DSA - Experiment 1

Name: Divyesh Khunt Sapid:60009210116 Batch:A/3

AIM: Implement and Analyse Tower of Hannoi

THEORY: Tower of Hanoi is a mathematical puzzle where we have three rods (A ,B ,C and) and N disks. The smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod in the same manner as it was in the start.

RULES OF THE GAME:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Time complexity: $O(2^n)$

$T(n)=2T(n-1)+1$

Backward substitution:

$T(n-1) = 2T(n-2) + 2$

$T(n-2) = 2T(n-3) + 2$

$2T(n-2) = 2^2\ T(n-3) + 4$

$2^2\ T(n-3) = 2^3\ T(n-3) + 8$

$T(1)=0+2^n$

General equation is $T(n)=2^n$

Space complexity:O(n)

Space of recursive stack is of order n so space complexity is O(n)

CODE:

main.c

```c
#include<conio.h>
#include <stdio.h>

void tower(int n,char frompeg,char topeg,char auxpeg)
{
    if(n==1)
    {
        printf("Move disk 1 form peg %c to peg %c \n",frompeg ,topeg);
        return;
    }
    tower(n-1,frompeg,auxpeg,topeg);     //recall

     printf("Move disk %d form peg %c to peg %c\n",n,frompeg ,topeg);
     tower(n-1,auxpeg,topeg,frompeg);
}
int main()
{
printf("Divyesh khunt\n60009210116\n");
   int disk;
   char A,B,C;
   printf("enter no. of disks\n");
   scanf("%d",&disk);
    tower(disk, 'A','C','B' );
    return 0;
}
```

OUTPUT:

```
Divyesh khunt
60009210116
enter no. of disks
3
Move disk 1 form peg A to peg C
Move disk 2 form peg A to peg B
Move disk 1 form peg C to peg B
Move disk 3 form peg A to peg C
Move disk 1 form peg B to peg A
Move disk 2 form peg B to peg C
Move disk 1 form peg A to peg C


...Program finished with exit code 0
Press ENTER to exit console.
```

CONCLUSION:

Thus the code of tower of hannoi was analysed and implemented.

The time complexity of tower of hannoi is $2^n - 1$

DSA - Experiment 2

Name: Divyesh Khunt **Sapid:**60009210116 **Batch: A/3**

Aim: To implement and analyze Insertion and Selection sort.

Selection Sort

Theory:

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

Time Complexity: the time complexity of Selection Sort is O(N2) as there are two nested loops

- one loop to select an element of Array one by one = O(N)
- Another loop to compare that element with ever other Array element

O(N)

therefore overall complexity O(N*N) =$O(N^2)$

CODE:

```c
int main()
{

int a[100], n, i, j, min, temp;
printf("Enter value of n: ");
scanf("%d", &n);
printf("Enter array: ");
for(i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
for(i=0; i<=n-2; i++)
{
min=i;
for(j=i+1; j<=n-1; j++)
{ if(a[j]<a[min]) min=j;

}
temp=a[i]; a[i]=a[min]; a[min]=temp;
} printf("The sorted array is: ");
for (i=0; i<n; i++) {

printf("%d\n", a[i]);
}
}
```

OUTPUTS:

```
Enter value of n: 5
Enter array: 23
45
1
45
7
The sorted array is: 1
7
23
45
45


--------------------------------
Process exited after 5.355 seconds with return value 0
Press any key to continue . . .
```

Insertion sort

Theory:

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Time Complexity :

Since there is a while loop enclosed by the for loop the time complexity adds up to $O(N^2)$=O(N*N) CODE:

```c
#include<stdio.h>
#include<conio.h>
int main()
 {
int a[100], i, n, j, temp;
 printf("Enter value of n: ");
 scanf("%d", &n);
printf("Enter the array: ");
for(i=0; i<n; i++) {
scanf("%d", &a[i]);
}
for(i=1; i<=n-1; i++)
 {
 temp=a[i]; j=i-1;
 while(j>=0 && a[j]>temp)
    {
    a[j+1]=a[j];
    j=j-1;
    } a[j+1]=temp;

  }
 printf("The sorted array is: ");
 for(i=0; i<n; i++)
{
printf("%d \n", a[i]);
}
}
```

OUTPUTS:

```
Enter value of n: 5
Enter the array: 53
1
43
87
6
The sorted array is: 1
6
43
53
87

--------------------------------
Process exited after 13.57 seconds with return value 0
Press any key to continue . . .
```

Conclusion: Thus insertion and selection sort were implemented.

EXP 3

NAME: DIVYESH KHUNT **SAPID:60009210116**

Aim: To implement and analyse Merge sort and Quick sort.

Merge Sort

Theory:

The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

Time Complexity:

O (N log(N)), Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$T(n) = 2T(n/2) + \theta(n)$ CODE:

```
#include<stdio.h>
#include<conio.h>
void quicksort(int number[25], int first, int last)
{
    int i, j, pivot, temp;
    if(first<last)
    {
        pivot=first;
        i=first;
        j=last;
        while(i<j)
        {
            while(number[i]<=number[pivot] && i<last)
            i++;
            while(number[j]>number[pivot])
            j--;
            if(i<j)
            {
                temp=number[i];
                number[i]=number[j];
                number[j]=temp;
            }
        }
        temp=number[pivot];
        number[pivot]=number[j];
        number[j]=temp;
        quicksort(number, first, j-1);
        quicksort(number, j+1, last);
    }
}
```

```
int main()
{
    int a[100], i, n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Enter the array: ");
    for(i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
    quicksort(a, 0, n-1);
    printf("The sorted array is: ");
    for(i=0; i<n; i++)
    {
        printf("%d", a[i]);
    }
}
```

OUTPUTS:

```
Enter a number: 5
Enter the array: 5 4 3 2 1
The sorted array is: 12345
```

Quick sort

Theory:

Quicksort is a Divide and Conquer algorithm. It picks an Like Merge Sort element as a pivot and partitions the given array around the picked pivot. There are many different versions of quicksort that pick pivot in different ways.

1. Always pick the first element as a pivot.
2. Always pick the last element as a pivot (implemented below)
3. Pick a random element as a pivot.
4. Pick median as the pivot.

