09/02/2021

**Experiment No:24**

**HASHING**

**AIM:**

1. Implement a Hash table using Chaining method. Let the size of hash table be 10 so that the index varies from 0 to 9.
2. Implement a Hash table that uses Linear Probing for collision resolution

**DATA STRUCTURES USED:**

Arrays

**ALGORITHM:**

Algorithm Hashing(using Chaining)

START

1. index = value % 10

2. ptr = hash\_table[index]

3. while (ptr->LINK != NULL)

4. ptr = ptr->LINK

5. endwhile

6. new = GetNode(NODE)

7. new->value = value

8. new->link=NULL

9. ptr->link=new

STOP

Algorithm Hashing(using Linear Probing)

START

1. index = value % size

2. if (hash\_table[index] == ∞)

3. hash\_table[index] = key

4. else

5. for i = index+1 till size

6. if (hash[i] == ∞)

7. hash[i] = key

8. return

9. endfor

10. for i = 0 till index-1

11. if (hash[i] == ∞)

12. hash[i] = key

13. return

14. endfor

15 print "Hash table is full!"

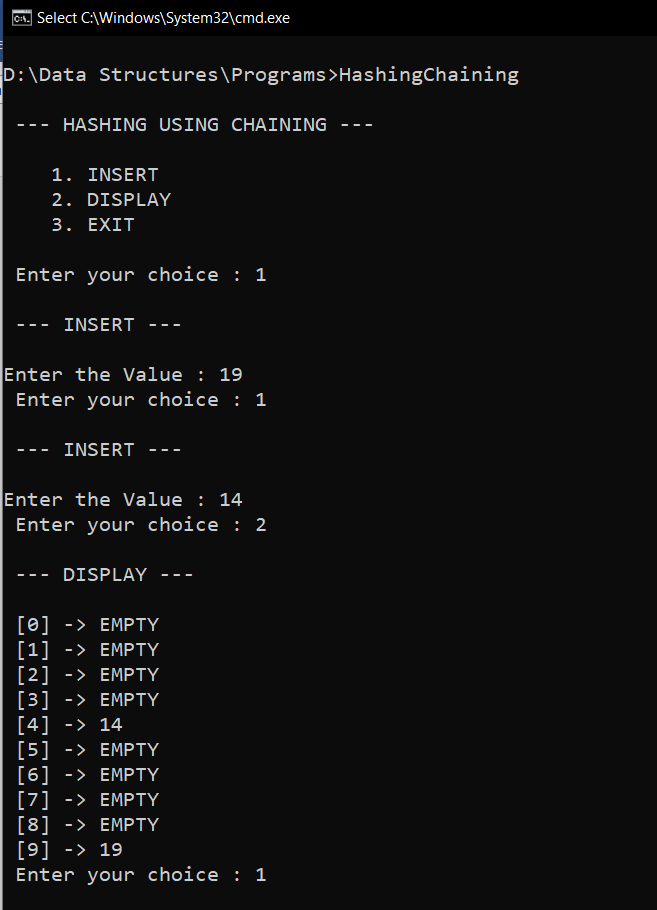
16. endif

