PROGRAM-1: Write a program to simulate the working of stack using an array with the following a).Push b).Pop c).Display. The program should print appropriate messages for stack overflow and stack underflow.

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

int top=-1, stack[MAX];
void push();
void pop();
void display();

int main()
{
   int ch;
   while(1)
   {
   printf("\n**Stack Menu**");
   printf("\n\n1. Push\n2. Pop\n3. Display\n4. Exit");
   printf("\n\n Enter Your Choice(1-4):");
   scanf("%d",&ch);

   switch(ch)
   {
     case 1: push();
        break;
     case 2: pop();
        break;
     case 3: display();
        break;
     case 4: exit(0);
     default: printf("\n Wrong Choice");
   }
}
```

```
return 0;
}
void push()
{
int val;
if (top == MAX-1)
{ printf("\n Stack is Full");
}
else
{ printf("\n Enter Elements to Push :");
Scanf("%d", &val);
top = top+1;
Stack[top] = val;
}
}
void pop()
{
if (top == -1)
{ printf("\n Stack is Empty");
}
else
{ printf("\n Deleted element is %d", stack[top]);
top = top-1;
}
}
void display()
{ int i;
if (top == -1)
{ printf("\n Stack is Empty");
}
else
{ printf("\n Stack is .....");
```

for (i=top; i>=0; --i)
printf("%d\n", stack[i]);
}
}

Output :

** Stack Menu **
1. Push
2. Pop
3. display
4. exit

Enter Your choice (1-4): 1
Enter Element to Push: 3

** Stack Menu **
1. Push
2. Pop
3. display
4. Exit
Enter Your choice (1-4): 2

Deleted Element is 3
** Stack Menu **
1. Push
2. pop
3. display
4. Exit

PROGRAM-2: WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators "+" (plus) , "-" (minus), "*" (multiply) and "/" (divide).

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

int F(char symbol)
{
 switch (symbol)
 {
  case '+':
  case '-': return 2;
  case '*':
  case '/': return 4;
  case '^':
  case '$': return 5;
  case 'c': return 0;
  case '#': return -1;
   default: return 8;
 }
}
int G(char symbol)
{
  switch (symbol)
   {
  case '+':
  case '-': return 1;
  case '*':
  case '/': return 3;
  case '^':
  case '$': return 6;
  case 'c': return 9;
  case '(': return 0;
  default: return 7;
```

```
}
}
void infix_postfix (char infix[], char
postfix[])
{
int top, i, j;
char s[30], symbol;
top = -1;
s[++top] = '#';
j = 0;

for (i = 0; i < strlen (infix); i++)
 {
  symbol = infix[i];
 while (F(s[top]) >G (symbol))
 {
  postfix[j] = s[top--];
  j++;
 }
 if (F (s[top]) != G (symbol))
  s[++top] = symbol;
  else
  top--;
 }
 while (s[top] != '#')
{
 postfix[j++] = s[top--];
}
 postfix[j] = '\0';
}
  void main ()
{
```

Date __/__/__
Page
SPLASH

```
char imfix [20];
char postfix [20];
printf("Enter the Valid imfix expression");
scanf("%s", imfix);
imfix_postfix (imfix, postfix);
printf("The Postfix expression is \n");
printf("%s\n", postfix);
}
```

Output:-

Enter the Valid imfix expression:
(a+b)*(c-d)/(e/f)
The post fix expression is
ab+cd-*ef/

PROGRAM-3: WAP to simulate the working of a queue of integers using an array. Provide the following operations a).Insert b).Delete c).Display. The program should print appropriate messages for queue empty and queue overflow conditions.

LINER QUEUE

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3

void insert();
void delete();
void display();
int queue_array[MAX];
int rear = -1;
int front = -1;
int main()
{
    int choice;
    while()
    {
        printf("1. Insert element to queue \n");
        printf("2. Delete element from queue \n");
        printf("3. Display all element of queue \n");
        printf("4. Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                insert();
            break;
            case 2:
                delete();
            break;
            case 3:
                display();
            break;
            case 4:
```

```
exit (1);
default:
    Print f ("Wrong choice \n");
}
}
}
Void insert()
{
    int add_item;
    if (rear == MAX - 1)
    printf ("Queue Overflow \n");
else
{
    if (front == -1)
    front = 0;
    Print f (" Insert the element
in queue : ");
    Scanf ("%d", &add_item);
    rear = rear + 1;
    queue_array [rear] = add_item;
}
}
Void delete()
{
    if (front == -1 || front > rear)
{
    printf (" Queue Underflow \n");
    return;
}
else
{
    printf ("Element deleted from queue is %d \n");
```

