1st Assignment
DSC

① Write a C program to perform multiplication of two matrices

Ans) Program :-

```
#include <stdio.h>
int main(){
    int a[10][10], b[10][10], mul[10][10], r, c, i, j, k;
    printf("Enter the number of row = ");
    scanf("%d", &r);
    printf("Enter the first number of column =");
    scanf("%d", &c);
    printf("Enter the first matrix elements = \n");
    for(i=0; i<r; i++){
        for(j=0; j<c; j++){
            scanf("%d", &a[i][j]);
        }
    }
    printf("Enter the second matrix elements = \n");
    for(i=0; i<r; i++){
        for(j=0; j<c; j++){
            scanf("%d", &b[i][j]);
        }
    }
    printf("multiply of the matrix = \n");
    for(i=0; i<r; i++){
        for(j=0; j<c; j++){
            mul[i][j]=0;
            for(k=0; k<c; k++){
                mul[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    for(i=0; i<r; i++){
        for(j=0; j<c; j++){
            printf("%d \t", mul[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output :-

```
Enter the number of row = 2
Enter the number of column = 2
Enter the first matrix elements =
4
5
3
2
Enter the Second matrix elements =
3
4
8
9
multiply of the matrix =
52   61
25   30
```

② Write Various algorithms for basic operations on Stack.

A Stack is defined as linear data structure in which the insertion and deletion operations are performed on the same end of the stack. Stack follows LIFO (Last-In-first-Out) mechanism.

Representation of a Stack :-

A Stack can be represented in the memory in two ways

(1) Array Representation of Stacks.
(2) Linked-List Representation of Stacks.

The basic operations required to manipulate a stack are :

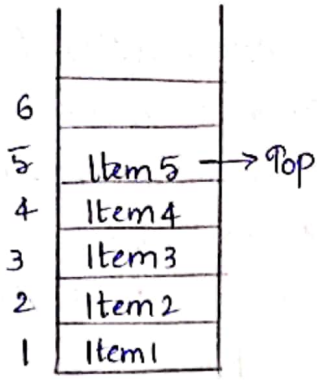
(1) PUSH → To insert an item into a stack
(2) POP → To remove an item from a Stack
(3) STATUS → To know the present status (state) of a stack.

1. The Operators for a stack represented with Arrays :-

1. Algorithm - Push - Array :-

Steps :-

1. If Top ≥ size then
2. print "Stack is full".
3. Else
4. Top = Top + 1
5. A[Top] = ITEM
6. EndIf
7. Stop



2. Algorithm Pop - Array :-

Steps :-

1. If Top < 1 then
2. print "Stack is empty".
3. Else
4. ITEM = A[Top]
5. Top = Top - 1
6. EndIf
7. Stop

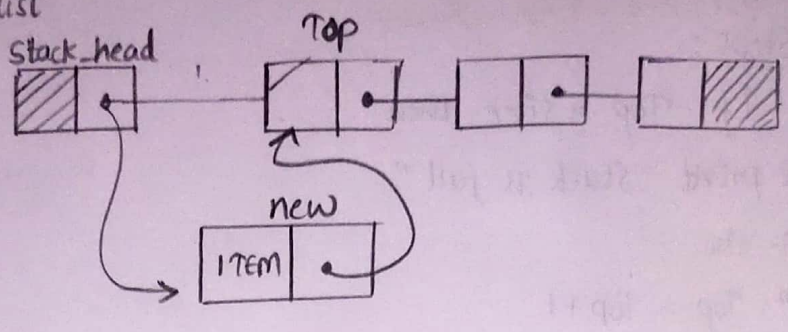
3. Algorithm Status - Array :-

Steps :-

1. If Top < 1 then
2. print "Stack is Empty".
3. Else
4. If (Top ≥ size) then
5. print "Stack is full".
6. Else
7. print "The Element at Top is ", A[Top]
8. free = (size - Top) / size * 100
9. print "percentage of free stack is ", free
10. EndIf
11. EndIf
12. Stop

The Operators for a Stack represented with Linked lists :-

1) Algorithm PUSH – Linked list



Steps :-

1. new = GetNode(NODE)
2. new → DATA = ITEM
3. new → LINK = TOP
4. TOP = new
5. Stack_head → LINK = TOP
6. Stop

GetNode Algorithm :-

Steps :-

1. If (AVAIL = NULL)
2. Return (NULL)
3. Print "Insufficient memory : Unable to allocate memory".
4. Else
5. ptr = AVAIL
6. While (sizeof (ptr) ≠ sizeOf(NODE)) and (ptr → LINK ≠ NULL) do
7. ptr1 = ptr
8. ptr = ptr → LINK
9. EndWhile
10. If (sizeOf(ptr) = sizeof(NODE))
11. ptr1 → LINK = ptr → LINK
12. Return(ptr)
13. Else
14. print "The memory block is too large to fit"
15. Return(NULL)
16. EndIf
17. EndIf
18. Stop

2) Algorithm POP - Linked list

Steps :-

1. If Top = NULL
2. print "Stack is Empty"
3. Exit
4. Else
5. Ptr = Top → LINK
6. ITEM = Top → DATA
7. Stack_head → LINK = Ptr
8. Top = Ptr
9. EndIf
10. Stop

3) Algorithm Status - Linked list

Steps :-

1. Ptr = stack_head → LINK
2. If (Ptr = NULL) then
3. print "Stack is Empty".
4. Else
5. node_count = 0
6. while (ptr ≠ NULL) do
7. node_count = node_count + 1
8. ptr = ptr → LINK
9. Endwhile
10. print "The item at the front is ", TOP → DATA, "Stack contains", node_count, "Number of items".
11. EndIf
12. Stop

3. Convert infix expression into its equivalent post fix expression

A * (B+D) / E - F * (G + H/K)

Input A * (B+D) / E - F * (G + H/K)

Read Symbol	Stack	Output
Initial		
1		A
2	*	A
3	*(	A
4	*(	AB
5	*(+	AB
6	*(+	ABD
7	/	ABD+*
8	/	ABD+*E
9	-	ABD+*E/
10	-	ABD+*E/F
11	-*	ABD+*E/F
12	-*(	ABD+*E/F
13	-*(	ABD+*E/FG
14	-*(+	ABD+*E/FG
15	-*(+	ABD+*E/FGH
16	-*(+/	ABD+*E/FGH
17	-*(+/	ABD+*E/FGHK
18		ABD+*E/FGHK/+*-

Output :- postfix Expression is ABD+*E/FGHK/+*-