                        x=(key+(i*i))%H_SIZE;
                        if(Hash[x]==0)
                        {
                                Hash[x]=value;
                                break;
                        }
                }
        }
                //printf("\nCollision Occured..");
}
```

```
void Delete()
{
int i,x;
        printf("\nEnter value:");
        scanf("%d",&value);
        key=(value%H_SIZE);
        if(Hash[key]==value)
        {
        printf("\nDeleted Value = %d",Hash[key]);
        Hash[key]=0;
        }
        else
        {
                for(i=1;i<H_SIZE;i++)
                {
                        x=(key+(i*i))%H_SIZE;
                        if(Hash[x]==value)
                        {
                                printf("\nDeleted Value = %d",Hash[x]);
                                Hash[x]=0;
                                break;
                        }
                }
        }
}

void Search()
{
int flag=0,i,x;
```

```
        printf("\nEnter value to Search:");
        scanf("%d",&value);
        key=value%H_SIZE;
                if(value==Hash[key])
                        flag=1;
                else
                {
                for(i=1;i<H_SIZE;i++)
                {
                        x=(key+(i*i))%H_SIZE;
                        if(Hash[x]==value)
                        {
                                flag=1;
                                break;
                        }
                }
                }
                if(flag==1)
                        printf("\nElement Found..");
                else
                        printf("\nElement Not Found..");
}

void Display()
{
int i;
printf("\nHash Table:");
        for(i=0;i<H_SIZE;i++)
        if(Hash[i]== 0)
                printf("\nHash[%d] =  ",i);
```

```
          else
                    printf("\nHash[%d] = %d",i,Hash[i]);
}

int main()
{
          int ch;
printf("\nHash operations Using Division Method");
do
{
          printf("\n1. Insert");
          printf("\n2. Delete");
          printf("\n3. Search");
          printf("\n4. Display");
          printf("\n0. Exit");
          printf("\nEnter Ur Choice:");
          scanf("%d",&ch);
          switch(ch)
          {
                    case 1:
                              Insert();
                              break;
                    case 2:
                              Delete();
                              break;
                    case 3:
                              Search();
                              break;
                    case 4:
                              Display();
                    break;
            case 0:
                    exit(0);
            default:
                    printf("\nInvalid Option...");
                    break;
        }
}while(ch!=0);
return 0;
}
```

SEPARATE CHAINING

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


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


struct HashTable {
   int size;
   struct Node** array;
};


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 HashTable* createHashTable(int size) {
       int i;
   struct HashTable* hashTable = (struct HashTable*)malloc(sizeof(struct
HashTable));
   if (hashTable == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   hashTable->size = size;
   hashTable->array = (struct Node*)malloc(size * sizeof(struct Node));
   if (hashTable->array == NULL) {
      printf("Memory allocation failed\n");
      exit(1);
   }

   for (i = 0; i < size; i++) {
      hashTable->array[i] = NULL;
   }
   return hashTable;
}


void insert(struct HashTable* hashTable, int key) {
   int index = key % hashTable->size;
   struct Node* newNode = createNode(key);

   if (hashTable->array[index] == NULL) {
      hashTable->array[index] = newNode;
   } else {
      struct Node* temp = hashTable->array[index];
      while (temp->next != NULL) {
          temp = temp->next;
      }
      temp->next = newNode;
   }
}



void display(struct HashTable* hashTable) {
       int i;
   for (i = 0; i < hashTable->size; i++) {
      printf("%d -> ", i);
      struct Node* temp = hashTable->array[i];
      while (temp != NULL) {
          printf("%d -> ", temp->data);
          temp = temp->next;
      }
      printf("NULL\n");
   }
}
int main() {
   struct HashTable* hashTable = createHashTable(10);
   insert(hashTable, 10);
   insert(hashTable, 20);
   insert(hashTable, 30);
   insert(hashTable, 15);
   insert(hashTable, 25);

   display(hashTable);

   return 0;
}
```

