AMAL JYOTHI

COLLGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University)
Koovapally P.O, Kanjirapally-686 507

LAB RECORD ON

DATA STRUCTURE LAB
(ITL 201)

SUBMITTED BY

Name	**Roshan Reji**
Roll No.	**43**
Semester & Branch	**S3 - INFORMATION TECHNOLOGY**
Reg. No.	**AJC19IT043**

2019-2023 BATCH

DEPARTMENT OF INFORMATION TECHNOLOGY

AMAL JYOTHI

COLLGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University)
Koovapally P.O, Kanjirapally-686 507

CERTIFICATE

Certified that this is a Bonafide Record of Practical work done in **DATA STRUCTURE LAB (ITL201)** *Laboratory by*……Roshan Reji…… **Reg.No**……AJC19IT043…… **of S3-INFORMATION TECHNOLOGY DEPARTMENT** *in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Information Technology under APJ Abdul Kalam Technological University during the year 2020-2021*

Head of the Department **Faculty-in-charge**

External Examiner **Internal Examiner**

Experiment No: 1
Date: 07/09/2020
Aim: Develop a C program to implement insertion sort, Selection sort and bubble sort.
Description: Program to sort a given number sets using the insertion,selection,and bubble sorting algorithm

Algorithm

Insertion sort
Input: size(s), Array (a)
Output : Array in sorted form
Data structure: Array
Steps
1. insertionSort(array)
2. mark first element as sorted
3. for each unsorted element X
4. 'extract' the element X
5. for j <- lastSortedIndex down to 0
6. if current element j > X
7. move sorted element to the right by 1
8. break loop and insert X here
9. end insertionSort

Selection sort
Input: size(s), Array (a)
Output : Array in sorted form
Data structure: Array
Steps
1. selectionSort(array, size)
2. repeat (size - 1) times
3. set the first unsorted element as the minimum
4. for each of the unsorted elements
5. if element < currentMinimum
6. set element as new minimum
7. swap minimum with first unsorted position
8. end selectionSort

Bubble sort
Input: size(s), Array (a)
Output : Array in sorted form
Data structure: Array
Steps
1. bubble Sort(array)
2. for i <- 1 to indexOfLastUnsortedElement-1
3. if leftElement > rightElement
4. swap leftElement and rightElement
5. end bubbleSort

Program:

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
 int p[20],q[20],n,r[20],j,i,t;
 printf("Enter the size of the array : ");
 scanf("%d",&n);
 printf("Enter the array: ");
 for(i=0;i<n;i++)
 scanf("%d,",&p[i]);
 for(i=0;i<n;i++)
{
 q[i]=p[i];
 r[i]=p[i];
}
 for(i=0;i<n-1;i++)
 {
 for(j=0;j<n-i-1;j++)
 {
 if(p[j]>p[j+1])
 {
 t=p[j];
 p[j]=p[j+1];
 p[j+1]=t;
 }
 }
 }
printf("**********Insertion Sort**********\n");
for(i=0;i<n;i++)
{
printf("%d ",r[i]);
}
}
printf("***********Bubble Sort*************\n");
for(i=0;i<n;i++)
{
 printf("%d",p[i]);
 if(i<n-1)
 printf(",");
}
for(i=0;i<n;i++)
{
 m=i;
 for(j=i+1;j<n;j++)
 {
 if(q[m]>q[j])
 m=j;
```

```
 }
 t=q[m];
 q[m]=q[i];
 q[i]=t;
}
printf("**********Selection Sort: "**********\n);
for(i=0;i<n;i++)
{
 printf("%d",q[i]);
 if(i<n-1)
 printf(",");
}
for(i=1;i<n;i++)
{
 k=r[i];
 j=i-1;
 while(j>=0 && r[j]>k)
{
 r[j+1]=r[j];
 j=j-1;
}
r[j+1]=k;
}
```

Output:

```
Enter the size of the array:8
Enter the array: 5
6
7
3
45
23
56
9
Insertion Sort:-
5 6 7 3 45 23 56 9
Bubble Sort array :-
3,5,6,7,9,23,45,56
Selection Sort :-
3,5,6,7,9,23,45,56
```

Result:

The given array is sorted and the output is printed

Experiment No: 2
Date: 29/10/2020
Aim: Design a program to Implement i) Quick sort ii) Merge sort(Self-Study).
Description: Program to implement Quick sort and merge sort

Algorithm

1. Quick sort

Input: size(s), Array (a)
Output : Array in sorted form
Data structure: Array
Steps
1. 7uicksort(a, l, r)
2. if (l < r)
3. pivotIndex <- partition(a,l,r)
4. 7uicksort(a, l, pivotIndex)
5. 7uicksort(a, pivotIndex + 1,r)
6. partition(a, l,r)
7. set r as pivotIndex
8. storeIndex <- l – 1
9. for l <- l + 1 to r
10. if element[i] < pivotElement
11. swap element[i] and element[storeIndex]
12. storeIndex++
13. swap pivotElement and element[storeIndex+1]
14. return storeIndex + 1

2. Merge sort

Input: size(s), Array (a)
Output : Array in sorted form
Data structure: Array
Steps
1. MergeSort(A, p, r):
2. if p > r
3. return
4. q = (p+r)/2
5. mergeSort(A, p, q)
6. mergeSort(A, q+1, r)
7. merge(A, p, q, r)
8. end if
9. end mergesort

Program

1.Quick sort

```
# include<stdio.h>
# include<conio.h>
void quicksort(int, int);
int partition(int, int);
void interchange(int, int);
int array[25];
int main()
{
int num, i = 0;
clrscr();
printf( "Enter the number of elements: " );
scanf( "%d", &num);
printf( "Enter the elements: " );
for(i=0; i < num; i++)
scanf( "%d", &array[i] );
quicksort(0, num -1);
printf( "\nThe elements after sorting are: " );
for(i=0; i < num; i++)
printf("%d ", array[i]);
return 0;
}
void quicksort(int low, int high)
{
int pivotpos;
if( low < high )
{
pivotpos = partition(low, high + 1);
quicksort(low, pivotpos - 1);
quicksort(pivotpos + 1, high);
}
}
int partition(int low, int high)
{
int pivot = array[low];
int up = low, down = high;
do
{
do
up = up + 1;
while(array[up] < pivot );
do
down = down - 1;
while(array[down] > pivot);
if(up < down)
interchange(up, down);
}while(up < down);
array[low] = array[down];
array[down] = pivot;
return down;
```

```
}
void interchange(int i, int j)
{
int temp;
temp = array[i];
array[i] = array[j];
array[j] = temp;
}
```

2.Merge sort

```
#include <stdio.h>
void merge(int arr[], int p, int q, int r) {
 int n1 = q - p + 1;
 int n2 = r - q;
 int L[n1], M[n2];
 for (int i = 0; i < n1; i++)
 L[i] = arr[p + i];
 for (int j = 0; j < n2; j++)
 M[j] = arr[q + 1 + j];
int i, j, k;
 i = 0;
 j = 0;
 k = p;
 while (i < n1 && j < n2) {
 if (L[i] <= M[j]) {
 arr[k] = L[i];
 i++;
 } else {
 arr[k] = M[j];
 j++;
 }
 k++;
 }
 while (i < n1) {
 arr[k] = L[i];
 i++;
 k++;
 }
 while (j < n2) {
 arr[k] = M[j];
 j++;
 k++;
 }
}
void mergeSort(int arr[], int l, int r) {
 if (l < r) {
 int m = l + (r - l) / 2;
 mergeSort(arr, l, m);
```

```
 mergeSort(arr, m + 1, r);
 merge(arr, l, m, r);
 }
}
void printArray(int arr[], int size) {
 for (int i = 0; i < size; i++)
 printf("%d ", arr[i]);
 printf("\n");
}
int main() {
 int arr[] = {6, 5, 12, 7, 10, 9, 1, 23};
 int size = sizeof(arr) / sizeof(arr[0]);
 mergeSort(arr, 0, size - 1);
 printf("Sorted array: \n");
 printArray(arr, size);
}
```

Output:

1.Quick sort

Enter the number of elements: 5
Enter the elements: 4
7
8
9
6
The elements after sorting are: 4 6 7 8 9

2.Merge sort

Sorted array:
1 5 6 7 9 10 12 23

Result:

The given array is sorted and the output is printed using the two implementation

Experiment No: 3
Date: 18/10/2020
Aim: Create Programs for i) Linear Search ii) Binary Search.
Description: Create a program for linear and binary Search

Algorithm

(1).Linear search

Input: A key,Element to be Searched
Output: Index of the key or a Message on Failure
Data Structure: Array
1. Let array a[n] stores n elements. Determine whether element 'x' is present or not.
2. linsrch(a[n], x)
3. {
4. index = 0;
5. flag = 0;
6. while (index < n) do
7. {
8. if (x == a[index])
9. {
10. flag = 1;
11. break;
12. }
13. index ++;
14. }
15. if(flag == 1)
16. printf("Data found at %d position", index);
17. else
18. printf("data not found");
19. }

(2).Binary search

Input: A Sorted Array, key Element to be Searched
Output: Index of the key or a Message on Failure
Data Structure: Array
1. Let array a[n] of elements in increasing order, n 0, determine whether 'x' is present, and if so, set j
2. such that x = a[j] else return 0.
3. binsrch(a[], n, x)
4. {
5. low = 1; high = n;
6. while (low < high) do
7. {
8. mid = (low + high)/2
9. if (x < a[mid])
10. high = mid – 1;

