MODULE - 02

* STACK :

1. Define Stack (DS). Explain the different operation that can be performed on stack using C function & show them using a diagramatic representation ?

(or)

Give the implementation of push, pop & display function. Include check for-empty & full conditions.

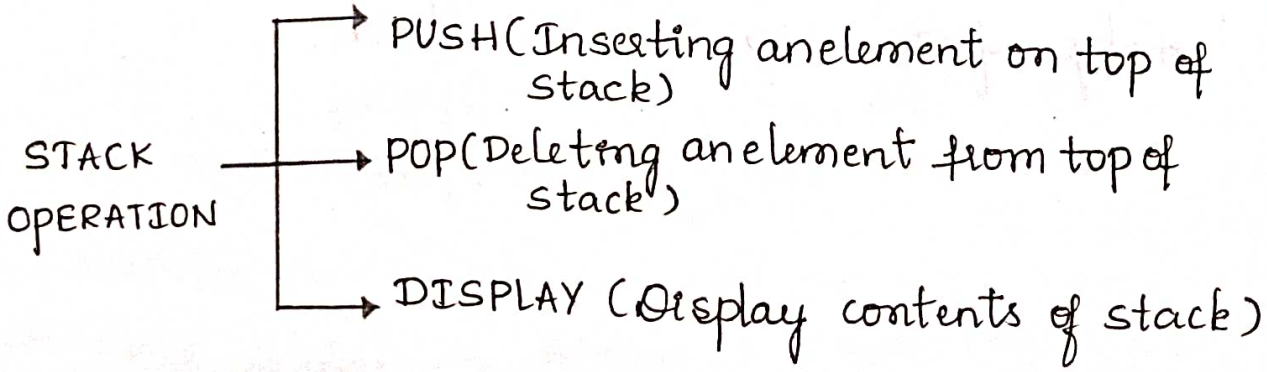
[June/July 2019, Aug/Sep 2020, Dec/Jan 2019-20, Jab/Feb 2021 Jan 2019-20]

→ Stack : A stack is a special type of data structure (linear data structure) where elements are inserted from one end and elements are deleted from the same end called 'Top'.

It is also called as

LIFO : Last in First Out.

FILO : First in Last out.

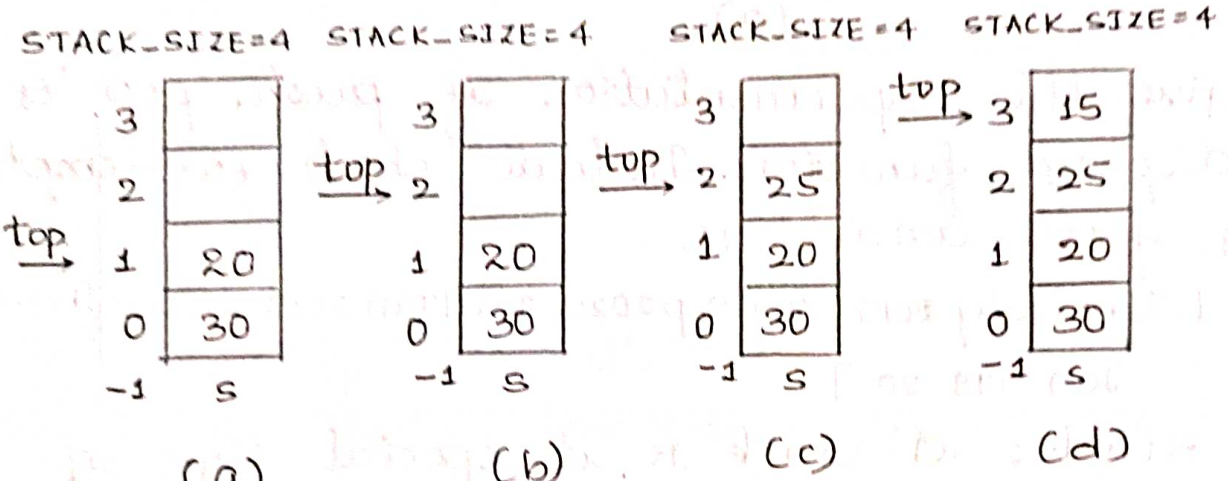


1. PUSH Operation:

→ Inserting an element into the stack is called push operation.

- Only one item is inserted at a time and item has to be inserted only from top of stack.

→ Diagramatic representation with an example:



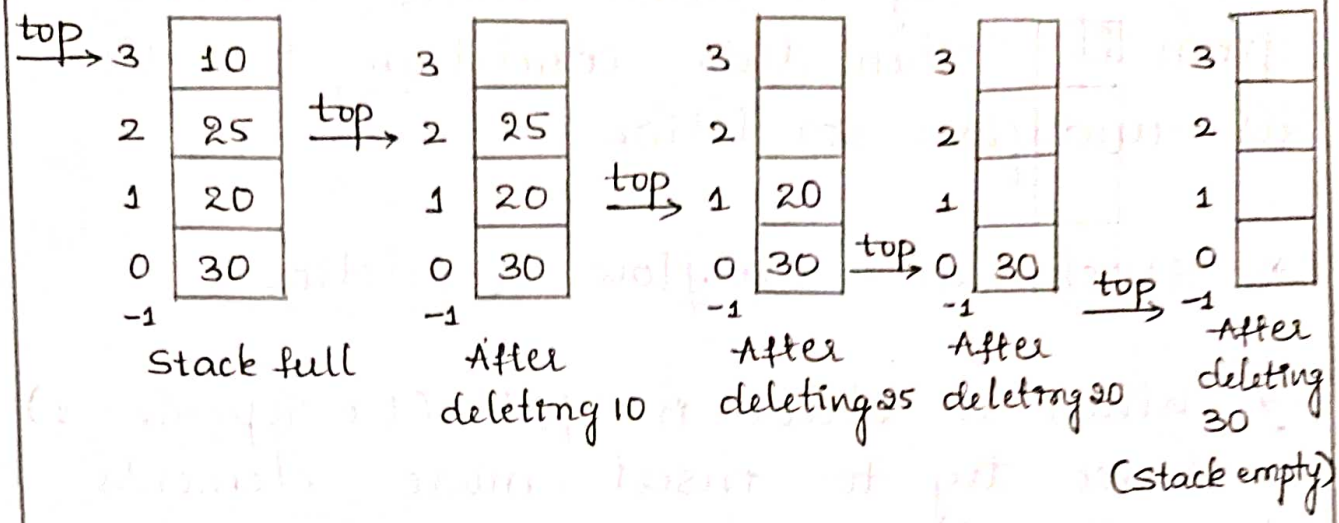
⇒ push operation. diagramatic represen

2. POP operation:

→ Deleting an element from the stack is called pop operation.

- Only one item can be deleted at a time and item has to be deleted only from top of the stack.

→ Diagramatic representation of Pop with an example.



3. Display stack items:

→ In display procedure, if the stack already has same items, all those items are displayed one after the other.

- If no items are present, the appropriate error message is displayed.

~> Diagramatic representation:

Design	Output
printf("%d\n", s[0]);	30
printf("%d\n", s[1]);	20
printf("%d\n", s[2]);	25

In general, we can use printf("%d\n", s[i]); Note: i=0 to 2 (i.e, top)

* Stack empty / Underflow Condition:

→ When a stack is empty (i.e Top = -1) and we try to delete more elements from it. Then this condition is called as underflow condition.

* Stack Full / Overflow Condition:

→ When a stack is full (i.e Top = size-1) and we try to insert more elements from it. Then this condition is called as overflow condition.

