DAYANANDA SAGAR COLLEGE OF ENGINEERING
(An Autonomous Institution affiliated by VTU, Belagavi)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING (DATA SCIENCE)

DATA STRUCTURES LAB MANUAL

Prepared By
Prof. Ankita Mandore.
Assistant Professor,
CSE (Data Science)

DATA STRUCTURES LAB			
Course code:	22CDL35	Credits:	1
L: T:P:	0:0:2	CIE Marks:	50
Exam Hours:	03	SEE Marks:	50
Total Hours:	15		

Course Objectives	
1	To understand and implement the concept of recursion in real time approach.
2	To implement a program for stack, queue and lists to solve the problems.
3	Conversions of reverse polish notation.
4	To write and execute programs to solve problems using data structures such as hash tables and search trees.
5	To learn the implementation of various sorting and searching algorithms.

Course Outcomes: At the end of the course, student will be able to	
CO1	Apply the knowledge of data structure to solve the practical problems using C programming.
CO2	Analyze the various methods to implement memory allocation for different Data Structures.
CO3	Design the program by using data structure according to the problem statement.
CO4	Develop a menu driven program to do different operations on the same data structures.

Expt No	Content of the Experiments	Hours
1	To solve the tower of Hanoi problem using recursion.	2
2	Design and develop a program to perform following operations on Stack. push : Adds an element to the top of the stack. pop : Removes the topmost element from the stack. isEmpty : Checks whether the stack is empty. isFull : Checks whether the stack is full. top : Displays the topmost element of the stack.	2
3	Design and develop a program to perform following operations on Queue. enqueue(): this method adds an element to the end/rear of the queue dequeue(): this method removes an element from the front of the queue top(): returns the first element in the queue initialize(): creates an empty queue	2
4	Design and develop a program to perform insert, delete and display Operations on Queue using runtime memory allocation functions malloc(), calloc(), alloc().	
5	Design, Develop and Implement a menu driven Program in C for the following operations on Priority QUEUE Insertion in a Priority Queue Deletion in a Priority Queue Peek in a Priority Queue	2
6	Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE. (Array Implementation of Queue with maximum size MAX) Insert an Element onto Circular QUEUE Delete an Element from Circular QUEUE Demonstrate Overflow and Underflow situations on Circular QUEUE Display the status of Circular QUEUE Exit Support the program with appropriate functions for each of the above operations	2
7	Implement a program to convert the Infix to Postfix expression(Polish notation).	2

8	Design, develop and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The operators are +(add), -(subtract), *(multiply), /(divide)	2
9	Design, develop and execute a program in C to implement a singly linked list where each node consists of integers. The program should support the following functions. Create a singly linked list Insert a new node Delete a node if it is found, otherwise display appropriate message Display the nodes of singly linked list	2
10	Design, develop and execute a program in C to implement a doubly linked list where each node consists of integers. The program should support the following functions. Create a doubly linked list Insert a new node at start, middle and end Delete a node at start,middle and end Display the nodes of doubly linked list	2
11	Design, develop and execute a program in C to implement a circular singly linked list where each node consists of integers. The program should support the following functions. Create a doubly linked list Insert a new node Delete a node Display the nodes of circular singly linked list	2
12	Implementation of various operations on Binary Tree like – creating a tree, displaying a tree, copying tree, mirroring a tree, counting the number of nodes in the tree, counting only leaf nodes in the tree	2

Experiment 1. To solve the tower of Hanoi problem using recursion.

Algorithm:

To write an algorithm for Tower of Hanoi, first we need to learn how to solve this problem with lesser amount of disks, say → 1 or 2. We mark three towers with name, source, destination and aux (only to help moving the disks). If we have only one disk, then it can easily be moved from source to destination pole.

If we have 2 disks −

First, we move the smaller (top) disk to aux pole.

Then, we move the larger (bottom) disk to destination pole.

And finally, we move the smaller disk from aux to destination pole.

```
START
Procedure Hanoi(disk, source, dest, aux)

  IF disk == 1, THEN
    move disk from source to dest
  ELSE
    Hanoi(disk - 1, source, aux, dest)     // Step 1
    move disk from source to dest          // Step 2
    Hanoi(disk - 1, aux, dest, source)     // Step 3
  END IF

END Procedure
STOP
```

Program:

```
#include <stdio.h>

void towers(int, char, char, char);
```

```
int main()
{
   int num;

   printf("Enter the number of disks : ");
   scanf("%d", &num);
   printf("The sequence of moves involved in the Tower of Hanoi are :\n");
   towers(num, 'A', 'C', 'B');
   return 0;
}
void towers(int num, char beg, char aux, char end)
{
   // Base Condition if no of disks are
   if (num >= 1)

 {
   // Recursively calling function twice
   towers(num - 1, beg, end, aux);
   printf("\n Move disk %d from peg %c to peg %c", num, beg, end);
   towers(num - 1, aux, beg, end);
}
}
```

Output:

```
Enter the number of disks : 3
The sequence of moves involved in the Tower of Hanoi are :

 Move disk 1 from peg A to peg B
 Move disk 2 from peg A to peg C
 Move disk 1 from peg B to peg C
```

Move disk 3 from peg A to peg B

Move disk 1 from peg C to peg A

Move disk 2 from peg C to peg B

Move disk 1 from peg A to peg B

Experiment 2: Design and develop a program to perform following operations on Stack.

push: Adds an element to the top of the stack.

pop: Removes the topmost element from the stack.

isEmpty: Checks whether the stack is empty.

isFull: Checks whether the stack is full.

top: Displays the topmost element of the stack.

Algorithm:

Push Operation

1) [Overflow?]