   Time Complexity :

   T(n) = T(k)+T(n-k-1)+O(n)Where k is no. of elements that are smaller than pivot.

CODE:

```c
#include<stdio.h>

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
int a[30],n,i;
printf("Enter no of elements:");
scanf("%d",&n);
printf("Enter array elements:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
mergesort(a,0,n-1);
printf("\nSorted array is :");
for(i=0;i<n;i++)
printf("%d ",a[i]);
return 0;
}

void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
mid=(i+j)/2;
mergesort(a,i,mid);
```

```
26  mid=(i+j)/2;
27  mergesort(a,i,mid);
28  mergesort(a,mid+1,j);
29  merge(a,i,mid,mid+1,j);
30  }
31  }
32
33  void merge(int a[],int i1,int j1,int i2,int j2)
34  {
35  int temp[50];
36  int i,j,k;
37  i=i1;
38  j=i2;
39  k=0;
40  while(i<=j1 && j<=j2)
41  {
42  if(a[i]<a[j])
43  temp[k++]=a[i++];
44  else
45  temp[k++]=a[j++];
46  }
47  while(i<=j1)
48  temp[k++]=a[i++];
49  while(j<=j2)
50  temp[k++]=a[j++];
51  for(i=i1,j=0;i<=j2;i++,j++)
52  a[i]=temp[j];
53  }
```

OUTPUTS:

```
Enter no of elements:5
Enter array elements:9 1 7 4 8

Sorted array is :1 4 7 8 9
```

Conclusion: Thus insertion and selection sort were implemented.

EXP 4
LINKED LIST

NAME: DIVYESH KHUNT **SAPID:60009210116**

Aim: To create and implement a linked list in c programming

Theory:

Linked is a data structure which uses dynamic memory allocation. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers. They include a series of connected nodes.
Here, each node stores the data and the address of the next node.
Advantages of Linked Lists over arrays:

- Dynamic Array.
- Ease of Insertion/Deletion.

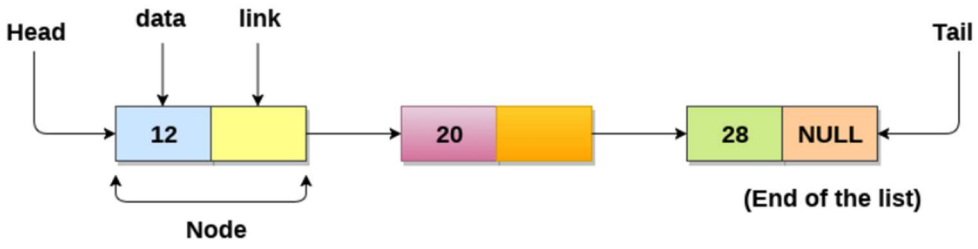
Drawbacks of Linked Lists:

- Random access is not allowed. We have to access elements sequentially starting from the first node(head node). So we cannot do a binary search with linked lists efficiently with its default implementation.
- Extra memory space for a pointer is required with each element of the list.
- Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Time Complexity :

For searching is O(n)

For insertion and deletion is O(1)



