**DAY 15 – HASH TABLE**

18.. Write a menu driven C program to implement hash table and the

following collision resolution techniques-(i) Linear Probing (ii) Quadratic

Probing (iii) Chaining

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

int lin\_prob\_array[MAX];

int quad\_prob\_array[MAX];

int size, c1, c2;

struct node

{

    int key;

    struct node \*next;

}\*chaining\_array[MAX] = {NULL};

void display\_lin\_quad(int *a*[MAX])

{

    int i;

    printf("Hash table: ");

    for(i = 0; i < size; i++)

    {

        if(*a*[i] != -1)

        printf("%d ", *a*[i]);

        else

        printf("- ");

    }

}

void display\_chaining()

{

    int i;

    struct node \*ptr;

    printf("Hash table:\n");

    for(i = 0; i < size; i++)

    {

        printf("%d - ", i);

        ptr = chaining\_array[i];

        if(chaining\_array[i] == NULL)

        printf("NULL");

        else

        {

            while(ptr != NULL)

            {

                printf("%d -> ", ptr -> key);

                ptr = ptr -> next;

            }

            printf("NULL");

        }

        printf("\n");

    }

}

void insert\_lin\_prob(int *key*)

{

    int i, hash, finalhash, flag = 1;

    for(i = 0; i < size; i++)

    {

        if(lin\_prob\_array[i] == -1)

        flag = 0;

    }

    if(flag == 1)

    {

        printf("\nHash table is full.");

        return;

    }

    finalhash = hash = *key* % size;

    if(lin\_prob\_array[hash] != -1)

    {

        for(i = 0; i < size; i++)

        {

            finalhash = (hash + i) % size;

            if(lin\_prob\_array[finalhash] == -1)

            break;

        }

    }

    lin\_prob\_array[finalhash] = *key*;

    if(*key*!= -1)

    display\_lin\_quad(lin\_prob\_array);

}

void insert\_quad\_prob(int *key*)

{

    int i, hash, finalhash, flag = 1;

    for(i = 0; i < size; i++)

    {

        if(lin\_prob\_array[i] == -1)

        flag = 0;

    }

    if(flag == 1)

    {

        printf("\nHash table is full.");

        return;

    }

    finalhash = hash = *key* % size;

    if(quad\_prob\_array[hash] != -1)

    {

        for(i = 0; i < size; i++)

        {

            finalhash = (hash + c1 \* i + c2 \* i \* i) % size;

            if(quad\_prob\_array[finalhash] == -1)

                break;

        }

    }

    quad\_prob\_array[finalhash] = *key*;

    if(*key* != -1)

        display\_lin\_quad(quad\_prob\_array);

}

void insert\_chaining(int *key*)

{

    int i, hash;

    struct node \*newnode, \*ptr;

    hash = *key* % size;

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

    newnode -> key = *key*;

    newnode -> next = NULL;

    if(chaining\_array[hash] == NULL)

        chaining\_array[hash] = newnode;

    else

    {

        ptr = chaining\_array[hash];

        while(ptr -> next != NULL)

        ptr = ptr -> next;

        ptr -> next = newnode;

    }

    if(*key* != -1)

        display\_chaining();

}

void main()

{

    int ch, key,i;

    for(i=0;i<MAX;i++)

    {

        lin\_prob\_array[i]=-1;

        quad\_prob\_array[i]=-1;

    }

    printf("Enter hash table size: ");

    scanf("%d", &size);

    printf("Choose collision resolution method -\n");

    printf("1. Linear Probing, 2. Quadratic Probing, 3. Chaining: ");

    scanf("%d", &ch);

    if(ch == 2)

    {

        printf("Enter values of c1 and c2: ");

        scanf("%d%d", &c1, &c2);

    }

    do

    {

        printf("\nEnter key to insert (-1 to exit): ");

        scanf("%d", &key);

        switch(ch)

        {

        case 1: insert\_lin\_prob(key);

                break;

        case 2: insert\_quad\_prob(key);

                break;

        case 3: insert\_chaining(key);