IF Top=Max-1 then

Print: Overflow and Return

2) [Increase Top by 1]

SET Top=Top+1

3) [Assign Element at Top position]

SET Stack [Top] =Element

4) Exit

Pop Operation

1) [Underflow]

IF Top=-1 then

Print: Underflow and Return

2) [Assign Top element to Variable]

SET Element= Stack [Top]

3) [Decrease Top by 1]

SET Top=Top-1

4) Exit

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10

int stack_arr[MAX];
int top = -1;

void push(int item);
int pop();
int peek();
int isEmpty();
int isFull();
void display();

int main()
{
	int choice,item;
	while(1)
	{
		printf("\n1.Push\n");
		printf("2.Pop\n");
		printf("3.Display the top element\n");
		printf("4.Display all stack elements\n");
		printf("5.Quit\n");
		printf("\nEnter your choice : ");
		scanf("%d",&choice);
		switch(choice)
		{
```

```
            case 1 :
                printf("\nEnter the item to be pushed : ");
                scanf("%d",&item);
                push(item);
                break;
            case 2:
                item = pop();
                printf("\nPopped item is : %d\n",item );
                break;
            case 3:
                printf("\nItem at the top is : %d\n", peek() );
                break;
            case 4:
                display();
                break;
            case 5:
                exit(1);
            default:
                printf("\nWrong choice\n");
            }/*End of switch*/
        }/*End of while*/

        return 0;

}/*End of main()*/

void push(int item)
{
        if( isFull() )
        {
```

```
        printf("\nStack Overflow\n");
        return;
    }
    top = top+1;
    stack_arr[top] = item;
}/*End of push()*/

int pop()
{
    int item;
    if( isEmpty() )
    {
        printf("\nStack Underflow\n");
        exit(1);
    }
    item = stack_arr[top];
    top = top-1;
    return item;
}/*End of pop()*/

int peek()
{
    if( isEmpty() )
    {
        printf("\nStack Underflow\n");
        exit(1);
    }
    return stack_arr[top];
}/*End of peek()*/
```

```
int isEmpty()
{
	if( top == -1 )
		return 1;
	else
		return 0;
}/*End of isEmpty*/

int isFull()
{
	if( top == MAX-1 )
		return 1;
	else
		return 0;
}/*End of isFull*/

void display()
{
	int i;
	if( isEmpty() )
	{
		printf("\nStack is empty\n");
		return;
	}
    printf("\nStack elements :\n\n");
	for(i=top;i>=0;i--)
		printf(" %d\n", stack_arr[i] );
	printf("\n");
}/*End of display()*/
```

Output:

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 1

Enter the item to be pushed : 10

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 1

Enter the item to be pushed : 20

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 1

Enter the item to be pushed : 30

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 1

Enter the item to be pushed : 40

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 3

Item at the top is : 40

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 4

Stack elements :

40

30

20

10

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 2

Popped item is : 40

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 3

Item at the top is : 30

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 2

Popped item is : 30

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 3

Item at the top is : 20

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 4

Stack elements :

20

10

1.Push

2.Pop

3.Display the top element

4.Display all stack elements

5.Quit

Enter your choice : 5

Experiment 3:

Design and develop a program to perform following operations on Queue.

enqueue(): this method adds an element to the end/rear of the queue

dequeue(): this method removes an element from the front of the queue

top(): returns the first element in the queue

initialize(): creates an empty queue

Algorithm:

Algorithm to Insert an element in the queue

Step 1: IF REAR = MAX - 1
Write OVERFLOW
Go to step
[END OF IF]

Step 2: IF FRONT = -1 and REAR = -1
SET FRONT = REAR = 0
ELSE
SET REAR = REAR + 1
[END OF IF]

Step 3: Set QUEUE[REAR] = NUM

Step 4: EXIT

Algorithm to delete an element from the queue

Step 1: IF FRONT = -1 or FRONT > REAR
Write UNDERFLOW
ELSE
SET VAL = QUEUE[FRONT]
SET FRONT = FRONT + 1
[END OF IF]

Step 2: EXIT

Program:

```
#include<stdio.h>

#include<stdlib.h>

#define maxsize 5

void insert();

void delete();
```

```
void display();
int front = -1, rear = -1;
int queue[maxsize];
void main ()
{
  int choice;
  while(choice != 4)
  {
    printf("\nMain Menu\n");
    printf("\n1.insert an element\n2.Delete an element\n3.Display the
queue\n4.Exit\n");
    printf("\nEnter your choice ?");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
      insert();
      break;
      case 2:
      delete();
      break;
      case 3:
      display();
      break;
      case 4:
      exit(0);
      break;
      default:
      printf("\nEnter valid choice??\n");
    }
```

```
    }
}
void insert()
{
    int item;
    printf("\nEnter the element\n");
    scanf("\n%d",&item);
    if(rear == maxsize-1)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    if(front == -1 && rear == -1)
    {
        front = 0;
        rear = 0;
    }
    else
    {
        rear = rear+1;
    }
    queue[rear] = item;
    printf("\nValue inserted ");

}
void delete()
{
    int item;
    if (front == -1 || front > rear)
    {
```

```
        printf("\nUNDERFLOW\n");
        return;

    }
    else
    {
        item = queue[front];
        if(front == rear)
        {
            front = -1;
            rear = -1 ;
        }
        else
        {
            front = front + 1;
        }
        printf("\nvalue deleted ");
    }
}

void display()
{
    int i;
    if(rear == -1)
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nprinting values .....\n");
        for(i=front;i<=rear;i++)
```

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

Output:

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

10

Value inserted

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

20

Value inserted

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

30

Value inserted

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter the element

40

Value inserted

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

10

20

30

40

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

value deleted

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

20

30

40

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

value deleted

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

30

40

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?4

Experiemnt 4:

Design and develop a program to perform insert, delete and display

Operations on Queue using runtime memory allocation functions malloc(), calloc(), alloc().

Algorithm to Insert:

Step 1: Allocate the space for the new node PTR

Step 2: SET PTR -> DATA = VAL

Step 3: IF FRONT = NULL
SET FRONT = REAR = PTR
SET FRONT -> NEXT = REAR -> NEXT = NULL
ELSE
SET REAR -> NEXT = PTR
SET REAR = PTR
SET REAR -> NEXT = NULL
[END OF IF]

Step 4: END

Algorithm to delete

Step 1: IF FRONT = NULL
Write " Underflow "
Go to Step 5
[END OF IF]

Step 2: SET PTR = FRONT

Step 3: SET FRONT = FRONT -> NEXT

Step 4: FREE PTR

Step 5: END

Program:

```
#include<stdio.h>

#include<stdlib.h>

struct node

{

   int data;
```

```
    struct node *next;
};
struct node *front;
struct node *rear;
void insert();
void delete();
void display();
void main ()
{
    int choice;
    while(choice != 4)
    {
        printf("\nMain Menu\n");
        printf("\n1.insert an element\n2.Delete an element\n3.Display the
queue\n4.Exit\n");
        printf("\nEnter your choice ?");
        scanf("%d",& choice);
        switch(choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
```

```
            break;
            default:
            printf("\nEnter valid choice??\n");
        }
    }
}
void insert()
{
    struct node *ptr;
    int item;

    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr -> data = item;
        if(front == NULL)
        {
            front = ptr;
            rear = ptr;
            front -> next = NULL;
            rear -> next = NULL;
        }
        else
```

```
        {
            rear -> next = ptr;
            rear = ptr;
            rear->next = NULL;
        }
    }
}
void delete ()
{
    struct node *ptr;
    if(front == NULL)
    {
        printf("\nUNDERFLOW\n");
        return;
    }
    else
    {
        ptr = front;
        front = front -> next;
        free(ptr);
    }
}
void display()
{
    struct node *ptr;
    ptr = front;
    if(front == NULL)
    {
        printf("\nEmpty queue\n");
    }
```

```
    else
    {   printf("\nprinting values .....\n");
        while(ptr != NULL)
        {
            printf("\n%d\n",ptr -> data);
            ptr = ptr -> next;
        }
    }
}
```

Output:

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

10

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

20

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

30

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?1

Enter value?

40

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

10

20

30

40

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

20

30

40

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?2

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?3

printing values .....

30

40

Main Menu

1.insert an element

2.Delete an element

3.Display the queue

4.Exit

Enter your choice ?4

Experiment 5:

Design, Develop and Implement a menu driven Program in C for the following operations on Priority QUEUE

Insertion in a Priority Queue

Deletion in a Priority Queue

Peek in a Priority Queue

Algorithm:

Program:

```
#include<stdio.h>

#include<stdlib.h>


#define MAX 5


void insert_by_priority(int);

void delete_by_priority(int);

void create();

void check(int);

void display_pqueue();


int pri_que[MAX];

int front, rear;


void main()

{

   int n, ch;


   printf("\n1 - Insert an element into queue");

   printf("\n2 - Delete an element from queue");

   printf("\n3 - Display queue elements");
```

```
printf("\n4 - Exit");

create();

do
{
   printf("\nEnter your choice : ");
   scanf("%d", &ch);

   switch (ch)
   {
   case 1:
      printf("\nEnter value to be inserted : ");
      scanf("%d",&n);
      insert_by_priority(n);
      break;
   case 2:
      printf("\nEnter value to delete : ");
      scanf("%d",&n);
      delete_by_priority(n);
      break;
   case 3:
      display_pqueue();
      break;
   case 4:
      exit(0);
   default:
      printf("\nChoice is incorrect, Enter a correct choice");
   }
}while(ch!=4);
```

```
}

/* Function to create an empty priority queue */
void create()
{
   front = rear = -1;
}

/* Function to insert value into priority queue */
void insert_by_priority(int data)
{
   if (rear >= MAX - 1)
   {
      printf("\nQueue overflow no more elements can be inserted");
      return;
   }
   if ((front == -1) && (rear == -1))
   {
      front++;
      rear++;
      pri_que[rear] = data;
      return;
   }
   else
      check(data);
   rear++;
}

/* Function to check priority and place element */
void check(int data)
```

```
{
   int i,j;

   for (i = 0; i <= rear; i++)
   {
      if (data >= pri_que[i])
      {
         for (j = rear + 1; j > i; j--)
         {
            pri_que[j] = pri_que[j - 1];
         }
         pri_que[i] = data;
         return;
      }
   }
   pri_que[i] = data;
}

/* Function to delete an element from queue */
void delete_by_priority(int data)
{
   int i;

   if ((front==-1) && (rear==-1))
   {
      printf("\nQueue is empty no elements to delete");
      return;
   }

   for (i = 0; i <= rear; i++)
```

```
    {
        if (data == pri_que[i])
        {
            for (; i < rear; i++)
            {
                pri_que[i] = pri_que[i + 1];
            }

        pri_que[i] = -99;
        rear--;

        if (rear == -1)
            front = -1;
        return;
        }
    }
    printf("\n%d not found in queue to delete", data);
}

/* Function to display queue elements */
void display_pqueue()
{
    if ((front == -1) && (rear == -1))
    {
        printf("\nQueue is empty");
        return;
    }

    for (; front <= rear; front++)
    {
```

```
        printf(" %d ", pri_que[front]);
    }

    front = 0;
}
```

Output:

```
1 - Insert an element into queue
2 - Delete an element from queue
3 - Display queue elements
4 - Exit
Enter your choice : 1

Enter value to be inserted : 10

Enter your choice : 1

Enter value to be inserted : 50

Enter your choice : 1

Enter value to be inserted : 30

Enter your choice : 3
 50  30  10
Enter your choice : 2

Enter value to delete : 30
```

Enter your choice : 2

Enter value to delete : 0

0 not found in queue to delete

Enter your choice : 3

50 10

Enter your choice : 4

Experiment 6:

Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE. (Array Implementation of Queue with maximum size MAX)

Insert an Element onto Circular QUEUE

Delete an Element from Circular QUEUE

Demonstrate Overflow and Underflow situations on Circular QUEUE

Display the status of Circular QUEUE

Exit

Support the program with appropriate functions for each of the above operations

Algorithm:

Program:

```
/* C Program to implement circular queue using arrays */


#include<stdio.h>

#include<stdlib.h>

#define MAX 5


int cqueue_arr[MAX];

int front=-1;

int rear=-1;


void display( );

void insert();

int del();

int peek();

int isEmpty();

int isFull();


int main()
```

```
{
    int choice;
    while(1)
    {
        printf("\n1.Insert\n");
        printf("2.Delete\n");
        printf("3.Peek\n");
        printf("4.Display\n");
        printf("5.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
        case 1 :

            insert();
            break;
        case 2 :
            printf("\nElement deleted is : %d\n",del());
            break;
        case 3:
            printf("\nElement at the front is  : %d\n",peek());
            break;
        case 4:
            display();
            break;
        case 5:
            exit(1);
        default:
```