CODE:

```c
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<stdlib.h>
struct node {
 int data ;
 struct node *next ;
};
struct node *start = NULL ;
struct node *create_ll (struct node *) ;
struct node *insert_begin (struct node *) ;
struct node *insert_end (struct node *) ;
struct node *insert_before (struct node *) ;
struct node *insert_after (struct node *) ;
struct node *display (struct node *) ;
struct node *delete_beg (struct node *) ;
struct node *delete_end (struct node *) ;
struct node *delete_after (struct node *) ;
struct node *delete_node (struct node *) ;
struct node *delete_list (struct node *) ;
struct node *sort_ll (struct node *) ;

int main () {

 int ch ;

 printf("Press 1 to create a linked list\n") ;
 printf("Press 2 to insert at the begin\n") ;
 printf("Press 3 to insert at the end\n") ;
 printf("Press 4 to insert before an element\n") ;
 printf("Press 5 to insert after an element\n") ;
 printf("Press 6 to display the linked list\n") ;
 printf("Press 7 to delete the first element\n") ;
 printf("Press 8 to delete the last element\n") ;
 printf("Press 9 to delete after an element\n") ;
 printf("Press 10 to delete a particular node\n") ;
 printf("Press 11 to delete the entire linked list\n") ;
 printf("Press 12 to sort the linked list\n") ;
 printf("Press 13 to exit this program\n") ;
```

```
41  do {
42  scanf("%d", &ch) ;
43  switch (ch) {
44  case 1 :
45  start = create_ll (start) ;
46  printf("list is created\n") ;
47  break ;
48  case 2 :
49  start = insert_begin (start) ;
50  break ;
51  case 3 :
52  start = insert_end (start) ;
53  break ;
54  case 4 :
55  start = insert_before (start) ;
56  break ;
57  case 5 :
58  start = insert_after (start) ;
59  break ;
60  case 6 :
61  start = display (start) ;
62  break ;
63  case 7 :
64  start = delete_beg (start) ;
65  break ;
66  case 8 :
67  start = delete_end (start) ;
68  break ;
69  case 9 :
70  start = delete_after (start) ;
71  break ;
72  case 10 :
73  start = delete_node (start) ;
74  break ;
75  case 11 :
76  start = delete_list (start) ;
77  break ;
78  case 12 :
79  start = sort_ll (start) ;
```

```
80      break ;
81    }
82    }while(ch != 13) ;
83  }
84
85  struct node *create_ll(struct node *start) {
86
87    struct node *nn, *ptr ;
88    int x ;
89    printf("enter -1 to stop\n") ;
90    printf("enter a number: \n") ;
91    scanf("%d", &x) ;
92
93    while(x != -1) {
94    nn = (struct node *)malloc(sizeof(struct node)) ;
95    nn -> data = x ;
96    if(start == NULL) {
97    nn -> next = NULL ;
98    start = nn ;
99    }
100   else {
101   nn -> next = NULL ;
102   ptr = start ;
103   while(ptr -> next != NULL) {
104   ptr = ptr -> next ;
105   }
106   ptr -> next = nn ;
107   }
108   printf("Enter a no: ") ;
109   scanf("%d", &x) ;
110   }
111   return start ;
112 }
```

```
114 struct node *insert_begin (struct node * start) {
115
116   struct node *nn ;
117   int x ;
118   printf("Enter a number to at beg: ") ;
119   scanf("%d" , &x) ;
120   nn = (struct node *)malloc(sizeof(struct node)) ;
121   nn -> data = x ;
122   nn -> next = start ;
123   start = nn ;
124   return start ;
125 }
```

```
127 struct node *insert_end (struct node *start) {
128   struct node *nn, *ptr ;
129   int x ;
130   printf("Enter a no to add at end: ") ;
131   scanf("%d", &x) ;
132   nn = (struct node *)malloc(sizeof(struct node)) ;
133   nn -> data = x ;
134   ptr = start ;
135   while(ptr -> next != NULL) {
136   ptr = ptr -> next ;
137   }
138   ptr -> next = nn ;
139   nn -> next = NULL ;
140
141   return start ;
142 }
```

```
144 struct node *insert_before (struct node *start) {
145   struct node *nn, *pp, *ptr ;
146   int x, val ;
147   printf("enter a number befor which your no. is to added: ") ;
148   scanf("%d", &x) ;
149   nn = (struct node *)malloc(sizeof(struct node)) ;
150   nn -> data = x ;
151   printf("enter the number you want to insert: ") ;
152   scanf("%d", &val) ;
153   ptr = start ;
154   while(ptr -> data != val) {
155   pp = ptr ;
156   ptr = ptr -> next ;
157   pp -> next = nn ;
158   nn -> next = ptr ;
159   }
160   return start ;
161 }
162 struct node *insert_after (struct node *start) {
163   struct node *nn, *pp, *ptr ;
164   int x ,val ;
165   printf("enter a number after which your no. is to added: ") ;
166   scanf("%d", &x) ;
167   nn = (struct node *)malloc(sizeof(struct node)) ;
168   nn -> data = x ;
169   printf("Enter the value: ") ;
170   scanf("%d", &val) ;
171   pp = start ;
172   ptr = start ;
173   while(pp -> data != val) {
174   pp = ptr ;
175   ptr = ptr -> next ;
176   }
177   pp -> next ;
178   nn-> next = ptr ;
179   return start ;
180 }
```

```
181 struct node *display (struct node *start) {
182   struct node *ptr ;
183   ptr = start ;
184   while (ptr != NULL) {
185   printf("Data is : %d\n", ptr -> data) ;
186   ptr = ptr -> next ;
187   }
188   return start ;
189 }
190 struct node *delete_beg (struct node *start) {
191   struct node *ptr ;
192   ptr = start ;
193   start = start -> next ;
194   printf("Data to be deleted is: %d\n", ptr -> data) ;
195   free(ptr) ;
196   return start ;
197 }
198 struct node *delete_end (struct node * start) {
199   struct node *ptr, *pp ;
200   pp = start ;
201   ptr = start ;
202
203   while(ptr -> next != NULL) {
204   pp = ptr ;
205   ptr = ptr -> next ;
206   }
207   pp -> next = NULL ;
208   printf("Node to be deleted is: %d\n", ptr -> data) ;
209   free(ptr) ;
210   return start ;
211 }
```

```
212 struct node *delete_after (struct node *start) {
213   int val ;
214   struct node *pp, *ptr ;
215   ptr = start ;
216   pp = start ;
217   printf("Enter a value after which u want to delete a node: ") ;
218   scanf("%d", &val) ;
219   while(pp -> data != val) {
220   pp = ptr ;
221   ptr = ptr -> next ;
222   }
223   pp -> next - ptr -> next ;
224   printf("deleted node is %d\n", ptr -> data ) ;
225   free(ptr) ;
226   return start ;
227 }
```

```
228 struct node *delete_node (struct node *start) {
229   struct node *ptr = start, *pp;
230   int x ;
231   printf("Enter a value to delete: ") ;
232   scanf("%d", &x) ;
233   while(ptr -> data != x) {
234   pp = ptr ;
235   ptr = ptr -> next ;
236   }
237   pp -> next = ptr -> next ;
238   free(pp) ;
239   return start ;
240 }
241 struct node *delete_list (struct node *start) {
242   while(start != NULL) {
243   printf("Data deleted %d\n", start -> data) ;
244   start = delete_beg(start) ;
245   }
246   return start ;
247 }
```

```
248 struct node *sort_ll (struct node *start) {
249   struct node *ptr1, *ptr2 ;
250   int temp ;
251   ptr1 = start ;
252   while (ptr2 -> next = NULL) {
253   ptr2 = ptr1 -> next ;
254   while(ptr2 != NULL) {
255   if (ptr1 -> data > ptr2 -> data) {
256   temp = ptr1 -> data ;
257   ptr1 -> data = ptr2 -> data ;
258   ptr2 -> data = temp ;
259   }
260   ptr2 = ptr2 -> next ;
261   }
262   ptr1 = ptr1 -> next ;
263   }
264   return start ;
265
266 }
```

OUTPUTS:

Creation and display of LL

```
Press 1 to create a linked list
Press 2 to insert at the begin
Press 3 to insert at the end
Press 4 to insert before an element
Press 5 to insert after an element
Press 6 to display the linked list
Press 7 to delete the first element
Press 8 to delete the last element
Press 9 to delete after an element
Press 10 to delete a particular node
Press 11 to delete the entire linked list
Press 12 to sort the linked list
Press 13 to exit this program
1
enter -1 to stop
enter a number:
23
Enter a no: 46
Enter a no: 32
Enter a no: 78
Enter a no: 1
Enter a no: -1
list is created
6
Data is : 23
Data is : 46
Data is : 32
Data is : 78
Data is : 1
```