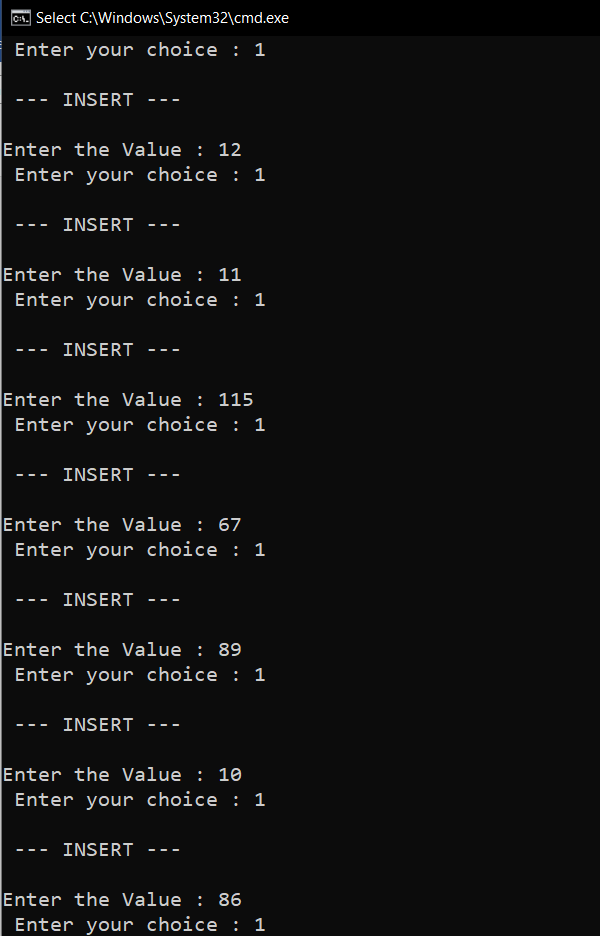
STOP

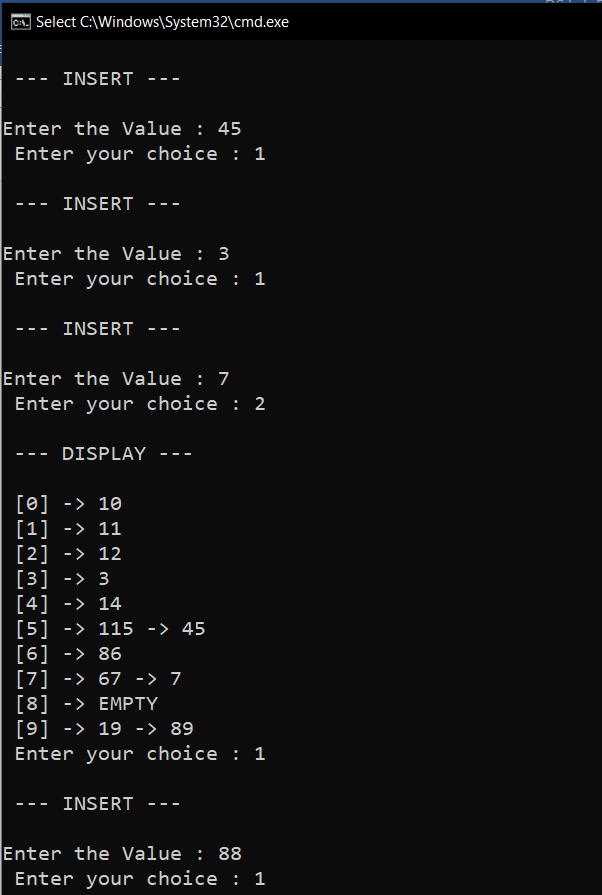
**PROGRAM(1):**

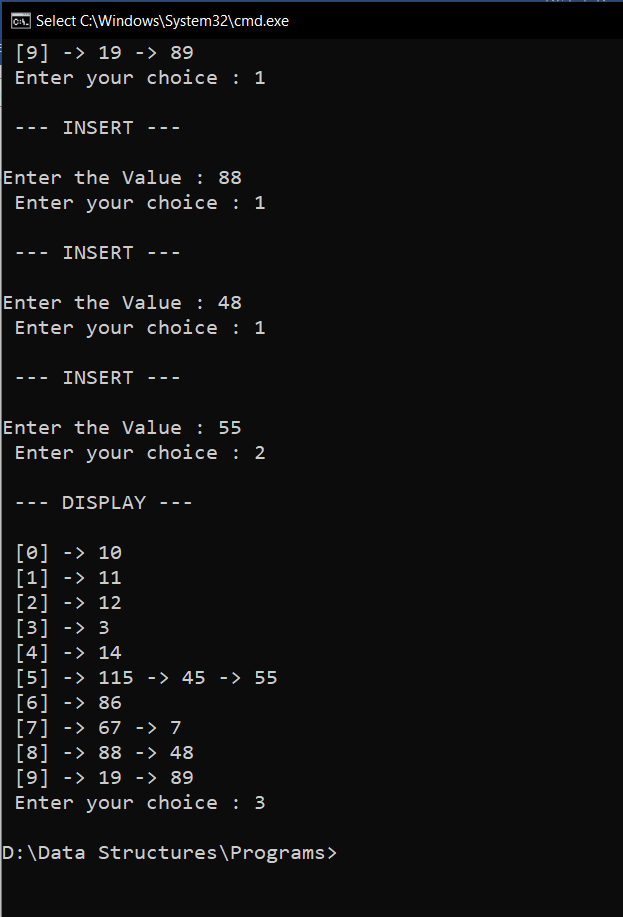
#include<stdio.h>  
#include<stdlib.h>  
struct node {  
 int value;  
 struct node \*link;  
};  
void insert(struct node hash\_table[],int value){  
 int index = value%10;  
 struct node \*ptr = hash\_table;  
 struct node \*new = (struct node\*)malloc(sizeof(struct node));  
 new->link=NULL;  
 new->value = value;  
 ptr=ptr+index;  
 if(ptr->link==NULL){  
 ptr->link=new;  
 }else{  
 while(ptr->link != NULL){  
 ptr=ptr->link;  
 }  
 ptr->link=new;  
 }  
}  
void display(struct node hash\_table[]){  
 struct node \*ptr = hash\_table;  
 for(int i=0;i<10;i++){  
 if(ptr->link==NULL){  
 printf(" [%d] -> EMPTY ",i);  
 }else{  
 printf(" [%d] ",i);  
 struct node \*ptr1=ptr;  
 while(ptr1->link!=NULL){  
 ptr1=ptr1->link;  
 printf("-> %d ",ptr1->value);  
 }  
 }  
 printf("\n");  
 ptr++;  
 }  
  