```
                printf("\nWrong choice\n");
            }/*End of switch*/
        }/*End of while */

        return 0;

}/*End of main()*/

void insert()
{
    int item;

        if( isFull() )
        {
            printf("\nQueue Overflow\n");
            return;
        }
    printf("\nInput the element for insertion : ");
    scanf("%d",&item);
        if(front == -1 )
            front=0;

        if(rear==MAX-1)/*rear is at last position of queue*/
            rear=0;
        else
            rear=rear+1;
        cqueue_arr[rear]=item ;
}/*End of insert()*/

int del()
```

```
{
	int item;
	if( isEmpty() )
	{
		printf("\nQueue Underflow\n");
		exit(1);
	}
	item=cqueue_arr[front];
	if(front==rear) /* queue has only one element */
	{
		front=-1;
		rear=-1;
	}
	else if(front==MAX-1)
		front=0;
	else
		front=front+1;
	return item;
}/*End of del() */

int isEmpty()
{
	if(front==-1)
		return 1;
	else
		return 0;
}/*End of isEmpty()*/

int isFull()
{
```

```
	if((front==0 && rear==MAX-1) || (front==rear+1))
		return 1;
	else
		return 0;
}/*End of isFull()*/

int peek()
{
	if( isEmpty() )
	{
		printf("\nQueue Underflow\n");
		exit(1);
	}
	return cqueue_arr[front];
}/*End of peek()*/

void display()
{
	int i;
	if(isEmpty())
	{
		printf("\nQueue is empty\n");
		return;
	}
	printf("\nQueue elements :\n");
	i=front;
	if( front<=rear )
	{
		while(i<=rear)
			printf("%d ",cqueue_arr[i++]);
```

```
    }
    else
    {
        while(i<=MAX-1)
            printf("%d ",cqueue_arr[i++]);
        i=0;
        while(i<=rear)
            printf("%d ",cqueue_arr[i++]);
    }
    printf("\n");
}/*End of display() */
```

Output:

Experiment 7: Implement a program to convert the Infix to postfix expression (Polish notation).

Algorithm for Conversion of Infix to Postfix using Stack in C

1. Scan all the symbols one by one from left to right in the given Infix Expression.
2. If the reading symbol is an operand, then immediately append it to the Postfix Expression.
3. If the reading symbol is left parenthesis '(', then Push it onto the Stack.
4. If the reading symbol is right parenthesis ')', then Pop all the contents of the stack until the respective left parenthesis is popped and append each popped symbol to Postfix Expression.
5. If the reading symbol is an operator (+, –, *, /), then Push it onto the Stack. However, first, pop the operators which are already on the stack that have higher or equal precedence than the current operator and append them to the postfix. If an open parenthesis is there on top of the stack then push the operator into the stack.
6. If the input is over, pop all the remaining symbols from the stack and append them to the postfix.

Program:

```
#include<stdio.h>

#include<string.h>

#include <stdlib.h>

#include<ctype.h>

int priority(char);

void main()

{

  char q[30],p[30],stk[10];

  int i=0,j=0,top=-1,l,k;

  printf("Enter Infix expression");

  gets(q);

  stk[++top]='(';

  strcat(q,")");

  printf("\nlement \t Stack \t Postfix");

  while(top!=-1)

  {

    if(isalpha(q[i])||isdigit(q[i]))
```

```
    p[j++]=q[i];
    else
    if(q[i]=='(')
    stk[++top]='(';
    else
    if(q[i]=='+'||q[i]=='-'||q[i]=='*'||q[i]=='/'||q[i]=='%'||q[i]=='^')
    {
        while(priority(stk[top])>=priority(q[i]))
        p[j++]=stk[top--];
        stk[++top]=q[i];
    }
    else
    if(q[i]==')')
    {
        while(stk[top]!='(')
        p[j++]=stk[top--];
        top--;
    }
    printf("\n");
    printf("%c",q[i]);
    printf("\t");
    for(k=0;k<=top;k++)
    printf("%c",stk[k]);
    printf("\t");
    for(l=0;l<=j;l++);
    printf("%c",p[l]);
    i++;
}
p[j]='\0';
printf("\n Postfix is =%s",p);
```

```
}
int priority(char ch)
{
   switch(ch)
   {
      case '^':return 3;
      case '*':
      case '/':
      case '%':return 2;
      case '+':
      case '-':return 1;
      default:return 0;
   }
}
```

Output:

```
Enter Infix expressionA-B/(C*D^E)
Element   Stack   Postfix
A      (
-      (-
B      (-
/      (-/
(      (-/(
C      (-/(
*      (-/(*
D      (-/(*
^      (-/(*^
E      (-/(*^
)      (-/
)
 Postfix is =ABCDE^*/-
```

Experiment 8: Design, develop and execute a program in C to evaluate a valid postfix expression using stack. Assume that the postfix expression is read as a single line consisting of non-negative single digit operands and binary arithmetic operators. The operators are +(add), -(subtract), *(multiply), /(divide).

Algorithm:

1. Create a stack to store operands (or values).
2. Scan the given expression from left to right and do the following for every scanned element.
3. If the element is a number, push it into the stack.
4. If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
5. When the expression is ended, the number in the stack is the final answer.

Program:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_SIZE 100

// Stack implementation

int stack[MAX_SIZE];

int top = -1;

void push(int item) {

  if (top >= MAX_SIZE - 1) {

printf("Stack Overflow\n");

    return;

  }

  top++;

  stack[top] = item;

}

int pop() {

  if (top < 0) {

printf("Stack Underflow\n");

    return -1;
```

```
    }
    int item = stack[top];
    top--;
    return item;
}
int is_operator(char symbol) {
    if (symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/') {
        return 1;
    }
    return 0;
}
int evaluate(char* expression) {
    int i = 0;
    char symbol = expression[i];
    int operand1, operand2, result;

    while (symbol != '\0') {
        if (symbol >= '0' && symbol <= '9') {
            int num = symbol - '0';
            push(num);
        }
        else if (is_operator(symbol)) {
            operand2 = pop();
            operand1 = pop();
            switch(symbol) {
                case '+': result = operand1 + operand2; break;
                case '-': result = operand1 - operand2; break;
                case '*': result = operand1 * operand2; break;
                case '/': result = operand1 / operand2; break;
            }
```

```
            push(result);
        }
i++;
        symbol = expression[i];
    }
    result = pop();
    return result;
}

int main() {
    char expression[30];
    int result;
    printf("Enter Postfix Expression");
    gets(expression);
    result = evaluate(expression);
printf("Result= %d\n", result);
return 0;
}
```