② → Program: The different operation that can be performed on stack using C function :

```
# include < stdio.h>
# include < stdlib.h>
#  define MAX 5
int s [MAX];
int top = -1;

void push();
void pop();
void display();

void main()
{
    int choice;
    while (1)
    {
        printf ("1. PUSH \n");
        printf ("2. POP\n");
        printf ("3. DISPLAY \n");
        printf ("4. EXIT\n");
        printf ("Enter your choice \n");
        scanf ("%d", & choice );

       switch( choice)
       {
```

```
            case 1: push();   break;
            case 2: pop();    break;
            case 3: display(); break;
            case 4: exit(0);  break;
            default: printf("Enter valid choice\n");
        }
    }
}
void push()
{
    int ele;
    if (top == MAX-1)
    {
        printf ("stack Overflow \n");
        return;
    }
    printf ("Enter an element to be pushed \n");
    scanf ("%d", &ele);
    top = top+1;
    s[top] = ele;
    return;
}
void pop()
{
    if (top == -1)
    {
        printf ("stack Underflow \n");
        return;
    }
```

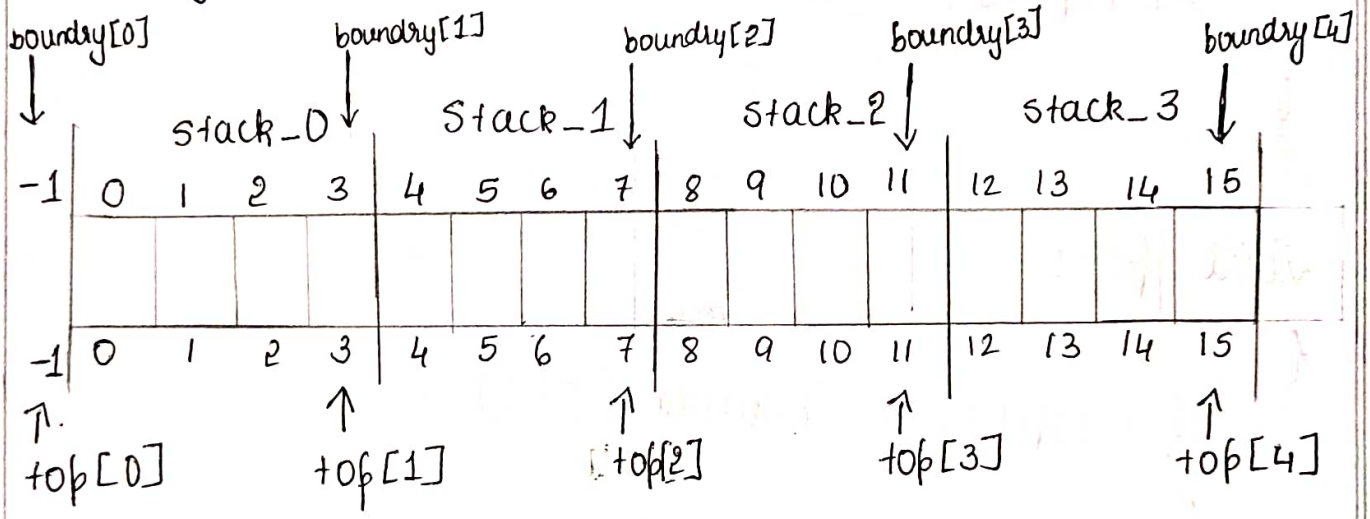
```
        printf ("Elements popped is %d\n", s[top]);
        top = top - 1;
        return;
    }
void display()
{
    int i;
    if (top == -1)
    {
        printf ("stack is Empty\n");
        return;
    }
    printf (" stack elements are\n");
    for (i=top; i>=0; i--)
    printf ("|%d|\n", s[i]);
    return;
}
```

③. Explain in details multiple stacks with relevant function in C. June/July 2019

A single array having more than one stack is called multiple stack.

A single stack is sometime not sufficient to store large amount of data. To overcome this problem we use multiple stack.

Diagramatic representation



C Program to implement multiple stacks

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 20
int s[MAX_SIZE];
#define MAX_STACKS 10          // Maximum no of stacks
int boundary[MAX_STACKS];
int n, i, item;
```

```
void push()
{
    if (top[i] == boundary[i+1])
    {
        printf("stack %d is full\n", i);
        return;
    }
    top[i]++;
    s[top[i]] = item;
}

int pop()
{
    if (top[i] == boundary[i])
    {
        return -1;          // stack is empty
    }
    return s[top[i]--];
}
```

```
void display()
{
    int j;
    if (top[i] == boundary[i])
    {
        printf("Stack %d is empty\n", i);
        return;
    }
    printf("Contents of stack are:\n");
    for (j = boundary[i] + 1; j <= top[i]; j++)
    {
        printf("%d\n", S[j]);
    }
}

void main()
{
    int choice, j;
    printf("Enter number of queues\n");
    scanf("%d", &n);
    for (j = 0; j <= n; j++)
    {
        boundary[j] = top[j] = MAX_SIZE/n * j - 1;
    }
```

```
for(;;)
{
    printf("stack number:");
    for(j=0;j<n;j++)
    printf("%d",j);
    printf("Enter stack number:");
    scanf("%d",&i);
    printf("1.PUSH\n 2:POP\n 3. Display\n 4. Exit
                                        \n");
    printf("Enter the choice\n");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1: printf("Enter the item to be Inserted\n");
                scanf("%d",&item);
                push();
                break;
        case 2: item = pop();
                if(item==-1)
                    printf("stack is empty\n");
                else
                    printf("Item deleted =%d\n",item);
                break;
        case 3: display();
                break;
        default: exit(0);
    }
}
```

④ Define Recursion? & Write a C recursion function for
(i) Tower of Honoi
(ii) Factorial of a given number.

→ Recursion is the process in which a function calls itself up to n-number of times.

(i) Tower of Honoi :

* C recursive function.

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

void tower (int n, int source, int temp, int destination)
{
    if (n==0)
    return;
    tower (n-1, source, destination, temp);
    printf ("\n Move disc %d from %c to %c",
                    n, source, destination);
    tower (n-1, temp, source, destination);
}

void main()
{
   int n, res;
   printf ("\n Enter the number of discs:\n");
   scanf ("%d", &n);
   tower (n, 'A', 'B', 'C');
   res = pow(2, n)-1;
```

```
    printf ("\n total number of moves are %d",
                                    res);

}
```

(ii) Factorial of a given number: C recursive function.

```
# include <stdio.h>
void find_factorial (int);
void main()
{
    int num, fact;
    printf ("Enter a number to find factorial\n");
    scanf ("%d", &num);
    fact = find_factorial (num);
    printf ("Factorial of %d is : %d \n", num, fact);
    return;
}

void find_factorial (int)
{
    if (num == 0)
    return (1);

    return (n* find_factorial (n-1));
}
```

⑤ Properties of Recursive Procedure list out it?

(or)

What are the properties of Recursive Procedure.

→ A recursive function can go infinite like a loop. To avoid infinite running of recursive function, there are two properties that a recursive function must have -

Base Case:

1) Base criteria - There must be at least one base criteria or condition, such that, when this condition is met the function stops calling itself recursively.

General Case:

2) Progressive approach - The recursive calls should progress in such a way that each time a recursive call is made it comes closer to the base criteria.

⑥ Differentiate recursion & iteration process.

Recursion	Iteration.
1. The statement in a body of function calls the function itself	1. Allows the set of instructions to be repeatedly executed.
2. In recursive function only termination condition (base case) is specified.	2. Iteration includes initialization, condition, execution of statement within loop and update the control variable.

3. A conditional statement is included in the body of the function to force the function to return without recursion call being executed.	3. The iteration statement is repeatedly executed until a certain condition is reached.
4. Infinite recursion can crash the system	4. Infinite loop uses CPU cycles repetatedly.
5. Recursion is always applied to functions.	5. Iteration is applied to iteration statements or "loops"
6. The stack is used to store the set of new local variable & parameters each time the function is called.	6. Does not uses stack here.
7. Slow in execution.	7. Fast in execution.
8. Recursion reduces the size of the code.	8. Iteration makes the code longer.
9. Recursion reduces the size of the code.	9. Iteration makes the code longer
10. If the function does not converge to some condtion called (base cas), it leads to infinite recursion.	10. If the control condition in the iteration statement never become false, it leads to infinite iteration.

(7) Define Ackerman's function and find the value of A(1,3) using Ackerman's function

→ Ackerman's function is an example of total computable function and that is also called as non-premitive recursive function It's a function with two arguments each of which can be assinged any non-negative integer. It is in the form A(m,n)
m,n → two arguements.