}  
void main(){  
 int ans=1,op,value,i;  
 struct node hash\_table[10];  
 for(i=0;i<10;i++){  
 hash\_table[i].link=NULL;  
 }  
 printf("\n --- HASHING USING CHAINING --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. DISPLAY \n");  
 printf(" 3. EXIT \n\n");  
 while(ans==1){  
 printf(" Enter your choice : ");  
 scanf("%d",&op);  
 switch(op){  
 case 1 : printf("\n --- INSERT ---\n\n");  
 printf("Enter the Value : ");  
 scanf("%d",&value);  
 insert(hash\_table,value);  
 break;  
 case 2 : printf("\n --- DISPLAY ---\n\n");  
 display(hash\_table);  
 break;  
 case 3 : ans=0;  
 break;  
 default : printf("\n Enter a Valid Input \n");  
 }  
 }  
}

**OUTPUT-(1):**





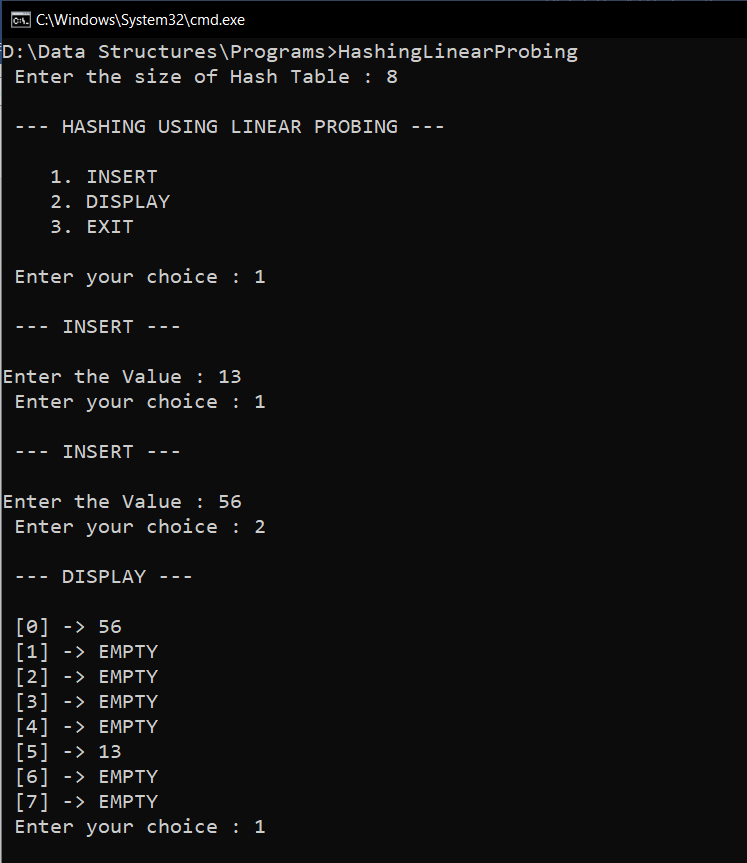


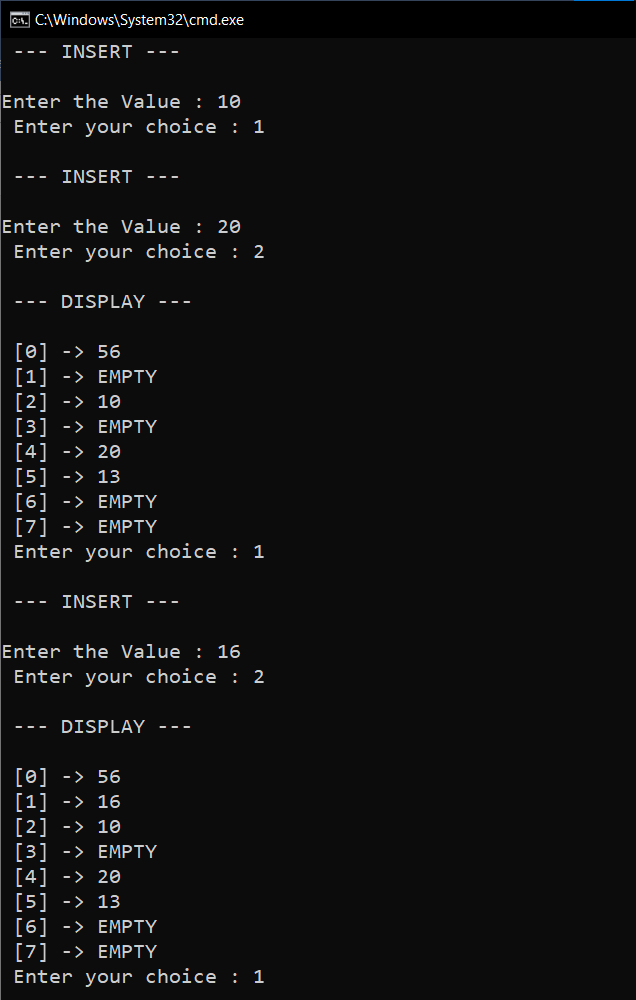


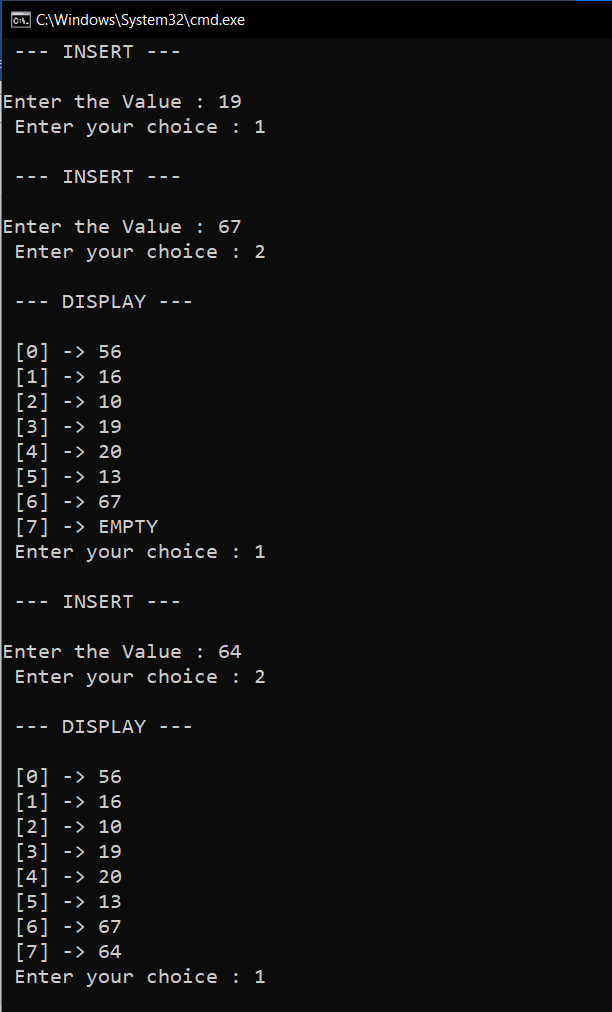