Output:

```
Enter Postfix Expression534*-82/+7+
Result= 4
```

Experiment 9: Design, develop and execute a program in C to implement a singly linked list where each node consists of integers. The program should support the following functions.

Create a singly linked list

Insert a new node

Delete a node if it is found, otherwise display appropriate message

Display the nodes of singly linked list

Algorithm:

Insertion in singly linked list at beginning

Step 1: IF PTR = NULL

Write OVERFLOW
Go to Step 7
[END OF IF]

Step 2: SET NEW_NODE = PTR

Step 3: SET PTR = PTR → NEXT

Step 4: SET NEW_NODE → DATA = VAL

Step 5: SET NEW_NODE → NEXT = HEAD

Step 6: SET HEAD = NEW_NODE

Step 7: EXIT

Insertion in singly linked list at the end

Step 1: IF PTR = NULL Write OVERFLOW
Go to Step 1
[END OF IF]

Step 2: SET NEW_NODE = PTR

Step 3: SET PTR = PTR - > NEXT

Step 4: SET NEW_NODE - > DATA = VAL

Step 5: SET NEW_NODE - > NEXT = NULL

Step 6: SET PTR = HEAD

Step 7: Repeat Step 8 while PTR - > NEXT != NULL

Step 8: SET PTR = PTR - > NEXT
[END OF LOOP]

Step 9: SET PTR - > NEXT = NEW_NODE

Step 10: EXIT

Insertion in singly linked list after specified Node

STEP 1: IF PTR = NULL

WRITE OVERFLOW
GOTO STEP 12
END OF IF

STEP 2: SET NEW_NODE = PTR

STEP 3: NEW_NODE → DATA = VAL

STEP 4: SET TEMP = HEAD

STEP 5: SET I = 0

STEP 6: REPEAT STEP 5 AND 6 UNTIL I<loc< li=""></loc<>

STEP 7: TEMP = TEMP → NEXT

STEP 8: IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"
GOTO STEP 12
END OF IF
END OF LOOP

STEP 9: PTR → NEXT = TEMP → NEXT

STEP 10: TEMP → NEXT = PTR

STEP 11: SET PTR = NEW_NODE

STEP 12: EXIT

Deletion in singly linked list at beginning

Step 1: IF HEAD = NULL

Write UNDERFLOW
Go to Step 5
[END OF IF]

Step 2: SET PTR = HEAD

Step 3: SET HEAD = HEAD -> NEXT

Step 4: FREE PTR

Step 5: EXIT

Deletion in singly linked list at the end

Step 1: IF HEAD = NULL

Write UNDERFLOW
  Go to Step 8
 [END OF IF]

Step 2: SET PTR = HEAD

Step 3: Repeat Steps 4 and 5 while PTR -> NEXT!= NULL

Step 4: SET PREPTR = PTR

Step 5: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 6: SET PREPTR -> NEXT = NULL

Step 7: FREE PTR

Step 8: EXIT

Deletion in singly linked list after the specified node :

STEP 1: IF HEAD = NULL

WRITE UNDERFLOW
   GOTO STEP 10
  END OF IF

STEP 2: SET TEMP = HEAD

STEP 3: SET I = 0

STEP 4: REPEAT STEP 5 TO 8 UNTIL I<loc< li=""></loc<>

STEP 5: TEMP1 = TEMP

STEP 6: TEMP = TEMP → NEXT

STEP 7: IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"
   GOTO STEP 12
   END OF IF

