SEPARATE HASHING:

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

#define size 10

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

struct node *chain[size];

void init()
{
   int i;
   for(i = 0; i < size; i++)
      chain[i] = NULL;
}

void insert(int value)
{

   struct node *newNode = malloc(sizeof(struct node));
   newNode->data = value;
   newNode->next = NULL;


   int key = value % size;
```

```
    if(chain[key] == NULL)
        chain[key] = newNode;

    else
    {

        struct node *temp = chain[key];
        while(temp->next)
        {
            temp = temp->next;
        }

        temp->next = newNode;
    }
}

void print()
{
    int i;

    for(i = 0; i < size; i++)
    {
        struct node *temp = chain[i];
        printf("chain[%d]-->",i);
        while(temp)
        {
            printf("%d -->",temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
```

```
}

int main()
{

   init();

   insert(7);
   insert(0);
   insert(13);
   insert(23);
   insert(4);
   insert(5);

   print();

   return 0;
}
```

OUTPUT:

```
chain[0]-->0 -->NULL
chain[1]-->NULL
chain[2]-->NULL
chain[3]-->13 -->23 -->NULL
chain[4]-->4 -->NULL
chain[5]-->5 -->NULL
chain[6]-->NULL
chain[7]-->7 -->NULL
chain[8]-->NULL
chain[9]-->NULL
```

LINEAR PROBING:

```
// C program for the above approach
#include <stdio.h>
#include <stdlib.h>

struct HashNode {
        int key;
        int value;
};

const int capacity = 20;
int size = 0;

struct HashNode** arr;
struct HashNode* dummy;

// Function to add key value pair
void insert(int key, int V)
{

        struct HashNode* temp
                = (struct HashNode*)malloc(sizeof(struct HashNode));
        temp->key = key;
        temp->value = V;

        // Apply hash function to find
        // index for given key
        int hashIndex = key % capacity;

        // Find next free space
        while (arr[hashIndex] != NULL
```

```
                && arr[hashIndex]->key != key
                && arr[hashIndex]->key != -1) {
                hashIndex++;
                hashIndex %= capacity;
        }

        // If new node to be inserted
        // increase the current size
        if (arr[hashIndex] == NULL
                || arr[hashIndex]->key == -1)
                size++;

        arr[hashIndex] = temp;
}

// Function to delete a key value pair
int delete (int key)
{
        // Apply hash function to find
        // index for given key
        int hashIndex = key % capacity;

        // Finding the node with given
        // key
        while (arr[hashIndex] != NULL) {
                // if node found
                if (arr[hashIndex]->key == key) {
                        // Insert dummy node here
                        // for further use
                        arr[hashIndex] = dummy;

                        // Reduce size
```

```
                        size--;

                        // Return the value of the key
                        return 1;
                }
                hashIndex++;
                hashIndex %= capacity;
        }

        // If not found return null
        return 0;
}

// Function to search the value
// for a given key
int find(int key)
{
        // Apply hash function to find
        // index for given key
        int hashIndex = (key % capacity);

        int counter = 0;

        // Find the node with given key
        while (arr[hashIndex] != NULL) {

                int counter = 0;
                // If counter is greater than
                // capacity
                if (counter++ > capacity)
                        break;
```

```
                // If node found return its
                // value
                if (arr[hashIndex]->key == key)
                        return arr[hashIndex]->value;

                hashIndex++;
                hashIndex %= capacity;
        }

        // If not found return
        // -1
        return -1;
}

// Driver Code
int main()
{
        // Space allocation
        arr = (struct HashNode**)malloc(sizeof(struct HashNode*)
                                                                * capacity);
        // Assign NULL initially
        for (int i = 0; i < capacity; i++)
                arr[i] = NULL;

        dummy
                = (struct HashNode*)malloc(sizeof(struct HashNode));

        dummy->key = -1;
        dummy->value = -1;

        insert(1, 5);
        insert(2, 15);
```

```
        insert(3, 20);
        insert(4, 7);

        if (find(4) != -1)
                printf("Value of Key 4 = %d\n", find(4));
        else
                printf("Key 4 does not exists\n");

        if (delete (4))
                printf("Node value of key 4 is deleted "
                        "successfully\n");
        else {
                printf("Key does not exists\n");
        }

        if (find(4) != -1)
                printf("Value of Key 4 = %d\n", find(4));
        else
                printf("Key 4 does not exists\n");
}
```