**SHRI SHANKARACHARYA GROUP OF INSTITUTIONS**

FACULTY OF ENGINEERING AND TECHNOLOGY

# CERTIFICATE

THIS IS TO CERTIFY THAT THIS PRACTICAL RECORD CONTAINS THE BONAFIDE PRACTICAL WORK FOR THE SUBJECT

## "DATA STRUCTURES LAB MANUAL USING'C' "

## VINOD KUMAR SARTHI

DURING THE ACADEMIC SESSION 2018-2019
OF 4th SEMESTER SECTION "C"

ROLL NO. 36
DATE :24/04/2019

SIGNATURE OF HOD

SIGNATURE OF LECTURER

| Sr no. | EXPERIMENT NAME | EXP. DATE | SUBMISSION DATE | SIGNATURE |
|---|---|---|---|---|
| 1 | Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary). | 22/01/19 | 29/01/19 | |
| 2 | Write a program to implement stack and perform push, pop operation. | 29/01/19 | 5/02/19 | |
| 3 | Write a program to convert Infix expression to postfix expression using stack | 29/01/19 | 5/02/19 | |
| 4 | Write a program to perform the following operations in linear queue – addition, deletion, and traversing. | 5/02/19 | 12/02/19 | |
| 5 | Write a program to perform the following operations in circular queue – addition, deletion, and traversing | 5/02/19 | 12/02/19 | |
| 6 | Write a program to perform the following operations in singly linked list – creation, insertion, and deletion. | 12/02/19 | 19/02/19 | |
| 7 | Write a program to perform the following operations in singly linked list – creation, insertion, and deletion. | 19/02/19 | 26/02/19 | |
| 8 | Write a program to perform the following operations in doubly linked list – creation, insertion, and deletion | 26/02/19 | 5/03/19 | |
| 9 | Write a program to implement polynomial in linked list and perform the following<br>a. Arithmetic.<br>b. Evaluation. | 5/03/19 | 12/03/19 | |
| 10 | Write programs to implement linked stack and linked queue. | 12/03/19 | 26/03/19 | |
| 11 | Write programs to perform Insertion sort, Selection sort, and Bubble sort. | 26/03/19 | 2/04/19 | |

| | | | | |
|---|---|---|---|---|
| 12 | Write a program to perform Quick sort | 2/04/19 | 9/04/19 | |
| 13 | Write a program to perform Merge sort. | 2/04/19 | 9/04/19 | |
| 14 | Write a program to perform Heap sort. | 9/04/19 | 16/04/19 | |
| 15 | Write a program to create a binary search tree and perform – insertion, deletion, and traversal. | 9/04/19 | 16/04/19 | |
| 16 | Write a program for traversal of graph (B.F.S., D.F.S.). | 16/04/19 | 16/04/19 | |

# DATA STRUCTURES LAB MANUAL USING‘C’

# LIST OF EXPERIMENTS

**1)** Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).
**2)** Write a program to implement stack and perform push, pop operation.
**3)** Write a program to convert Infix expression to postfix expression using stack.
**4)** Write a program to perform the following operations in linear queue – addition, deletion, and traversing.
**5)** Write a program to perform the following operations in circular queue – addition, deletion, and traversing.
**6)** Write a program to perform the following operations in double ended queue – addition, deletion, and traversing.
**7)** Write a program to perform the following operations in singly linked list – creation, insertion, and deletion.
**8)** Write a program to perform the following operations in doubly linked list – creation, insertion, and deletion.
**9)** Write a program to implement polynomial in linked list and perform the following
    c. Arithmetic.
    d. Evaluation.
**10)** Write programs to implement linked stack and linked queue.
**11)** Write programs to perform Insertion sort, Selection sort, and Bubble sort.
**12)** Write a program to perform Quick sort.
**13)** Write a program to perform Merge sort.
**14)** Write a program to perform Heap sort.
**15)** Write a program to create a binary search tree and perform – insertion, deletion, and traversal.
**16)** Write a program for traversal of graph (B.F.S., D.F.S.).

# EXPERIMENT No.1 (a)

## Aim:- Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).

## Theory:

1. Locate the position where the element in to be inserted (position may be user-specified in case of an unsorted list or may be decided by search for a sorted list).
2. Reorganize the list and create an 'empty' slot.
3. Insert the element.

Example: (Sorted list)

| Data: | 345 | 358 | 490 | 501 | 513 | 555 | 561 | 701 | 724 | 797 |
|---|---|---|---|---|---|---|---|---|---|---|
| Location: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Insert 505 onto the above list:

1. Locate the appropriate position by performing a binary search. 505 should be stored in location 4.
2. Create an 'empty' slot

| Data: | 345 | 358 | 490 | 501 | 513 | 555 | 561 | 701 | 724 | 797 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Location: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

3. Insert 505

| Data: | 345 | 358 | 490 | 501 | 505 | 513 | 555 | 561 | 701 | 724 | 797 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Location: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

## Source Code:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 20
/********** Function Declaration begins **********/
int insert(int[],int,int,int);
void traverse(int[],int);
/********** Function Declaration ends **********/
void main()
{
        int i=0,A[SIZE],n,pos,item;
        clrscr();
        printf("\n\n\t\t Program to insert element in 1-Dimensional array: ");
        printf("\n\n\t\tHow many number you want to store in the array: ");
        scanf("%d",&n);
        while(i<n)
        {
                printf("\n Enter value A[%d]: ",i);
```

