1. 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

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


#define MAX 2
Int qu[MAX];
Int front =-1;
Int rear = -1;

Void insert();
Int delete_q();
Void display();

Int main(){
   While(1){
      Int choice;
      Printf("\n1. Insert \t 2.delete \t 3.display \t 4.exit\n");
      Scanf("%d",&choice);
      Switch(choice){
         Case 1:
            Insert();
            Break;
```

```
            Case 2:
                Delete_q();
                Break;
            Case 3:
                Display();
                Break;
            Case 4:
                Exit(0);
        }
    }
}

Void insert(){
    If(rear == MAX -1){
        Printf(“Queue is Full\n”);
        Return;
    }
    Printf(“Enter the element to be inserted\n”);
     Int a;
     Scanf(“%d”,&a);
    If((front == -1) && (rear == -1)) {
        Front=rear=0;
    }
    Else{
        Rear++;
    }
        Qu[rear]=a;
}
```

```
Int delete_q(){
   If(front == rear && rear != 0){
      Printf(“stack is empty”);
      Exit(0);
   }
   If(front == rear == 0){
      Int x = qu[front];
      Front = rear = -1;
      Return x;
   }
   Else if(front != -1 && rear>front){
      Int x = qu[front];
      Front ++;
      Return x;
   }
}
Void display(){
   Printf(“the elements are:\n”);
   For(int i = front;i<=rear;i++){
      Printf(“%d \n”,qu[i]);
   }
}
```

OUTPUT:

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

10

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

20

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

30

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

40

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

50

1. Insert 2. Delete 3. Display 4. Exit

1

Queue is Full

1. Insert 2. Delete 3. Display 4. Exit

3

The elements are:

10

20

30

40

50

1. Insert 2. Delete 3. Display 4. Exit

2

The number popped is: 10

1. Insert 2. Delete 3. Display 4. Exit

2

The number popped is: 20

1. Insert 2. Delete 3. Display 4. Exit

2

The number popped is: 30

1. Insert 2. Delete 3. Display 4. Exit

2

The number popped is: 40

1. Insert 2. Delete 3. Display 4. Exit

2

The number popped is: 50

1. Insert 2. Delete 3. Display 4. Exit

2

Queue is Empty

1. Insert 2. Delete 3. Display 4. Exit

4

2. Write a program 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), / (divide) and ^ (power).

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

#define MAX 100

Void push(char st[],char ch);
Char pop(char st[]);
Void infix_to_postfix(char src[],char ans[]);
Int isalpha_numeric(char ch);
Int isOperator(char ch);
Int isPrior(char ch);
Int top = -1;
Char st[MAX];

Int main(){
   Char postfix[100],infix[100];
   Printf(“Enter the infix expression\n”);
   Scanf(“%s”,infix);
   Strcpy(postfix,””);
   Infix_to_postfix(infix,postfix);
   Printf(“The postfix expression is\n”);
   Printf(“%s\n”,postfix);

}
Int isalpha_numeric(char ch){
```

```
    If((ch>= ‘a’ && ch<=’z’)||(ch >=’A’ && ch <= ‘Z’)||(ch >= ‘0’ && ch <= ‘9’)){
        Return 1;
    }else{
        Return 0;
    }
}
Int isOperator(char ch){
    If(ch == ‘+’ || ch == ‘-‘ || ch == ‘*’ || ch == ‘/’ ||ch == ‘%’ ){
        Return 1;
    }else{
        Return 0;
    }
}
Int isPrior(char ch){
    If( ch == ‘*’ || ch == ‘/’ ||ch == ‘%’){
        Return 1;
    }else{
        Return 0;
    }
}

Void infix_to_postfix(char src[],char ans[]){
    Int i=0;
    Int j =0;
    While(src[i]!=’\0’) {
        If(src[i] == ‘(‘){
            Push(st,src[i]);
        }
        Else if(isalpha_numeric(src[i])){
```

```
        Ans[j]= src[i];
        ++j;
      }
      Else if(isOperator(src[i])){
        While(top != -1 && st[top] != '(' && (isPrior(st[top]) >= isPrior(src[i]))){
          Ans[j] = pop(st);
          ++j;
        }
        Push(st,src[i]);
      }else if(src[i] == ')'){
        While(top != -1 && st[top] != '('){
          Ans[j]= pop(st);
          ++j;
        }
        Pop(st);
      }
      Else{
        Printf("invalid expression");
        Exit(0);
      }
       ++i;

   }
    While(top != -1 && st[top] != '('){
        Ans[j] = pop(st);
        ++j;
      }
      Ans[j]='\0';
}
```

```
Void push(char st[],char ch){
   If(top == MAX-1){
      Printf(“Stack overflow\n”);
   }
   Else{
      ++top;
      St[top] = ch;
   }
}
Char pop(char st[]){
   Char ch = ‘\0’;
   If(top ==-1){
      Printf(“Stack underflow\n”);
   }
   Else{
      Ch = st[top];
      --top;
   }

   Return ch;
}
```