* Ackermann function is defined as :

$$A(m,n) = \begin{cases} n+1 & \text{if } m=0 \\ A(m-1,1) & \text{if } m>0 \text{ and } n=0 \\ A(m-1, A(m,n-1)) & \text{if } m>0 \text{ and } n>0 \text{ or } m,n \neq 0. \end{cases}$$

→ C program for this function

```
# include < stdio.h>
int A( int m, int n)
{
    if (m==0)    return n+1 ;          // case 1
    if (n==0)    return A(m-1, 1);     // case 2
    return A (m-1 , A(m, n-1));        // case 3
}
```

```
void main()
{
    int m, n, res;
    printf("Enter m and n \n");
    scanf("%d %d", &m, &n);

    res = A(m, n);

    printf("A (%d, %d) = %d \n", m, n, res);
}
```

→ To find value of A(1, 3) using 3 conditions

1) A(1, 3) = A(0, A(1, 2))
= A(0, A(0, A(1, 1)))
= A(0, A(0, A(0, A(1, 0))))
= A(0, A(0, A(0, A(0, 1))))
= A(0, A(0, A(0, 2)))
= A(0, A(0, 3))
= A(0, 4)
= 5.

2) A(1, 2) = A(0, A(1, 1))
= A(0, A(0, A(1, 0)))
= A(0, A(0, A(0, 1)))
= A(0, A(0, 2))
= A(0, 3)
= 4

8]

* GCD - Greatest Commun Divisor of Two numbers.

The GCD of two given numbers is the largest integer that divides both of them. GCD of two numbers is defined only for positive integers but, not defined for negative integers and floating point numbers.

```
#include <stdio.h>
int gcd(int m, int n)
{
        if (n==0)
            return m;
        return gcd(n, m%n);
}
void main()
{
        int m,n,res;
        printf("Enter the value of m and n \n");
        scanf("%d %d", &m, &n);
        res = gcd(m,n);
        printf("gcd(%d,%d) = %d \n", m,n, res);
}
```

*(9) CIRCULAR QUEUE Operations.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 5

int front=0,rear=-1,count=0;
char cq[MAX],ele;

void insert();
void display();
void delete();

void main()
{
    int choice;
    while(1)
    {
        printf("\n\t 1.Insert an element
        on to CIRCULAR QUEUE\n\t
        2. Delete an element from
        CIRCULAR QUEUE\n\t & 3. DISPLAY
        the status of CIRCULAR QUEUE\n\t
        4. EXIT\n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: insert(); break;
            case 2: delete(); break;
            case 3: display(); break;
```

```
                case 4 : exit(0);
                default : printf("enter a
                                valid choice\n");
            }
        }
    }
void insert()
{
    if(count == MAX)
    {
        printf("Circular Queue is full, elements
        can not be inserted \n");
        return;
    }
    rear = (rear+1) % MAX;
    printf("\n Enter the element to be inserted
    into the circular Queue \n");
    ele = getche();
    cq[rear] = ele;
    count++;
}.
void delete()
{
    if(count == 0)
    {
        printf("Circular Queue is empty, no
        elements to delete \n");
        return;
    }
```

```
        ele = cq[front];
        front = (front+1)% MAX;
        printf("The element deleted is %c\n",
                                ele);
        count--;
}

void display()
{
        int i;
        if(count==0)
        {
            printf("CIRCULAR QUEUE is empty,
            no element to display\n");
            return;
        }.
        printf("Circular Queue contents
                are\n");
        for(i=front; i!=rear; i=(i+1)%MAX)
        {
                printf("%c\t", cq[i]);
        }
        printf("%c", cq[i]);

}.
```

(10) * What are the advantages of Circular Queue over ordinary Queue.

→

- As the insertion in the queue is from the rear end and in the case of linear Queue of fixed size insertion is not possible when rear reaches the end of the queue.
- But in the case of Circular Queue, the rear end moves from the last position to the front position circularly.
- In the circular queue, elements can be inserted easily if there are vacant locations until it is not fully occupied, whereas in the case of a ordinary queue insertion is not possible once if there are empty locations present in the queue.
- In the circular queue, there is no wastage of memory as it uses the unoccupied space, and memory is used properly in a valuable and effective manner as compared to a ordinary queue.
- In Ordinary Queue, FIFO is followed, so the element inserted at first is the element to be deleted first. This is not the scenario in the case of circular queue as the rear and front are not fixed

So the order of insertion-deletion can be changed, which is very useful.

11. * C; Recursive Program for:

- Sum of n odd numbers.

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

int sum(int n)
{
    if(n<=0)
      return 0;
    if(n%2==0)
    {
      return sum(n-1);
    }
    return n + sum(n-2);
}
void main()
{
    int a, s;
    printf("Enter the number: \n");
    scanf("%d", &a);
    s = sum(a);
    printf("%d", s);
}
```

(12) Sum of n even numbers

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

int sum(int n)
{
    if (n<=0)
        return 0;
    if(n%2==1)
    {   return sum(n-1);
    }
    return n + sum(n-2);
}
void main()
{
    int a, s;
    printf("Enter the number:\n");
    scanf("%d", &a);
    s=sum(a);
    printf("%d", s);
}.
```

* C Function to find n^{th} Fibanacci series:

→ C program to find n^{th} Fibanacci number