```
11. else if (x > a[mid])
12. low = mid + 1;
13. else return mid;
14. }
15. return 0;
16. }
```

Program

(1).Linear search

```
# include <stdio.h>
Void main()
{
int number[25], n, data, i, flag = 0;
printf("\n Enter the number of elements: ");
scanf("%d", &n);
printf("\n Enter the elements: ");
for(i = 0; i < n; i++)
scanf("%d", &number[i]);
printf("\n Enter the element to be Searched: ");
scanf("%d", &data);
for( i = 0; i < n; i++)
{
if(number[i] == data)
{
flag = 1;
break;
}
}
if(flag == 1)
printf("\n Data found at location: %d", i+1);
else
printf("\n Data not found ");
}
```

(2).Binary search

```
# include <stdio.h>
main()
{
int number[25], n, data, i, flag = 0, low, high, mid;
printf("\n Enter the number of elements: ");
scanf("%d", &n);
printf("\n Enter the elements in ascending order: ");
for(i = 0; i < n; i++)
scanf("%d", &number[i]);
printf("\n Enter the element to be searched: ");
scanf("%d", &data);
```

```
low = 0; high = n-1;
while(low <= high)
{
mid = (low + high)/2;
if(number[mid] == data)
{
flag = 1;
break;
}
else
{
if(data < number[mid])
high = mid - 1;
else
low = mid + 1;
}
}
if(flag == 1)
printf("\n Data found at location: %d", mid + 1);
else
printf("\n Data Not Found ");
}
```

Output:

(1).Linear search

```
Enter the number of elements: 5
Enter the elements: 5
6
75
3
2
Enter the element to be Searched: 75
Data found at location: 3
```

(2). Binary search

```
Enter the number of elements: 6
Enter the elements in ascending order: 4
5
6
7
8
6
Enter the element to be searched: 7
Data found at location: 4
```

Result:

The Element is found using Linear Search and Binary Search Algorithms.

Experiment No: 4
Date: 03/10/2020
Aim: Create a menu driven program to implement singly linked list operations with options for insertion, deletion, search and traversal.
Description: Implementing a menu driven program for implementing singly linked list operations with options for insertion, deletion, search and traversal.

Algorithm

INSERTION

1.Insertion at end

Input: header is the pointer to the header node and item is the element to be inserted.
Output: A node with element item is inserted at end of the list.
Data Structure: Linked list
Steps
1. NODE: temp1,temp2
2. Temp1=GETSPACE(node)
3. Temp1.data=item;
4. Temp1.link=NULL;
5. If (header=NULL) then //list is empty
1. Header=temp1
2. exit
6. Else
1. Temp2=header
2. While(temp2.link!=NULL)
1. Temp2=temp2.link
3. EndWhile
4. Temp2.link=temp1
7. EndIf
8. Stop

2.Insertion at beginning

Input: header is the pointer to the header node and item is the element to be inserted.
Output: A node with element item is inserted at front of the list if list is not null.
Data Structure: Linked list
Steps
1. NODE: temp
2. Temp=GETSPACE(node)
3. Temp.data=item
4. If(header=NULL) then
1. Temp.link=NULL
2. Header=temp
3. Exit
5. else
1. Temp.link=header
2. Header=temp
6. EndIf

7. Stop

3.Insertion at particular position

Input: header is the pointer to the header node and item is the element to be inserted and key being the data one node after which the item to be inserted.
Output: A node with element item is inserted at given position if the list is not null.
Data Structure: Linked list
Steps
1.NODE: temp1,temp2
2.temp1=GETSPACE(node)
3.Flag=0
4.If(header=NULL) then
1.Print("List is Empty..")
2.Exit
5.EndIf
6.Temp2=header
7.While(temp2!=NULL)
1.If(temp2.data=key)
1.Flag=1
2.ExitWhile
2.EndIf
3.Temp2=temp2.link
8.EndWhile
9.If(flag=0) then
1.Print("Key not found in the list")
1.Exit
10.Else
1.Temp1.data=item
2.Temp1.link=temp2.link
3.Temp2.link=temp1
11.Endif
12.Stop
Deletion

1.Deletion from the front of the list

Input: Header is the pointer to the header node of the linked list
Output: A single linked list eliminating the node at the front.
Data Structure: Linked list
Steps
1. NODE : temp
2. If(header=NULL)
1. Printf("List empty")
2. Exit
3. EndIf
4. Temp=header
5. Header=header.link
6. FREESPACE(temp)
7. Stop

2.Deletion from the end of the list

Input: Header is the pointer to the header node of the linked list
Output: A single linked list eliminating the node at the end.
Data Structure: Linked list
Steps
1. NODE : temp,prev
2. If(header=NULL)
1. Printf("List is empty..")
2. Exit
3. EndIf
4. If(header.link=NULL)
1. Item=header.data
2. FREESPACE(header)
3. Header=NULL
4. Exit
5. Else
1. Temp=header
2. While(temp.link!=NULL)
1. Prev=temp;
2. Temp=temp.link
3. EndWhile
4. Item=temp.data
5. FREESPACE(temp)
6. Prev.link=NULL
6. EndIf
7. Stop

3.Deletion from any position of the list

Input: Header is the pointer to the header node of the linked list, key is the data content of the node to be deleted.
Output: A single linked list except the node with content as key.
Data Structure: Linked list.
Steps
1. NODE: temp,prev
2. Flag=0
3. If(header=NULL) then
1. Printf("List is Empty")
2. Exit
4. ElseIf(header.data=key AND header.link=NULL)
1. Item=header.data
2. FREESPACE(header)
3. Header=NULL
4. Exit
5. ElseIf(header.data=key AND header.link!=NULL)
1. Item=header.data
2. Temp=header
3. Header=header.link
4. FREESPACE(temp)
5. Exit

6. Else
1. Temp=header
2. While(temp.link!=NULL)
1. Prev=temp
2. Temp=temp.link
3. If(temp.data=key)
1. Flag=1
2. ExitWhile
4. EndIf
3. EndWhile
4. If(flag=0)
1. Print("Key not found)
2. Exit
5. Else
1. Item=temp.data
2. Prev.link=temp.link
3. FREESPACE(temp)
6. EndIf
7. EndIf
8. Stop

Searching

1.Flag=0 , e
2.If(header=NULL) then
1.Print("List is Empty..")
2.Exit
3.EndIf
4.temp=header
5.While(temp!=NULL)
1.If(temp.data=key)
1.Flag=1
2.ExitWhile
2.EndIf
3.temp=temp.link
6.EndWhile
7.If(flag=0) then
1.Print("Element not found in the list")
1.Exit
8.else
1.Print("Element found ") / display its position
1.Exit
9.Stop

Traversal

Input: Header is the pointer to the first node if not null.
Output: Elements in the list is displayed.
Data Structure: A single linked list.
Steps
1. NODE: temp
2. If (header=NULL) then
1. Print(“List is empty”)
2. Exit
3. Else
1. Temp=header
2. While(temp!=NULL)
1. Print(temp.data)
2. Temp=temp.link
3. EndWhile
4. stop

Program

```
#include<stdlib.h>
#include <stdio.h>
void ib();
void ip();
void inend();
void db();
void delete_end();
void delete_pos();
void search();
void display();
struct node
{
 int info;
 struct node *next;
};
struct node *start=NULL;
int main()
{
 int choice;
 while(1)
 {
 printf("\n 1.Insert at front \n");
 printf("\n 2.Insert at a position \n");
 printf("\n 3.Insert at the end \n");
 printf("\n 4.Delete at front \n");
 printf("\n 5.Delete from the end \n");
 printf("\n 6.Delete from specified position \n");
 printf("\n 7.Search an element \n");
 printf("\n 8.Traverse \n");
 printf("\n 9.Exit \n");
```

```
 printf(" Enter your choice:>>\t");
 scanf("%d",&choice);
 switch(choice)
 {
 case 1:ib();
 break;
 case 2:ip();
 break;
 case 3:inend();
 break;
 case 4:db();
 break;
 case 5:delete_end();
 break;
 case 6:delete_pos();
 break;
 case 7:search();
 break;
 case 8:display();
 break;
 case 9: exit(0);
 break;
 default: printf("\n [The choice entered is wrong] \n");
 break;
 }
 }
 return 0;
}
 void ib()
{
 struct node *temp;
 temp=(struct node *)malloc(sizeof(struct node));
 if(temp==NULL)
 {
 printf("\n [Memory is out of Space]\n");
 return;
 }
 printf("\nEnter the data value for the node:>> \t" );
 scanf("%d",&temp->info);
 temp->next =NULL;
 if(start==NULL)
 {
 start=temp;
 }
 else
 {
 temp->next=start;
 start=temp;
 }
```