OUTPUT:

Enter the infix expression

(a+b/c*(d+e)-f)

The postfix expression is

Abc/de+*+f-

3) 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 condition

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

#define MAX 100

Void push(char st[],char ch);
Char pop(char st[]);
Void infix_to_postfix(char src[],char ans[]);
Int isalpha_numeric(char ch);
Int isOperator(char ch);
Int isPrior(char ch);
Int top = -1;
Char st[MAX];

Int main(){
   Char postfix[100],infix[100];
   Printf(“Enter the infix expression\n”);
```

```
    Scanf("%s",infix);
    Strcpy(postfix,"");
    Infix_to_postfix(infix,postfix);
    Printf("The postfix expression is\n");
    Printf("%s\n",postfix);



}
Int isalpha_numeric(char ch){
    If((ch>= 'a' && ch<='z')||(ch >='A' && ch <= 'Z')||(ch >= '0' && ch <= '9')){
        Return 1;
    }else{
        Return 0;
    }
}
Int isOperator(char ch){
    If(ch == '+' || ch == '-' || ch == '*' || ch == '/' ||ch == '%' ){
        Return 1;
    }else{
        Return 0;
    }
}
Int isPrior(char ch){
    If( ch == '*' || ch == '/' ||ch == '%'){
        Return 1;
    }else{
        Return 0;
    }
}
```

```
Void infix_to_postfix(char src[],char ans[]){
    Int i=0;
    Int j =0;
    While(src[i]!='\0') {
        If(src[i] == '('){
            Push(st,src[i]);
        }
        Else if(isalpha_numeric(src[i])){
            Ans[j]= src[i];
            ++j;
        }
        Else if(isOperator(src[i])){
            While(top != -1 && st[top] != '(' && (isPrior(st[top]) >= isPrior(src[i]))){
                Ans[j] = pop(st);
                ++j;
            }
            Push(st,src[i]);
        }else if(src[i] == ')'){
            While(top != -1 && st[top] != '('){
                Ans[j]= pop(st);
                ++j;
            }
            Pop(st);
        }
        Else{
            Printf("invalid expression");
            Exit(0);
        }
```

```
        ++i;

    }
    While(top != -1 && st[top] != '('){
        Ans[j] = pop(st);
        ++j;
      }
      Ans[j]='\0';
}
Void push(char st[],char ch){
   If(top == MAX-1){
      Printf("Stack overflow\n");
   }
   Else{
      ++top;
      St[top] = ch;
   }
}
Char pop(char st[]){
   Char ch = '\0';
   If(top ==-1){
      Printf("Stack underflow\n");
   }
   Else{
      Ch = st[top];
      --top;
   }

   Return ch;
```

}

OUTPUT :

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

2

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

4

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

6

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

8

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

18

1. Insert 2. Delete 3. Display 4. Exit

1

Queue is Full

1. Insert 2. Delete 3. Display 4. Exit

3

The elements are:

2 4 6 8 18

1. Insert 2. Delete 3. Display 4. Exit

2

The number poped is : 2

1. Insert 2. Delete 3. Display 4. Exit

1

Enter the element to be inserted

100

1. Insert 2. Delete 3. Display 4. Exit

3

The elements are:

4 6 8 18 100

1. Insert 2. Delete 3. Display 4. Exit

4