**PROGRAM(2):**

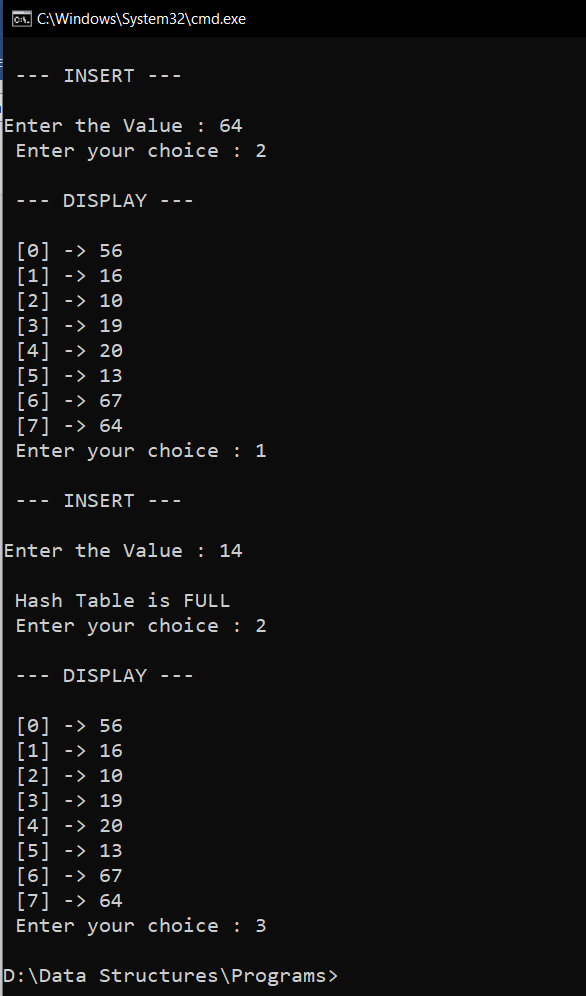
#include<stdio.h>  
#include<stdlib.h>  
#include<math.h>  
void insert(int hash\_table[],int value,int size){  
 int index = value%size;  
 if(hash\_table[index]==(int)INFINITY){  
 hash\_table[index]=value;  
 }else{  
 for(int i= index+1;i<size;i++){  
 if(hash\_table[i]==(int)INFINITY){  
 hash\_table[i]=value;  
 return;  
 }  
 }  
 for(int i=0;i<index;i++){  
 if(hash\_table[i]==(int)INFINITY){  
 hash\_table[i]=value;  
 return;  
 }  
 }  
 printf("\n Hash Table is FULL \n");  
 }  
}  
void display(int hash\_table[],int size){  
  
 for(int i=0;i<size;i++){  
 if(hash\_table[i]==(int)INFINITY){  
 printf(" [%d] -> EMPTY ",i);  
 }else{  
 printf(" [%d] -> %d ",i,hash\_table[i]);  
 }  
 printf("\n");  
 }  
  
}  
void main(){  
 int ans=1,op,value,i,size;  
 printf(" Enter the size of Hash Table : ");  
 scanf("%d",&size);  
 int hash\_table[size];  
 for(i=0;i<size;i++){  
 hash\_table[i]=(int)INFINITY;  
 }  
 printf("\n --- HASHING USING LINEAR PROBING --- \n\n");  
 printf(" 1. INSERT \n");  
 printf(" 2. DISPLAY \n");  
 printf(" 3. EXIT \n\n");  
 while(ans==1){  
 printf(" Enter your choice : ");  
 scanf("%d",&op);  
 switch(op){  
 case 1 : printf("\n --- INSERT ---\n\n");  
 printf("Enter the Value : ");  
 scanf("%d",&value);  
 insert(hash\_table,value,size);  
 break;  
 case 2 : printf("\n --- DISPLAY ---\n\n");  
 display(hash\_table,size);  
 break;  
 case 3 : ans=0;  
 break;  
 default : printf("\n Enter a Valid Input \n");  
 }  
 }  
}

OUTPUT(2):









**RESULT:**

Hash tables are implemented using open hashing (chaining) and closed hashing (linear probing).