```
#include<stdio.h>
int F(int n)
{
   if(n==0)    return 0; /* Base case: 1st number */
   if(n==1)    return 1; /* Base case: 2nd number */
   return F(n-1)+F(n+2);  /* General case: computation
                             of */
                          /* subsequent Fibanacci
                             numbers */
}
void main()
{
    int n;
    printf("Enter n \n");
    scanf("%d", &n);
    printf("fib(%d) = %d \n", n, F(n));
}
```

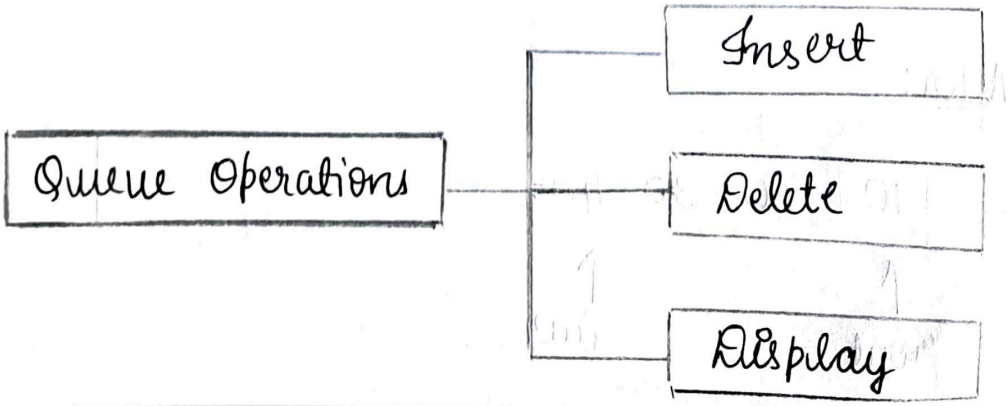
Queues

① Define queues. Write a C Program and algorithm for inserting, deleting and displaying an element from a queue.

A Queue is a special type of non primitive linear data structure where elements are inserted from one end and elements are deleted from the other end. The element at which new elements are added is called the rear and the end from which elements are deleted is called front.

They are also called FIFO [First in first out] data structures.

Various operations that can be performed on queues are shown below



i) Insert

It is the process of inserting an element at the rear end of the queue.

Algorithm:

Step 1: Check if queue is full.

Step 2: If the queue is full then print as "Queue overflows" and return.

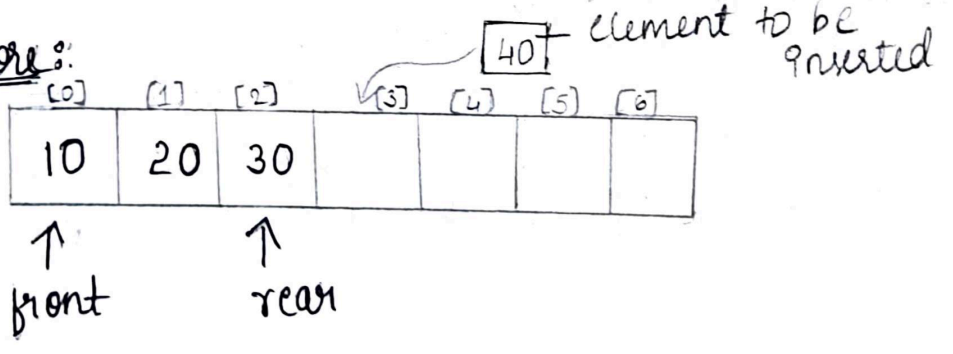
Step 3: If the queue is not full, increment rear pointer to point the next empty space.

Step 4: Add data element to the queue location, where the rear is pointing.

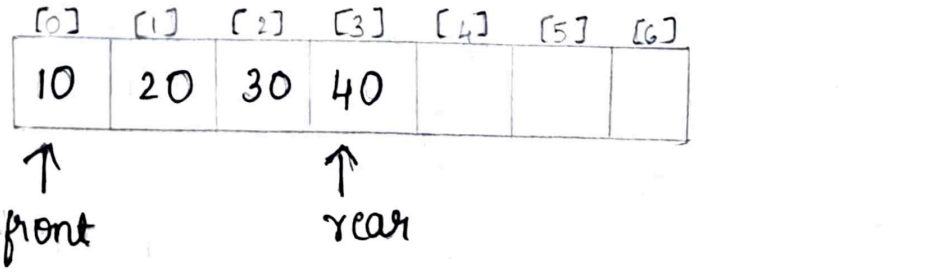
Step 5: Return from the C function

Diagramatic Representation:

Before:



After:



ii) Delete

It is a process of deleting an element from the front end of the queue.

Algorithm:

Step 1 : Check if queue is empty.

Step 2 : If the queue is empty, print as "Queue is underflow" and return

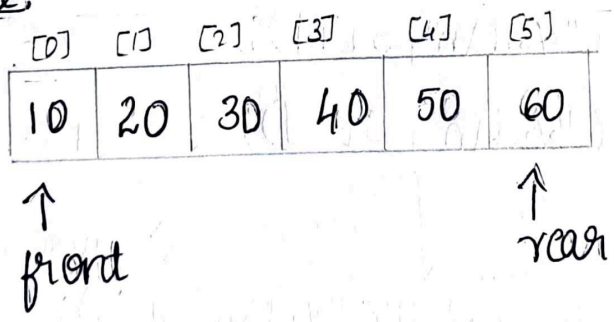
Step 3 : If the queue is not empty, access the data where front is pointing.

Step 4 : Increment front pointer to point to the next available data element.

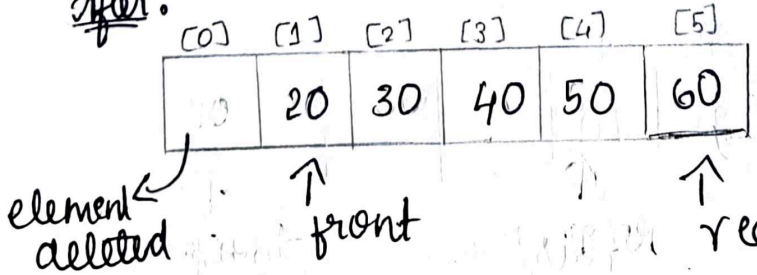
Step 5 : Return from the C function.

Diagramatic Representation

Before:



After:



iii) Display

It is the process of displaying the elements present in the queue.

Algorithm:

Step 1: Check if queue is empty.

Step 2: If the queue is empty then print as "Queue is empty" and return

Step 3: If queue is not empty then dislay the elements present in queue.

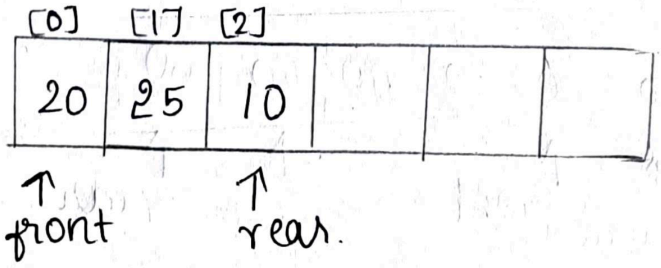
Step 4: Return from the C function

Diagramatic Representation:

```
printf("%d\n", s[0]);        Output
printf("%d\n", s[1]);          20
printf("%d\n", s[2]);          25
                               10
```

note: i= front to rear

In general, printf("%d\n", s[i]);



→ Program:

```
#Include <stdio.h>
# Include <stdlib.h>
# define SIZE 5
Int choice, ele, r = -1, f = 0, Q[SIZE];

void Insert();
void delete();
void display();

void main()
{
    while(1)
    {
        printf(" 1. Insert \n 2. delete \n 3. display \n
                                    4. exit \n");
        printf(" Enter your choice \n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1 : Insert();
                     break;
            case 2 : delete();
                     break;
```

```
                case 3 : display ();
                         break;
                case 4 : exit (0);
                default: printf ("Enter valid choice\n");
            }
        }
}
void insert ()
{
    if ( r == SIZE -1)
    {
        printf ("queue overflow\n");
        return ;
    }
    printf (" Enter the element to insert \n");
    scanf ("%d, & ele);
    q [++r] = ele ;
    return ;
}
void delete()
{
    if (f > r)
    {
```