```
                scanf("%d",&A[i]);
                i++;
        }
        traverse(A,n);
        printf("\nEnter the index to insert new number:
        "); scanf("%d",&pos);
        printf("\nEnter the number: ");
        scanf("%d",&item);
        n = insert(A,n,pos,item);
        traverse(A,n);
        getch();
}
/********** Traversing array elements **********/
/********** Function Definition begins **********/
void traverse(int A[], int n)
{
        int i=0;
        printf("\n\n\t\t elements of array
        are:\n"); while(i<n)
        {
                printf("A[%d]:  ",i);
                printf("%d\n",A[i]);
                i++;
        }
        printf("\n");
}
/********** Function Definition ends **********/

/********** inserting array element **********/
/********** Function Definition begins **********/
int insert(int A[], int n, int pos, int item)
{
        int i;
        for(i=n;i>=pos;i—)
                A[i+1] = A[i];
        A[pos] = item;
        n= n+1;
        return n;
}
/********** Function Definition ends **********/
```

## **Output:**

```
Program to insert an element from 1-Dimensional array:
How many number you want to store in the array:6

Enter value A[0]: 11
Enter value A[1]: 22
Enter value A[2]: 33
Enter value A[3]: 44
Enter value A[4]: 55
Enter value A[5]: 66
                elements of array are:
A[0]: 11
A[1]: 22
A[2]: 33
A[3]: 44
A[4]: 55
A[5]: 66

Enter the index to insert new number:
3 Enter the number: 88
                elements of array are:
A[0]: 11
A[1]: 22
A[2]: 33
A[3]: 88
A[4] :44
A[5]: 55
A[6]: 66
```

# EXPERIMENT No.1 (b)

## Aim:- Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).

## Theory:

1. Locate the element in the list (this involves searching).
2. Delete the element.
3. Reorganize the list and index.

**Example:**

Data: 345 358 490 501 513 555 561 701 724 797 Location: 0 1 2 3 4 5 6 7 8 9

Delete 358 from the above list:

1. Locate 358: If we use 'linear search', we'll compare 358 with each element of the list starting from the location 0.
2. Delete 358: Remove it from the list (space=10).

   Data: 345 490 501 513 555 561 701 724 797

   Location: 0 1 2 3 4 5 6 7 8 9
3. Reorganize the list: Move the remaining elements. (Space=9)

   Data: 345 490 501 513 555 561 701 724 797 ? (797)

   Location: 0 1 2 3 4 5 6 7 8 9

## Source code:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define SIZE 20
/********** Function Declaration begins **********/
int deletion(int[],int,int);
void traverse(int[],int);
/********** Function Declaration ends **********/

void main()
{
        int i=0,A[SIZE],n,pos;
        clrscr();
        printf("\n\n\t\t Program to delete an element from 1-Dimensional array: ");
        printf("\n\n\t\t How many number you want to store in the array: ");
        scanf("%d",&n);
        while(i<n)
        {
```

```
                printf("\nEnter value A[%d]:
                ",i); scanf("%d",&A[i]);
                i++;
        }
        traverse(A,n);
        printf("\nEnter the index for deleting the number: ");
        scanf("%d",&pos);
        n = deletion(A,n,pos);
        traverse(A,n); getch();

}

/********** Traversing array elements **********/
/********** Function Definition begins **********/
void traverse(int A[], int n)
{
        int i=0;

        while(i<n)
        {
                printf("\n A[%d]:",i);
                printf("%d\n",A[i]);
                i++;
        }
        printf("\n");
}
/********** Function Definition ends **********/
/********** Deleting array element **********/
/********** Function Definition begins **********/
int deletion(int A[], int n, int pos)
{
        int item;

        item = A[pos];
        printf("Deleted item from the index %d is :
        %d\n",pos,item); while(pos<=n)
        {
                A[pos] = A[pos+1];
                pos++;
        }
        n= n-1;
        return n;
}
/********** Function Definition ends **********/
```

**Output:**

```
Program to delete an element from 1-Dimensional array:
How many number you want to store in the array: 6
Enter value A[0]: 11
Enter value A[1]: 22
Enter value A[2]: 33
Enter value A[3]: 44
Enter value A[4]: 55
Enter value A[5]: 66
A[0]: 11
 A[1]: 22
 A[2]: 33
 A[3]:44
 A[4]:55
 A[5]:66
Enter the index for deleting the number:
3 Deleted item from the index 3 is: 44
A[0]:11
 A[1]:22
 A[2]:33
 A[3]:55
 A[4]:66
```

# EXPERIMENT No.1(c) (Linear search)

## Aim:- Write a program to perform the following in one dimensional array, Insertion, Deletion, and Searching (Linear and Binary).

## Theory:

In this algorithm in the set of '*N*' data item is given—$D1, D2 .... Dn$ having $k1, k2 .... kN$, '*N*' distinct respective keys. If the desired record is located that contains the key '$k_i$' then the search is successful otherwise unsuccessful. We assume that $N \geq 1$.

Step 1 Initialization
Set i ← 1.