Deletion of data

```
Press 3 to insert at the end
Press 4 to insert before an element
Press 5 to insert after an element
Press 6 to display the linked list
Press 7 to delete the first element
Press 8 to delete the last element
Press 9 to delete after an element
Press 10 to delete a particular node
Press 11 to delete the entire linked list
Press 12 to sort the linked list
Press 13 to exit this program
1
enter -1 to stop
enter a number:
5
Enter a no: 4
Enter a no: 3
Enter a no: -1
list is created
7
Data to be deleted is: 5
6
Data is : 4
Data is : 3
```

```
Press 1 to create a linked list
Press 2 to insert at the begin
Press 3 to insert at the end
Press 4 to insert before an element
Press 5 to insert after an element
Press 6 to display the linked list
Press 7 to delete the first element
Press 8 to delete the last element
Press 9 to delete after an element
Press 10 to delete a particular node
Press 11 to delete the entire linked list
Press 12 to sort the linked list
Press 13 to exit this program
1
enter -1 to stop
enter a number:
1
Enter a no: 2
Enter a no: 3
Enter a no: 4
Enter a no: 5
Enter a no: -1
list is created
11
Data deleted 1
Data to be deleted is: 1
Data deleted 2
Data to be deleted is: 2
Data deleted 3
Data to be deleted is: 3
Data deleted 4
Data to be deleted is: 4
Data deleted 5
Data to be deleted is: 5
```

Conclusion:

Thus linked list was created and manny operations on it were performed successfully

EXPERIMENT 5

NAME: DIVYESH KHUNT **SAPID:60009210116**

PARENTESES

AIM: To implement parentheses checker i.e. to check the expression is valid or not.

THEORY:

The balanced parentheses problem is one of the common programming problems that is also known as Balanced brackets. This problem is commonly asked by the interviewers where we have to validate whether the brackets in a given string are balanced on not.
Characters such as "(", ")", "[", "]", "{", and "}" are considered brackets

Time Complexity

The **time complexity** of the parenthesis checker implementation using stack is O(n) where n is the length of the input expression, as we are traversing the string character by character using for loop.

Space Complexity

The space complexity of the parenthesis checker implementation using stack is O(n) where n is the length of the input expression, as we are storing the opening parenthesis characters in a stack.

CODE:

```c
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 10
int top =-1;
int stk[MAX];
void push(char);
char pop();
int main() {
    char exp[MAX],temp;
    int i, flag=1;
    printf("Enter an expression : ");
    gets(exp);
    for(i=0; i<strlen(exp); i++) {
        if(exp[i]=='(' || exp[i]=='{' || exp[i]=='[')
            push(exp[i]);
        if(exp[i]==')' || exp[i]=='}' || exp[i]==']')
            if(top == -1)
                flag=0;
            else {
                temp=pop();
                if(exp[i]==')' && (temp=='{' || temp=='['))
                    flag=0;
                if(exp[i]=='}' && (temp=='(' || temp=='['))
                    flag=0;
                if(exp[i]==']' && (temp=='(' || temp=='{'))
                    flag=0;
            }
    }
    if(top>=0)
        flag=0;
    if(flag==1)
        printf("\n Valid expression");
    else
        printf("\n Invalid expression");
}
void push(char c) {
    if(top == (MAX-1))
        printf("Stack Overflow\n");
    else {
        top=top+1;
        stk[top] = c;
    }
}
char pop() {
    if(top == -1)
        printf("\n Stack Underflow");
    else
        return(stk[top--]);
}
```

OUTPUT:

```
Enter an expression : (A+(B-C))

 Valid expression
--------------------------------
Process exited after 13.13 seconds with return value 0
Press any key to continue . . .
```

```
Enter an expression : (A+{B*D))

 Invalid expression
--------------------------------
Process exited after 34.22 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION:

Thus we can check if the expression is valid or not with the help of stack.

EXPERIMENT 6

NAME: DIVYESH KHUNT **SAPID:60009210116**

INFIX TO POSTFIX

AIM: To convert an infix expression to postfix expression

Theory:

Infix Expression: In infix expression, an operator is placed between the two operands. Example: x + y, here operator + is placed between operands x and y.

Postfix Expression: In postfix expression, an operator is placed after the operands. Example: xy+, here operator + is placed after the operands x and y.

Infix Character Scanned	Stack	Postfix Expression
	(	
A	(	A
-	(-	A
(	(- (	A
B	(- (	A B
/	(- (/	A B
C	(- (/	A B C
+	(- (+	A B C /
(	(- (+ (	A B C /
D	(- (+ (	A B C / D
%	(- (+ (%	A B C / D
E	(- (+ (%	A B C / D E
*	(- (+ (% *	A B C / D E
F	(- (+ (% *	A B C / D E F
)	(- (+	A B C / D E F * %
/	(- (+ /	A B C / D E F * %
G	(- (+ /	A B C / D E F * % G
)	(-	A B C / D E F * % G / +
*	(- *	A B C / D E F * % G / +
H	(- *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * -

The precedence of these operators can be given as follows:
Higher priority *, /, %
Lower priority +, –

CODE:

```c
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
#include<stdlib.h>
#define MAX 100
char st[MAX];
int top = -1;
void push (char st[], char);
char pop (char st[]);
void InfixtoPostfix (char source[], char target[]);
int getPriority (char);
int main ()
{
char infix[100], postfix[100];
printf ("\n Enter any infix expression : ");
gets (infix);
strcpy (postfix, "");
InfixtoPostfix (infix, postfix);
printf ("\n The corresponding postfix expression is : ");
puts (postfix);
getch ();
return 0;
}
void InfixtoPostfix (char source[], char target[])
{
int i = 0, j = 0;
char temp;
strcpy (target, "");
while (source[i] != '\0')
    {
if (source[i] == '(')
    {
    push (st, source[i]);
    i++;
    }
      else if (source[i] == ')')
  {
```

```
while ((top != -1) && (st[top] != '('))
    {
    target[j] = pop (st);
    j++;
    }

temp = pop (st);
    i++;
}
    else if (isdigit (source[i]) || isalpha (source[i]))
    {
target[j] = source[i];
j++;
i++;
}
    else if (source[i] == '+' || source[i] == '-' || source[i] == '*'
        ||
source[i] == '/' || source[i] == '%')

    {

while ((top != -1) && (st[top] != '(') && (getPriority (st[top]) >getPriority (source[i])))
      {
target[j] = pop (st);
j++;
}
  push (st, source[i]);
  i++;
}
    else
    {
printf ("\n INCORRECT ELEMENT IN EXPRESSION");
exit(1);
}
}
```

