Practical-10

1.Hashing.

Aim:

To implement hashing in a fixed-size hash table using separate chaining with linked list and allow insertion of single values or an entire array.

Theory:

Hashing is used to store elements in a fixed-size array called a hash table. A hash function maps each value to an index. When two values map to the same index, a collision occurs. We handle this using separate chaining, where each index stores a linked list of collided values.

Code:

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

#define TABLE_SIZE 10

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

struct node* createnode(int i)
{
   struct node* n = (struct node*)malloc(sizeof(struct node));
   if(n == NULL)
   {
      printf("\nMemory is full");
      exit(0);
   }
```

```
    n->info = i;
    n->next = NULL;
    return n;
}

int hashFunction(int key)
{
    return key % TABLE_SIZE;
}

struct node* insertValue(int key, struct node* start)
{
    struct node* n = createnode(key);

    if(start == NULL)
    {
        start = n;
    }
    else
    {
        n->next = start;
        start = n;
    }
    return start;
}

void display(struct node* hashTable[])
{
    int i;
```

```
    for(i = 0; i < TABLE_SIZE; i++)
    {
        printf("\nIndex %d: ", i);
        struct node* temp = hashTable[i];
        while(temp != NULL)
        {
            printf("%d-->", temp->info);
            temp = temp->next;
        }
        printf("NULL");
    }
}

int main()
{
    struct node* hashTable[TABLE_SIZE];
    int i;
    for(i = 0; i < TABLE_SIZE; i++)
        hashTable[i] = NULL;

    while(1)
    {
        int ch;

        printf("\n\nEnter the number for following choices=\n");
        printf("1.Insert a single value\n");
        printf("2.Insert an array of values\n");
        printf("3.Display hash table\n");
        printf("4.To exit\n");
```

```
scanf("%d", &ch);

switch(ch)
{
  case 1:
  {
    int val;
    printf("\nEnter the value to be stored = ");
    scanf("%d", &val);
    int index = hashFunction(val);
    hashTable[index] = insertValue(val, hashTable[index]);
    break;
  }

  case 2:
  {
    int n, val, i;
    printf("\nEnter how many values you want to insert = ");
    scanf("%d", &n);

    printf("Enter %d values = ", n);
    for(i = 0; i < n; i++)
    {
      scanf("%d", &val);
      int index = hashFunction(val);
      hashTable[index] = insertValue(val, hashTable[index]);
    }
    break;
  }
```

```
        case 3:
            display(hashTable);
            break;

        case 4:
            exit(0);

        default:
            printf("Invalid choice");
            exit(0);
      }
   }
}
```

Output:

```
PS C:\Users\breez\OneDrive - pdpu.ac.in\PDEU S
U STUDY\Sem 3\DSA Lab\Practise-10\" ; if ($?)
Enter the number for following choices=
1.Insert a single value
2.Insert an array of values
3.Display hash table
4.To exit
2

Enter how many values you want to insert = 5
Enter 5 values = 1
2
3
4
5


Enter the number for following choices=
1.Insert a single value
2.Insert an array of values
```

```
Enter the number for following choices=
1.Insert a single value
2.Insert an array of values
3.Display hash table
4.To exit
3

Index 0: NULL
Index 1: 1-->NULL
Index 2: 2-->NULL
Index 3: 3-->NULL
Index 4: 4-->NULL
Index 5: 5-->NULL
Index 6: NULL
Index 7: NULL
Index 8: NULL
Index 9: NULL
```

```
Index 9: NULL

Enter the number for following choices=
1.Insert a single value
2.Insert an array of values
3.Display hash table
4.To exit
1

Enter the value to be stored = 11


Enter the number for following choices=
1.Insert a single value
2.Insert an array of values
3.Display hash table
4.To exit
3
```

```
3.Display hash table
4.To exit
3

Index 0: NULL
Index 1: 11-->1-->NULL
Index 2: 2-->NULL
Index 3: 3-->NULL
Index 4: 4-->NULL
Index 5: 5-->NULL
Index 6: NULL
Index 7: NULL
Index 8: NULL
Index 9: NULL

Enter the number for following choices=
1.Insert a single value
2.Insert an array of values
3.Display hash table
4.To exit
4
PS C:\Users\breez\OneDrive - pdpu.ac.in\
```