```
}
void ip()
{
 struct node *ptr,*temp;
 int i,pos;
 temp=(struct node *)malloc(sizeof(struct node));
 if(temp==NULL)
 {
 printf("\n[Memory is out of Space]\n");
 return;
 }
 printf("\nEnter the position for the new node to be inserted:>>\t");
 scanf("%d",&pos);
 printf("\nEnter the data value of the node:>>\t");
 scanf("%d",&temp->info) ;
 temp->next=NULL;
 if(pos==0)
 {
 temp->next=start;
 start=temp;
 }
 else
 {
 for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;
 if(ptr==NULL)
 {
 printf("\n[Position is not found:]\n");
 return;
 }
 }
 temp->next =ptr->next ;
 ptr->next=temp;
 }
}
void inend()
{
 int ele;
 struct node*newnode,*temp1;
 printf("\nEnter the element to be inserted: ");
 scanf("%d",&ele);
 newnode=(struct node*)malloc(sizeof(struct node));
 newnode->info=ele;
 newnode->next=NULL;
 if(start==NULL)
 start=newnode;
 else
 {
 temp1=start;
 while(temp1->next!=NULL)
```

```
 {
 temp1=temp1->next;
 }
 temp1->next=newnode;
 }
 printf("\nSuccessfully inserted\n");
 temp1=start;
 while(temp1!=NULL)
 {
 printf("%d -> ",temp1 ->info);
 temp1=temp1->next;
 }
}
void db()
{
 struct node *ptr;
 if(ptr==NULL)
 {
 printf("\n[List is Empty] \n");
 return;
 }
 else
 {
 ptr=start;
 start=start->next ;
 printf("\nThe deleted element is:>>%d\t \n",ptr->info);
 free(ptr);
 }
}
void delete_end()
{
 struct node *temp,*ptr;
 if(start==NULL)
 {
 printf("nList is Empty:");
 exit(0);
 }
 else if(start->next ==NULL)
 {
 ptr=start;
 start=NULL;
 printf("nThe deleted element is:%dt",ptr->info);
 free(ptr);
 }
 else
 {
 ptr=start;
 while(ptr->next!=NULL)
 {
```

```
 temp=ptr;
 ptr=ptr->next;
 }
 temp->next=NULL;
 printf("nThe deleted element is:%dt",ptr->info);
 free(ptr);
 }
}
void delete_pos()
{
 int i,pos;
 struct node *temp,*ptr;
 if(start==NULL)
 {
 printf("nThe List is Empty:n");
 exit(0);
 }
 else
 {
 printf("nEnter the position of the node to be deleted:t");
 scanf("%d",&pos);
 if(pos==0)
 {
 ptr=start;
 start=start->next ;
 printf("nThe deleted element is:%dt",ptr->info );
 free(ptr);
 }
 else
 {
 ptr=start;
 for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;
 if(ptr==NULL)
 {
 printf("nPosition not Found:n");
 return;
 }
 }
 temp->next =ptr->next ;
 printf("nThe deleted element is:%dt",ptr->info );
 free(ptr);
 }
 }
}
void search()
{
 struct node *temp;
 int flag=-1,ele;
 printf("Enter the data to be searched:");
```

```
 scanf("%d",&ele);
 if(start==NULL)
 {
 printf("\nList is empty and the element not existing...\n");
 return;
 }
 else
 {
 temp=start;
 while(temp->next!=NULL)
 {
 temp=temp->next;
 if(temp->info==ele)
 {
 flag=0;
 break;
 }
 }
 if(flag==-1)
 printf("The data is not existing...\n");
 else
 printf("The data has been found...\n");
 }
}
void display()
{
 struct node *ptr;
 if(start==NULL)
 {
 printf("\nList is empty:\n");
 return;
 }
 else
 {
 ptr=start;
 printf("\nThe List elements are:\n");
 while(ptr!=NULL)
 {
 printf("%d\t",ptr->info );
 ptr=ptr->next ;
 }
 }
}
```

Output

```
 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>  1
Enter the data value for the node:>>   34

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>  1
Enter the data value for the node:>>   56

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>  1
Enter the data value for the node:>>   43

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>  2
Enter the position for the new node to be inserted:>>  2
```

```
Enter the data value of the node:>>     32

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>   3
Enter the element to be inserted: 78
Successfully inserted
43 -> 56 -> 32 -> 34 -> 78 ->

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>   4
The deleted element is:>>43

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>   6
Enter the position of the node to be deleted: 2
The deleted element is:34

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
```

```
 9.Exit
 Enter your choice:>>  5
The deleted element is:78

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>  7
Enter the data to be searched:32
The data has been found...

 1.Insert at front
 2.Insert at a position
 3.Insert at the end
 4.Delete at front
 5.Delete from the end
 6.Delete from specified position
 7.Search an element
 8.Traverse
 9.Exit
 Enter your choice:>>  8
The List elements are:
56      32
```

Result

The Operations on Singly Linked List is successfully Implemented.

Experiment No : 5

Date: 22/10/2020

Aim: Design a menu driven program to implement Doubly linked list operations with options for insertion at front, insertion at end, deletion at front, deletion at end and traversal.

Description: Implementing a program for the operation of insertion at front, insertion at end, deletion at front, deletion at end and traversal of Doubly Linked Lists.

Algorithm

Insertion at front

Input: Item is the data content of the node to be inserted
Output: A double linked list enriched with the node containing data to be added at the front
Data Structure: Linked List
Steps
1. NODE: newnode
2. Newnode=GETSPACE(NODE)
3. Newnode.data=item
4. If(header=NULL)
1. Newnode.llink=NULL
2. Newnode.rlink=NULL
3. Header=newnode
5. Else
1. Newnode.llink=NULL
2. Newnode.rlink=header
3. Header.llink=newnode
4. Header=newnode
6. EndIf
7. Stop

Insertion at end

Input: Item is the data content of the node to be inserted
Output: A double linked list enriched with the node containing data at the end of the list.
Data Structure: Linked List
Steps
1. NODE: current,newnode
2. Newnode=GETSPACE(node)
3. Newnode.data=item
4. If(header=NULL)
1. Newnode.llink=NULL
2. Newnode.rlink=NULL
3. Header=newnode
5. Else
1. Current=header
2. While(current.rlink!=NULL)
1. Current=current.rlink
3. EndWhile
4. Current.rlink=newnode

5. Newnode.llink=current
6. Newnode.rlink=NULL
6. EndIf
7. Stop

Deletion at front

Input: Double linked list with data
Output: List with front node removed
Data Structure: Linked List
Steps
1. NODE: current
2. If (header=NULL) then
1. Print("List is Empty…");
3. ElseIf(header.rlink=NULL)
1. Item=header.data
2. FREESPACE(header)
3. Header=NULL
4. Else
1. Current=header
2. Item=current.data
3. Header=header.rlink
4. Header.llink=NULL
5. FREESPACE(current)
5. EndIf
6. Stop

Deletion at end

Input: A double linked list with data
Output: A list with removed last node
Data Structure: Linked List
Steps
1. NODE: current,prev
2. If(header=NULL) then
1. Print("List is Empty")
3. Else if(header.rlink=NULL)
1. Item=header.data
2. FREESPACE(header)
3. Header=NULL
4. Else
1. Current=header
2. While(current.rlink!=NULL)
1. Current=current.rlink
3. Item=current.data
4. Prev=current.llink
5. Prev.rlink=NULL;
6. FREESAPCE(current)
5. EndIf
6. Stop

Traversal

Input: Header is the pointer to the first node if not null.
Output: Elements in the list is displayed.
Data Structure: A Doubly linked list.
Steps
1.NODE: temp
2.If (header=NULL) then
1.Print("List is empty")
2.Exit
3.else
1.Temp=header
2.While(temp!=NULL)
1.Print(temp.data)
2.Temp=temp.link
3.EndWhile
4.stop

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *prev;
   struct node *next;
   int data;
};
struct node *head;
void inbeg();
void inlast();
void delbeg();
void dellast();
void display();
int main ()
{
int choice =0;
   while(choice != 9)
   {
     printf("\n1.Insert at the begining\n2.Insert at the last\n3.Delete from Beginning\n4.Delete from
     last\n5.Display\n6.Exit\n");
       printf("\nEnter your choice-->");
       scanf("\n%d",&choice);

       switch(choice)
       {
          case 1:
          inbeg();
          break;
          case 2:
```

```
            inlast();
            break;
            case 3:
            delbeg();
            break;
            case 4:
            dellast();
            break;
            case 5:
            display();
            break;
            case 6:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void inbeg()
{
   struct node *ptr;
   int item;
   ptr = (struct node *)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\n overfollow.....");
   }
   else
   {
    printf("\nEnter Item value");
    scanf("%d",&item);

   if(head==NULL)
   {
      ptr->next = NULL;
      ptr->prev=NULL;
      ptr->data=item;
      head=ptr;
   }
   else
   {
      ptr->data=item;
      ptr->prev=NULL;
      ptr->next = head;
      head->prev=ptr;
      head=ptr;
   }
   printf("\nNode is inserted\n");
```

```
}
}
void inlast()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *) malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nover flow.....");
  }
  else
  {
     printf("\nEnter value");
     scanf("%d",&item);
      ptr->data=item;
     if(head == NULL)
     {
        ptr->next = NULL;
        ptr->prev = NULL;
        head = ptr;
     }
     else
     {
       temp = head;
       while(temp->next!=NULL)
       {
          temp = temp->next;
       }
       temp->next = ptr;
       ptr ->prev=temp;
       ptr->next = NULL;
     }
    printf("\nnode inserted\n");
  }
void delbeg()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == NULL)
   {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }
   else
```

```
    {
        ptr = head;
        head = head -> next;
        head -> prev = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void dellast()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}
void display()
{
    struct node *ptr;
    printf("\n printing values...\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
```

```
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("\nitem found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        if(flag==1)
        {
            printf("\nItem not found\n");
        }
    }
}
```

Output:

```
1.Insert at the begining
2.Insert at the last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
Enter your choice-->1
Enter Item value34
Node is inserted

1.Insert at the begining
2.Insert at the last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit

Enter your choice-->1
Enter Item value67
Node is inserted

1.Insert at the begining
2.Insert at the last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit

Enter your choice-->1
Enter Item value54
Node is inserted

1.Insert at the begining
2.Insert at the last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit

Enter your choice-->2
Enter value34
node inserted

1.Insert at the begining
2.Insert at the last
3.Delete from Beginning
4.Delete from last
```

```
5.Display
6.Exit

Enter your choice-->3
node deleted

1.Insert at the begining
2.Insert at the last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit

Enter your choice-->4
node deleted

1.Insert at the begining
2.Insert at the last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit

Enter your choice-->5
 printing values...
67
```

Experiment No : 6

Date: 20/10/2020

Aim: Apply linked list concept to perform polynomial addition.