Step 2 Loop, Comparison
while ( i < = N)
{
if (k = $k_i$) then
{
message : ''successful search''
display (k) go to step 4
}
else
Set i ← i + 1
}
End of loop.

Step 3 If no match
If (k ≠ $k_i$) then
message : "unsuccessful search".

Step 4 Finish
Exit.

## Source Code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[100],n,i,item,loc=-
1; clrscr();
```

```
printf("\nEnter the number of element:");
scanf("%d",&n);
printf("Enter the number:\n");
for(i=0;i<=n-1;i++)
  {
   scanf("%d",&a[i]);
  }
printf("Enter the no. to be
search\n"); scanf("%d",&item);
for(i=0;i<=n-1;i++)
  {
   if(item==a[i])
     {
      loc=i;
      break;
    }
  }
if(loc>=0)

  printf("\n%dis found in position%d",item,loc+1);
  else
  printf("\nItem does not
  exits"); getch();
    }
```

## **Output:**

```
How many elements:
5
Enter element of the
array: 2 5 8 1 3
Enter the element to be
searched: 8
Search is Successful
Position of the item searched , 3.


How many elements:
7
Enter element of the
array: 2 5 8 1 3 12 45
Enter the element to be
searched: 4
Search is Unsuccessful
```

```
/***************Function Declaration End**********/
void main()
{
    node *START=NULL;
    int ch;
    do
    {
        printf("\n\t\t Program for singly linked list\n");
        printf("\n\t\t\t Menu:\n");
        printf("\n\t\t1.Create");
        printf("\n\t\t2.Insert");
        printf("\n\t\t3.Delete");
        printf("\n\t\t4.Display");
        printf("\n\t\t5.Exit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch); switch(ch)
            {
                case 1 :
                    START = SLcreation(START);
                    break;
                case 2:
                    START =
                    SLinsertion(START); break;
                case 3:
                    START = SLdeletion(START);
                    break;
                case 4:
                    printf("\n***** Linked list *****\n");
                    SLdisplay(START);
                    break;
                case 5:
                    exit(0);
                default :
                    printf("\nWrong choice:");
            }
        }
```

```
        while (ch!=5);
        printf("\n");
}

/********** Creating of linked list MENU **********/
/********** Function Definition begins **********/
node *SLcreation(node *START)
{
    node *temp,*prev;
    int item;
    char ch;
    prev = START =
    NULL; do
    {
        printf("\n\t\t Menu:");
        printf("\n\t\t1.Add node");
        printf("\n\t\t2. Display:");
        printf("\n\t\t3. Quit:");
        printf("\n\t\tEnter choice:");
        scanf("%d",&ch); switch(ch)

        {
            case 1:
                printf("\nEnter data:");
                scanf("%d",&item);
                temp = (node*)malloc(sizeof(node));
                temp->data = item;
                temp->link = NULL;
                if (START == NULL)
                    START = temp;
                else
                    prev->link = temp;
                prev = temp;
                break;
            case 2:
                printf("\n***** Linked list *****\n");
                SLdisplay(START);
```

```
                case 3:
                        break;
                default:
                                printf("\nWrong choice:");
                }
        }while (ch != 3);
        return START;
}
/********** Function Definition ends **********/

/********** Insertion of node in linked list **********/
/********** Function Definition begins **********/
node* SLinsertion(node *START)
{
        node *new_node, *temp;
        int i,item,pos;


        printf("\nEnter data to be inserted : ");
        scanf("%d",&item);
        do
        {
                printf("\nEnter the position of insertion :
                "); scanf("%d",&pos);
        }
        while (pos < 1);


        new_node = (node*)malloc(sizeof(node));
        new_node->data = item;
        if ((pos == 1) || (START == NULL))
        {
                new_node->link = START;
                START = new_node;
        }
        else
        {
                temp =
                START; i = 2;
```

```
            while ((i <pos)
            && (temp->link !
            = NULL))
            {
                  temp = temp->link;
                  ++i;
            }
            new_node->link = temp->link;
            temp->link = new_node;
      }
      return START;
}
/********** Function Definition ends **********/

/********** Deletion of node in linked list **********/
/********** Function Definition begins **********/
node *SLdeletion(node *START)
{
      node *temp, *prev;
      int item;


      printf("\nEnter data to be deleted : ");
      scanf("%d",&item);
      if (START == NULL)
            printf("\nCan't delete - list empty\n");
      else
      {
            prev = NULL;
            temp = START;
            while ((temp != NULL) && (temp->data != item))
            {
                  prev = temp;
                  temp = temp->link;
            }
            if (temp == NULL)
                  printf("Element not found\n");
            else
```

```
{
        if (prev == NULL)
```

```
                    START = START->link;
                else
                    prev->link = temp->link;
            printf("\n***** Linked list *****\n");
            }
        }
        return START;
}
/********** Function Definition ends **********/

/********** Displaying nodes of linked list **********/
/********** Function Definition begins **********/
void SLdisplay(node *START)
{
    printf("\nSTART->");
        while (START != NULL)
        {
            printf("%d->",START->data);
            START = START->link;
        }
        printf("->NULL\n\n");
}
/********** Function Definition ends **********/
```

## **Output:**

```
Program for singly linked
list Menu:
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice : 1