HEAP SORT

```
#include <stdio.h>


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


    int temp = *a;
    *a = *b;
    *b = temp;
}


void heapify(int arr[], int N, int i)
{


    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < N && arr[left] > arr[largest])

        largest = left;

    if (right < N && arr[right] > arr[largest])

        largest = right;

    if (largest != i) {

        swap(&arr[i], &arr[largest]);
        heapify(arr, N, largest);
    }
}

void heapSort(int arr[], int N)
{
          int i;
    for (i = N / 2 - 1; i >= 0; i--)

        heapify(arr, N, i);



    for (i = N - 1; i >= 0; i--) {

        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}


void printArray(int arr[], int N)
{
          int i;
    for (i = 0; i < N; i++)
        printf("%d ", arr[i]);
    printf("\n");
}


int main()
```

```
{
        int n,i;
        printf("\nEnter How many Elements");
        scanf("%d",&n);
        int a[n];
        printf("\nEnter Elements:");
        for(i=0;i<n;i++)
                scanf("%d",&a[i]);
   heapSort(a,n);
   printf("Sorted array is\n");
   printArray(a,n);
   return 0;
}
```

BINARY SEARCH TREE

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

struct BST
{
        int key;
        struct BST *left,*right;
};

struct BST *root=NULL;

struct BST* Create(int key)
{
        struct BST *New;
        New=(struct BST *)malloc(sizeof(struct BST));
        New->key=key;
        New->left=NULL;
        New->right=NULL;
        return New;
}


struct BST* Insert(struct BST *Node,int key)
{
        if(Node==NULL)
                return Create(key);
        if(key < Node->key)
                Node->left=Insert(Node->left,key);
        else if(key > Node->key)
```

```
                Node->right=Insert(Node->right,key);

        return Node;
}

void Inorder(struct BST *Node)
{
        if(Node!=NULL)
        {
                Inorder(Node->left);
                printf("%d ",Node->key);
                Inorder(Node->right);
        }
}

void preOrder(struct BST *Node)
{
        if(Node!=NULL)
        {
                printf("%d ",Node->key);
                preOrder(Node->left);
                preOrder(Node->right);
        }
}

void minValue()
{
        struct BST *Node;
        Node=root;
        while(Node->left!=NULL)
```

```
                Node=Node->left;
        printf("\nMinimun Value = %d",Node->key);
}

void maxValue()
{
        struct BST *Node;
        Node=root;
        while(Node->right!=NULL)
                Node=Node->right;
        printf("\nMaximum Value = %d",Node->key);
}

void search()
{
        struct BST *Node;
        int key,flag=0;
        Node=root;
        printf("\nEnter Searching Value:");
        scanf("%d",&key);
        while(Node!=NULL)
        {
                if(key==Node->key)
                {
                flag=1;
                break;
                }
                if(key<Node->key)   Node=Node->left;
                if(key>Node->key)   Node=Node->right;
        }
        if(flag==1)
                printf("\nElement Found..");
        else
                printf("\nElement Not Found..");
}

struct BST *Successor(struct BST *Node)
{
        struct BST *current;
        current=Node;
        while(current && current->left !=NULL)
                current=current->left;
        return current;
}

struct BST *deleteNode(struct BST *Node,int key)
{
        struct BST *temp;
        if(Node==NULL)
                return Node;
        if(key<Node->key)
                Node->left=deleteNode(Node->left,key);
        else if(key>Node->key)
                Node->right=deleteNode(Node->right,key);
        else
        {
                if(Node->left==NULL)
                {
                        temp=Node->right;
                        free(Node);
```

```
                                return temp;
                        }
                        else if(Node->right==NULL)
                        {
                                temp=Node->left;
                                free(Node);
                                return temp;
                        }
                temp=Successor(Node->right);
                Node->key=temp->key;
                Node->right=deleteNode(Node->right,key);
                }
                return Node;
}
int main()
{
                int i,n,key;
                printf("\nEnter Number of Nodes:");
                scanf("%d",&n);
                for(i=0;i<n;i++)
                {
                        printf("\nEnter Value:");
                        scanf("%d",&key);
                        root=Insert(root,key);
                }
                printf("\nInorder Traversal:");
                Inorder(root);
                printf("\nPreorder Traversal:");
                preOrder(root);
                minValue();
```

```
        maxValue();
        search();
        printf("\nEnter Node to Delete:");
        scanf("%d",&key);
        root=deleteNode(root,key);
        Inorder(root);
return 0;
}
```