Description: Implementing a program for polynomial addition using linked list

Algorithm

Input: Two polynomial P1 and P2 whose header pointers are p1header and p2header

Output: A polynomial R is the sum of P1 and P2 with header rheader

Data Structure: Polynomial is implemented by linked List

Steps

1. NODE p1current, p2current, newnode, rcurrent
2. P1current=p1header
3. P2current=p2header
4. Rheader=NULL
5. While(p1current!=NULL AND p2current!=NULL)

1. If(p1current.exp=p2current.exp)

1. Newnode=GETSPACE(NODE)
2. Newnode.coeff=p1current.coeff+p2current.coeff
3. Newnode.exp=p1current.exp
4. Newnode.link=NULL
5. P1current=p1current.link
6. P2current=p2current.link

2. ElseIf(p1current.exp>p2current.exp)

1. Newnode=GETSPACE(NODE)
2. Newnode.coeff=p1current.coeff
3. Newnode.exp=p1current.exp
4. Newnode.link=NULL
5. P1current=p1current.link

3. ElseIf(p2current.exp>p1current.exp)

1. Newnode=GETSPACE(NODE)
2. Newnode.coeff=p2current.coeff

3. Newnode.exp=p2current.exp

4. Newnode.link=NULL

5. P2current=p2current.link

4. EndIf

5. If(rheader=NULL)

1. Rheader=newnode

6. Else

1. Rcurrent=rheader

2. While(rcurrent.link!=NULL)

1. Rcurrent=rcurrent.link

3. EndWhile

4. Rcurrent.link=newnode

7. EndIf

6. EndWhile

7. While(p1current!=NULL)

1. Newnode=GETSPACE(NODE)

2. Newnode.coeff=p1current.coeff

3. Newnode.exp=p1current.exp

4. Newnode.link=NULL

5. P1current=p1current.link

2. If(rheader=NULL)

1. Rheader=newnode

3. Else

1. Rcurrent=rheader

2. While(rcurrent.link!=NULL)

1. Rcurrent=rcurrent.link

3. EndWhile

4. Rcurrent.link=newnode

4. EndIf

8. EndWhile

9. While(p2current!=NULL)

1. Newnode=GETSPACE(NODE)

2. Newnode.coeff=p2current.coeff

3. Newnode.exp=p2current.exp

4. Newnode.link=NULL

5. P2current=p2current.link

2. If(rheader=NULL)

1. Rheader=newnode

3. Else

1. Rcurrent=rheader

2. While(rcurrent.link!=NULL)

1. Rcurrent=rcurrent.link

3. EndWhile

4. Rcurrent.link=newnode

4. EndIf

10. EndWhile

11. Stop

Program

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
	int coeff;
	int pow;
	struct Node* next;
};
void readPolynomial(struct Node** poly)
{
	int coeff, exp, cont;
	struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
	*poly = temp;
	do{
		printf("Enter the Coefficents: ");
```

```
                scanf("%d", &coeff);
                printf("Enter the Exponent: ");
                scanf("%d", &exp);
                temp->coeff = coeff;
                temp->pow = exp;
                temp-> next = NULL;
                printf("Enter 1 for continuing or enter 0 for exiting first polynomial section: ");
                scanf("%d", &cont);
                if(cont)
                {
                        temp->next = (struct Node*)malloc(sizeof(struct Node));
                        temp = temp->next;
                        temp->next = NULL;
                }
        }while(cont);
}
void displayresultPolynomial(struct Node* poly)
{
        printf("\n\nThe final sum expression is: ");
        while(poly != NULL)
        {
                printf("%dX^%d", poly->coeff, poly->pow);
                poly = poly->next;
                if(poly != NULL)
                        printf("+");
        }
}
void displayPolynomial(struct Node* poly)
{
        printf("Polynomial expression is: ");
        while(poly != NULL)
        {
```

```
                printf("%dX^%d", poly->coeff, poly->pow);
                poly = poly->next;
                if(poly != NULL)
                        printf("+");
        }
}
void addPolynomials(struct Node** result, struct Node* first, struct Node* second)
{
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        temp->next = NULL;
        *result = temp;
        while(first && second)
        {
                if(first->pow > second->pow)
                {
                        temp->coeff = first->coeff;
                        temp->pow = first->pow;
                        first = first->next;
                }
                else if(first->pow < second->pow)
                {
                        temp->coeff = second->coeff;
                        temp->pow = second->pow;
                        second = second->next;
                }
                else
                {
                        temp->coeff = first->coeff + second->coeff;
                        temp->pow = first->pow;
                        first = first->next;
                        second = second->next;
                }
```

```
            if(first && second)
            {
                temp->next = (struct Node*)malloc(sizeof(struct Node));
                temp = temp->next;
                temp->next = NULL;
            }
        }
        while(first || second)
        {
            temp->next = (struct Node*)malloc(sizeof(struct Node));
            temp = temp->next;
            temp->next = NULL;

            if(first)
            {
                temp->coeff = first->coeff;
                temp->pow = first->pow;
                first = first->next;
            }

            else if(second)
            {
                temp->coeff = second->coeff;
                temp->pow = second->pow;
                second = second->next;
            }
        }
}
int main()
{
        struct Node* first = NULL;
```

```
    struct Node* second = NULL;
    struct Node* result = NULL;
    printf("\nEnter the data:-\n");
    printf("\nFirst polynomial:\n");
    readPolynomial(&first);
    displayPolynomial(first);
    printf("Second polynomial:\n");
    readPolynomial(&second);
    displayPolynomial(second);
    addPolynomials(&result, first, second);
    displayresultPolynomial(result);
    return 0;
}
```

Output

Enter the data:-

First polynomial:

Enter the Coefficents: 1

Enter the Exponent: 3

Enter 1 for continuing or enter 0 for exiting first polynomial section: 0

Polynomial expression is: 1X^3Second polynomial:

Enter the Coefficents: 5

Enter the Exponent: 1

Enter 1 for continuing or enter 0 for exiting first polynomial section: 0

Polynomial expression is: 5X^1

The final sum expression is: 1X^3+5X^1

Result:

Polynomial Addition is successfully Implemented using Linked Lists.

Experiment No: 7

Date: 28/09/2020

Aim: Conversion of infix expression to postfix expression

Description: Implementing a program for the conversion of infix to postfix

Algorithm

Input: Infix expression

Output: Postfix expression

Data Structure: array to represent infix to postfix form and a stack

Steps

1. Push "(" onto Stack, and add ")" to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
    [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.
    [End of If]
    [End of If]
7. END.

To Get Precidence [GETPREC]

Input: An operator to find the precedence

Output: An integer representing precedence

Steps:

1. If symbol='(' then

    1. Prec=0

2. ElseIf symbol='+' OR symbol='-'

    1. Prec=1

3. ElseIf symbol='*' OR symbol='/' OR symbol='%'

1. Prec=2

4. Elself symbol='^'

1. Prec=3

5. Endif

6. Stop

Program

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define SIZE 100
char stack[SIZE];
int top = -1;
void push(char c);
char pop();
int isoperator(char symbol);
int precedence(char symbol);
void InfixToPostfix(char infix_exp[], char postfix_exp[]);
void main()
{
char infix[SIZE], postfix[SIZE];
printf("#####################################");
printf("\n\n Enter Infix expression : ");
gets(infix);
InfixToPostfix(infix,postfix);
printf("#####################################");
printf("\n Postfix Expression: ");
puts(postfix);
}
void InfixToPostfix(char infix_exp[], char postfix_exp[])
```

```
{
int i, j;
char item, x;
push('(');
strcat(infix_exp,")");
i=0;
j=0;
item=infix_exp[i];
while(item != '\0')
{
if(item == '(')
{
push(item);
}
else if( isdigit(item) || isalpha(item))
{
postfix_exp[j] = item;
j++;
}
else if(isoperator(item) == 1)
{
x=pop();
while(isoperator(x) == 1 && precedence(x)>= precedence(item))
{
postfix_exp[j] = x;
j++;
x = pop();
}
push(x);
push(item);
```

```
}
else if(item == ')')
{
x = pop();
while(x != '(')
{
postfix_exp[j] = x;
j++;
x = pop();
}
}
else
{
printf("######################################");
printf("\nInvalid infix Expression.\n");
break;
}
i++;

item = infix_exp[i];
}
if(top > 0)
printf("######################################");
postfix_exp[j] = '\0';
}
void push(char c)
{
if (top >= SIZE-1)
printf("\n Stack Overflow.");
else
```

```
{
top++;
stack[top] = c;
}
}
char pop()
{
char c;
c='\0';
if(top < 0)
printf("\n Stack Underflow.");
else
{
c = stack[top];
top--;
}
return c;
}
int isoperator(char symbol)
{
if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol == '-')
return 1;
else
return 0;
}
int precedence(char symbol)
{
if(symbol == '^')
return(5);
else if(symbol == '/')
```

```
return(4);
else if(symbol == '*')
return(3);
else if(symbol == '+')
return(2);
else if(symbol == '-')
return(1);
else
return(0);
}
```

Output

```
#####################################
 Enter Infix expression : A+(B*C-(D/E^F)*G)*H
#####################################
 Postfix Expression: ABC*DEF^/G*-H*+
```

Result

Infix expression is converted into the postfix form and the postfix evaluation is evaluated

Experiment No: 8

Date: 30/09/2020

Aim: c program to perform postfix evaluation

Description: C program to implement postfix evaluation of the expression.