```
            printf ("Queue is Underflow\n");
            return ;
        }
        ele = q[f];
        printf ("Element deleted is : %d\n", ele);
        f++;
        return;
}
void display()
{
        if (f > r)
        {
            printf ("Queue is Empty\n");
            return;
        }
        for (i = f; i <= r, i++)
            printf ("%d \t", q[i];
        return;
}
```

② List the different types of queues

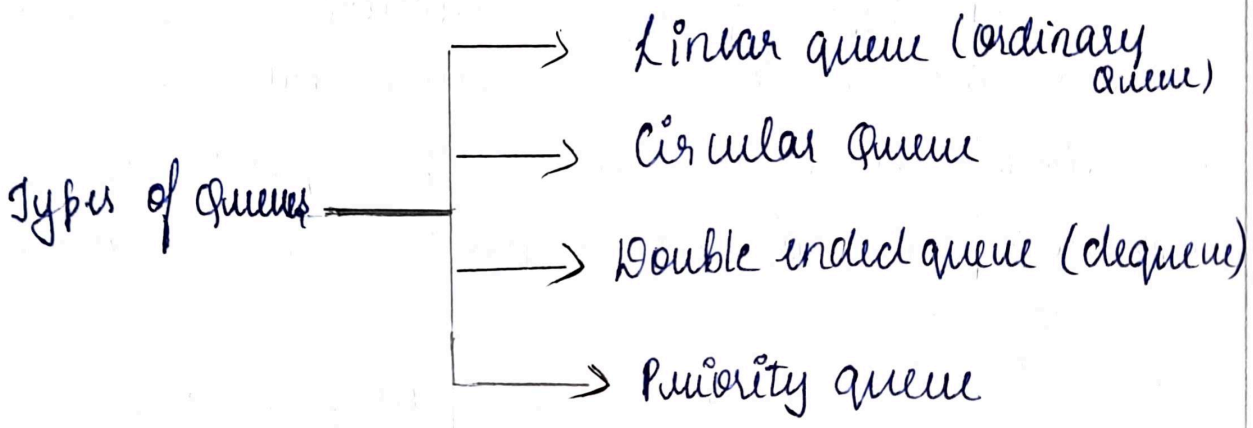
(Aug/Sep 2020)

(OR)

Write a note on Dequeue and priority queue along with their operations

(Dec/Jan 2019-20,
Jan/Feb 2021
Dec/Jan 2018-19)

Based on the methods of Insertion and deletions the queues are classified as shown below



① Linear queue

An ordered collection of items where elements are inserted from one end and elements are deleted from the other end. namely rear and front respectively.

Operation on Linear queue:

Refer question no ① for the operation.

② Circular Queue

In circular queue, the elements of a given queue can be stored efficiently in an array so as to "wrap around" so that rear end of the queue is followed by the front queue.

Operation of Circular Queue

Refer Question no ⑨ for the operation

③ Double Ended Queue [Dequeue]

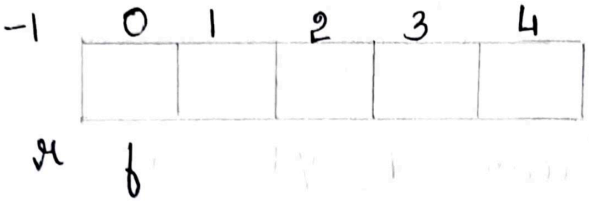
A type of Data structure in which insertion are done from both ends and deletions are done at both ends. The operations that can be performed on dequeues are shown below.

Operations performed on dequeues

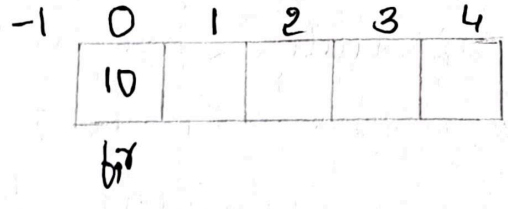
- Insert an item from front end
- Insert an item from rear end
- delete an item from front end
- delete an item from rear end
- display the contents of queue

Diagramatic representation

① When queue is empty

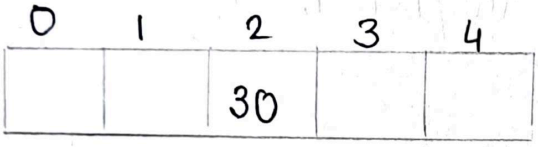


a) Before insert

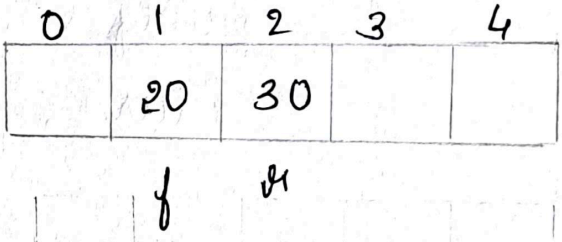


b) After insert.

② When some items are deleted

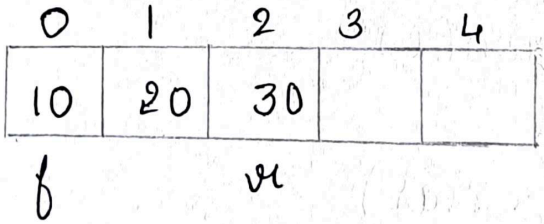


a) After deleting 2 items



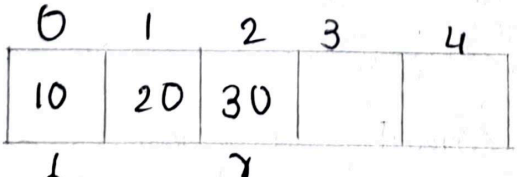
b) After inserting 20

③ Some items are inserted

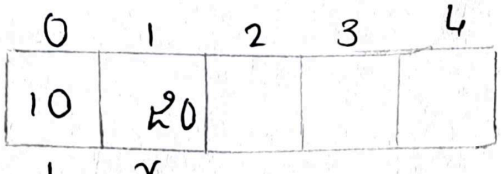


a) queue after inserting 10, 20, 30

④ Delete from the rear end



a) After inserting 3 items



b) After deleting from rear

C Program to implement double ended queue

```
# include <stdio.h>
# include <process.h>
# define QUESIZE 5
int choice, item, front=0, rear=-1, q[10];

void Insert_Rear()
{
    if (rear == QUE_SIZE -1)
    {
        printf ("Queue Overflow\n");
        return;
    }
    rear = rear + 1;
    q[rear] = item;
}

int Delete_Front()
{
    if (front > rear)
    {
    return -1;
    }
    return q[front++];
}
```

```
void Insert_Front()
{
    if (front == 0 && rear == -1)
    {
        q[++rear] = item;
        return;
    }
    if (front != 0)
    {
        q[--front] = item;
        return;
    }
    printf("Front insertion not possible\n");
}

int Delete_Rear()
{
    int item;
    if (front > rear)
        return -1;
    item = q[rear--];
    if (front > rear)
        front = 0, rear = -1;
    return item;
}
```

```
void Display()
{
    int i;
    if (front > rear)
    {
        printf ("Queue is empty\n");
        return;
    }
    printf ("contents of queues are : \n");
    for (i = front ; i <= rear ; i++)
    {
        printf ("%d \n", q[i]);
    }
}

void main()
{
    int choice, item, front, rear, q[10];
    front = 0;
    rear = -1;
    for (;;)
    {
        printf ("1: Insert_Front\n2 : Insert_rear\n
                 3 : Delete_Front\n 4 : Delete_rear\n");
        printf (" 5. Display\n 6 : Exit\n");
        printf ("Enter the choice\n");
        scanf ("%d", &choice);
```

```
switch(choice)
{
    case 1: printf("Enter the item to be inserted\n");
            scanf("%d", &item);
            Insert_Front();
            break;
    case 2: printf("Enter the item to be
                                inserted\n");
            scanf("%d", &item);
            Insert_Rear();
            break;
    case 3: item = Delete_Front();
            if(item == -1)
                printf("Queue is empty\n");
            else
                printf("Item deleted is %d\n",
                                item);
            break;
    case 4: item = Delete_Rear();
            if(item == -1)
                printf("Queue is empty\n");
            else
                printf("Item deleted is %d\n",
                                item);
            break;
    case 5: Display();
            break;
    default: exit(0);
}
}
```

(4) Priority Queue

A queue in which we are able to insert items or remove items from any position based on some priority is often referred to as a priority queue.

Always an element with highest priority is processed before processing any of the lower priority elements. If the elements in the queue are of same priority then the element, which is inserted first into the queue is processed.

Priority Queue is further classified into

(i) Ascending priority queue

In an ascending priority queue. elements can be inserted in any order. But, while deleting an element from the queue, only the smallest element is removed first.

(ii) Descending priority queue.

In descending priority also elements can be inserted in any order. But, while deleting an element from the queue, only the largest element is deleted first.

C Program to implement priority Queue.

```
#include <stdio.h>
#include <process.h>
#define QUE_SIZE 5

int Delete_Front()
{
    if (front > rear)
        return -1;
    return q[front++];
}
void display()
{
    int i;
    if (front > rear)
    {
        printf("Queue is empty\n");
        return;
    }
    printf("contents of queues are :\n");
    for (i= front; i<= rear; i++)
    {
        printf("%d\n", q[i]);
    }
}
```

```
void Insert_Item (int item, int q[], int *r)
{
    int j;
    if (*r == Que_SIZE-1)
    {
        printf ("Q is full\n");
        return;
    }
    j = *r;
    while (j >=0 && item <q[j])
    {
        q[j+1] = q[j];
        j = j-1;
    }
    q[j+1] = item;
    *r = *r+1;
}

void main()
{
    int choice, item, front, rear, q[10];
    front = 0; rear = -1;
    for (;;)
    {
        printf ("1: Insert\n 2: Delete\n 3: Display
                                    \n Exit\n");
        printf ("Enter your choice:");
```

