**Assignment 8**

**Aim:**

A node consists of <Key,Value> pair where nodes are compared and inserted on the basis of key

Build a structure such that it should provide a facility of adding a new key,update meaning of a key,

delete a key(linear probing without chaining)

**Objective:**

Understand the concept of Linear probing without chaining

**Theory:**

**a) Linear Probing:** In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also.  
let **hash(x)** be the slot index computed using hash function and **S** be the table size

If slot hash(x) % S is full, then we try (hash(x) + 1) % S

If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S

If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S

..................................................

..................................................

Let us consider a simple hash function as “key mod 7” and sequence of keys as 50, 700, 76, 85, 92, 73, 101.



**Disadvantage of Linear Probing :**

**Clustering:** The main problem with linear probing is clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search an element.

**Algorithm :**

1. Take the size of the hash table from the user and store it in size variable
2. Create a structure to store key and value and make array of objects
3. Initialize the hash keys to -1
4. For Insert

Start Loop from I=0 and insert at h[(key+i)%size] if this value is -1 or -2(if value deleted)

1. For Search

Take the key to be searched

Start a loop from I=0 and check h[(key+i)%2] is the key searched

If yes

break the loop and return location

Else

Continue

Return(-1) if not found

1. For Delete

Take the key to be deleted and Search the location and update its value to null and set its key to -2

**Code:**

#include<iostream>

#include<string.h>

using namespace std;

struct product

{

int serialNo;

char name[20];

};

void insertKey(product ht[],int size );

void searchValue(product ht[],int size);

void updateValue(product ht[],int size);

void deleteKey(product ht[],int size);

void display(product ht[],int size);

int hashKey(int,int);

int main()

{

int choise;

int size;

cout<<"Enter the size of hash table : ";

cin>>size;

product hashTable[size];

for(int i=0;i<size;i++)

{

hashTable[i].serialNo = -1;

strcpy(hashTable[i].name,"");

}

do

{

cout<<"\n------------------------------------------------------------\n";

cout<<"\t\t\t Linear Probing Without Chaining \n\n";

cout<<"\t\t1.Insert a new Product \n";

cout<<"\t\t2.Search for Product details with serial number \n";

cout<<"\t\t3.Update the Product data \n";

cout<<"\t\t4.Delete the Product data \n";

cout<<"\t\t5.Display the hashing table data \n";

cout<<"\t\t6.Exit\n";

cout<<"\tYour choise : ";

cin>>choise;

switch(choise)

{

case 1:

insertKey(hashTable,size);

break;

case 2:

searchValue(hashTable,size);

break;

case 3:

updateValue(hashTable,size);

break;

case 4:

deleteKey(hashTable,size);

break;

case 5:

display(hashTable,size);

break;

}

}while(choise<6);

}

void insertKey(product ht[],int size)

{

int key;char name[20];

cout<<"Enter the Serial Number of product : ";

cin>>key;

cout<<"Enter the Name of Product : ";

cin>>name;

int hash = hashKey(key,size);

if(ht[hash].serialNo==(-1))

{

ht[hash].serialNo = key;

strcpy(ht[hash].name,name);

cout<<"\t\tProduct inserted Succesfully in table \n";

}

else

{

for(int i=0;i<size;i++)

{

if(ht[(hash+i)%size].serialNo == (-1))

{

ht[(hash+i)%size].serialNo = key;

strcpy(ht[(hash+i)%size].name,name);

cout<<"\t\tProduct inserted Succesfully in table \n";

break;

}

}

}

}

void display(product ht[],int size)

{

cout<<"\n\t\t\tThe data present in hash table is as follows \n\n";

cout<<"\t\tSerial Number\tProduct Name\n";

for (int i = 0; i < size; i++)

{

cout<<"\t\t "<<ht[i].serialNo<<"\t\t\t"<<ht[i].name<<endl;

}

}

void searchValue(product ht[],int size)

{

int key,flag=0;

cout<<"Enter the Serial Number to be search : ";

cin>>key;

int hash = hashKey(key,size);

if(ht[hash].serialNo==key)

{

cout<<"Key found !\nThe details stored at key "