Algorithm

Postfix Evaluation

Input: An expression in postfix form
Output: Result of the expression
Data Structure: A Stack to store the operand and intermediate result and an array
Steps:
1. Postfix=READ(expression)
2. Length=STRLENGTH(expression),i=0,result;
3. While(i<length)
1. Ch=postfix[i]
2. If(ch=operand)
1. PUSH(ch)
3. Else
1. Operator=ch
2. Oprnd1=POP()
3. Oprnd2=POP()
4. PERFORM (Result=oprnd2 operator oprnd1)
5. PUSH(result)
4. EndIf
5. i=i+1;
4. EndWhile
5. Result=POP()
6. Stop

Program

```
#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include<math.h>

#define max 100

int top=-1;

int stack[max];
```

```
char posfix[max];
int check(char);
int perform(int,int,int);
int main()
{
 int i,top=-1,result,val,op1,op2,op;
 char ch;
 printf("Enter the posfix expression as string : ");
 scanf("%s",posfix);
 for(i=0;posfix[i]!='\0';++i)
 {
 ch=posfix[i];
 op=check(ch);
 if(isalpha(ch))
 {
 printf("Enter the Value of %c: ",ch);
 scanf("%d",&val);
 push(val);
 }
 else if(op!=0)
 {
 op2=pop();
 op1=pop();
 val=perform(op,op1,op2);
 push(val);
```

```
 }
 else
 {
 printf(" Invalid Input! ");
 exit(0);
 }
 }
 result=pop();
 printf("Evaluated Postfix result is : %d",result);
 return 0;
}
int check(char c)
{
 switch(c)
 {
 case '^': return 6;
 case '*': return 5;
 case '/': return 4;
 case '%': return 3;
 case '-': return 2;
 case '+': return 1;
 default : return 0;
 }
}
int perform(int op , int op1 , int op2)
```

```
{
 switch(op)
 {
 case 6: return pow(op1,op2);
 case 5: return op1*op2;
 case 4: return op1/op1;
 case 3: return op1%op2;
 case 2: return op1-op2;
 case 1: return op1+op2;
 default : return 0;
 }
}
void push(int x)
{
 top=top+1;
 stack[top]=x;
}
int pop()
{
 return stack[top--];
}
```

Output

```
Enter the posfix expression as string : abc++
Enter the Value of a: 2
Enter the Value of b: 3
Enter the Value of c: 6
Evaluated Postfix result is : 11
```

Result

Infix expression is converted into the postfix form and the postfix evaluation is evaluated

Experiment No: 9
Date: 14/10/2020
Aim: To design a c program stack implementation using 1). array 2).linked list
Description: C program to implement stack using 1). array 2).linked list

Algorithm

1) Array

Push()

Input: Item to be inserted at Top
Output: A new element inserted at top
Data Structure: Array

1. If top = MAX – 1
2. Then
3. Print "Stack is full"
4. Return
5. Else
6. Top = top + 1
7. A[top] = x
8. End if
9. End PUSH( )

Pop()

Input : Item to be deleted from Top
Output : Stack with last element deleted
Data structure : Linked List

1. If top = -1
2. Then
3. Print "Stack is empty"
4. Return
5. Else
6. Item = A[top]
7. A[Top] = 0
8. Top = top – 1
9. Return item
10. End if
11. End POP( )

2) Linked list

Push()

Input: Item to be inserted at Top
Output: A new element inserted at top
Data Structure: Linked List
Steps
1.NODE:newnode
2.newnode = Getspace(node)
3.newnode-> = item
4.newnode->link = NULL
5.if (top == NULL)
    1.newnode->link = NULL
    2.top = newnode
6.else
    1.newnode->link = top
   4.top = newnode
7.endif
8.Stop

Pop()

Input : Item to be deleted from Top
Output : Stack with last element deleted
Data structure : Linked List
Steps
1. NODE:temp
2. if (top == NULL)
1. print("stack is empty")
3.else
1. temp = top
2. top = top->link
3. free(temp)
2. temp = top
4.End if
5.Stop

Program

1) Array

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 20

int push(int*,int);
int pop(int*,int);
void stats(int);

void main()
{
   int ch,stack[MAX],top=-1;
```

```
    int *st=stack;
    while(1)
    {
        printf("\nMenu\n1.Push\n2.Pop\n3.Status\n4.Exit\n");
        printf("enter the choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:top=push(st,top);
                   break;
            case 2:top=pop(st,top);
                   break;
            case 3:stats(top);
                   break;
            case 4: exit(0);
            deflaut:printf("wrong input");
                    break;
        };
    }
}

int push(int*st,int t)
{
    int item;
        printf("enter the element to be push: ");
        scanf("%d",&item);
        if(t>=MAX)
    {
        printf("stack overflow");
        return t;
    }
        else
    {
        *(st+t+1)=item;
        printf("successfully pushed");
        t++;
        return t;
    }
}

int pop(int*st,int t)
{
    int item;
    if(t==-1)
    {
      printf("stack underflow");
      return t;
    }
    else
```

```
    {
        item=*(st+t);
        printf("%d is poped",item);
        t--;
        return t;
    }
}

void stats(int t)
{
    int x;
    if(t==-1)
    {
        printf("Stack under flow");
    }
    else if(t>=MAX)
    {
        printf("stack over flow");
    }
    else
    {
        x=MAX-t-1;
        printf("free space:%d",x);
    }
}
```

2)Linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*link;
}*top=NULL;
int c=0;
void main()
{
int o;
while(1)
{
printf("\n1) Push\n2) Pop\n3) Status\n4) Exit\nEnter the option: ");
scanf("%d",&o);
switch(o)
{
case 1:
push();
break;
case 2:
```

```
pop();
break;
case 3:
status();
break;
case 4:
exit(0);
default:printf("Invalid option");
}
}
}
void push()
{
int i;
printf("Enter the element to be inserted: >>");
scanf("%d",&i);
struct node* t;
t=(struct node*)malloc(sizeof(struct node));
t->data=i;
if(top==NULL)
{
t->link=NULL;
top=t;
}
else
{
t->link=top;
top=t;
}
c=c+1;
}
void pop()
{
struct node* t;
if(top==NULL)
printf("Stact is empty!");
else
{
c=c-1;
printf("The Element at the top:>>> %d\n",top->data);
t=top;
top=top->link;
free(t);
t=top;
}
}
void status()
{
if(top == NULL)
```

```
printf("Stack is empty!!!");
else
{
printf("Number of elements present in stack are: %d\n",c);
printf("Element at top of the stack are : %d\n",top->data);
struct node* t;
t=top;
printf("Stack is : ");
while(t != NULL)
{
printf("%d ",t->data);
t=t->link;
}
}
}
```

Output:

1) Array

```
Menu
1.Push
2.Pop
3.Status
4.Exit
enter the choice:1
enter the element to be push: 54
successfully pushed

Menu
1.Push
2.Pop
3.Status
4.Exit
enter the choice:1
enter the element to be push: 34
successfully pushed

Menu
1.Push
2.Pop
3.Status
4.Exit
enter the choice:1
enter the element to be push: 90
successfully pushed

Menu
1.Push
2.Pop
3.Status
4.Exit
enter the choice:2
90 is poped

Menu
1.Push
2.Pop
3.Status
4.Exit
enter the choice:3
free space:18
```

2) Linked list

1) Push
2) Pop
3) Status
4) Exit
Enter the option: 1
Enter the element to be inserted: >>67

1) Push
2) Pop
3) Status
4) Exit
Enter the option: 1
Enter the element to be inserted: >>54

1) Push
2) Pop
3) Status
4) Exit
Enter the option: 1
Enter the element to be inserted: >>98

1) Push
2) Pop
3) Status
4) Exit
Enter the option: 2
The Element at the top:>>> 98

1) Push
2) Pop
3) Status
4) Exit
Enter the option: 3
Number of elements present in stack are: 2
Element at top of the stack are : 54
Stack is : 54 67

Result:

The jmplementation of stack using and array and linked list is successfully implemented

Experiment No: 10

Date: 14/10/2020

Aim: To design a c program to Implement queue using 1).Array and 2) linked list

Description: C program to Implement queue using linked list.