```
        queue_array[front]);
        front = front + 1;
    }
}
void display()
{
int i;
if (front == -1)
    printf("Queue is Empty \n");
else
{
    printf("Queue is: \n");
    for (i = front; i <= rear; i++)
        printf("%d", queue_array[i]);
    printf("\n");
}
}
```

PROGRAM-4: WAP to simulate the working of a circular queue of integers using an array. Provide the following operations a).Insert b).Delete c).Display. The program should print appropriate messages for queue empty and queue overflow conditions.

Date __/__/__
Page _____
SPLASH

Circular queue

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 3
int a[SIZE], x;
int front = -1;
int rear = -1;

int IsEmpty()
{ if (rear == -1 && front == -1)
  return 1;
  else
  return 0;
}
  int IsFull()
{
  if (front == (rear+1) % SIZE)
  return 1;
  else
  return 0;
}
  void Enqueue(int x)
{
    if (IsFull())
  printf("The queue is full \n");
  else if (IsEmpty())
  {
    front = 0;
    rear = 0;
    a[rear] = x;
}
```

```
else
}
    rear = (rear+1) % SIZE;
    a[rear] = x;
  }
}
int dequeue()
{ int x;
  if (IsEmpty())
  printf("The queue is empty");
  else if (front == rear)
{
    x = a[front];
    front = -1;
    rear = -1;
    printf("The element was removed");
}
else
{
    x = a[front];
    front = (front+1) % Size;
  printf("The element was removed\n");
}
return x;
}
void display()
{
  if (front == -1)
{
  printf("\n Queue is Empty");
  return;
}
```

```
}
printf("\n Elements in Circular Queue are: \n");
    if (rear >= front)
    {
      for (int i = front; i <= rear; i++)
        printf("%d \n", a[i]);
    }
    else
    {
    for (int i = front; i < Size; i++)
      printf("%d \n", a[i]);
  for (int i = 0; i <= rear; i++)
    printf("%d \n", a[i]);
}
}
int main()
{
  int n, a;
  while (1)
  {
    printf("Enter the operation \n
1-Insert \n 2-Delete \n 3-Display \n 4-Exit \n");
  scanf("%d", &n);
  switch (n)
{
  case 1: printf("Enter the element \n");
    scanf("%d", &a);
  Enqueue(a);
    break;
  case 2: Dequeue();
      break;
```

Date __/__/__
Page
SPLASH

```
case 3: display();
        break;
case 4: exit(0);
default: printf("There is no such operation\n");
}
}
return 0;
}
```

PROGRAM-5: WAP to Implement Singly Linked List with following operations a).Create a linked list b).Insertion of a node at first position, at any position and at end of the list c).Display the contents of the linked list.

Date / /
Page
SPLASH

LAB PROGRAM-5

```
#include <stdio.h>
#include <malloc.h>
Struct node
{
int info;
Struct node *link;
};
typedef Struct node * NODE;
NODE getnode()
{
NODE x;
x = (NODE) malloc (sizeof (Struct node));
if (x == NULL)
{
printf ("mem full\n");
exit (0);
}
return x;
}
int freenode (NODE x)
{
free (x);
return 0;
}
NODE insert_front (NODE first, int item)
{
NODE temp;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL)
```

```
return temp;
temp->link = first;
first = temp;
return first;
}
NODE insert_rear(NODE first, int item)
{
NODE temp, cur;
temp = getnode();
temp->info = item;
temp->link = NULL;
if(first == NULL)
return temp;
cur = first;
while(cur->link != NULL)
cur = cur->link;
cur->link = temp;
return first;
}
void display(NODE first)
{
NODE temp;
if(first == NULL)
printf("List empty cannot display\n");
for(temp = first; temp != NULL; temp = temp->link)
printf("%d\n", temp->info);
}
NODE insert_pos(int item, int pos, NODE first)
{
NODE temp;
```

Date / /
Page
SPLASH

```
NODE *prev, *cur;
int count;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL && pos == 1)
{
return temp;
}
{
printf("invalid position\n");
return first;
}
if (pos == 1)
{
temp -> link = first;
return temp;
}
count = 1;
prev = NULL;
cur = first;
while cur != NULL && count != pos
{
prev = cur;
cur = cur -> link;
count++;
}
if(count == pos)
{
prev -> link = temp;
temp -> link = cur;
return first;
}
```

```
{
cprev ptr,
    cur = cur -> link;
    count++;
}
if (count == pos)
{
prev -> link = temp;
temp -> link = cur;
return first;
}
insertion()
{ int item, choice, pos;
  NODE first = NULL;
  system("cls");
  for(;;)
```

2.

```
printf("\n1.Insert from \n2.Insert at specific position\n3.
Insert end\n4.Display\n5.Exit\n");
  printf("Enter the choice\n");
  scanf("%d", &choice);
  switch(choice)
  {
case 1: printf("Enter the item at front end\n");
  scanf("%d", &item);
  first = insert_front(first, item);
  break;
  case 2: printf("Enter the item to be inserted:\n");
          scanf("%d", &item);
          printf("Enter the position at which the item
          to be inserted\n");
          scanf("%d", &pos);
```

Date __/__/__
Page
SPLASH

```
break;
case 3: printf("Enter the item at rear - \n");
        scanf("%d", &item);
        first = insert_rear(first, item);
        break;
case 4: display(first);
        break;
default: exit(0);
        break;
}
}
getch();
return 0;
}
```