<<key<<" is as : Serial Number : "

<<ht[hash].serialNo<<"\t Product name : "<<ht[hash].name<<endl;

flag =1;

}

else{

for(int i=0;i<size;i++)

{

if(ht[(hash+i)%size].serialNo == key)

{

cout<<"Key found !\nThe details stored at key "

<<key<<" is as : Serial Number : "

<<ht[(hash+i)%size].serialNo<<"\t Product name : "<<ht[(hash+i)%size].name<<endl;

flag =1;

break;

}

}

}

if(flag==0)

cout<<"\tOops !! Looks like the key is not in table\n";

}

void updateValue(product ht[],int size)

{

int key,flag =0;

char uName[20];

cout<<"Enter the Serial Number to be Update : ";

cin>>key;

int hash = hashKey(key,size);

if(ht[hash].serialNo ==key)

{

cout<<"Enter the updated name of product : ";

cin>>uName;

strcpy(ht[hash].name,uName);

cout<<"\tSuccess ! Data updated at Serial Number : "

<<ht[hash].serialNo<<endl;

flag =1;

}

else

{

for(int i=0;i<size;i++)

{

if(ht[(hash+i)%size].serialNo == key)

{

cout<<"Enter the updated name of product : ";

cin>>uName;

strcpy(ht[(hash+i)%10].name,uName);

cout<<"\tSuccess ! Data updated at Serial Number : "

<<ht[(hash+i)%10].serialNo<<endl;

flag =1;

break;

}

}

}

if(flag==0)

cout<<"\tLooks like the key is not in table\n";

}

void deleteKey(product ht[],int size)

{

int key,flag =0;

char uName[20];

cout<<"Enter the Serial Number to be Delete : ";

cin>>key;

int hash = hashKey(key,size);

if(ht[hash].serialNo ==key)

{

ht[hash].serialNo = -1;

strcpy(ht[hash].name,"");

cout<<"\tSuccess ! Data deleted at Serial Number : "

<<key<<endl;

flag =1;

}

else

{

for(int i=0;i<size;i++)

{

if(ht[(hash+i)%size].serialNo == key)

{

ht[(hash+i)%size].serialNo = -1;

strcpy(ht[(hash+i)%size].name,"");

cout<<"\tSuccess ! Data deleted at Serial Number : "

<<key<<endl;

flag =1;

break;

}

}

}

if(flag==0)

cout<<"\tLooks like the key is not in table\n";