Menu:
1.Add node
```

2. D
i
s
p
l
a
y
:
3. Quit:
Enter choice:1

Enter data:11
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:1

Enter data:22
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:1

Enter data:33
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:2

***** Linked list *****

START->11->22->33->->NULL
Menu:
1.Add node
2. Display:
3. Quit:
Enter choice:3

Program for singly linked
list Menu:
1.Create
2.Insert
3.Delete
4.Display
5.Exit

Enter choice : 3

Enter data to be deleted : 22

***** Linked list *****

Program for singly linked
list Menu:

1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice :4

***** Linked list *****

START->11->33->->NULL

Program for singly linked
list Menu:
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice :5

# EXPERIMENT No.8

## Aim:- Write a program to perform creation , insertion , deletion of doubly linked list.

## Theory:

### Procedure Dcreate (START, END)

This procedure creates an empty list. The pointer variable START and END are assigned a sentinel value to indicate the list is empty in the beginning.

Step1 Initialization.

Set START ← NULL
Set END ← NULL

Step 2 R return at the point of call.

return

## Source code:

```
#include <stdio.h>
#include <malloc.h>
#include<process.h>

typedef struct DList_tag
{
    int data;
    struct DList_tag *rlink, *llink;
}node;

/**************Function Declaration Begin**********/
node *DLcreation(node **);
void DLinsertion(node **, node **, int, int);
void DLdeletion(node **, node**);
void DLdisplay(node *, node *);
/**************Function Declaration End**********/

void main()
{
    node *left=NULL,*right;
    int item,pos,ch;
    printf("\n\t\tProgram for doubly linked list\n");
```

```
    do
    {
        printf("\n\t\tMenu");
        printf("\n\t\t1.Create");
        printf("\n\t\t2.Insert");
        printf("\n\t\t3.Delete");
        printf("\n\t\t4.Display");
        printf("\n\t\t5.Exit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                left = DLcreation(&right);
                break;
            case 2:
                printf("\nEnter data :");
                scanf("%d",&item); do

                {
                    printf("\nEnter position of insertion :");
                    scanf("%d",&pos);
                }while(pos < 1);
                DLinsertion(&left,&right,item,pos);
                break;
            case 3:
                DLdeletion(&left,&right);
                break;
            case 4:
                printf("\n\t**** Doubly linked list *****\n");
                DLdisplay(left,right);
                break;
            case 5:
                exit(0);
            default:
                printf("\n Wrong Choice");
        }
    }while(ch!=5);
    printf("\n");
}

/********** Creating of double linked list MENU **********/
```

```
/********** Function Definition begins **********/
node *DLcreation( node **right )
{
    node *left, *new_node;
    int item,ch;
    *right = left = NULL;
    do
    {
        printf("\n\t\tMenu");
        printf("\n\t\t1.Add node");
        printf("\n\t\t2.Quit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                printf("\n Enter data:");
                scanf("%d",&item);
                new_node = (node *)malloc(sizeof(node));
                new_node->data = item;
                new_node->rlink = NULL;
                if(left == NULL)
                    {
                        new_node->llink =
                        NULL; left = new_node;
                    }
                else
                    {
                        new_node->llink = (*right);
                        (*right)->rlink = new_node;
                    }
                (*right) = new_node;
                if(left != NULL)
                    (*right) =
                new_node; break;
            case 2:
                break;
            default:
                printf("\n Wrong Choice");
        }
    }while(ch!=2);
    return left;
}
```

```
/********** Function Definition ends **********/

/********** Insertion of node in double linked list **********/
/********** Function Definition begins **********/
void DLinsertion(node **start, node **right,int item, int pos)
{
    node *new_node, *temp;
    int i;
    if((pos == 1) ||((*start) == NULL))
    {
        new_node = (node *)malloc(sizeof(node));
        new_node->data = item;
        new_node->rlink  =  *start;
        new_node->llink  =  NULL;
        if((*start) != NULL)
            (*start)->llink = new_node;
        else
            (*right) = new_node;
        *start = new_node;
    }
    else
    {
        temp = *start;
        i = 2;
        while((i < pos) && (temp->rlink != NULL))
        {
            temp = temp->rlink;
            ++i;
        }
        new_node = (node *)malloc(sizeof( node));
        new_node->data = item;
        new_node->rlink = temp->rlink;
        if(temp->rlink != NULL)
            temp->rlink->llink = new_node;
            new_node->llink = temp; temp-
            >rlink = new_node;
    }
    if(new_node->rlink == NULL)
        *right = new_node;
}
/********** Function Definition ends **********/

/********** Deletion of node in linked list **********/
/********** Function Definition begins **********/
```

```
void DLdeletion( node **start, node **right)
{
    node *temp,
    *prec; int item;
    printf("\nElement to be deleted :");
    scanf("%d",&item);
    if(*start != NULL)
    {
        if((*start)->data == item)
        {
            if((*start)->rlink == NULL)
                *start = *right = NULL;
            else
            {
                *start = (*start)->rlink;
                (*start)->llink = NULL;
            }
        }
        else
        {
            temp = *start;
            prec = NULL;
            while((temp->rlink != NULL) && (temp->data != item))
            {
                prec = temp;
                temp = temp->rlink;
            }
            if(temp->data != item)
                printf("\n Data in the list not found\n");
            else
            {
                if(temp == *right)
                    *right = prec;
                else
                    temp->rlink->llink = temp-
                >llink; prec->rlink = temp->rlink;
            }
        }
    }
    else
        printf("\n!!! Empty list !!!\n");
    return;
}
/********** Function Definition ends **********/
```