```
while ((top != -1) && (st[top] != '('))
    {
target[j] = pop (st);
j++;
}
target[j] = '\0';
}
int getPriority (char op)
{
if (op == '/' || op == '*' || op == '%')
return 1;
  else if (op == '+' || op == '-')
return 0;
}
```

```
89   void push (char st[], char val)
90   {
91
92       if (top == MAX-1)
93
94       printf ("\n STACK OVERFLOW");
95        else
96       {
97       top++;
98       st[top] = val;
99       }
100  }
101
102  char pop (char st[])
103  {
104  char val = ' ';
105  if (top == -1)
106      printf ("\n STACK UNDERFLOW");
107  else
108      {
109      val = st[top];
110      top --;
111      }
112  return val;
113  }
```

OUTPUT

```
Enter any infix expression : (A+B)/(C+D)-(D*E)

The corresponding postfix expression is : AB+CD+/DE*-
```

CONCLUSION:

Thus, an application of stack to convert infix to postfix was performed successfully

EXP 7

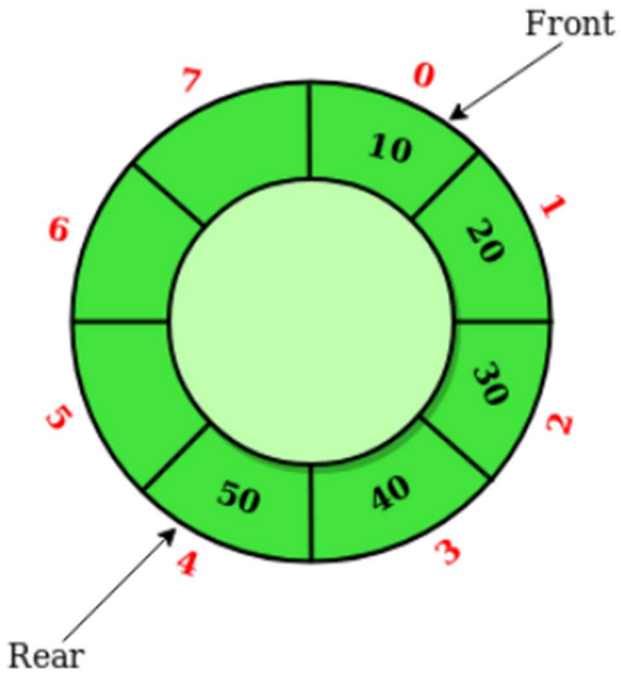
NAME: DIVYESH KHUNT **SAPID:60009210116**

Circular Queue

AIM: To create and implement circular queue as an abstract data type

THEORY:

A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle.



Time Complexity: Time complexity of dequeue() operation is O(1) as there is no loop in any of the operation.

CODE:

```c
#include <stdio.h>
#include <conio.h>
#define MAX 10
int queue[MAX];
int front=-1, rear=-1;
void insert(void);
int delete_element(void);
int peek(void);
void display(void);
int main() {
    int option, val;
    do {
        printf("\n Press 1 Insert an element");
        printf("\n Press 2 Delete an element");
        printf("\n Press 3 Peek");
        printf("\n Press 4 Display the queue");
        printf("\n Press 5 EXIT");
        printf("\n Enter your option : ");
        scanf("%d", &option);
        switch(option) {
            case 1:
                insert();
                break;
            case 2:
                val = delete_element();
                if(val!=-1)
                    printf("\n The number deleted is : %d", val);
                break;
            case 3:
                val = peek();
                if(val!=-1)
                    printf("\n The first value in queue is : %d", val);
                break;
            case 4:
                display();
                break;
        }
    } while(option!=5);
```

```
    return 0;
}
void insert() {
    int num;
    printf("\n Enter the number to be inserted in the queue : ");
    scanf("%d", &num);
    if(front==0 && rear==MAX-1)
        printf("\n OVERFLOW");
    else if(front==-1 && rear==-1) {
        front=rear=0;
        queue[rear]=num;
    } else if(rear==MAX-1 && front!=0) {
        rear=0;
        queue[rear]=num;
    } else {
        rear++;
        queue[rear]=num;
    }
}
int delete_element() {
    int val;
    if(front==-1 && rear==-1) {
        printf("\n UNDERFLOW");
        return -1;
    }
    val = queue[front];
    if(front==rear)
        front=rear=-1;
    else {
        if(front==MAX-1)
            front=0;
        else
            front++;
    }
    return val;
}
int peek() {
    if(front==-1 && rear==-1) {
        printf("\n QUEUE IS EMPTY");
        return -1;
    } else {
        return queue[front];
    }
}
void display() {
    int i;
    printf("\n");
    if (front ==-1 && rear==-1)
        printf ("\n QUEUE IS EMPTY");
    else {
        if(front<rear) {
            for(i=front; i<=rear; i++)
                printf("\t %d", queue[i]);
        } else {
            for(i=front; i<MAX; i++)
                printf("\t %d", queue[i]);
            for(i=0; i<=rear; i++)
                printf("\t %d", queue[i]);
        }
    }
}
```

OUTPUT:

```
Press 1 Insert an element
Press 2 Delete an element
Press 3 Peek
Press 4 Display the queue
Press 5 EXIT
Enter your option : 1

Enter the number to be inserted in the queue : 25

Press 1 Insert an element
Press 2 Delete an element
Press 3 Peek
Press 4 Display the queue
Press 5 EXIT
Enter your option : 1

Enter the number to be inserted in the queue : 23

Press 1 Insert an element
Press 2 Delete an element
Press 3 Peek
Press 4 Display the queue
Press 5 EXIT
Enter your option : 4

        25      23
```

```
Press 1 Insert an element
Press 2 Delete an element
Press 3 Peek
Press 4 Display the queue
Press 5 EXIT
Enter your option : 2

The number deleted is : 25
Press 1 Insert an element
Press 2 Delete an element
Press 3 Peek
Press 4 Display the queue
Press 5 EXIT
Enter your option : 3