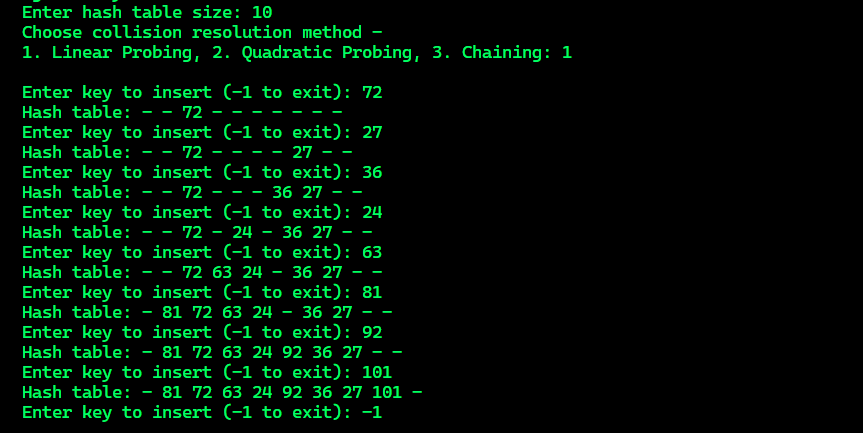
                    break;

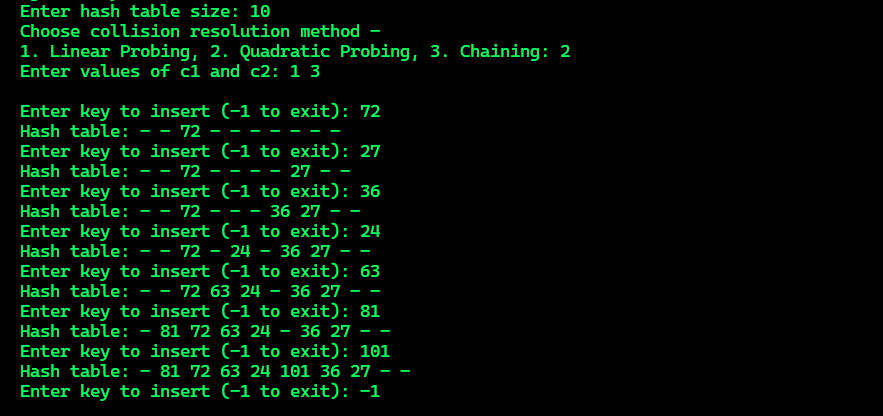
        }

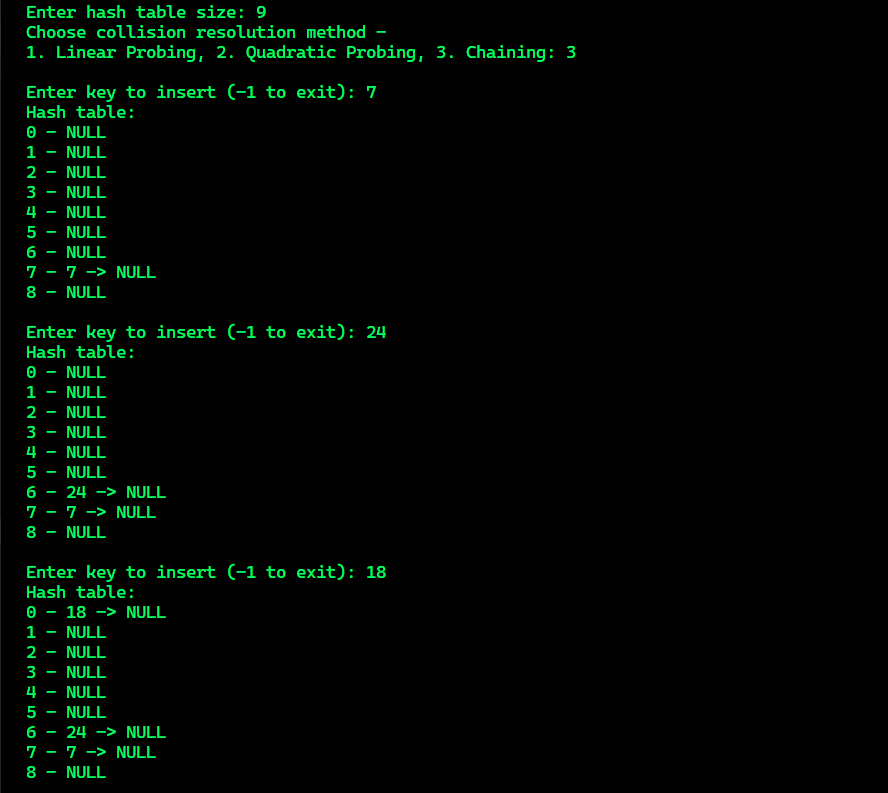
    }while(key != -1);