```
/********** Displaying nodes of double linked list **********/
/********** Function Definition begins **********/
void DLdisplay(node *start, node *right)
{
    printf("\n***** Traverse in Forward direction *****\n left->");
    while(start != NULL)
    {
        printf("%d-> ",start->data);
        start = start->rlink;
    }
    printf("right");
    printf("\n***** Traverse in Backward direction *****\n right->");
    while(right != NULL)
    {
        printf("%d-> ",right->data);
        right = right->llink;
    }
    printf("left");
}
/********** Function Definition ends **********/
```

## Output:

```
Program for doubly linked
list Menu
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice : 1
Menu
1.Add node
2.Quit
Enter choice : 1

Enter data:11
Menu
1.Add node
2.Quit
Enter choice : 1
```

Enter data:22

Menu
1.Add node
2.Quit
Enter choice : 1

Enter data:33

Menu
1.Add node
2.Quit
Enter choice : 1

Enter data:44

Menu
1.Add node
2.Quit
Enter choice : 1

Enter data:55

Menu
1.Add node
2.Quit
Enter choice : 2

Menu
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice : 2

Menu
1.Create
2.Insert
3.Delete
4.Display
5.Exit
Enter choice : 2

Enter data :99

Enter position of insertion :

3 Menu
1.Create
2.Insert
3.Delete
4.Display

```
      5.Exit
      Enter choice :4

*       Doubly linked list *****

*         Traverse in Forward direction *****
 left->11-> 22-> 99-> 33-> 44-> 55-> right
*         Traverse in Backward direction *****
 right->55-> 44-> 33-> 99-> 22-> 11-> left

      Menu
      1.Create
      2.Insert
      3.Delete
      4.Display
      5.Exit
      Enter choice : 3

Element to be deleted :33
      Menu
      1.Create
      2.Insert
      3.Delete
      4.Display
      5.Exit
      Enter choice : 4

   *       Doubly linked list *****

*         Traverse in Forward direction *****
 left->11-> 22-> 99-> 44-> 55-> right
*         Traverse in Backward direction *****
 right->55->   44->   99->   22->   11->
      left Menu
      1.Create
      2.Insert
      3.Delete
      4.Display
      5.Exit
   Enter choice :5
```

# EXPERIMENT No. 9

## Aim:-Write a program to implement polynomial in link list and perform

**(a) Arithmetic.**

**(b) Evaluation.**

## Theory:-

Linked lists are widely used to represent and manipulate polynomials. Polynomials are the expressions containing number of terms with non zero coefficients and exponents.

Consider the following polynomial.

$p(X)=a^{n}x^{e}{}_{n}+a_{n-1}x^{e}{}_{n-1}+\ldots+a_{1}x_{1}^{e}+a$

where $a_i$ a are nonzero coefficients.

$e_i$ a are exponents such that

In the linked representation of polynomials, each term is considered as a node. And such a node contains three fields.

1.Coefficent field 2.Exponent field 3.Link field.

The coefficient field holds the value of the coefficient of a term and the exponent field contains the exponent value of that term and the exponent field contains the exponent value of that term. And the link field contains the addresses of the next term in the polynomial.

The logical representation of the above node is given below:

```
struct polynode
{
int coeff;
int expo;
struct polynode *ptr;
};
typedef struct polynode PNODE;
```

Two polynomials can be added. And the steps involved in adding two polynomials are given below:

1.Read the number of terms in the first polynomial P.

2.Read the coefficients and exponents of the first polynomial.

3.Read the number of terms in the second polynomial Q. 4.Read the coefficients and exponents in the second polynomial.

5.Set the temporary pointers p and q to traverse the two polynomials respectively.

6.Compare the exponents of two polynomials starting from the first nodes.

(a) If both exponents are equal then add the coefficients and store it in the resultant linked list.

(b) If the exponent of the current term in the first polynomial P is less than the exponent of the current term of the second polynomial is added to the resultant linked list. And move the pointer q to point to the next node in the second polynomial Q.

(c) If the exponent of the current term in the first polynomial P is greater than the exponent of the current term in the second polynomial Q then the current term of the first polynomial is added to the resultant linked list. And move the pointer p to the next node.

(d) Append the remaining nodes of either of the polynomials to the resultant linked list.

## Source Code:-

```
#include<stdio.h>
#include<conio.h>
#include<limits.h>
int select();
struct rec
{
float coef;
int exp;
struct rec
*next; };
struct rec *rear;
struct rec *create(struct rec *list);
void *add(struct rec *first,struct rec *second);
struct rec *insert(double coef,int exp,struct rec
*rear); void *display(struct rec *list);
int nodes;
void main()
{
struct rec *first=NULL,*second=NULL;
int choice;
do
{
choice=select();
switch(choice)
{
case 1: first=create(first);continue;
case 2: second=create(second);continue;
case 3: add(first,second);continue;
case 4: puts("END");exit(0);
}
}while(choice!=4);
}
int select()
{
int selection;
```