STEP 8: I = I+1

END OF LOOP

STEP 9: TEMP1 → NEXT = TEMP → NEXT

STEP 10: FREE TEMP

STEP 11: EXIT

Program:

```
// C program for the all operations in
// the Singly Linked List

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

// Linked List Node
struct node {
	int info;
	struct node* link;
};
struct node* start = NULL;

// Function to create list with n nodes initially
void createList()
{
	if (start == NULL) {
		int n;
		printf("\nEnter the number of nodes: ");
		scanf("%d", &n);
		if (n != 0) {
			int data;
			struct node* newnode;
			struct node* temp;
			newnode = malloc(sizeof(struct node));
			start = newnode;
			temp = start;
```

```
                printf("\nEnter number to"
                        " be inserted : ");
                scanf("%d", &data);
                start->info = data;

                for (int i = 2; i <= n; i++) {
                        newnode = malloc(sizeof(struct node));
                        temp->link = newnode;
                        printf("\nEnter number to"
                                " be inserted : ");
                        scanf("%d", &data);
                        newnode->info = data;
                        temp = temp->link;
                }
        }
        printf("\nThe list is created\n");
    }
    else
        printf("\nThe list is already created\n");
}

// Function to traverse the linked list
void traverse()
{
    struct node* temp;

    // List is empty
    if (start == NULL)
        printf("\nList is empty\n");
```

```
        // Else print the LL
        else {
                temp = start;
                while (temp != NULL) {
                        printf("Data = %d\n", temp->info);
                        temp = temp->link;
                }
        }
}

// Function to insert at the front
// of the linked list
void insertAtFront()
{
        int data;
        struct node* temp;
        temp = malloc(sizeof(struct node));
        printf("\nEnter number to"
                " be inserted : ");
        scanf("%d", &data);
        temp->info = data;

        // Pointer of temp will be
        // assigned to start
        temp->link = start;
        start = temp;
}

// Function to insert at the end of
// the linked list
```

```
void insertAtEnd()
{
    int data;
    struct node *temp, *head;
    temp = malloc(sizeof(struct node));

    // Enter the number
    printf("\nEnter number to"
        " be inserted : ");
    scanf("%d", &data);

    // Changes links
    temp->link = 0;
    temp->info = data;
    head = start;
    while (head->link != NULL) {
        head = head->link;
    }
    head->link = temp;
}

// Function to insert at any specified
// position in the linked list
void insertAtPosition()
{
    struct node *temp, *newnode;
    int pos, data, i = 1;
    newnode = malloc(sizeof(struct node));

    // Enter the position and data
```

```c
	printf("\nEnter position and data :");
	scanf("%d %d", &pos, &data);

	// Change Links
	temp = start;
	newnode->info = data;
	newnode->link = 0;
	while (i < pos - 1) {
		temp = temp->link;
		i++;
	}
	newnode->link = temp->link;
	temp->link = newnode;
}

// Function to delete from the front
// of the linked list
void deleteFirst()
{
	struct node* temp;
	if (start == NULL)
		printf("\nList is empty\n");
	else {
		temp = start;
		start = start->link;
		free(temp);
	}
}

// Function to delete from the end
```

```
// of the linked list
void deleteEnd()
{
	struct node *temp, *prevnode;
	if (start == NULL)
		printf("\nList is Empty\n");
	else {
		temp = start;
		while (temp->link != 0) {
			prevnode = temp;
			temp = temp->link;
		}
		free(temp);
		prevnode->link = 0;
	}
}

// Function to delete from any specified
// position from the linked list
void deletePosition()
{
	struct node *temp, *position;
	int i = 1, pos;

	// If LL is empty
	if (start == NULL)
		printf("\nList is empty\n");

	// Otherwise
	else {
```

```
		printf("\nEnter index : ");

		// Position to be deleted
		scanf("%d", &pos);
		position = malloc(sizeof(struct node));
		temp = start;

		// Traverse till position
		while (i < pos - 1) {
			temp = temp->link;
			i++;
		}

		// Change Links
		position = temp->link;
		temp->link = position->link;

		// Free memory
		free(position);
	}
}
int main()
{
	createList();
	int choice;
	while (1) {

		printf("\n\t1 Display list\n");
		printf("\t2 For insertion at"
			" starting\n");
```

```
printf("\t3 For insertion at"
        " end\n");
printf("\t4 For insertion at "
        "any position\n");
printf("\t5 For deletion of "
        "first element\n");
printf("\t6 For deletion of "
        "last element\n");
printf("\t7 For deletion of "
        "element at any position\n");
printf("\t8 To exit\n");
printf("\nEnter Choice :\n");
scanf("%d", &choice);

switch (choice) {
case 1:
        traverse();
        break;
case 2:
        insertAtFront();
        break;
case 3:
        insertAtEnd();
        break;
case 4:
        insertAtPosition();
        break;
case 5:
        deleteFirst();
        break;
```

```
        case 6:
            deleteEnd();
            break;
        case 7:
            deletePosition();
            break;
        case 8:
            exit(1);
            break;
        default:
            printf("Incorrect Choice\n");
        }
    }
    return 0;
}
```

Output:

```
Enter the number of nodes: 2

Enter number to be inserted : 10

Enter number to be inserted : 20

The list is created

    1 Display list
    2 For insertion at starting
    3 For insertion at end
    4 For insertion at any position
```

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 10

Data = 20

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

2

Enter number to be inserted : 30

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 30

Data = 10

Data = 20

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

3

Enter number to be inserted : 40

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 30

Data = 10

Data = 20

Data = 40

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

4

Enter position and data :3 50

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 30

Data = 10

Data = 50

Data = 20

Data = 40

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

5

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 10

Data = 50

Data = 20

Data = 40

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

6

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 10

Data = 50

Data = 20

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

7

Enter index : 2

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 10

Data = 20

1 Display list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

8

Experiment 10: Design, develop and execute a program in C to implement a doubly linked list where each node consists of integers. The program should support the following functions.

Create a doubly linked list

Insert a new node at start, middle and end

Delete a node at start, middle and end

Display the nodes of doubly linked list

Algorithm:

Insertion at the Beginning

1. START
2. Create a new node with three variables: prev, data, next.
3. Store the new data in the data variable
4. If the list is empty, make the new node as head.
5. Otherwise, link the address of the existing first node to the

next variable of the new node, and assign null to the prev variable.

6. Point the head to the new node.
7. END

Insertion at the End

1. START
2. If the list is empty, add the node to the list and point
   the head to it.
3. If the list is not empty, find the last node of the list.
4. Create a link between the last node in the list and the
   new node.
5. The new node will point to NULL as it is the new last node.
6. END