}

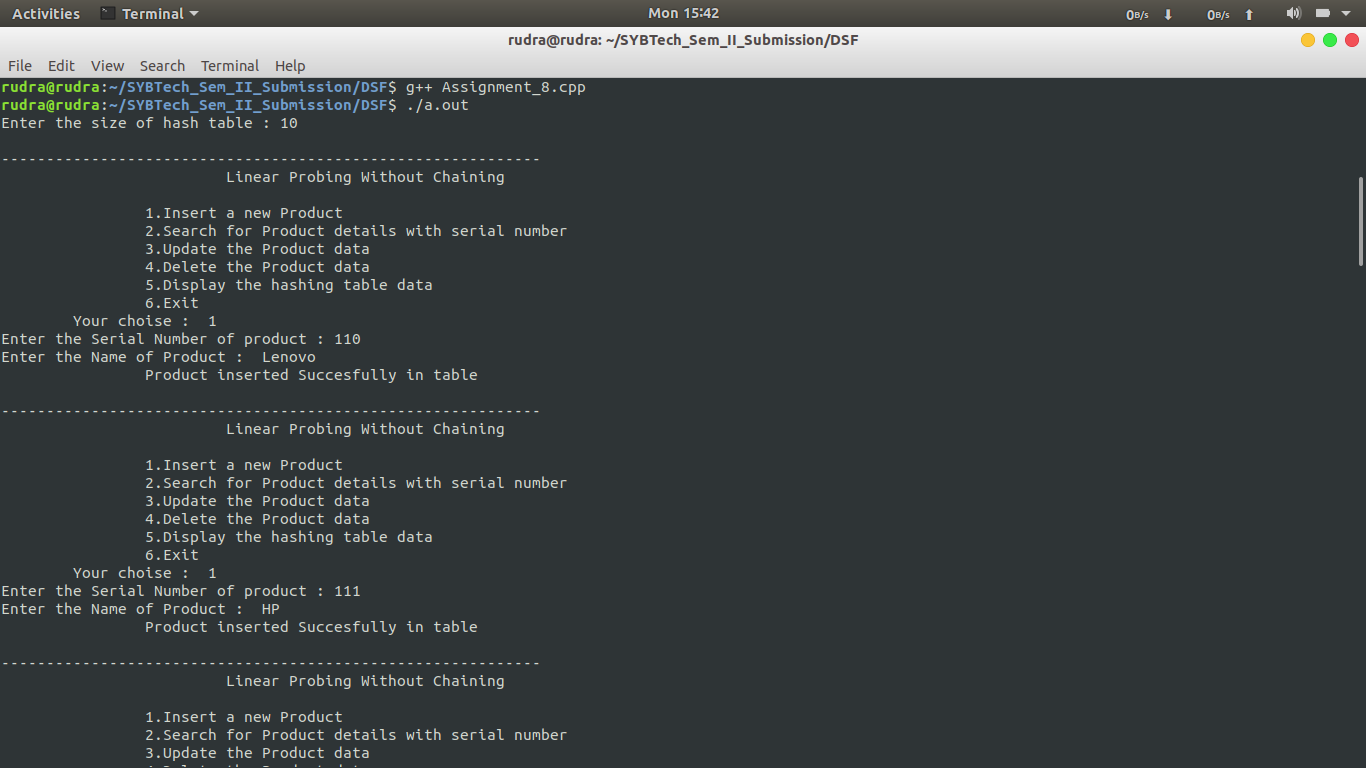
int hashKey(int key,int size)