```
do
{
puts("Enter 1: create the first list");
puts("Enter 2: create the second
list"); puts("Enter 3: add the two
list"); puts("Enter 4: END");
puts("Entr your choice");
scanf("%d",&selection); }
while((selection<1)||(selection>4));
return (selection);
}


struct rec *create(struct rec *x)
{
float coef;
int exp;
int endexp=INT_MAX;
struct rec *element;
puts("Enter coefs &exp:exp in descending order:""to quit enter 0 for
exp"); x=(struct rec *)malloc(sizeof(struct rec));
x->next=NULL;
rear=x;
for(;;)
{
puts("Enter coefficient");
element=(struct rec*)malloc(sizeof(struct rec));
scanf("%f",&coef);
element->coef=coef;
if(element->coef==0.0)break;
puts("Enter exponent");
scanf("%d",&exp); element-
>exp=exp;
if((element->exp<=0)||(element->exp>=endexp))
{
puts("Invalid
exponent"); break;
}
element->next=NULL;
rear->next=element;
rear=element;
}
x=x->next;
return(x);
}
void *add(struct rec *first,struct rec *second)
```

```
{
float total;
struct rec *end,*rear,*result;
result=(struct rec *)malloc(sizeof(struct rec));
rear=end; while((first!=NULL)&&(second!
=NULL))
{
if(first->exp==second->exp)
{
if((total=first->exp+second->exp)!=0.0)
rear=insert(total,first->exp,rear);
first=first->next; second=second->next;

}
Else


if(first->exp>second->exp)
{
rear=insert(first->coef,first-
>exp,rear); first=first->next;
}else
{
rear=insert(second->coef,second-
>exp,rear); second=second->next;
}
}
for(;first;first=first->next) rear=insert(first-
>coef,first->exp,rear);
for(;second;second=second->next)
rear=insert(second->coef,second-
>exp,rear); rear->next=NULL;
display(end->next);
free(end);
}
void *display(struct rec *head)
{
while(head!=NULL)
{
printf("%2lf",head->coef);
printf("%2d",head->exp);
head=head->next;
}
printf("\n");
}
struct rec *insert(double coef,int exp,struct rec *rear)
```

```
{
rear->next=(struct rec *)malloc(sizeof(struct
rec)); rear=rear->next;
rear->coef=coef;
rear->exp=exp;
return(rear);
}
```

## Output:

```
Enter 1 : Create the first list
Enter 2 : Create the second list
Enter 3 : Add the two list
Enter 4 : END
Enter  your  choice

1
Enter coefs & exp : exp in descending order : to quit enter 0 for
exp Enter coefficient
5
Enter exponent
4
Enter
coefficient 7
Enter exponent
9
Enter
coefficient 1
Enter exponent
3
Enter
coefficient 0
Enter 1 : Create the first list
Enter 2 : Create the second
list Enter 3 : Add the two list
Enter 4 : END
Enter  your  choice

2
Enter coefs & exp : exp in descending order : to quit enter 0 for
exp Enter coefficient
9
Enter exponent
```

3
Enter
coefficient 2
Enter exponent
2
Enter
coefficient 11
Enter exponent
1
Enter
coefficient 5
Enter exponent
0
Invalid exponent
Enter 1 : Create the first list
Enter 2 : Create the second
list Enter 3 : Add the two list
Enter 4 : END
Enter your choice
3
5.000000 47.000000 96.000000 32.000000 211.000000 1 Enter
1 : Create the first list
Enter 2 : Create the second
list Enter 3 : Add the two list
Enter 4 : END
Enter your choice
4

# EXPERIMENT No. 10(a)

## Aim:- Write programs to implement linked stack and linked queue.

## Theory:-

### Pushing:-

1. Input the data element to be pushed.
2. Create a NewNode.
3. NewNode → DATA=DATA.
4. NewNode → Next=TOP.
5. TOP=NewNode.
6. Exit.

### Popping:-

1. If(TOP is equal to NULL)
   (a) Display "The Stack is empty".
2. Else
   (a) TEMP=TOP.
   (b) Display "The popped element is TOP→DATA".
   (c) TOP=TOP→Next.
   (d) TEMP→Next=NULL.
   (e) Free the TEMP node.
3. EXIT.

## Source Code:-

```
#include <stdio.h>
#include <malloc.h>
#include<process.h>
typedef struct link_tag
{
     int data;
     struct link_tag *link;
}node;

/********** Function Declaration begins **********/
node *push(node *);
node *pop(node *);
```

```
void display(node *);
/********** Function Declaration ends **********/

void main()
{
    node *start=NULL;
    int ch;


    printf("\n\t\t Program of stack using linked list");

    do
    {
        printf("\n\t\tMenu");
        printf("\n\t\t1.Push");
        printf("\n\t\t2.Pop");
        printf("\n\t\t3.Display");
        printf("\n\t\t4.Exit");
        printf("\n\t\tEnter choice : ");
        scanf("%d",&ch); switch(ch)
        {
            case 1:
                start =
                push(start); break;
            case 2:
                start = pop(start);
                break;
            case 3:
                printf("\n\t**** Stack *****\n");
                display(start);
                break;
            case 4:
                exit(0);
            default:
                printf("\nwrong choice : ");
        }
    }
    while (ch!=4);
    printf("\n");
}

