ADT using Hashing

Revanth. NM
1BM18CS081

```
typedef struct list
{
    int data;
    struct list *next;
} node_type;

class Dictionary
{
  Public:
        int index;
        Dictionary ();
        void search(int);
        void insert (int);
        void delete_ele (int);
};

Dictionary :: Dictionary ()
{
    intex = -1;
    for(int i=0 ;i<max; i++)
    {
        root [i] = NULL;
        ptr[i] = NULL;
        temp[i] = NULL;
    }
}

void Dictionary :: search (int Key)
{
    int flag=0;
    index = int (Key %max);
    temp [index] = root [index];
    while (temp [index] != NULL) {
        if (temp [index] -> data == key ){
            cout << "\n Search key is found";
            flag=1
            break;
        }
```

Revauth.NM
1BM18CS081

```
        else
            temp [index] = temp [index] -> next ;
    }
    if (flag ~~=~~==0)
        cout << "\n Search Key not found ";
}

void Dictionary :: delete_ele (int Key)
{
    index = int ( Key % max);
    temp[index] = root [index] ;
    while (temp [index] -> data != Key && temp [index] != NULL)
    {
        ptr [index] = temp [index];
        temp[index] = temp [index] -> next;
    }
    ptr [index] -> next = temp [index] -> next ;
    cout << "\n" << temp [index] -> data << " has been deleted.";
    temp [index] -> data = -1;
    temp [index] = NULL;
    free (temp [index]);
}
void Dictionary :: insert (int Key)
{
    index = int ( Key % max);
    ptr[index] = (node_type*) malloc (sizeof(node_type));
    ptr [index] -> data = Key;
    if (root [index] == NULL) {
        root [index] = ptr [index];
        root [index] -> next = NULL;
        temp [index] = ptr[index];
    }
    else {
        temp [index] = root [index];
        while (temp [index] -> next != NULL)
            temp [index] = temp [index] -> next;
        temp [index] -> next = ptr [index];
    }
}
```

Revauth.NM