The first value in queue is : 23
```

CONCLUSION: Thus circular queue was implemented successfully

EXP 8

NAME: DIVYESH KHUNT **SAPID:60009210116**

Priority Queue

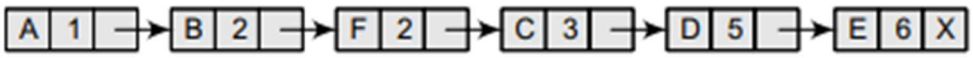
Aim: To create an type of queue (priority queue)

THEORY:

Priority Queue is an abstract data type that is similar to a queue, and every element has some priority value associated with it. The priority of the elements in a priority queue determines the order in which elements are served (i.e., the order in which they are removed).

FRONT	REAR
3	3
1	3
4	5
4	1

	1	2	3	4	5
1			A		
2	B	C	D		
3				E	F
4	I			G	H



CODE:

```c
#include <stdio.h>
#include <malloc.h>
#include <conio.h>
struct node {
    int data;
    int priority;
    struct node *next;
};
struct node *start=NULL;
struct node *insert(struct node *);
struct node *del(struct node *);
void display(struct node *);
int main() {
    int option;

    do {
                printf("\n Press 1 to INSERT");
                printf("\n Press 2 to DELETE");
                printf("\n Press 3 to DISPLAY");
                printf("\n Press 4 to EXIT");
                printf("\n Enter your option : ");
                scanf( "%d", &option);
        switch(option) {
        case 1:
            start=insert(start);
                break;
            case 2:
                start = del(start);
                break;
            case 3:
                display(start);
                break;
        }
    } while(option!=4);
}
```

```
36  struct node *insert(struct node *start) {
37      int val, pri;
38      struct node *ptr, *p;
39      ptr = (struct node *)malloc(sizeof(struct node));
40      printf("\n Enter the value and its priority : " );
41      scanf( "%d %d", &val, &pri);
42      ptr->data = val;
43      ptr->priority = pri;
44      if(start==NULL || pri < start->priority ) {
45          ptr->next = start;
46          start = ptr;
47      } else {
48          p = start;
49          while(p->next != NULL && p->next->priority <= pri)
50              p = p->next;
51          ptr->next = p->next;
52          p->next = ptr;
53      }
54      return start;
55  }
56  struct node *del(struct node *start) {
57      struct node *ptr;
58      if(start == NULL) {
59          printf("\n UNDERFLOW" );
60          return start;
61      } else {
62          ptr = start;
63          printf("\n Deleted item is: %d", ptr->data);
64          start = start->next;
65          free(ptr);
66      }
67      return start;
68  }
```

```
69  void display(struct node *start) {
70      struct node *ptr;
71      ptr = start;
72      if(start == NULL)
73          printf("\nQUEUE IS EMPTY" );
74      else {
75          printf("\n PRIORITY QUEUE IS : " );
76          while(ptr != NULL) {
77              printf( "\t%d[priority=%d]", ptr->data, ptr->priority );
78              ptr=ptr->next;
79          }
80      }
81  }
```

OUTPUT:

```
 Press 1 to INSERT
 Press 2 to DELETE
 Press 3 to DISPLAY
 Press 4 to EXIT
 Enter your option : 1

 Enter the value and its priority : 2
1

 Press 1 to INSERT
 Press 2 to DELETE
 Press 3 to DISPLAY
 Press 4 to EXIT
 Enter your option : 1

 Enter the value and its priority : 12
3

 Press 1 to INSERT
 Press 2 to DELETE
 Press 3 to DISPLAY
 Press 4 to EXIT
 Enter your option : 1

 Enter the value and its priority : 6
2

 Press 1 to INSERT
 Press 2 to DELETE
 Press 3 to DISPLAY
 Press 4 to EXIT
 Enter your option : 3

 PRIORITY QUEUE IS :    2[priority=1]   6[priority=2]   12[priority=3]
```

```
Press 1 to INSERT
Press 2 to DELETE
Press 3 to DISPLAY
Press 4 to EXIT
Enter your option : 2

Deleted item is: 2
Press 1 to INSERT
Press 2 to DELETE
Press 3 to DISPLAY
Press 4 to EXIT
Enter your option : 3

PRIORITY QUEUE IS :    6[priority=2]   12[priority=3]
```

EXP 9

NAME: DIVYESH KHUNT **SAPID:60009210116**

Binary Search Tree

AIM: To implement an hierarchical data structure known as TREE

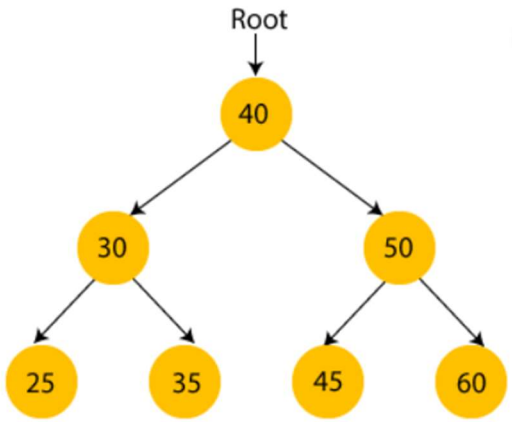
THEORY:

A binary search tree, also known as an ordered binary tree, is a variant of binary trees in which the nodes are arranged in an order. In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node. Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node. The same rule is applicable to every sub-tree in the tree.

Basic Operations

Following are the basic operations of a tree −

- **Search** − Searches an element in a tree.
- **Insert** − Inserts an element in a tree.
- **Pre-order Traversal** − Traverses a tree in a pre-order manner.
- **In-order Traversal** − Traverses a tree in an in-order manner.
- **Post-order Traversal** − Traverses a tree in a post-order manner.



CODE:

```c
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node {
    int data;
    struct node *left;
    struct node *right;
};
struct node *tree;
void create_tree(struct node *);
struct node *insertElement(struct node *, int);
void preorderTraversal(struct node *);
void inorderTraversal(struct node *);
void postorderTraversal(struct node *);
struct node *findSmallestElement(struct node *);
struct node *findLargestElement(struct node *);
int totalNodes(struct node *);
struct node *deleteTree(struct node *);
int main() {
    int option, val;
    struct node *ptr;
    create_tree(tree);
    while(option!=9) {
        printf("Press");
        printf("\n 1. Insert Element");
        printf("\n 2. Preorder Traversal");
        printf("\n 3. Inorder Traversal");
        printf("\n 4. Postorder Traversal");
        printf("\n 5. Find the smallest element");
        printf("\n 6. Find the largest element");
        printf("\n 7. Count total number of nodes in tress");
        printf("\n 8. Delete the tree");
        printf("\n 9. Exit");
        printf("\n\n Enter your option : ");
        scanf("%d", &option);
```