Deletion at the Beginning

1. START

2. Check the status of the doubly linked list

3. If the list is empty, deletion is not possible

4. If the list is not empty, the head pointer is shifted to the next node.

5. END

Program:

```
// C program for the all operations in
// the Doubly Linked List
#include <stdio.h>
#include <stdlib.h>

struct node {
	int info;
	struct node *prev, *next;
};
struct node* start = NULL;

void traverse()
{
	if (start == NULL) {
		printf("\nList is empty\n");
		return;
	}
	struct node* temp;
	temp = start;
	while (temp != NULL) {
		printf("Data = %d\n", temp->info);
		temp = temp->next;
	}
```

```
}
void insertAtFront()
{
	int data;
	struct node* temp;
	temp = (struct node*)malloc(sizeof(struct node));
	printf("\nEnter number to be inserted: ");
	scanf("%d", &data);
	temp->info = data;
	temp->prev = NULL;

	temp->next = start;
	start = temp;
}
void insertAtEnd()
{
	int data;
	struct node *temp, *trav;
	temp = (struct node*)malloc(sizeof(struct node));
	temp->prev = NULL;
	temp->next = NULL;
	printf("\nEnter number to be inserted: ");
	scanf("%d", &data);
	temp->info = data;
	temp->next = NULL;
	trav = start;

	if (start == NULL) {

		start = temp;
```

```
	}
	else {
		while (trav->next != NULL)
			trav = trav->next;
		temp->prev = trav;
		trav->next = temp;
	}
}
void insertAtPosition()
{
	int data, pos, i = 1;
	struct node *temp, *newnode;
	newnode = malloc(sizeof(struct node));
	newnode->next = NULL;
	newnode->prev = NULL;
	printf("\nEnter position : ");
	scanf("%d", &pos);
	if (start == NULL) {
		start = newnode;
		newnode->prev = NULL;
		newnode->next = NULL;
	}
	else if (pos == 1) {
	insertAtFront();
	}
	else {
	printf("\nEnter number to be inserted: ");
	scanf("%d", &data);
	newnode->info = data;
	temp = start;
```

```
        while (i < pos - 1) {
            temp = temp->next;
            i++;
        }
        newnode->next = temp->next;
        newnode->prev = temp;
        temp->next = newnode;
        temp->next->prev = newnode;
    }
}
void deleteFirst()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else {
        temp = start;
        start = start->next;
        if (start != NULL)
            start->prev = NULL;
        free(temp);
    }
}
void deleteEnd()
{
    struct node* temp;
    if (start == NULL)
        printf("\nList is empty\n");
    temp = start;
    while (temp->next != NULL)
```

```
            temp = temp->next;
        if (start->next == NULL)
            start = NULL;
        else {
            temp->prev->next = NULL;
            free(temp);
        }
}
void deletePosition()
{
        int pos, i = 1;
        struct node *temp, *position;
        temp = start;
        if (start == NULL)
            printf("\nList is empty\n");
        else {
            printf("\nEnter position : ");
            scanf("%d", &pos);
            if (pos == 1) {
                deleteFirst();
                if (start != NULL) {
                    start->prev = NULL;
                }
                free(position);
                return;
            }
            while (i < pos - 1) {
                temp = temp->next;
                i++;
            }
```

```
                position = temp->next;
                if (position->next != NULL)
                        position->next->prev = temp;
                temp->next = position->next;
                free(position);
        }
}
int main()
{
        int choice;
        while (1) {

                printf("\n\t1 To see list\n");
                printf("\t2 For insertion at"
                        " starting\n");
                printf("\t3 For insertion at"
                        " end\n");
                printf("\t4 For insertion at "
                        "any position\n");
                printf("\t5 For deletion of "
                        "first element\n");
                printf("\t6 For deletion of "
                        "last element\n");
                printf("\t7 For deletion of "
                        "element at any position\n");
                printf("\t8 To exit\n");
                printf("\nEnter Choice :\n");
                scanf("%d", &choice);

                switch (choice) {
```

```
        case 1:
            traverse();
            break;
        case 2:
            insertAtFront();
            break;
        case 3:
            insertAtEnd();
            break;
        case 4:
            insertAtPosition();
            break;
        case 5:
            deleteFirst();
            break;
        case 6:
            deleteEnd();
            break;
        case 7:
            deletePosition();
            break;

        case 8:
            exit(1);
            break;
        default:
            printf("Incorrect Choice. Try Again \n");
            continue;
        }
}
```

```
	return 0;
}
```

Output:

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

List is empty

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

2

Enter number to be inserted: 10

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

2

Enter number to be inserted: 20

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

3

Enter number to be inserted: 30

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 20

Data = 10

Data = 30

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

4

Enter position : 2

Enter number to be inserted: 40

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 20

Data = 40

Data = 10

Data = 30

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

3

Enter number to be inserted: 50

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 20

Data = 40

Data = 10

Data = 30

Data = 50

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

7

Enter position : 4

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 20

Data = 40

Data = 10

Data = 50

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

5

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 40

Data = 10

Data = 50

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

6

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 40

Data = 10

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

8

Experiment 11: Design, develop and execute a program in C to implement a circular singly linked list where each node consists of integers. The program should support the following functions.

Create a doubly linked list

Insert a new node

Delete a node

Display the nodes of circular singly linked list

Algorithm:

Inserting a Node at the Beginning of a Circular Linked List :-

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 11

[END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET AVAIL = AVAIL -> NEXT

Step 4: SET NEW_NODE -> DATA = VAL

Step 5: SET PTR = START

Step 6: Repeat Step 7 while PTR -> NEXT != START

Step 7: PTR = PTR -> NEXT

[END OF LOOP]

Step 8: SET NEW_NODE -> NEXT = START

Step 9: SET PTR -> NEXT = NEW_NODE

Step 10: SET START = NEW_NODE

Step 11: EXIT

Inserting a Node at the End of a Circular Linked List :-

Step 1: IF AVAIL = NULL

Write OVERFLOW

Go to Step 1

[END OF IF]

Step 2: SET NEW_NODE = AVAIL

Step 3: SET AVAIL = AVAIL -> NEXT

Step 4: SET NEW_NODE -> DATA = VAL

Step 5: SET NEW_NODE -> NEXT = START

Step 6: SET PTR = START

Step 7: Repeat Step 8 while PTR -> NEXT != START

Step 8: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 9: SET PTR -> NEXT = NEW_NODE

Step 10: EXIT

Deleting the first node from a circular linked list :-

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 8

[END OF IF]

Step 2: SET PTR = START

Step 3: Repeat Step 4 while PTR -> NEXT != START

Step 4: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 5: SET PTR -> NEXT = START -> NEXT

Step 6: FREE START

Step 7: SET START = PTR -> NEXT

Step 8: EXIT

Deleting the last node from a circular linked list :-

Step 1: IF START = NULL

Write UNDERFLOW

Go to Step 8

[END OF IF]

Step 2: SET PTR = START

Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START

Step 4: SET PREPTR = PTR

Step 5: SET PTR = PTR -> NEXT

[END OF LOOP]

Step 6: SET PREPTR -> NEXT = START

Step 7: FREE PTR

Step 8: EXIT

Program:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;

void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void main ()
{
   int choice =0;
   while(choice != 7)
   {
      printf("\nMain Menu\n");
```