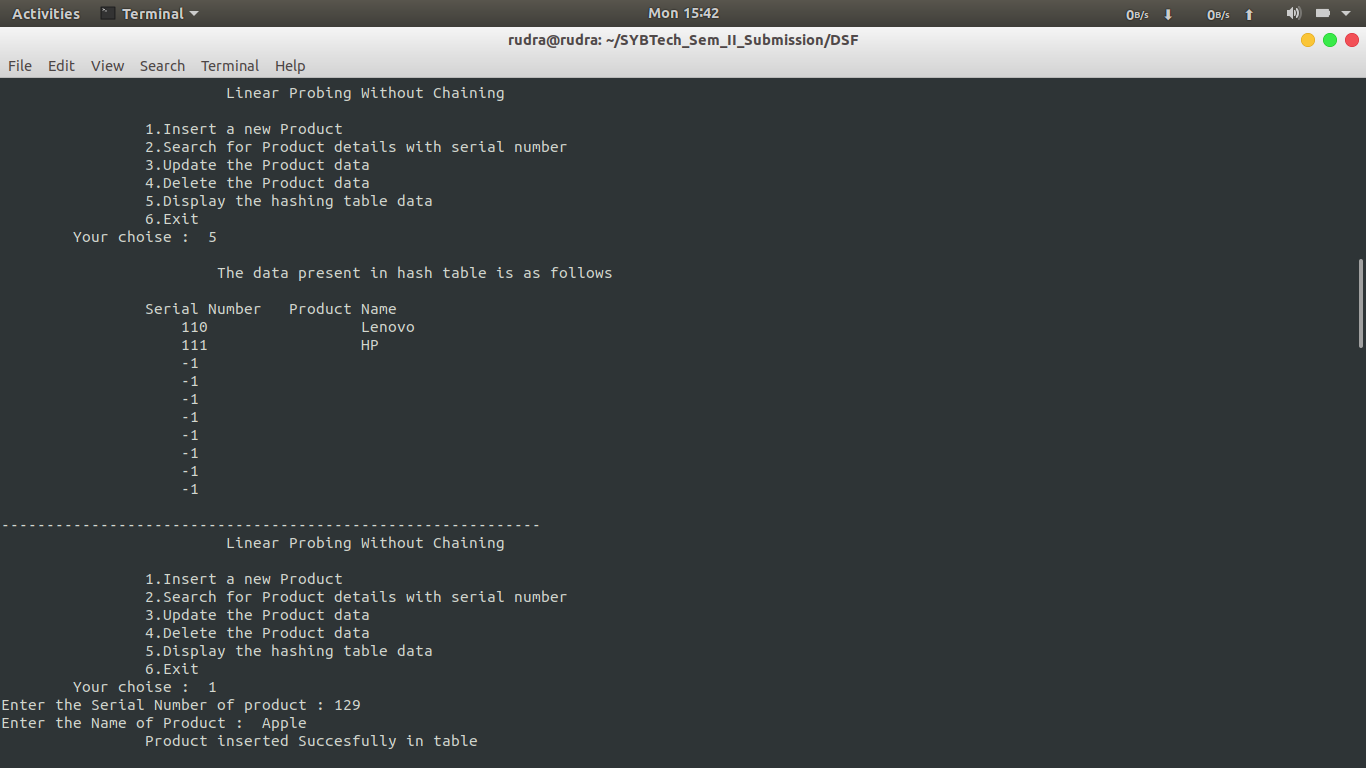
{

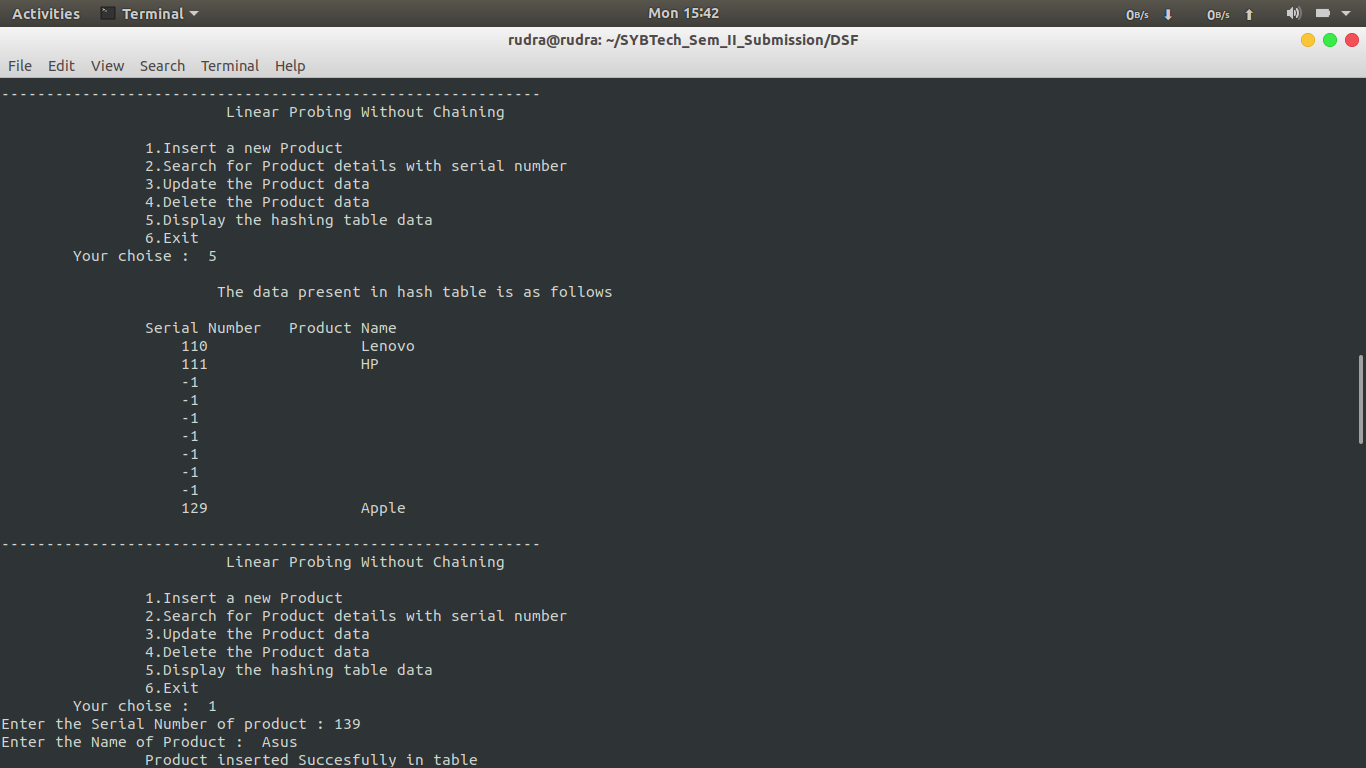
return(key%size);