```
scanf("%d", &choice);
switch (choice)
{
    case 1 : printf ("Enter the item to be inserted");
             scanf ("%d", &item);
             Insert_Item (item, qv, &rear);
             break;
    case 2 : item = Delete_Front (&front,
                                  &rear, qv)
             if ( item == -1)
                 printf ("Queue is empty\n");
             else
                 printf (" Deleted Item = %d\n",
                                  item);
             break;
    case 3 : Display ( front, rear, qv);
             break;
    default : exit (0);
}
}
}
```

(3) Write a C function to double the circular Queue capacity dynamically.

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

int *CQ;
int SIZE = 1;
int r = -1, f = 0;
int count = 0;
char ele;

void insert()
{
    if (count == SIZE)
    {
        printf("Queue overflow\n");
        return;
    }
    SIZE = SIZE*2;
    CQ = (int*) realloc (CQ, SIZE * sizeof(int));
    r = (r+1)%SIZE;
    printf("Enter the element into circular Queue\n");
    scanf("%c", &ele);
    CQ[r] = ele;
    count++;
}
```

```
void delete()
{
   if (count = 0)
   {
      printf ("Queue underflow\n");
      return ;
   }
   ele = CQ[f];
   printf ("Element deleted is %c\n", ele);
   f = (f+1)% SIZE ;
   SIZE = SIZE-1;
   CQ=(int*) realloc (CQ,SIZE * sizeof(int));
   count--;
   return;
}

void display()
{
   int i;
   if (count == 0)
   {
      printf ("Circular Queue is Empty\n");
      return ;
   }
   printf ("Circular Queue elements are\n");
```

```
        for (i=f; i!=r; i=(i+1)%SIZE)
        {
            printf("%c\n", CQ[i]);
        }
        printf("%c\n", CQ[i]);
    }

void main()
{
    CQ = (int*) malloc (SIZE * sizeof(int));
    while (1)
    {
        int choice;
        printf("1. insert\n 2. delete\n 3. display\n
                                        4. exit\n");
        printf("Enter your choice\n");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1 : insert();
                     break;
            case 2 : delete();
                     break;
```

```
            case 3 : display();
                     break;
            case 4 : exit(0);
            default: printf("Enter valid choice\n");

        }

    }

}
```

→ (4) Evaluation of Suffix expression with single digit operands and operators: +, −, *, /,

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

int i, top = - 1
mt op1, op2, res , s[20];
char postfix[50], symbol;

void push (int item)
{
  top = top + 1 ;
  s[top] = item;
}

int pop ()
{
  mt item ;
  item = s[top];
  top = top - 1;
  return item;
}

Void main()
{
  printf ("\n Enter a valid postfix expression:\n");
  scanf ("%s", postfix);
```

Decimal	Character
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

```
for (i=0 ; postfix[i] != '\0' ; i++)
{
    symbol = postfix[i]
    if (isdigit ( symbol))
    {
      push (symbol - '0');
    }
    else
    {
      op2 = pop();
      op1 = pop();

  switch (symbol)
  {
case '+' : push (op1 + op2);    break;
case '-' : push (op1 - op2);    break;
case '*' : push (op1 * op2);    break;
case '/' : push (op1 / op2);    break;
case '%' : push (op1 % op2);    break;
case '$':
case '^' : push(pow (op1, op2)); break;
default : printf ("Invalid operator \n") exit (0);
  }
 }
}
res = pop();
printf ("\n Result = %d", res)
;
}
```

⑤ Write an algorithm to evaluate postfix expression.

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

int input(char symbol)
{
      switch(symbol)
      {
          case '+':
          case '-': return 1;
          case '*':
          case '/': return 3;
          case '^':
          case '$': return 6;
          case '(': return 9;
          case ')': return 0;
          default: return 7;
      }
}
int stack(char symbol)
{
     switch(symbol)
     {
          case '+':
          case '-': return 2;
          case '*':
          case '/': return 4;
```

```
        case '^':
        case '$': return 5;
        case '(': return 0;
        case '#': return -1;
        default: return 8;
    }
}
infixtopostfix(char infix[], char postfix[])
{
    int top=-1, i, J=0;
    char S[30], symbol;
    S[++top] = '#';
    for(i=0; i<strlen(infix); i++)
    {
        symbol= infix[i];
        while(stack(S[top]) > input(symbol))
        {
            postfix[J++] = S[top--];
        }
        if(stack(S[top]) != input(symbol))
            S[++top] = symbol;
        else
            top++;
    }
```

```
        while (s[top] != '#')
        {
            postfix[j++] = s[top--];
        }
        postfix[j] = '\0';
        return;

}

void main()
{
    char infix[20], postfix[20];
    printf("Enter the infix expression \n");
    scanf("%s", infix);
    infixtopostfix(infix, postfix);
    printf("Postfix expression is: %s \n",
                postfix);

}
```

Convert the following expression to Postfix expression

① (A + B * C) * ((D + E - F) / J) [Aug/Sept 2020]

symb-ol	s(top)	stack[s(top)] input(symbol)	Action	Stack	Postfix
#			PUSH #	#	
(	#	-1 < 9	PUSH (	# (	
A	(	0 < 7	PUSH A	# (A	
+	A	8 > 1	POP A	# (	A
+	(	0 < 1	PUSH +	# (+	A
B	+	2 < 7	PUSH B	# (+ B	A
*	B	8 > 3	POP B	# (+	A B
*	+	2 < 3	PUSH *	# (+ *	A B
C	*	4 < 7	PUSH C	# (+ * C	A B
)	C	8 > 0	POP C	# (+ * C	A B C
)	*	4 > 0	POP *	# (+	A B C *
)	+	2 > 0	POP +	# (	A B C * +
)	(	0 == 0	POP (	#	A B C * +
*	#	-1 < 3	PUSH *	# *	A B C * +
(	*	4 < 9	PUSH (	# * (	A B C * +
(	(	0 < 9	PUSH (	# * ((	A B C * +
D	(	0 < 7	PUSH D	# * ((D	A B C * +
+	D	8 > 1	POP D	# * ((D	A B C * + D
+	(	0 < 1	PUSH +	# * ((D +	A B C * + D

E	+	2 < 7	PUSH E	# * ((+ E	A B C * + D
-	E	8 > 1	POP E	# * ((+	A B C * + D E
-	+	2 > 1	POP +	# * ((	A B C * + D E +
-	(	0 < 1	PUSH -	# * ((-	A B C * + D E +
F	-	2 < 7	PUSH F	# * ((- F	A B C * + D E +
)	F	8 > 0	POP F	# * ((-	A B C * + D E + F
)	-	2 > 0	pop -	# * ((	A B C * + D E + F -
)	(	0 = 0	POP (	# * (	A B C * + D E + F -
/	(	0 < 3	PUSH /	# * (/	A B C * + D E + F -
J	/	4 < 7	PUSH J	# * (/ J	A B C * + D E + F -
)	J	8 > 0	POP J	# * (/	A B C * + D E + F - J
)	/	4 > 0	POP /	# * (	A B C * + D E + F - J /
)	(	0 = 0	POP (	# *	A B C * + D E + F - J /
'0'	*		POP *	#	A B C * + D E + F - J / *

The Postfix expression is A B C * + D E + F - J / *

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

[June/July 2019]

Symbol	s[top]	Stack(s(top)) Input (symbol)	Action	Stack	Postfix
#			PUSH #	#	
(	#	-1 < 9	PUSH (	#(	
(	(	0 < 9	PUSH (	#((	
a	(	0 < 7	PUSH a	#((a	
/	a	8 > 3	pop a	#((	a
/	(	0 < 3	PUSH /	#((/	a
(	/	4 < 9	PUSH (	#((/(	a
b	(	0 < 7	PUSH b	#((/(b	a
-	b	8 > 1	POP b	#((/(	ab
-	(	0 < 1	PUSH -	#((/(-	ab
G	-	2 < 7	PUSH G	#((/(-G	ab
+	G	8 > 1	POP G	#((/(-	abG
+	-	2 > 1	POP -	#((/(	abG-
+	(	0 < 1	PUSH +	#((/(+	abG-
d	+	2 < 7	PUSH d	#((/(+d	abG-
)	d	8 > 0	POP d	#((/(+	abG-d
)	+	2 > 0	POP +	#((/(	abG-d+
)	(	0 == 0	POP (	#((/	abG-d+
)	/	4 > 0	POP /	#((	abG-d+/
)	(	0 == 0	POP (	#(	abG-d+/