/********** Pushing an element in stack **********/
/********** Function Definition begins **********/
```

```
node *push(node *temp)
{
    node *new_node;
    int item;

    printf("Enter an data to be pushed : ");
    scanf("%d",&item);

    new_node = ( node *)malloc(sizeof( node));
    new_node->data = item;
    new_node->link = temp;
    temp = new_node;
    return(temp);
}
/********** Function Definition ends **********/


/********** Popping an element from stack **********/
/********** Function Definition begins **********/
node *pop(node *p)
{
    node *temp;


    if(p == NULL)
        printf("\n***** Empty *****\n");
    else
    {
        printf("Popped data = %d\n",p->data);
        temp = p->link;
        free(p); p
        = temp;
        if (p == NULL)
            printf("\n***** Empty *****\n");
    }
    return(p);
}
/********** Function Definition ends **********/


/********** Displaying elements of Multistack1 **********/
/********** Function Definition begins **********/
void display(node *seek)
{
```

```
    printf("\nTop"); while
    (seek != NULL)
    {
        printf("-> %d",seek->data);
        seek = seek->link;
    }
    printf("->NULL\n");
    return;
}
/********** Function Definition ends **********/
```

## Output:

Program of stack using linked list

```
    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
  Enter choice : 1

Enter an data to be pushed :
    11 Menu
  1.Push
  2.Pop
  3.Display
  4.Exit
  Enter choice : 1

Enter an data to be pushed :
    22 Menu
  1.Push
  2.Pop
  3.Display
  4.Exit
  Enter choice : 1

Enter an data to be pushed :
  33 Menu
  1.Push
    2.Pop
    3.Display
    4.Exit
```

```
    Enter choice : 3
**** Stack *****

Top-> 33-> 22-> 11->NULL

    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice : 2
Popped data = 33

    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice : 2
Popped data = 22

    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice : 3
**** Stack *****

Top-> 11->NULL

    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
    Enter choice :2
Popped data = 11
***** Empty *****

    Menu
    1.Push
    2.Pop
    3.Display
    4.Exit
Enter choice : 4
```

element from the queue.

Step 1 Initialization, loop.

for u  V1, V2, .....  R For each u in V
{
Set color [u]  white.
Set dist [u]  ∞.
Set pre [u] NULL.
}

Step 2 Intializing source S, placing ‘S’ in the queue.

Set color [S]  gray.
Set dist [S]  0.
Set Q  {S} R putting S in the queue.

Step 3 Loop, while no more adjacent vertices

while (Q NULL)
{
Set u  Dequeue (Q) R u is the next to visit.
for V  Vk .... Vn  R for each V in adj [u]
{
if (color [V] = white) then { R if neighbour unreached
Set color [V] gray R mark it reached.

Set dist [V] □ dist [u] + 1 R set its distance.

Set pre [V] □ u R set its predecessor.

call to Enqueue (Q, V) R put in the queue.

}

Set color [u] □ black R u is visited.

}

Step 4 Return at the point of call.

Return.

## Source Code:-

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
#define FALSE 0
#define TRUE 1

typedef int adj_mat[SIZE][SIZE];
int front=1,rear=1;
int q[SIZE];

typedef struct graph_t{
                        int nodes; int
                        *visited;
                        adj_mat mat;
                    }graph;

/***************Function Declaration Begin**********/
void BFS(graph *);
void add_queue(int[],int);
int delete_queue();
/***************Function Declaration End**********/
void main()
{
    graph G;
    clrscr();
    printf("\n\t\t Program shows Breath First Search in a graph ");
    printf("\n\t\t Enter number of nodes in the graph : ");
    scanf("%d",&G.nodes);
    BFS(&G);
    getch();
}

/********** breadth first searching **********/
/********** Function Definition begins **********/
void BFS( graph *G )
{
    int k,i,j; for(k=1;k<=G-
    >nodes;k++)
        G->visited[k] = FALSE;
    for(i=1;i<=G->nodes;i++)
    {
        for(j=1;j<=G->nodes;j++)
```

```
        {
            printf("\n Enter data of vertex %d for(%d,%d) : ",i,i,j);
            printf("\n Enter 1 for adjacent vertex and 0 otehrwise ");
            scanf("%d",&G->mat[i][j]);
        }
    }
    for(k=1;k<=G->nodes;k++)
    {
        if ( !G->visited[k] )
        {
                add_queue(q,k);
                do
                {
                    k= delete_queue(q); G-
                    >visited[k] = TRUE;
                    for(j=1;j<=G->nodes;j++)
                    {
                        if(G->mat[k][j] == 0)
                            continue;
                            if (!G->visited[j])
                            {
                                G->visited[j] = TRUE;
                                add_queue(q, j);
                            }
                    }
                }while(front!=rear);
        }
    }
        printf("\n Adjacency matrix of a graph is :\n");
    for(i=1;i<=G->nodes;i++)
    {
        for(k=1;k<=G->nodes;k++)
        {
            printf("%d\t",G->mat[i][k]);
        }
        printf("\n");
    }
    i=0;
    printf("\n Traversal of a given graph is \n");
    while(i<G->nodes)
    {
        printf("%d\t",q[++i]);
    }
}
/********** Function Definition ends **********/
```

```
/********** inserting element in queue **********/
/********** Function Definition begins **********/
void enqueue(int q[], int k)
{
      q[rear] = k;
      rear++;
}
/********** Function Definition ends **********/

/********** deleting element from queue **********/
/********** Function Definition begins **********/
int dequeue(int q[])
{
      int data;
      data = q[front];
      front++;
      if(front==SIZE)
      {
            front=1;
            rear=1;
      }
      return(data);
}
/********** Function Definition ends **********/
```

