UNIT-3

1. Write short notes on Data Structures and list its classifications with examples

A Data Structure is a particular way of organizing data in a computer so that it can be used effectively.

There are 2 types of data structures

→ Linear

→ Non-linear

Linear : The elements of a linear structure form a sequences.

Ex : Arrays, stacks, queues, linked list.

Non-linear : The elements of a non-linear structure do not form a sequence.

Ex : trees, graphs.

2 write a c program to Implement Stack using arrays. What are the advantages and disadvantages of this approach

A Program :

```
#include<stdio.h>
#include<stdlib.h>
#define STACK_SIZE 4
int Stack[STACK_SIZE];
int top = -1;
void push(int);
void POP();
void display();
void Peek();
int main()
{
int item, ch;
while(1)
{
scanf("%d", &ch);
switch(ch)
{
case 1 : scanf("%d", &item);
         push(item);
         break;
case 2 : POP();
         break;
case 3: display(); break;
case 4: Peek(); break;
case 5: exit(0);
default: printf("Invalid Choice");
}
}
}
```

```
void push (int element)
{
if (top == STACK_SIZE-1)
  printf("Stack Overflow - Cannot push");
else
{
  top= top+1;
  Stack[top] = element;
}
}
VOH POP()
{
if (top == -1) printf("In Stack underflow");
else{printf("In The POPPed element is :%d",
                                Stack[top]);
  top--;
}
}
void display()
{
int i;
if (top==-1)
  printf("In stack is empty");
else
{for (i=top; i>=0; i--)
  printf("\n%d", Stack[i]);
}
}
void Peek()
{
if (top==-1)
  Printf("In Stack Empty");
else
  printf("In The top element is %d", Stack[top]);
}
```

Advantages:

→ simplicity

→ Efficiency

→ LIFO

→ Limited memory usage

Disadvantages:

→ Limited access

→ potential for overflow

→ Not suitable for random access

→ limited capacity

3. Write a C program to implement circular queue operation like enqueue, dequeue and display

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 4
int queue[MAX];
int rear = -1, front = -1;
void enqueue (int);
void dequeue();
void display();
void main()
{
  int a, n, key, ch, i;
  while(1)
  {
    scanf("%d", &ch);
    switch(ch)
    {
      case 1: scanf("%d", &n);
              enqueue(n); break;
      case 2: dequeue(); break;
      case 3: display(); break;
      case 4: exit(0);
      default: printf("Invalid choice");
    }
  }
}
void enqueue (int n)
{
  if ((rear == MAX-1) && (front == 0) ||
                (rear == front - 1))
  {
    printf("\n overflow");
    exit(0);
  }
  else
  {
    if (front == -1)
    { rear++; front++; }
    else if (rear == MAX-1)
      rear = 0;
    else rear++;
    queue[rear] = n;
  }
}
void dequeue()
{
  int i;
  if (front == -1)
    printf("\n underflow");
  else
  {
    i = queue[front];
    printf("\n Dequeued element = %d", i);
    if (front == rear)
      front = rear = -1;
    else if (front == MAX-1)
      front = 0;
    else front++;
  }
}
void display()
{
  if (front == -1)
  { printf("empty");
    exit(0);
  }
  else
  { int i;
    printf("\n");
    if (front <= rear)
    { for (i=front; i<=rear; i++)
        printf("%d", queue[i]);
    }
    else
    { for (i=front; i<MAX; i++)
        printf("%d", queue[i]);
      for (i=0; i<=rear; i++)
        printf("%d", queue[i]);
    }
  }
}
```

3 Explain how you would handle overflow and underflow conditions in a circular queue and advantages of using a circular queue over a linear queue

4 Overflow condition

```
if ((rear == Maxsize-1) && (front == 0)) || (rear == front-1)
```

underflow condition