*	(	0<3	PUSH *	# (*	abc-d+/
(	*	4<9	PUSH (	# (*(	abc-d+/
e	(	0<7	PUSH e	#(*(e	abc-d+/
-	e	8>1	POP e	#(*(	abc-d+/e
-	(	0<1	PUSH -	#(*(-	abc-d+/e
f	-	2<7	PUSH f	#(*(-f	abc-d+/e
)	f	8>0	POP f	#(*(-	abc-d+/ef
)	-	2>0	POP -	#(*(	abc-d+/ef-
)	(	0==0	POP (	#(*	abc-d+/ef-
*	*	4>3	POP *	#(	abc-d+/ef-*
*	(	0<3	PUSH *	#(*	abc-d+/ef-*
g	*	4<7	PUSH g	#(*g	abc
)	g	8>0	POP g	#(*	abc-d+/ef-*g
)	*	4>0	POP *	#(	abc-d+/ef-*g*
)	(	0==0	POP (	#	abc-d+/ef-*g*

The postfix expression is abc-d+/ef-*g*

③ (A + B ↑ D) / (E - F) + G [Jan / Feb . 2021]

symb-ol	s(top)	stack[s(top)] input(symbol)	Action	Stack	Postfix
#			push #	#	
(	#	-1 < 9	PUSH (	# (	
A	(	0 < 7	PUSH A	# (A	
+	A	8 > 1	POP A	# (	A
+	(	0 < 1	PUSH +	# (+	A
B	+	2 < 7	PUSH B	# (+ B	A
-	B	8 > 1	POP B	# (+	A B
-	+	2 > 1	POP +	# (	A B +
-	(	0 < 1	PUSH -	# (-	A B +
^	-	2 < 6	PUSH ^	# (- ^	A B +
D	^	5 < 7	PUSH D	# (- ^ D	A B +
)	D	8 > 0	POP D	# (- ^	A B + D
)	^	5 > 0	POP ^	# (-	A B + D ^
)	-	2 > 0	POP -	# (	A B + D ^ -
)	(	0 == 0	POP (	#	A B + D ^ -
/	#	-1 < 3	PUSH /	# /	A B + D ^ -
(	/	4 < 9	PUSH (	# / (	A B + D ^ -
E	(	0 < 7	PUSH E	# / (E	A B + D ^ -
-	E	8 > 1	POP E	# / (	A B + D ^ - E
-	(	0 < 1	PUSH -	# / (-	A B + D ^ - E
F	-	2 < 7	PUSH F	# / (- F	A B + D ^ - E

)	F	8 > 0	POP F	# / (-	AB+DΛ-EF
)	-	2 > 0	POP -	# / (	AB+DΛ-EF-
)	(	0 = 0	POP (	# /	AB+DΛ-EF-
+	/	4 > 1	POP /	#	AB+DΛ-EF-/
+	#	-1 < 1	PUSH +	# +	AB+DΛ-EF-/
cr	+	2 < 7	PUSH cr	# + cr	AB+DΛ-EF-/
'\0'	cr		POP cr	# +	AB+DΛ-EF-/cr
'\0'	+		POP +	#	AB+DΛ-EF-/cr+

The Postfix expression is AB+DΛ-EF-/cr+

④ A+B*((-D/E$F)*C [AUC/sept.2020]

sym-bol	S(top)	stack[S(top)] input(symbol)	Action	Stack	Postfix
#			PUSH #	#	
A	#	-1 < 7	PUSH A	#A	
+	A	8 > 3	POP A	#	A
+	#	-1 < 1	PUSH +	#+	A
B	+	2 < 7	PUSH B	#+B	AB
*	B	8 > 3	POP B	#+	AB
*	+	2 < 3	PUSH *	#+*	AB
(	*	4 < 9	PUSH (	#+*(	AB
C	(	0 < 7	PUSH C	#+*(C	AB
-	C	8 > 1	POP C	#+*(	ABC
-	(	0 < 1	PUSH -	#+*(-	ABC
D	-	2 < 7	PUSH D	#+*(-D	ABC
/	D	8 > 3	POP D	#+*(-	ABCD
/	-	2 < 3	PUSH /	#+*(-/	ABCD
E	/	4 < 7	PUSH E	#+*(-/E	ABCD
$	E	8 > 6	POP E	#+*(-/	ABCDE
$	/	4 < 6	PUSH $	#+*(-/$	ABCDE
F	$	5 < 7	PUSH F	#+*(-/$F	ABCDE
)	F	8 > 0	POP F	#+*(-/$	ABCDEF
)	$	5 > 0	POP $	#+*(-/	ABCDEF$
)	/	4 > 0	POP /	#+*(-	ABCDEF$/

)	−	2 > 0	POP −	# + * (	A B C D E F $ / −
)	(	0 = = 0	POP (	# + *	A B C D E F $ / −
*	*	4 > 3	POP *	# +	A B C D E F $ / − *
*	+	2 < 3	PUSH *	# + *	A B C D E F $ / − *
↵	*	4 < 7	PUSH ↵	# + * ↵	A B C D E F $ / − *
'\0'	↵		POP ↵	# + *	A B C D E F $ / − * ↵
'\0'	*		POP *	# +	A B C D E F $ / − * ↵ *
'\0'	+		POP +	#	A B C D E F $ / − * ↵ * +

The Postfix expression is A B C D E F $ / − * ↵ * +

⑤ ((6+(3-2)*4) $ 5+7 [Aug/Sep 2020]

Symbol	S[top]	Stack[s(top] Input (Symbol)	Action	stack	Postfix
#			PUSH #	#	
(	#	-1<9	PUSH (	#(	
(	(	0<9	PUSH (	#((	
6	(	0<7	PUSH 6	#((6	
+	6	8>1	POP 6	#((	6
+	(	0<1	PUSH +	#((+	6
(	+	2<9	PUSH (	#((+(	6
3	(	0<7	PUSH 3	#((+(3	6
-	3	8>1	POP 3	#((+(	6 3
-	(	0<1	PUSH -	#((+(-	6 3
2	-	2<7	PUSH 2	#((+(-2	6 3
)	2	8>0	POP2	#((+(-	6 3 2
)	-	2>0	POP -	#((+(	6 3 2 -
)	(	0==0	POP(	#((+	6 3 2 -
*	+	2<3	PUSH*	#((+*	6 3 2 -
4	*	4<7	PUSH4	#((+*4	6 3 2 -
)	4	8>0	POP 4	#((+*	6 3 2 - 4
)	*	4>0	POP*	#((+	6 3 2 - 4 *
)	+	2>0	POP +	#((	6 3 2 - 4 * +
)	(	0==0	POP (	#(	6 3 2 - 4 * +
$	(	0<6	POP (	#(	6 3 2 - 4 * +

5	$	5 < ↑	PUSH 5	# ($5	6 32 - 4 * +
+	5	8 > 1	POP 5	# ($	6 32 - 4 * + 5
+	$	5 > 1	POP $	# (	6 32 - 4 * + 5 $
+	(	0 < 1	PUSH +	# (+	6 32 - 4 * + 5 $
↑	+	2 < ↑	PUSH ↑	# (+ ↑	6 32 - 4 * + 5 $
)	↑	8 > 0	POP ↑	# (+	6 32 - 4 * + 5 $ ↑
)	+	2 > 0	POP +	# (	6 32 - 4 * + 5 $ ↑ +
)	(	0 == 0	POP (	#	6 32 - 4 * + 5 $ ↑ +

The postfix expression is 6 32 - 4 * + 5 $ ↑ +

(a+b)*d+c/(f+g*h)+i.