## Output:

```
Program shows the traversal of graph using breadth first search
Enter number of nodes in the graph : 3

Enter data of vertex 1 for (1,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (1,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (1,3) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (2,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0
```

Enter data of vertex 1 for (2,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (2,3) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,4) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Adjacency matrix of the graph
is 0 1 1 0 0 0 0 0 0

Traversal of a given graph
is 1 2 3

# EXPERIMENT No. 16(b)

## Aim:- Write a program for traversal of graph (B.F.S., D.F.S.).

## Theory:-

**Procedure DFSvisit (u):**

The above Procedure subalgorithm processes the given vertex. It makes a recursive call to itself. The arrays used in this procedure have been previously described.

Step 1 Start search at u, mark u visited. Set color
[u] gray.
Set time time + 1. Set dis
[u] time.
Loop,
for V Vk .... Vn R for each V in Adj [u]
{
if (color [V] = White) then R if neighbour marked unreached Set pre
[V] u R set predecessor pointer.
Call to DFS visit (V) R processed V.
}
End Loop
Step 2 U, is visited
Set color [u] black. Set
time time + 1. Set pre [u]
time.
Step 3 Return at the point of call
Return.

**Procedur DFS (G):**

The above Procedure computes the depth-first-search of the given 'G' graph 'G'. It takes the advantage of Procedure DFS visit ( ). All the auxillary arrays used in this procedure have been previously described.

Step 1 Loop, initialization.
for u V1, V2 .... Vn R for each u in V
{
Set color [u] white. Set
pre [u] NULL.
}
Step 2 Setting time Set
time 0.
Step 3 Loop, finding unreached vertex and start new search

for u  V1, V2 .... V R for each u in V
{
if (color = white) R found unreached vertex
Call to DFS visit (u) R start a new search.
}

Step 4 Return at point of call
Return.

## Source Code:-

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
#define FALSE 0
#define TRUE 1
typedef int adj_mat[SIZE][SIZE];
typedef struct graph_t{
        int nodes[SIZE];
        int n;
        int *visited;
        adj_mat mat;
        }graph;

/**************Function Declaration Begin**********/
void DFS(graph *);
void visit(graph *,int);
/**************Function Declaration End**********/

static int find=0;
void main()
{
    graph G;
    //clrscr();
      printf("\n\t\t Program shows the traversal of graph using Depth First Search
");
    printf("\n\t\t Enter number of nodes in the graph : ");
    scanf("%d",&G.n);
    DFS(&G);
getch();
}
/********** depth first searching **********/
/********** Function Definition begins **********/
void DFS( graph *G )
```

```
{
    int k,i,j;
    for(k=1;k<=G->n;k++)
        G->visited[k] = FALSE;
    for(i=1;i<=G->n;i++)
    {
        for(j=1;j<=G->n;j++)
        {
            printf("\n Enter data of vertex %d for(%d,%d) :\n",i,i,j);
            printf("\n Enter 1 for adjacent vertex and 0 for otherwise :
            "); scanf("%d",&G->mat[i][j]);
        }
    }
    for(k=1;k<=G->n;k++)
    {
            if ( !G->visited[k] )
                visit(G, k);
    }
    printf("\n Adjacency matrix of the grpah is \n");
    for(i=1;i<=G->n;i++)
    {
        for(k=1;k<=G->n;k++)
        {
            printf("%d\t",G->mat[i][k]);
        }
        printf("\n");
    }
    i=0;
    printf("\n Traversal of a given graph is \n");
    while(i<G->n)
    {
        printf("%d\t",G->nodes[++i]);
    }
}
/********** Function Definition ends **********/
/********** visiting graph **********/
/********** Function Definition begins **********/
void visit( graph *G, int k )
{
  int j;
  G->visited[k] = TRUE;
  G->nodes[++find] = k;
  for(j=1;j<=G->n;j++)
  {
```

```
            if(G->mat[k][j] == 1)
            {
                  if (!G->visited[j])
                        visit( G, j );
            }
  }
}
/********** Function Definition ends **********/
```

## Output:

Program shows the traversal of graph using depth first search Enter number of nodes in the graph : 3

Enter data of vertex 1 for (1,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (1,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (1,3) :
Enter 1 for adjacent vertex and 0 for otherwise : 1

Enter data of vertex 1 for (2,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (2,2) :
Enter 1 for adjacent vertex and 0 for otherwise:0

Enter data of vertex 1 for (2,3):
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,1) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,2) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Enter data of vertex 1 for (3,4) :
Enter 1 for adjacent vertex and 0 for otherwise : 0

Adjacency matrix of the graph
is 0 1 1 0 0 0 0 0 0

Traversal of a given graph
is 1 2 3