```
        printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Show\n6.Exit\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);
            break;
            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 node 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;
            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 printing values ... \n");

        while(ptr -> next != head)
        {

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

}
```

Output:

Main Menu

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

4.Delete from last

5.Show

6.Exit

Enter your choice?

1

Enter the node data?10

node inserted

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

2

Enter Data?20

node inserted

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

1

Enter the node data?30

node inserted

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

5

printing values ...

30

10

20

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

2

Enter Data?40

node inserted

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

5

printing values ...

30

10

20

40

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

3

node deleted

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

5

printing values ...

10

20

40

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

4

node deleted

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

5

printing values ...

10

20

Main Menu

1.Insert in begining

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Show

6.Exit

Enter your choice?

6

Experiment 12: Implementation of various operations on Binary Tree like – creating a tree, displaying a tree, copying tree, mirroring a tree, counting the number of nodes in the tree, counting only leaf nodes in the tree.

Algorithm:

Insertion Operation

1. START
2. If the tree is empty, insert the first element as the root node of the tree. The following elements are added as the leaf nodes.
3. If an element is less than the root value, it is added into the left subtree as a leaf node.
4. If an element is greater than the root value, it is added into the right subtree as a leaf node.
5. The final leaf nodes of the tree point to NULL values as their child nodes.
6. END

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define COUNT 10
static int count = 0;
struct BST
{

    int data;
    struct BST *left;
    struct BST *right;

};
```

```
typedef struct BST NODE;

NODE *node;

NODE* createtree(NODE *node, int data)

{

    if (node == NULL)
    {

    NODE *temp;
    temp= (NODE*)malloc(sizeof(NODE));
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
    }

    if (data < (node->data))
    {
        node->left = createtree(node->left, data);
    }

    else if (data > node->data)
    {
        node -> right = createtree(node->right, data);
    }

    return node;

}
```

```
int countnodes(NODE *root)
{
   if(root != NULL)
   {
      countnodes(root->left);
      count++;
      countnodes(root->right);
   }
   return count;
}

void mirror(NODE *node)
{
   if (node == NULL)
      return;
   else {
      NODE *temp;

      /* do the subtrees */
      mirror(node->left);
      mirror(node->right);

      /* swap the pointers in this node */
      temp = node->left;
      node->left = node->right;
      node->right = temp;
   }
}

void print2DUtil(NODE * root, int space)
```

```
{
    // Base case
    if (root == NULL)
        return;

    // Increase distance between levels
    space += COUNT;

    // Process right child first
    print2DUtil(root->right, space);

    // Print current node after space
    // count
    printf("\n");
    for (int i = COUNT; i < space; i++)
        printf(" ");
    printf("%d\n", root->data);

    // Process left child
    print2DUtil(root->left, space);
}

// Wrapper over print2DUtil()
void print2D(NODE *root)
{
    // Pass initial space count as 0
    print2DUtil(root, 0);
}
unsigned int getLeafCount(NODE * node)
{
```

```
  if(node == NULL)
    return 0;
  if(node->left == NULL && node->right==NULL)
    return 1;
  else
    return getLeafCount(node->left)+
           getLeafCount(node->right);
}
int main()
{
     int data, ch, i, n;
     NODE *root=NULL;
     while (ch!=6)
     {
          printf("\n\n1.Insertion in Binary Search Tree");
          printf("\n2.Mirroring a Tree");
          printf("\n3.Display a Tree");
          printf("\n4.Count Total No. Of nodes in the Tree");
          printf("\n5.Count only Leaf Nodes");
          printf("\n6.Exit\n");
          printf("\nEnter your Choice: ");

          scanf("%d", &ch);

          switch (ch)
          {
               case 1:   printf("\nEnter size of tree: " );
                         scanf("%d", &n);
                         printf("\nEnter the elements of tree)\n");
```

```
            for(i=0; i<n; i++)
            {
                scanf("%d", &data);
                root=createtree(root, data);
            }
            break;

        case 2:
            mirror(root);
            printf("\nMIRRORING of the Tree is Done\n");
            break;

        case 3:  printf("\n PRINT a Tree: \n");
            print2D(root);
            break;

        case 4:  printf("\n Total no. of nodes in the Tree is %d \n", countnodes(root));
            break;

        case 5:  printf("\n No. of Leaf Nodes in the tree is %d \n",getLeafCount(root));
            break;

        case 6:   exit(0);

        default:  printf("\nEnter valid choice\n");
            break;

      }
   }
}
```

Output:

1.Insertion in Binary Search Tree

2.Mirroring a Tree

3.Display a Tree

4.Count Total No. Of nodes in the Tree

5.Count only Leaf Nodes

6.Exit

Enter your Choice: 1

Enter size of tree: 7

Enter the elements of tree)

50

30

35

20

60

55

65

1.Insertion in Binary Search Tree

2.Mirroring a Tree

3.Display a Tree

4.Count Total No. Of nodes in the Tree

```
5.Count only Leaf Nodes
6.Exit

Enter your Choice: 3

 PRINT a Tree:

          65

     60

          55

50

          35

     30

          20


1.Insertion in Binary Search Tree
2.Mirroring a Tree
3.Display a Tree
4.Count Total No. Of nodes in the Tree
5.Count only Leaf Nodes
6.Exit

Enter your Choice: 2
```

MIRRORING of the Tree is Done

1.Insertion in Binary Search Tree

2.Mirroring a Tree

3.Display a Tree

4.Count Total No. Of nodes in the Tree

5.Count only Leaf Nodes

6.Exit

Enter your Choice: 3

PRINT a Tree:

```
        20

    30

        35

50

        55

    60

        65
```

1.Insertion in Binary Search Tree

2.Mirroring a Tree

3.Display a Tree

4.Count Total No. Of nodes in the Tree

5.Count only Leaf Nodes

6.Exit

Enter your Choice: 4

Total no. of nodes in the Tree is 7

1.Insertion in Binary Search Tree

2.Mirroring a Tree

3.Display a Tree

4.Count Total No. Of nodes in the Tree

5.Count only Leaf Nodes

6.Exit

Enter your Choice: 5

No. of Leaf Nodes in the tree is 4

1.Insertion in Binary Search Tree

2.Mirroring a Tree

3.Display a Tree

4.Count Total No. Of nodes in the Tree

5.Count only Leaf Nodes

6.Exit

Enter your Choice: 6