Symbol	S[top]	Stack[S(top)] Infix[Symbol]	Action	Stack	Postfix
	#			#	
(	#	-1<9	PUSH c	#(	
a	(	0<7	PUSH a	#(a	
+	a	8>1	POP a	#(	a
+	(	0<1	PUSH +	#(+	a
b	+	2<7	PUSH b	#(+b	a
)	b	8>0	POP b	#(+	ab
)	+	2>0	POP +	#(	ab+
)	(	0==0	POP (	#	ab+
*	#	-1<3	PUSH *	#*	ab+
d	*	4<7	PUSH d	#*d	ab+
+	d	8>1	POP d	#*	ab+d
+	*	4>1	POP *	#	ab+d*
+	#	-1<1	PUSH +	#+	ab+d*
c	+	2<7	PUSH c	#+c	ab+d*
/	c	8>3	POP c	#+	ab+d*c
/	+	2<3	PUSH /	#+/	ab+d*c
(	/	4<9	PUSH (	#+/(	ab+d*c.
f	(	0<7	PUSH f	#+/(f	ab+d*c
+	f	8>1	POP f	#+/(	ab+d*cf
+	(	0<1	PUSH +	#+/(+	ab+d*cf
g	+	2<7	PUSH g	#+/(+g	ab+d*cf
+	g	8>1	POP g	#+/(+	ab+d*cfg
+	+	2>1	POP +	#+/(	ab+d*cfg+

+	(	0 < 1	PUSH +	#+/(+	ab+d*c-f+g+
h	+	2 < 7	PUSH h	#+/(+h	ab+d*c-f+g+
)	h	8 > 0	POP h	#+/(+	ab+d*c-f+g+h
)	+	2 > 0	POP +	#+/(	ab+d*c-f+g+h+
)	(	0 == 0	POP (	#+/	ab+d*c-f+g+h+
+	/	H > 1	POP /	#+	ab+d*c-f+g+h+/
+	+	2 > 1	POP +	#	ab+d*c-f+g+h+/+
+	#	-1 < 1	PUSH +	#+	ab+d*c-f+g+h+/+
i	+	2 < 7	PUSH i	#+i	ab+d*c-f+g+h+/+
\0	i		POP i	#+	ab+d*c-f+g+h+/+i
\0	+		POP +	#	ab+d*c-f+g+ h+/+i+

The Postfix expression is ab+d*c-f+g+h+/+i+

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

Symbol	S[top]	Stack[S(top)] infix(sym)	Action	Stack	Postfix
			PUSH #	#	
(	#	-1 < 9	PUSH (	#(	
a	(	0 < 7	PUSH a	#(a	
+	a	8 > 1	POP a	#(	a
+	(	0 < 1	PUSH +	#(+	a
b	+	2 < 7	PUSH b	#(+b	a
)	b	8 > 0	POP b	#(+	ab
)	+	2 > 0	POP +	#(	ab+
)	(	0 == 0	POP (	#	ab+
*	#	-1 < 3	PUSH *	#*	ab+
d	*	4 < 7	PUSH d	#*d	ab+
+	d	8 > 1	POP d	#*	ab+d
+	*	4 > 1	POP *	#	ab+d*
+	#	-1 < 1	PUSH +	#+	ab+d*
e	+	2 < 7	PUSH e	#+e	ab+d*
/	e	8 > 3	POP e	#+	ab+d*e
/	+	2 < 3	PUSH /	#+/	ab+d*e
(	/	4 < 9	PUSH (	#+/(	ab+d*e
f	(	0 < 7	PUSH f	#+/(f	ab+d*e
+	f	8 > 1	POP f	#+/(	ab+d*ef
+	(	0 < 1	PUSH +	#+/(+	ab+d*ef
a	+	2 < 7	PUSH a	#+/(+a	ab+d*ef
*	a	8 > 3	POP a	#+/(+	ab+d*efa
*	+	2 < 3	PUSH *	#+/(+*	ab+d*efa

d	*	4 < 7	PUSH d	#+/(+*d	ab+d*efa
)	d	8 > 0	POP d	#+/(+*	ab+d*efad
)	*	4 > 0	POP *	#+/(+	ab+d*efad*
)	+	2 > 0	POP +	#+/(	ab+d*efad*+
)	(	0 == 0	POP (	#+/	ab+d*efad*+
+	/	4 > 1	POP /	#+	ab+d*efad*+/
+	+	2 > 1	POP +	#	ab+d*efad*+/+
+	#	-1 < 1	PUSH +	#+	ab+d*efad*+/+
G	+	2 < 7	PUSH G	#+G	ab+d*efad*+/+
/0	G		POP G	#+	ab+d*efad*+/+G
/0	+		POP +	#	ab+d*efad*+/+G+

The Postfix expression is ab+d*efad*+/+G+

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

Symbol	S(top)	Stack[S(top)] infix(sym)	Action	stack	postfix
	#		PUSH #	#	
a	#	-1<7	PUSH a	# a	
*	a	8>3	pop a	#	a
*	#	-1<3	PUSH *	# *	a
(	*	4<9	PUSH (	# * (	a
b	(	0<7	PUSH b	# * (b	a
+	b	8>1	pop b	# * (	ab
+	(	0<1	PUSH +	# * (+	ab
c	+	2<7	PUSH c	# * (+ c	ab
)	c	8>0	pop c	# * (+	abc
)	+	2>0	pop +	# * (	abc+
)	(	0==0	pop (	# *	abc+
/	*	4>3	pop *	#	abc+*
/	#	-1<3	PUSH /	# /	abc+*
d	/	4<7	PUSH a	# / d	abc+*
-	d	8>1	pop a	# /	abc+*d
-	/	4>1	pop /	#	abc+*d/
-	#	-1<1	PUSH -	# -	abc+*d/
e	-	2<7	PUSH e	# - e	abc+*d/
*	e	8>3	pop e	# -	abc+*d/e
*	-	2<3	PUSH *	# - *	abc+*d/e
f	*	4<7	PUSH f	# - * f	abc+*d/e.

^	f	8>6	POP f	# - *	abc + * d / ef
^	*	4 < 6	PUSH ^	# - * ^	abc + * d / ef
g	^	5 < 7	PUSH g	# - * ^ g	abc + * d / ef
'/0'	g		POP g	# - * ^	abc + * d / efg
'/0'	^		POP ^	# - *	abc + * d / efg ^
'/0'	*		POP *	# -	abc + * d / efg ^ *
'/0'	-		POP -	#	abc + * d / efg ^ * -

The postfix expression is abc+*d/efg^*-

Evaluate the below postfix expression

① 6 5 1 - 4 * 2 3 ^ / + [Aug/Sep 2020]

Postfix	Symbol	OP_2	OP_1	OP_1 sym OP_2	Stack
6 5 1 - 4 * 2 3 ^ / +	6				6
	5				6 5
	1				6 5 1
	-	1	5	5 - 1	6 4
	4				6 4 4
	*	4	4	4 * 4	6 16
	2				6 16 2
	3				6 16 2 3
	^	3	2	2 ^ 3	6 16 8
	/	8	16	16 / 8	6 2
	+	2	6	6 + 2	8

The evaluated answer of the above expression is 8 //

Evaluate the following postfix expression

1 2 3 + * 3 2 1 - + *

[Jan/Feb 2021]

Postfix	Symbol	OP_2	OP_1	OP1 sym OP2	Stack
1 2 3 + * 3 2 1 - + *	1				1
	2				1 2
	3				1 2 3
	+	3	2	2 + 3	1 5
	*	5	1	1 * 5	5
	3				5 3
	2				5 3 2
	1				5 3 2 1
	-	1	2	2 - 1	5 3 1
	+	1	3	3 + 1	5 4
	*	5	4	4 * 5	20

The evaluated answer of the above expression is 20

Evaluate the given Post fix Expression.

ABC −D*+E$F+

A=6, B=3, C=2, D=5, E=1, F=7.

Postfix	Symbol	OP 2	OP 1	OP_1 Sym OP_2	Stack
6 3 2 - 5 * + 1 ^ 7 +	6				6
	3				6 3
	2				6 3 2
	-	2	3	3 - 2 = 1	6 1
	5				6 1 5
	*	5	1	1 * 5 = 5	6 5
	+	5	6	6 + 5 = 11	11
	1				11 1
	^	1	11	11 ^ 1 = 11	11
	7				11 7
	+	11	7	7 + 11 = 18	18

The evaluation Answer of the above expression is 18

Evaluate the given postfix expression

546+*493/++.

Postfix	Symbol	OP2	OP1	OP1 Sym OP2	Stack
546+*493/++	5				5
	4				5 4
	6				5 4 6
	+	6	4	4+6 = 10	5 10
	*	10	5	10*5 = 50	50
	4				50 4
	9				50 4 9
	3				50 4 9 3
	/	3	9	9/3 = 3	50 4 3
	+	3	4	4+3 = 7	50 7
	+	7	50	50+7 = 57	57.

The evaluation Answer of the above expression is 57.