Algorithm:

1).Array

Enqueue()

Input: Item to be inserted at the rear end of the queue, rear is the pointer to rear end

Output: A new element inserted at the rear end

Data structure: Array

Steps:

1. If rear = MAX – 1
2. Then
3. Print "Queue is full"
4. Return
5. Else
6. Rear = rear + 1
7. A[rear] = x
8. If front = -1
9. Then
10. Front = 0
11. End if
12. End if
13. End enqueue( )

Dequeue()

Input: A data to be deleted from the front end of the queue

Output: Queue will be shrinked after deletion

Data Structure: Linked List

1. If front = -1
2. Then
3. Print "Queue is Empty"
4. Return
5. Else
6. Item = A[front]
7. A[front] = 0

8. If front = rear
9. Then
10. Front = rear = -1
11. Else
12. Front = front + 1
13. End if
14. Return item
15. End if
16. End dequeue( )

2)Linked list

Enqueue()

Input: Item to be inserted at the rear end of the queue, rear is the pointer to rear end

Output: A new element inserted at the rear end

Data structure: linked List

Steps:

1. NODE: newnode

2. Newnode=GETSPACE(NODE)

3. Newnode.data=item

4. Newnode.link=NULL

5. If(front=NULL)

1. Front=rear=NULL

6. Else

1. Rear.link=newnode

2. Rear=newnode

7. EndIf

8. Stop

Dequeue()

Input: A data to be deleted from the front end of the queue

Output: Queue will be shrinked after deletion

Data Structure: Linked List

Steps:

1. NODE: current

2. If(front=NULL)

1. Print(“Queue is Empty”)

3. Else if(front=rear)

1. Item=front.data

2. Delete(front)

3. Front=rear=NULL

4. Else

1. Current=front

2. Item=current.data

3. Front=front.link

4. Delete(current)

5. EndIf

6. stop

Program

1).Array

```
#include <stdio.h>
#include<stdlib.h>
#define MAX 10
void enqueue();
void mdequeue();
void status();
int ar[MAX], rear = - 1, front = - 1;
void main()
{
   int choice;
   printf("    ****************************************\n");
   printf("         QUEUE OPERATIONS USING ARRAY\n");
   printf("    ****************************************");
```

```
    while (1)
    {
        printf("\n1.Insert \n2.Delete \n3.Display \n4.Exit \n");
        printf("*****************************************");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                enqueue();
                break;
            case 2:
                mdequeue();
                break;
            case 3:
                status();
                break;
            case 4:
                exit(0);
            default:
                printf("Wrong choice \n");
        }
    }
}
void enqueue()
{
    int x;
    if(rear==MAX-1)
    printf("Queue Overflow \n");
    else
```

```
    {
        if(front==-1)
        front=0;
        printf("Insert the element in queue : ");
        scanf("%d", &x);
        rear=rear+1;
        ar[rear] = x;
    }
}
void mdequeue()
{
    int i=0,item=0;
    if(front==-1)
    {
        printf("Queue Underflow \n");
    }
    else
    {
        item=ar[front];
        if(front==rear)
         {
            rear=-1;
            front=-1;
         }
        else
        {
           while(i<rear)
           {
              ar[i]=ar[i+1];
              i=i+1;
```

```
        }
        rear=rear-1;
        front=0;
        printf("Element Deleted from queue is : %d", item);
      }
   }
   }
void status()
{
   int r;
   printf("\nThe current top element is %d",ar[front]);
   if((MAX-1)==rear)
   printf("\nNo free space.");
   else
   r=(MAX-1)-rear;
   printf("\nTotal free space=%d",r);
}
```

2).Linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node*link;
}*front=NULL,*rear=NULL;
int c=0;
void main()
{
int o;
```

```
while(1)
{
printf("\n\n1) Enqueue\n2) Dequeue\n3) Status\n4) Exit\nEnter your option: ");
scanf("%d",&o);
switch(o)
{
case 1:
enqueue();
break;
case 2:
dequeue();
break;
case 3:
status();
break;
case 4: exit(0);
default:printf("Invalid option");
}
}
}
void enqueue()
{
int i;
printf("Enter the element to be inserted: ");
scanf("%d",&i);
struct node* t;
t=(struct node*)malloc(sizeof(struct node));
t->data=i;
t->link=NULL;
if(front==NULL && rear == NULL)
```

```
{
front=t;
rear=t;
}
else
{
rear->link=t;
rear=t;
}
c=c+1;
}
void dequeue()
{
struct node* t;
if(front==NULL)
printf("Queue is empty.");
else
{
printf("Element to be served: %d",front->data);
if(front==rear)
{
front=NULL;
rear=NULL;
}
else
{
t=front;
front=front->link;
free(t);
}
```

```
c=c-1;
}
}
void status()
{
if(c==0)
printf("Queue is empty!!!");
else
{
printf("Element at front: %d\nElement at rear: %d\n",(front->data),(rear->data));
printf("Number of elements present in the queue: %d\n",c);
printf("Elements in the queue: ");
struct node* t;
t=front;
while(t!=NULL)
{
printf("%d ",t->data);
t=t->link;
}
}
}
```

Output:

1) Array

```
   ****************************************
      QUEUE OPERATIONS USING ARRAY
   ****************************************
1.Insert
2.Delete
3.Display
4.Exit
*****************************************
Enter your choice : 1
Insert the element in queue : 23

1.Insert
2.Delete
3.Display
4.Exit
*****************************************
Enter your choice : 1
Insert the element in queue : 34

1.Insert
2.Delete
3.Display
4.Exit
*****************************************
Enter your choice : 2
Element Deleted from queue is : 23
1.Insert
2.Delete
3.Display
4.Exit
*****************************************
Enter your choice : 3

The current top element is 34
Total free space=9
```

2) Linked list

1) Enqueue

2) Dequeue

3) Status

4) Exit

Enter your option: 1

Enter the element to be inserted: 45

1) Enqueue

2) Dequeue

3) Status

4) Exit

Enter your option: 1

Enter the element to be inserted: 46

1) Enqueue

2) Dequeue

3) Status

4) Exit

Enter your option: 1

Enter the element to be inserted: 78

1) Enqueue

2) Dequeue

3) Status

4) Exit

Enter your option: 2

Element to be served: 45

1) Enqueue

2) Dequeue

3) Status

4) Exit

Enter your option: 3

Element at front: 46

Element at rear: 78

Number of elements present in the queue: 2

Elements in the queue: 46 78

Result:

The jmplementation of stack using and array and linked list is successfully implemented

Experiment No: 11
Date: 17/10/2020
Aim: Implementation of circular linked list.
Description: C program to implement a circular linked list

Algorithm

Insertion at front

Input: item to be inserted at the beginning.
Output: item inserted and last node contains the link to the header node.
Data structure: linked list
Steps:
1. NODE c, n
2. n=GETSPACE(NODE)
3. n.data=item
4. if(header=NULL)
1. header=n
2. n.link=header
5. Else
1. c=header
2. while(c.link != header)
1. c=c.link
3. End while
4. c.link=n
5. n.link=header
6. header=n
6. End if
7. Stop

Insertion at end

Input: item to be inserted at the end
Output: item inserted at the end and end node links to the header node.
Data structure: linked list
Steps:
1. NODE c, n
2. n=GETSPACE(NODE)
3. n.data=item
4. if(header=NULL)
1. header=n
2. header.link=n
5. Else
1. c=header
2. while(c.link != header)
1. c=c.link
3. End while
4. c.link=n
5. n.link=header
6. End if

7. Stop

Deletion from front

Input: circular linked list
Output: header element is deleted
Data structure: linked list
Steps:
1. NODE c, p
2. if(header=NULL)
1. print(“Empty”)
2. Exit
3. Else
1. item=header.data
2. if(header.link=header)
1. FREE(header)
2. header=NULL
3. Else
1. c=header
2. Do
1. p=c
2. c=c.link
3. while(c != header)
3. End do while
4. p.link=c.link
5. header=header.link
6. FREE(c)
4. End if
4. End if
5. Stop

Delete from rear

Input: circular linked list
Output: item at the rear is deleted
Data structure: linked list
Steps:
1. NODE p, c
2. if(header=NULL)
1. print(“Empty”)
2. Exit
3. Else if(header.link=header)
1. item=header.data
2. header=NULL
4. Else
1. c=header
2. while(c.link != header)
1. p=c
2. c=c.link
3. End while
4. item=c.data

5. p.link=header
6. FREE(c)
5. End if
6. Stop

Program:

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

struct node
{
   int data;
   struct node *next;
};
struct node *head;

void beginsert ();
void lastinsert ();
void begin_delete ();
void last_delete ();
void display ();

void main()
{
   int choice=0;

   while(choice!= 7)
   {
      printf("\n*********************************************\n");
      printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
last\n5.Display\n6.Exit\n");
      printf("\n*********************************************\n");
      printf("\nEnter your choice->\n");
      scanf("\n%d",&choice);
      switch(choice)
      {
         case 1:
         beginsert();
         break;
         case 2:
         lastinsert();
         break;
         case 3:
         begin_delete();
         break;
         case 4:
         last_delete();
```

```
            break;
            case 5:
            display();
            break;
            case 6:
            exit(0);

            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter the value");
        scanf("%d",&item);
        ptr -> data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
                temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
        }
        printf("\nnode inserted\n");
    }

}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
```

```
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp -> next != head)
            {
                temp = temp -> next;
            }
            temp->next=ptr;
            ptr->next=head;
        }

        printf("\nnode inserted\n");
    }

}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }

    else
    {   ptr = head;
        while(ptr -> next != head)
            ptr = ptr -> next;
        ptr->next = head->next;
```

```
        free(head);
        head = ptr->next;
        printf("\nnode deleted\n");

    }
}
void last_delete()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");

    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
        {
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
        free(ptr);
        printf("\nnode deleted\n");

    }
}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n Values u have entered ->> \n");

        while(ptr -> next != head)
        {
```

```
            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }

}
```