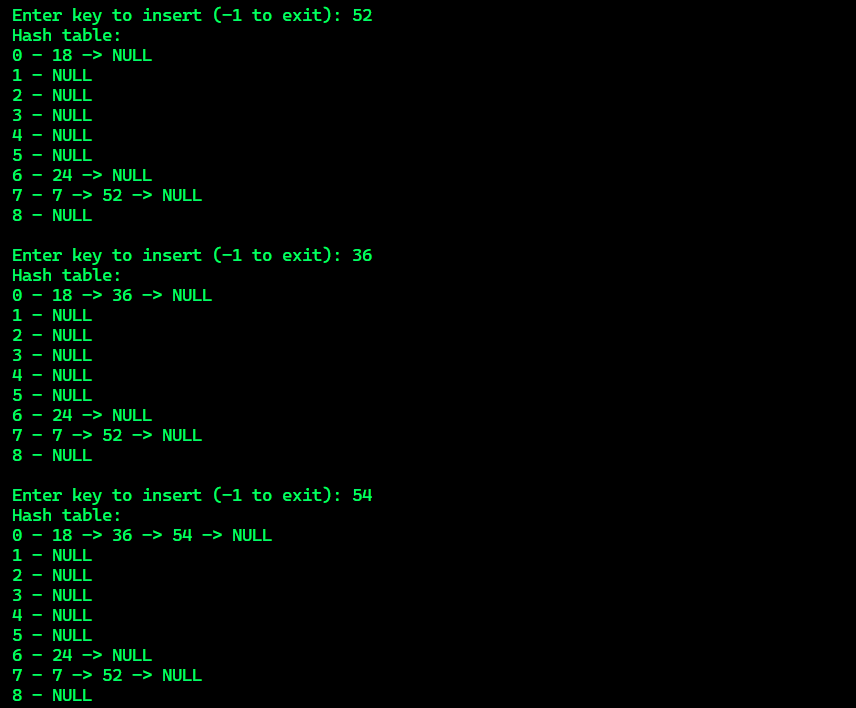
}

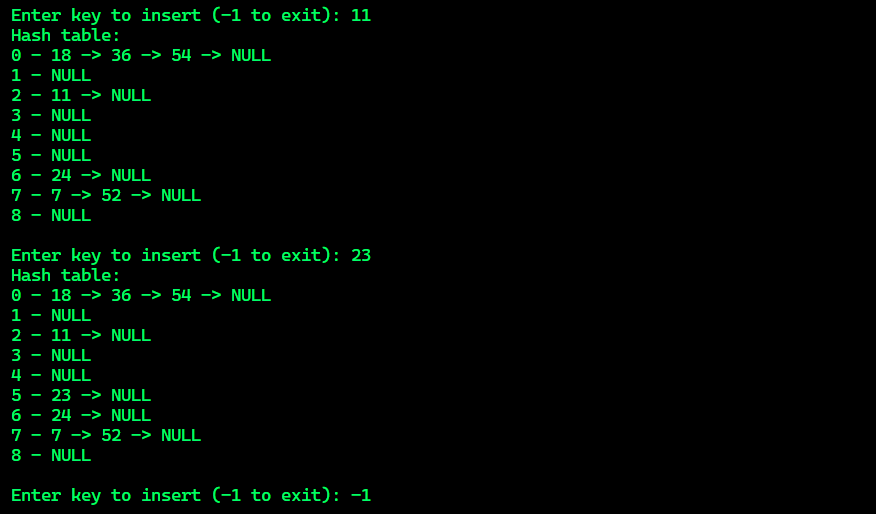
**OUTPUT**

****

****

****

****

****