```
if (front == -1)
```

Advantages

→ The circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full

→ It is possible to insert new elements, if and only if those locations are empty.

→ The main advantages are

- → Efficient use of memory
- → Easier for insertion - deletion
- → Better performance
- → Simplified Implementation
- → Improved flexibility
- → Reduced overhead.

5. Explain how you would initialize a circular queue with a fixed size and manage the front and rear pointers and write real-world examples where a circular queue would be more suitable than a linear queue.

A The circular queue will be initialized with fixed size n and front and rear values are initialised to -1.

The real-world examples where circular queue is suitable are

→ computer controlled traffic signal systems

→ CPU scheduling and memory management.

7. write a c program to evalute the infix expression to postfix and evaluate the postfix expression.

5, 4, 6, +, *, 4, 9, 3, /, +, * and represent the stack of each step

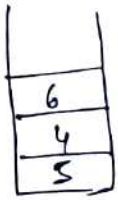
```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#define STACK_SIZE 100
int stack[STACK_SIZE];
int top=-1;
void push(int);
int pop();
int isOperand(char);
int isOperator(char);
void main()
{
 char postfix[STACK_SIZE],ch;
 int i=0,x,v,value,a,b,c;
 scanf("%s",postfix);
 for(i=0; postfix[i]!='\0';i++)
 {
  ch=postfix[i];
   if(isalpha(ch))
    {
     scanf("%d",&x);
     push(x);
    }
   else if(isOperand(ch))
   {
     v=ch-'0';
      push(v);
   }
  else if(isOperator(ch))
  {
    a=pop();
    b=pop();
    switch(ch)
    {
    case '+': c=b+a; break;
     case '-': c=b-a; break;
     case '/': c=b/a; break;
     case '*': c=b*a; break;
     case '%': c=b%a; break;
     case '^': c=b^a; break;
     default: printf("\nInvalid choice");
    }
     push(c);
  }
 }
 value=pop();
 if(top!=-1)
  printf("\nInsufficient postfix expr");
 else
  printf("\nThe result is :%d",value);
}
int isOperator(char ch)
{
 if(ch=='+'||ch=='-'||ch=='/'||
    ch=='*'||ch=='%'||ch=='^')
  return 1;
 else return 0;
}
int isOperand(char ch)
{
  if(ch>='0'&&ch<='9')
   return 1;
  else
 }  return 0;
```

5, 4, 6, +, *, 4, 9, 3, /, +, *

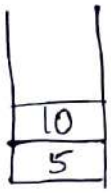
Step 1:

Step 2:

Step 3:

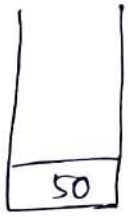


Step 4:



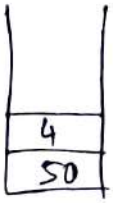
4 + 6 = 10

Step 5:

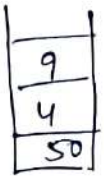


5 × 10 = 50

Step 6:

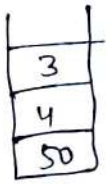


Step 7:



Step 8:

Step 9:



9 / 3 = 3

Step 10:

4 + 3 = 7

Step 11:

50 × 7 = 350

8. Write a C program to convert the infix expression to postfix and construct a postfix expression for A*(B^C+(D-E))/F using stack.

```
#include <stdio.h>
#include <ctype.h>
#define STACK_SIZE 100
char stack[STACK_SIZE];
int top=-1;
void push(char[], char);
char pop(char[]);
int getpriority(char);
int main()
{
char infix[100], postfix[100], temp;
int i, j=0;
scanf("%s", infix);
for(i=0; infix[i]!='\0'; i++)
{
 if(infix[i]=='(')
    push(stack, infix[i]);
 else if(isalpha(infix[i]) ||
         isdigit(infix[i]))
   {
     postfix[j] = infix[i];
     j++;
   }
else if(infix[i]=='+'||infix[i]=='-'
   ||infix[i]=='*'||infix[i]=='/'||
   infix[i]=='%'||infix[i]=='^')
 {
   while((getpriority(stack[top])>=
                 getpriority(infix[i])))
   {
     postfix[j] = pop(stack);
     j++;
   }
   push(stack, infix[i]);
}
else if(infix[i]==')')
{
  while((stack[top])!='(')
    {
      postfix[j] = pop(stack);
      j++;
    }
    temp=pop(stack);
}
}
while((top)!=-1))
{
  postfix[j]=pop(stack);
  j++;
}
  ~~temp=pop(stack);~~
  postfix[j]='\0';
printf("postfix exp = %s", postfix);
return 0;
}
int getpriority(char op)
{
  if(op=='^')
    return 2;
  else if(op=='/'||op=='*'||op=='%')
    return 1;
  else if(op=='+'||op=='-')
    return 0;
  else
    return -1;
}
```

A * B ^ C + (D * E) / F

Read Symbol	Stack	O/P
Initial	(	
1	(	A
2	(*	AB
3	(* ^	AB
4	(* ^	ABC
5	(*	ABC ^
6	(+	ABC ^ *
7	(+ (	ABC ^ *
8	(+ (	ABC ^ * D
9	(+ (*	ABC ^ * D
10	(+ (*	ABC ^ * DE
11	(+ (	ABC ^ * DE *
12	(+	ABC ^ * DE *
13	(+ /	ABC ^ * DE *
14	(+ /	ABC ^ * DE * F
15	(+	ABC ^ * DE * F /
16	-	ABC ^ * DE * F / +

9. Develop an algorithm for enqueue and dequeue operations of linear queue and construct a postfix expression for A + (B * (C - D) / E) using Stack.

A Algorithm for enqueue()

@ insert (maxsize, element)

Step 1: Initialize front = -1 and rear = -1

Step 2: If rear == maxsize - 1 print queue overflow and returns else goto step 3

Step 3: Set rear = rear + 1

Step 4: queue [rear] = element

Step 5: end if

Step 6: end

Algorithm for dequeue()

@ delete (maxsize, element)

Step 1: If front == -1 print queue is empty and returns else goto step 2

Step 2: element = queue [front]

Step 3: front = front + 1

Step 4: end

A + (B * (C - D) / E)

Read Symbol	Stack	O/P
Initial	(	
1	(	A
2	(+	A
3	(+ (	A
4	(+ (	AB
5	(+ (*	AB
6	(+ (* (	AB
7	(+ (* (	ABC
8	(+ (* (-	ABC
9	(+ (* (-	ABCD
10	(+ (*	ABCD -
11	(+ (/	ABCD - *
12	(+ (/	ABCD - * E
13	(+	ABCD - * E /
14		ABCD - * E / +