Output:

```
*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************

Enter your choice->
1
Enter the value34
node inserted

*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************

Enter your choice->
1
Enter the value46
node inserted

*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************

Enter your choice->
1
Enter the value87
node inserted

*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
```

```
5.Display
6.Exit
*****************************************

Enter your choice->
2
Enter Data?44
node inserted

*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************

Enter your choice->
2
Enter Data?78
node inserted

*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************

Enter your choice->
3
node deleted

*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************
Enter your choice->
4
node deleted

*****************************************
```

```
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************
Enter your choice->
5
 Values u have entered ->>
46
34
44

*****************************************
1.Insert in begining
2.Insert at last
3.Delete from Beginning
4.Delete from last
5.Display
6.Exit
*****************************************

Enter your choice->
6
```

Result:

The program for circular linked list is implemented successfully

Experiment No: 12
Date: 11/11/2020
Aim: develop a program to perform implementation of binary search tree
Description: C program for the implementation of binary search tree using linked list

Algorithm

Inorder Traversal

Input: root is the pointer to the root node of the binary search tree
Output: Visiting of all the nodes in preorder fashion
Data structure: linked structure
Steps:
1. Current=root
2. If(current!=NULL)
1. VISIT(current)
2. INORDER(current.llink)
3. INORDER(current.rlink)
3. EndIf
4. Stop

Postorder Traversal

Input: root is the pointer to the root node of the binary search tree
Output: Visiting of all the nodes in postorder fashion
Data structure: linked structure
Steps:
1. Current=root
2. If(current!=NULL)
1. POSTORDER(current.llink)
2. POSTORDER(current.rlink)
3. VISIT(current)
3. EndIf
4. Stop

Preorder Traversal

Input: root is the pointer to the root node of the binary search tree
Output: Visiting of all the nodes in postorder fashion
Data structure: linked structure
Steps:
1. Current=root
2. If(current!=NULL)
1. VISIT(current)
2. PREORDER(current.llink)
3. PREORDER(current.rlink)
3. EndIf
4. Stop

Insertion

Input: Item is the data component of a node that has to be inserted.

Output: Item inserted in to the proper place of the tree.
Data structure: Linked structure
Steps:
1. NODE: newnode, current, parent
2. Newnode=GETSPACE(newnode)
3. Newnode.data=item
4. Newnode.llink=NULL
5. Newnode.rlink=NULL
6. If(root=NULL)
1. Root=newnod
7. Else
1. Current=root
2. While(current!=NULL)
1.Parent=current
2.If(item>current.data)
1. Current=current->rlink
3.Else
1. Current=current->llink
4.EndIf
3. EndWhile
4. If(item>parent.data)
1.Parent.rlink=newnode
5. Else
1.Parent.llink=newnode
6. EndIf
8. EndIf
9. Stop.

Deletion

Input: item is the data to be removed
Output: if the node with data as item exist it is deleted else a message.
Data Structure: Linked structure of binary tree, root will point to the root node
Steps:
1. NODE: current, parent
2. Flag=0
3. Parent=NULL
4. If(root=NULL)
1. Print("Tree is Empty")
2. Exit
5. EndIf
6. Current=root
7. While(current!=NULL)
1. If(current.data=item)
1.Flag=1
2.ExitWhile
2. Else
1.Parent=current
2.If(item>current.data)
1. Current=current.rlink

3.Else
1. Current=current.llink
4.EndIf
3. EndIf
8. EndWhile
9. If(flag=0)
1. Print("No item found in the tree")
2. Exit
10. EndIf
11. If(current.llink=NULL AND current.rlink=NULL)
1. If(parent=NULL)
1.FREESPACE(current)
2.Root=NULL
2. ElseIf(parent.llink=current)
1.Parent.llink=NULL
2.FREESPACE(current)
3. ElseIf(parent.rlink=current)
1.Parent.rlink=NULL
2.FREESPACE(current)
4. EndIf
12. ElseIf(current.llink=NULL AND current.rlink!=NULL)
1. If(parent=NULL)
1.Root=root.rlink
2.FREESPACE(current)
2. ElseIf(parent.llink=current)
1.Parent.llink=current.rlink
2.FREESPACE(current)
3. ElseIf(parent.rlink=current)
1.Parent.rlink=current.rlink
2.FREESPACE(current)
4. EndIf
13. ElseIf(current.llink!=NULL AND current.rlink=NULL)
1. If(parent=NULL)
1.Root=root.llink
2.FREESPACE(current)
2. ElseIf(parent.llink=current)
1.Parent.llink=current.llink
2.FREESPACE(current)
3. ElseIf(parent.rlink=current)
1.Parent.rlink=current.llink
2.FREESPACE(current)
4. EndIf
14. ElseIf(current.llink!=NULL AND current.rlink!=NULL)
1. NODE temp
2. Temp=current.rlink
3. If(temp.llink=NULL AND temp.rlink=NULL)
1.Current.data=temp.data;
2.Current.rlink=NULL
3.FREESPACE(temp)

4. ElseIf(current.rlink.llink!=NULL)
1.NODE temp1,temp2
2.Temp1=current.rlink.llink
3.If(temp1.llink=NULL)
1. Temp2=current.rlink
4.Else
1. While(temp1.llink!=NULL)
1. Temp2=temp1
2. Temp1=temp1.llink
2. EndWhile
5.EndIf
6.Current.data=temp1.data
7.Temp2.link=NULL
8.FREESPACE(temp1)
5. ElseIf(current.rlink.llink=NULL)
1.NODE temp
2.Temp=current.rlink
3.Current.data=temp.data
4.Current.rlink=temp.rlink
5.FREESPACE(current)
6. EndIf
15. EndIf
16. Stop

Program

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
   int data;
   struct Node *left,*right;
}*root=NULL;
void insertionatbin();
void preorder(struct Node *temp);
void inorder(struct Node *temp);
void postorder(struct Node *temp);
void search();
void deletion();
int main()
{
   int ch;
   printf("\t################################################\n");
   while(1)
   {
      printf("\n1. Preorder Traverse\n2. Inorder Traverse\n3. Postorder Traverse\n4. Insertion");
      printf("\n5. Deletion\n6. Search\n7. Exit");
      printf("\nEnter the choice: ");
      scanf("%d",&ch);
```

```
      printf("\t################################################\n");
      switch(ch)
      {
         case 1: preorder(root);
                 break;
         case 2: inorder(root);
                 break;
         case 3: postorder(root);
                 break;
         case 4: insertionatbin();
                 break;
         case 5: deletion();
                 break;
         case 6: search();
                 break;
         case 7: exit(0);
         default:printf("Wrong input");
      };
   }
}
void insertionatbin()
{
   int ele;
   struct Node*newnode,*curr,*parnt;
   printf("\nEnter the element to be inserted: ");
   scanf("%d",&ele);
   newnode=(struct Node*)malloc(sizeof(struct Node));
   newnode->data=ele;
   newnode->left=NULL;
   newnode->right=NULL;
   if(root==NULL)
      root=newnode;
   else
   {
     curr=root;
     while(curr!=NULL)
     {
         parnt=curr;
         if(ele > curr->data)
            curr=curr->right;
         else
            curr=curr->left;
     }
      if(ele > parnt->data)
         parnt->right=newnode;
      else
         parnt->left=newnode;
   }
   printf("\nSuccessfully inserted");
```

```
}
void preorder(struct Node*temp)
{
    if(temp!=NULL)
   {
       printf("%d ",temp->data);
       preorder(temp->left);
       preorder(temp->right);
   }
   else
       return;
}
void inorder(struct Node *temp)
{
   if(temp!=NULL)
  {
     inorder(temp->left);
     printf("%d ",temp->data);
     inorder(temp->right);
   }
   else
       return;
}
void postorder(struct Node *temp)
{
   if(temp!=NULL)
  {
     postorder(temp->left);
     postorder(temp->right);
     printf("%d ",temp->data);
   }
   else
       return;
}
void search()
{
   int ele,flag=0;
   struct Node*curr;
   printf("\nEnter the element to be searched: ");
   scanf("%d",&ele);
   curr=root;
         while(curr!=NULL)
         {
       if(curr->data==ele)
       {
          flag=1;
          break;
       }
       else if(ele>curr->data)
```

```
          curr=curr->right;
       else
          curr=curr->left;
         }
         if (flag==1)
       printf("Found");
    else
       printf("Not found");