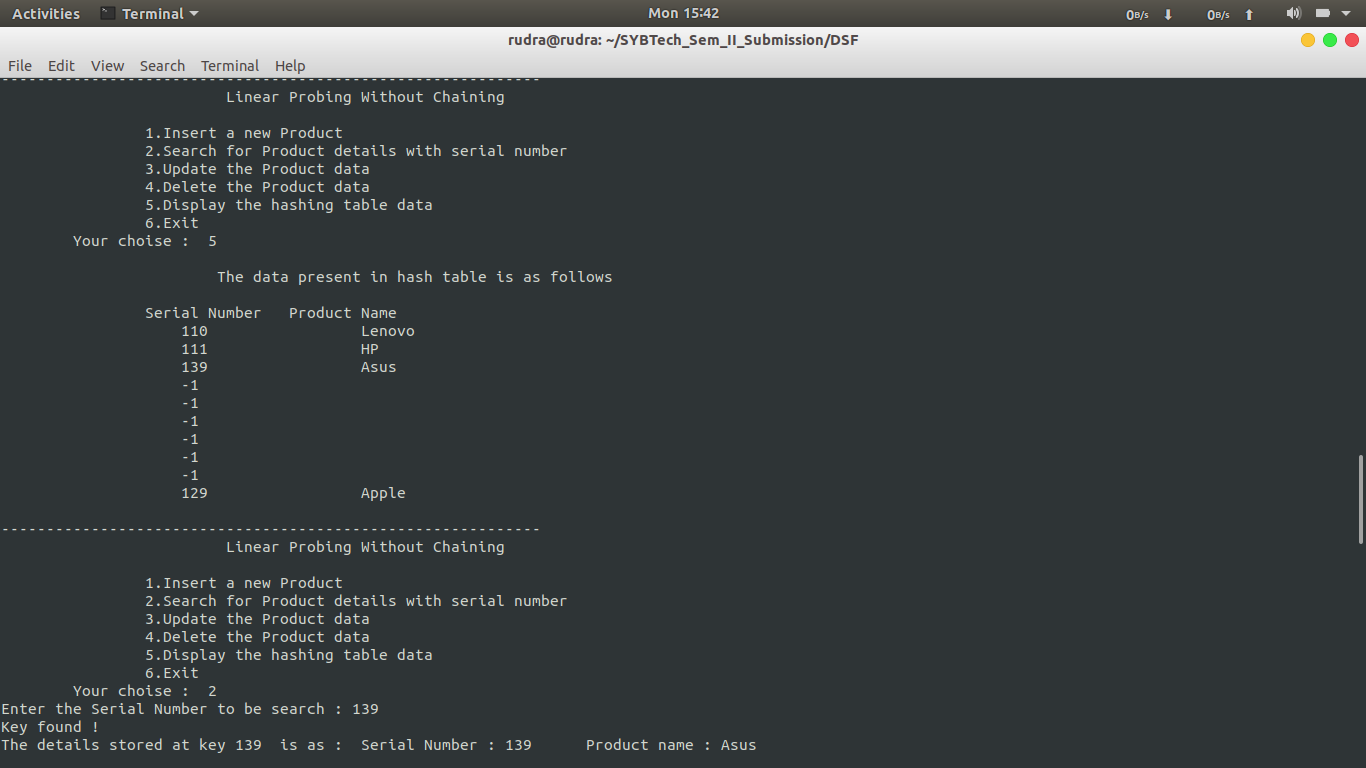
}

**Output:**









**Conclusion :**

Through this assignment we understand the concept of linear probing in hashing to avoid collisions .