```
        switch(option) {
            case 1:
                printf("\n Enter the value of the new node : ");
                scanf("%d", &val);
                tree = insertElement(tree, val);
                break;
            case 2:
                printf("\n The elements of the tree are : \n");
                preorderTraversal(tree);
                break;
            case 3:
                printf("\n The elements of the tree are : \n");
                inorderTraversal(tree);
                break;
            case 4:
                printf("\n The elements of the tree are : \n");
                postorderTraversal(tree);
                break;
            case 5:
                ptr = findSmallestElement(tree);
                printf("\n Smallest element is :%d",ptr ->data);
                break;
            case 6:
                ptr = findLargestElement(tree);
                printf("\n Largest element is : %d", ptr ->data);
                break;
            case 7:
                printf("\n Total no. of nodes = %d", totalNodes(tree));
                break;
            case 8:
                printf("The tree has been deleted");
                tree = deleteTree(tree);
                break;
        }
    }
    getch();
    return 0;
}
```

```
74  void create_tree(struct node *tree) {
75      tree = NULL;
76  }
77  struct node *insertElement(struct node *tree, int val) {
78      struct node *ptr, *nodeptr, *parentptr;
79      ptr = (struct node*)malloc(sizeof(struct node));
80      ptr ->data = val;
81      ptr ->left = NULL;
82      ptr ->right = NULL;
83      if(tree==NULL) {
84          tree=ptr;
85          tree ->left=NULL;
86          tree ->right=NULL;
87      } else {
88          parentptr=NULL;
89          nodeptr=tree;
90          while(nodeptr!=NULL) {
91              parentptr=nodeptr;
92              if(val<nodeptr ->data)
93                  nodeptr=nodeptr ->left;
94              else
95                  nodeptr = nodeptr ->right;
96          }
97          if(val<parentptr->data)
98              parentptr->left = ptr;
99          else
100             parentptr->right = ptr;
101     }
102     return tree;
103 }
104 void preorderTraversal(struct node *tree) {
105     if(tree != NULL) {
106         printf("%d\t", tree->data);
107         preorderTraversal(tree->left);
108         preorderTraversal(tree->right);
109     }
110 }
111 void inorderTraversal(struct node *tree) {
112     if(tree != NULL) {
113         inorderTraversal(tree->left);
114         printf("%d\t", tree->data);
115         inorderTraversal(tree->right);
116     }
117 }
118 void postorderTraversal(struct node *tree) {
119     if(tree != NULL) {
120         postorderTraversal(tree->left);
121         postorderTraversal(tree->right);
122         printf("%d\t", tree->data);
123     }
124 }
```

```
125 struct node *findSmallestElement(struct node *tree) {
126     if( (tree == NULL) || (tree->left == NULL))
127         return tree;
128     else
129         return findSmallestElement(tree ->left);
130 }
131 struct node *findLargestElement(struct node *tree) {
132     if( (tree == NULL) || (tree->right == NULL))
133         return tree;
134     else
135         return findLargestElement(tree->right);
136 }
137
138 struct node *deleteTree(struct node *tree) {
139     if(tree!=NULL) {
140         deleteTree(tree->left);
141         deleteTree(tree->right);
142         free(tree);
143     }
144 }
145 int totalNodes(struct node *tree) {
146     if(tree==NULL)
147         return 0;
148     else
149         return(totalNodes(tree->left) + totalNodes(tree->right) + 1);
150 }
```

OUTPUT:

Inserting a value

```
Press
 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Count total number of nodes in tress
 8. Delete the tree
 9. Exit

 Enter your option : 1

 Enter the value of the new node : 19
Press
 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Count total number of nodes in tress
 8. Delete the tree
 9. Exit

 Enter your option : 1

 Enter the value of the new node : 45
Press
 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Count total number of nodes in tress
 8. Delete the tree
 9. Exit

 Enter your option : 1

 Enter the value of the new node : 72
Press
 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Count total number of nodes in tress
```

```
Enter the value of the new node : 9
ress
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Count total number of nodes in tress
8. Delete the tree
9. Exit
```

Traversals

```
 Enter your option : 3

 The elements of the tree are :
9       19      45      72      Press
 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Count total number of nodes in tress
 8. Delete the tree
 9. Exit

 Enter your option : 2

 The elements of the tree are :
19      9       45      72      Press
 1. Insert Element
 2. Preorder Traversal
 3. Inorder Traversal
 4. Postorder Traversal
 5. Find the smallest element
 6. Find the largest element
 7. Count total number of nodes in tress
 8. Delete the tree
 9. Exit

 Enter your option : 4

 The elements of the tree are :
9       72      45      19      Press
```

Count nodes

```
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Count total number of nodes in tress
8. Delete the tree
9. Exit

Enter your option : 7

Total no. of nodes = 4Press
```

Find smallest and largest

```
Enter your option : 5

Smallest element is :9Press
1. Insert Element
2. Preorder Traversal
3. Inorder Traversal
4. Postorder Traversal
5. Find the smallest element
6. Find the largest element
7. Count total number of nodes in tress
8. Delete the tree
9. Exit

Enter your option : 6

Largest element is : 72Press
```

Conclusion:

In this experiment we saw how to implement binary search tree using linked list

EXP 10

NAME: DIVYESH KHUNT **SAPID:60009210116**

KRUSKAL ALGO

Aim: To implement and analyse Kruskal's Algorithm

Theory: In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached. The steps to implement Kruskal's algorithm are listed as follows –

- First, sort all the edges from low weight to high.
- Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
- Continue to add the edges until we reach all vertices, and a minimum spanning tree is created

CODE:

```c
#include<stdio.h>
#include<conio.h>
#define MAX 10
int adj[MAX][MAX], tree[MAX][MAX], n;
void readmatrix() {
    int i, j;
    printf("\n Enter the number of nodes in the Graph :");
    scanf("%d", &n);
    printf("\n Enter the adjacency matrix of the Graph\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &adj[i][j]);
}
int spanningtree(int src) {
    int visited[MAX], d[MAX], parent[MAX];
    int i, j, k, min, u, v, cost;
    for (i = 1; i <= n; i++) {
        d[i] = adj[src][i];
        visited[i] = 0;
        parent[i] = src;
    }
    visited[src] = 1;
    cost = 0;
    k = 1;
    for (i = 1; i < n; i++) {
        min = 9999;
        for (j = 1; j <= n; j++) {
            if (visited[j]==0 && d[j] < min) {
                min = d[j];
                u = j;
                cost += d[u];
            }
        }
        visited[u] = 1;
```

S

```
34          visited[u] = 1;
35          tree[k][1] = parent[u];
36          tree[k++][2] = u;
37          for (v = 1; v <= n; v++)
38              if (visited[v]==0 && (adj[u][v] < d[v])) {
39                  d[v] = adj[u][v];
40                  parent[v] = u;
41              }
42      }
43      return cost;
44  }
45  void display(int cost) {
46      int i;
47      printf("\n The Edges of the Mininum Spanning Tree are\n");
48      for (i = 1; i < n; i++)
49          printf(" %d %d \n", tree[i][1], tree[i][2]);
50      printf("\n The Total cost of the Minimum Spanning Tree is : %d\n", cost);
51  }
52  main() {
53      int source, treecost;
54      readmatrix();
55      printf("\n Enter the Source : ");
56      scanf("%d", &source);
57      treecost = spanningtree(source);
58      display(treecost);
59      return 0;
```

OUTPUTS:

```
 Enter the number of nodes in the Graph :3

 Enter the adjacency matrix of the Graph
1 2 3
4 5 6
7  8 9

 Enter the Source : 1

 The Edges of the Mininum Spanning Tree are
 1 2
 1 3

 The Total cost of the Minimum Spanning Tree is : 5

--------------------------------
Process exited after 20.38 seconds with return value 0
Press any key to continue . . .
```

```
 Enter the number of nodes in the Graph :3

 Enter the adjacency matrix of the Graph
1 2 3
4 5 6
7 8 9

 Enter the Source : 5

 The Edges of the Mininum Spanning Tree are
 5 1
 5 2

 The Total cost of the Minimum Spanning Tree is : 0

--------------------------------
Process exited after 16.78 seconds with return value 0
Press any key to continue . . .
```

Conclusion:

In this experiment we analysed and implemented Kruskal's algorithm code.

EXP 11

NAME: DIVYESH KHUNT **SAPID:60009210116**

LINEAR HASHING

AIM: To insert elements in array with help of linear hasing

Theory

Linear hashing is a dynamic data structure which implements a hash table and grows or shrinks one bucket at a time. The file structure of a dynamic hashing data structure adapts itself to changes in the size of the file, so expensive periodic file reorganization is avoided. A hash function is simply a mathematical function which then applied to a key, produces an integer which can be used as an index for the key in the hash table.

CODE:

```c
#include <stdio.h>
void display(int arr[]) {
    printf("Hash table is: ");
    for(int i=0; i<10; i++) {

        printf("%d ",arr[i]);
    }
    printf("\n");
}

void insert(int arr[],int val) {
    int i=0;
    int key=((val % 10)+i)%10;
    if ( arr[key] == -1 ) {
        arr[key] = val;
        display(arr);
    } else {
        if ( key < 9 ) {
            for ( i = key ; i < 10; i++ ) {
                if ( arr[i]!=-1 ) {
                    printf("collision detected at %d\n",i);
                } else  if ( arr[i] ==-1 ) {
                    arr[i] = val;
                    display(arr);
                    break;
                }
            }
        }
        if(i>9) {
            printf("Hash table is full\n");
        }
    }
}
```

```
}
    void search (int arr[], int val) {
        for(int i=0; i<10; i++) {
            if(arr[i]==val)
                printf("Element %d is found at %d",val,i+1);
            else
                printf("Element %d not found!",val);
        }
}
    void deleteVal(int arr[],int val) {
        for (int i = 0 ; i <=9; i++) {
            if( arr[i] ==val) {
                arr[i] ==-1 ;
                return;
            }
            printf("Element %d not found!",val);
            display( arr);
        }
    }
int main() {

    int option, val,i,num[10];
    for ( i=0; i<10; i++ ) {
        num[i] = -1;
    }
    int n=sizeof(num)/sizeof(int);
    do {
        printf("\n Press\n1.Insert \n2.Search \n3.Delete \n4.Exit");
        printf("\n Enter your option.");
        scanf("%d", &option);
        switch (option) {
            case 1:
                printf("Enter element to insert: ");
                scanf("%d",&val);
                insert(num,val);
                break;
            case 2:
                printf("enter element to search: ");
                scanf("%d",&val);
                search(num,val);
                break;
            case 3:
                printf("Enter element to delete: ");
                scanf("%d",&val);
                deleteVal(num,val);
                break;
            default:
                printf("\nInvalid choice entry!!!\n");
                break;
        }
    } while (option!=4);
    return 0;
```

OUTPUTS:

```
 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.1
Enter element to insert: 72
Hash table is: -1 -1 72 -1 -1 -1 -1 -1 -1 -1

 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.1
Enter element to insert: 27
Hash table is: -1 -1 72 -1 -1 -1 -1 27 -1 -1

 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.1
Enter element to insert: 36
Hash table is: -1 -1 72 -1 -1 -1 36 27 -1 -1

 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.1
Enter element to insert: 24
Hash table is: -1 -1 72 -1 24 -1 36 27 -1 -1

 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.1
Enter element to insert: 92
collision detected at 2
Hash table is: -1 -1 72 92 24 -1 36 27 -1 -1
```

```
 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.1
Enter element to insert: 101
Hash table is: -1 101 72 92 24 -1 36 27 -1 -1

 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.1
Enter element to insert: 81
collision detected at 1
collision detected at 2
collision detected at 3
collision detected at 4
Hash table is: -1 101 72 92 24 81 36 27 -1 -1
```

```
 Press
1.Insert
2.Search
3.Delete
4.Exit
 Enter your option.2
enter element to search: 92
Element 92 is found at 4
```

CONCLUSION:

Thus the elements were stored in array with help of linear hashing.