}
void deletion()
{
  int ele,flag=0;
    struct Node*curr,*parnt;
    printf("\nEnter the element to be deleted: ");
    scanf("%d",&ele);
    curr=root;
          while(curr!=NULL)
          {
       if(curr->data==ele)
       {
          flag=1;
          break;
       }
       else
       {
          parnt=curr;
          if(ele>curr->data)
             curr=curr->right;
          else
             curr=curr->left;
       }
       }
         if (flag==0)
         {
       printf("Element not existing!!");
       return;
         }
         if(curr->left==NULL && curr->right==NULL)          //if leafnode
         {
       if(parnt==NULL)
       {
          free(curr);
          root=NULL;
       }
       else if(parnt->left==curr)
       {
          parnt->left=NULL;
          free(curr);
```

```
        }
        else if(parnt->right==curr)
        {
            parnt->right=NULL;
            free(curr);
        }
    }
    else if(curr->left==NULL &&curr->right!=NULL)
    {
        if(parnt==NULL)
        {
            root=root->right;
            free(curr);
        }
        else if(parnt->left==curr)
        {
            parnt->left=curr->right;
            free(curr);
        }
        else if(parnt->right==curr)
        {
            parnt->right=curr->right;
            free(curr);
        }
    }
    else if(curr->left!=NULL &&curr->right==NULL)
    {
        if(parnt==NULL)
        {
            root=root->left;
            free(curr);
        }
        else if(parnt->left==curr)
        {
            parnt->left=curr->left;
            free(curr);
        }
        else if(parnt->right==curr)
        {
            parnt->right=curr->left;
            free(curr);
        }
    }
    printf("Element is deleted Successfully");
}
```

Output

```
1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 4
      Enter the element to be inserted: 65
Successfully inserted

1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 4
Enter the element to be inserted: 67
Successfully inserted

1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 4
Enter the element to be inserted: 78
Successfully inserted

1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 6
Enter the element to be searched: 78
Found

1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
```

4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 5
Enter the element to be deleted: 78
Element is deleted Successfully

1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 2
65 67

1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 1
65 67

1. Preorder Traverse
2. Inorder Traverse
3. Postorder Traverse
4. Insertion
5. Deletion
6. Search
7. Exit
Enter the choice: 3
67 65

Result

The operation on implementing binary search tree using linked list is successfully implementing

Experiment no: 13
Date: 25-11-2020
AIM: Perform BFS and DFS of a graph.
Description: BFS queue is used for performing the task and in DFS stack is used.

Algorithm

BFS
Input: adjacency matrix representation of the graph(G), number of nodes in the graph(n).
Output: nodes displayed after BFS.
Data structure: array
Steps:
1. rear=0,front=0
2. queue[r]=1
3. r=r+1
4. i=queue[front]
5. while(i<n+1)
1. j=1
2. while(j<n+1)
1. if(G[i][j]=1)
1. flag=0, k=0
2. while(k<rear)
1. if(queue[k]=j)
1. flag=1
2. k=k+1
3. End while
4. if(flag=0)
1. queue[rear]=j
2. front=front+1, rear=rear+1
5. End if
2. End if
3. j=j+1
3. End while
4. i=i+1
6. End while
7. Stop

DFS
Input: adjacency matrix representation of the graph(G), number of nodes(n)
Output: nodes displayed after DFS.
Data structure: array
Steps:
1. i=0, top=-1
2. while(i<n+1)
1. visit[i]=0
2. i=i+1

3. if(top=n)
1. printf(“Overflow”)
4. Else
1. top=top+1
2. stack[top]=1
5. while(top !=-1)
1. if(top=-1)
1. print(“underflow”)
2. Else
1. v=s[top]
2. top=top-1
3. if(visit[v]=0)
1. print(v)
2. visit[v]=1
4. i=1
5. while(i<=n)
1. if(G[v][i]=1 and visit[i]=0)
1. if(top=n)
1. printf(“Overflow”)
2. Else
1. top=top+1
2. stack[top]=i
3. End if
2. End if
3. i=i+1
6. End while
6. End while
7. Stop

Program

```
#include<stdio.h>
#include<stdlib.h>
int G[10][10],s[10],q[10],visit[10],n,top=-1;
void dfs()
{
int i,v;
for(i=0;i<n;i++)
visit[i]=0;
push(1);
printf("\nDepth first search result: ");
while(top != -1)
{
v=pop();
if(visit[v]==0)
{
printf("A%d ",v);
visit[v]=1;
}
```

```
for(i=1;i<=n;i++)
{
if(G[v][i]==1 && visit[i]==0)
push(i);
}
}
}
void push(int i)
{
if(top==n-1)
printf("Overflow condition!!!");
else
s[++top]=i;
}
int pop()
{
int i;
if(top==-1)
printf("Underflow condition!!!");
else
{
i=s[top--];
return i;
}
}
void bfs()
{
int r=0,f=0,i,j,re;
q[r++]=1;
for(i=q[f];i<n+1;i++)
{
for(j=1;j<n+1;j++)
{
if(G[i][j]==1)
{
re=0;
int k=0;
while(k<r)
{
if(q[k]==j)
re=1;
k++;
}
if(re == 0)
{
q[r++]=j;
f++;
}
}
```

```
}
}
int k;
printf("Breath first search result: ");
for(k=0;k<r;k++)
printf("A%d ",q[k]);
}
void main()
{
int i,j,o;
printf("Enter number of vertices:");
scanf("%d",&n);
for(i=1;i<n+1;i++)
{
for(j=1;j<n+1;j++)
{
printf("Do we have edge between A%d and A%d (1 yes/0 no): ",i,j);
scanf("%d",&G[i][j]);
}
}
printf("Adjacency matrix of the following input: \n");
for(i=1;i<n+1;i++)
{
for(j=1;j<n+1;j++)
printf("%d ",G[i][j]);
printf("\n");
}
bfs();
dfs();
return 0;
}
```

Output

```
Enter the number of vertices:3
Is theere an edge between A1 and A1 (Enter 1 for yes and 0 for no ): 1
Is theere an edge between A1 and A2 (Enter 1 for yes and 0 for no ): 0
Is theere an edge between A1 and A3 (Enter 1 for yes and 0 for no ): 1
Is theere an edge between A2 and A1 (Enter 1 for yes and 0 for no ): 1
Is theere an edge between A2 and A2 (Enter 1 for yes and 0 for no ): 1
Is theere an edge between A2 and A3 (Enter 1 for yes and 0 for no ): 0
Is theere an edge between A3 and A1 (Enter 1 for yes and 0 for no ): 0
Is theere an edge between A3 and A2 (Enter 1 for yes and 0 for no ): 0
Is theere an edge between A3 and A3 (Enter 1 for yes and 0 for no ): 1
The Adjacency matrix of the entered inputs are :
1 0 1
1 1 0
0 0 1
The results of breadth first search: A1 A3 A2
 The result of depth first search : A1 A3
```

Result

The program is executed and the required output is obtained.

Experiment no: 14
Date: 24-11-2020
Aim: Resolve the collisions if any using collision resolution techniques like closed hashing
Description:
Hashing is the method used for efficient storage and retrieval of data in the system.

Algorithm

Main

Input: element to be inserted(item), hashing function(%), size of array(size), pointer to point at the visit array(t), hash table(a).
Output: items inserted at proper order via hash function and collision is resolved via linear probing.
Data structure: array
Steps:
1. key=item%size
2. if(key<0)
1. key=key*-1
3. End if
4. i=key
5. t=0
6. if(check(key))
1. a[key]=item
2. visit[t]=key
3. t=t+1
7. Else
1. Do
1. key=key+1
2. key=key%size
3. if(check(key))
1. a[key]=item
2. visit[t]=key
3. t=t+1
4. End if
2. while(key != i)
3. if(key=i)
1. print("Not inserted!!!")
4. End if
8. End if
9. Stop

Function check

Input: key index to be checked if it is empty or not
Output: 1 if key index is empty, 0 if index is filled
Data structure: array
Steps:

1. check(a)
1. f=0,i=0
2. while(i<t)
1. if(visit[i]=a)
1. f=f+1
2. End if
3. i=i+1
3. End while
4. if(f==0)
1. return 1
5. Else
1. return 0
6. End if
2. Stop function

Program

```
#include<stdio.h>
int a[10],v[11],t=0;
int check(int a)
{
int i,f=0;
for(i=0;i<t;i++)
{
if(v[i]==a)
f+=1;
}
if(f==0)
return 1;
else
return 0;
}
void main()
{
int k,i,e,o;
while(1)
{
printf("Enter data: ");
scanf("%d",&e);
k=e%10;
if(k<0)
k*=-1;
i=k;
if(check(k))
{
a[k]=e;
v[t++]=k;
}
else
```

```
{
do
{
k++;
k=k%10;
if(check(k))
{
a[k]=e;
v[t++]=k;
break;
}
}while(k!=i);
if(k==i)
printf("Element not inserted!!!");
}
printf("\nDo you want to continue: (Yes=1/No=0) ");
scanf("%d",&o);
if(o==0)
break;
}
printf("Element now:\n");
for(i=0;i<10;i++)
{
if(!check(i))
printf("%d -> %d",i,a[i]);
printf("\n");
}
}
```

Output

Enter data: 7

Do you want to continue: (Yes=1/No=0) 1
Enter data: 2

Do you want to continue: (Yes=1/No=0) 1
Enter data: 3

Do you want to continue: (Yes=1/No=0) 1
Enter data: 3

Do you want to continue: (Yes=1/No=0) 0
Element now:

2 -> 2
3 -> 3
4 -> 3

7 -> 7

Result

The program is executed and the required result is obtained.