①
```
# Include <Stdio.h>
int main () {
  printf ("Name :- Atharva Patil
int marks [5];
int i , sum = 0;
float average ;
Printf (" Enter marks for 5 subjects :\n");
for (i=0 ; i<5 ; i++) {
Printf (" Subject %d:", i+1);
Scanf ("%d", & marks[i]);
sum += marks [i];
}
average = sum/5.0;
printf ("Average marks = %.2f\n", average);
return 0;
}
```

Output :-

```
Name :- Atharva Patil
Enter marks for 5 subjects;
  Subject 1 : 87
Subject 2 : 78
Subject 3 : 83
Subject 4 : 85
Subject 5 : 88

Average marks = 84.20
```

(2)

```
# Include <stdio.h>
int main () {
    printf ("Name :- Atharva Patil \n");
int number;
printf ("Enter a number :");
scanf ("%d", & number) ;
if (number %2 == 0) {
    printf ("%d is Even. \n", number);
} else {
    printf ("%d is Odd. \n", number);
}

return 0;
}
```

Output :- Name :- Atharva Patil
Enter a number :7
7 is odd.

③ # Include <stdio.h>

```
int main () {
    Printf ("Name :- Atharwa Patil\n");
int i, j;
for (i=2 ; i<=50 ; i++) {
  int is Prime =1;
for (j=2 ; j<i ; j++) {
  if (i% j==0) {
  is Prime =0;
  break;
  }
}
if (is Prime) {
  printf ("%d", i);
}
}
return 0;
}
```

Output :-

Name :- Atharwa Patil
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47

(4)

```
# include <stdio.h>
int main ()
{
int day;
Printf ("Day of the week & Type :\n\n");
for (day = 1; day <=7 ; day++)
{
switch (day)
{
case 1;
Printf ("Mon , Tue , Wed , Thurs , Fri - Weekday\n");
break;
case 2;
Printf ("Sat , Sun - weekends\n");
break;
}
}
return 0;
}
```

Output :- Day of the week & Type :

Mon , Tue , Wed , Thurs , Fri - Weekday
Sat , Sun - Weekends.

(5)

```
# include <stdio.h>
int main ()
{
  int n.o.x=0, r;
  Printf ("Enter an integer :");
  scanf ("%d", &n);
  o=n;
while(n!=0)
{
  r=n%10;
  x = x*10+r;
  n=n/10;
}
  if(o==x)
{
  printf ("%d is a palindrome.\n".o);
}
return0;
}
```

Output :- Enter an integer : 141
141 is a palindrome

Experiment: 1

*) find avg. of n numbers.

```
# include <stdio.h>
int main () {
  int n, i;
float sum = 0.0, avg, num;
printf ("Enter the number of elements :");
if (scanf ("%d", &n) != 1 || n <= 0) {
printf ("Invalid input! Number of elements must be a positive
        integer. \n");
  return 1;
}
for (i=0; i<n; i++) {
  printf ("Enter number %d:" i+1);
   if (scanf ("%of", &num) != 1) {
    printf ("Invalid input! Please enter a valid number. \n);
  return 1;
}
  sum += num;
printf ("The average = %.2f\n", avg);
  return 0;
}
```

Output:- Enter the value of n: 13
The average of first 13 numbers is : 7.00.

Page No.:
Date:

② Display the for loop pattern.

```
*
# #
* * *
# # # #
```

```
#include <stdio.h>
int main () {
int i, j ;
for (i=1 ; i<=4; i++) {
    for (j=1 ; j<=i ; j++) {
  if (i % 2 = 0) {
    printf ("*") ;
  }
  Else {
    printf ("#") ;
  }
  }
}
printf ("\n") ;
}
return 0;
}
```

(3) Find the greatest & smallest element in the array.

→

```
#include <stdio.h>
int main () {
    int arr [5] = {10, 4, 56, 3, 89};
    int i;
    int big = arr [0];
    int small = arr [0];
    for (i=1 ; i<5 ; i++) {
    if (arr [i] > big)
        big = arr [i];
    if (arr [i] < small)
        small = arr [i] ;
    }
    printf ("Greatest no. = %d\n", big);
    printf ("Smallest no. = %d\n", small);
    return 0;
}
```

Output :-

Greatest number = 89
Smallest number = 3

*) first first repeating number in an array.

→
```
# include <stdio.h>
int main ()
{
    int arr[6] = {2,4,3,5,7,4};
    int i, j ;
for(i=0 ; i<6 ; i++) {
    for(j= i+1 ; j<6 ; j++)
if (arr[i] == arr[j]) {
    Printf ("First repeating number is : %d|n", arr[i]);
    return 0;
}
}
}
Printf ("No repeating number found.|n");
return 0;
}
```

→ Output

First Reapeating number is : 4.

*) Squaring odd positions elements in an array.

→
```
# include <stdio.h>
int main () {
int arr [] = { 1,2,3,4,5,6,7 } ;
int n = sizeof (arr) / size of (arr [0]);
printf ("Original array:\n");
for int i=0; i<n ; i++ ){
    print ("%d", arr [i]) ;
}
for(int i= 1 ; i<n ; i += 2) {
    arr [i] = arr [i] * arr [i] :
}
Printf ("\n\n array after squarring elements at odd positions: \n");
for (int i=0 ; i<n ; i++) {
    printf ("%d", arr [i] );
}
return 0;
}
```

— Output

Original array:-
1 2 3 4 5 6 7

Array after squaring elements at odd positions:
1 4 3 16 5 36 7

Nyla
6/7/25

Practice

*) Find the following patterns.

i)→ 1
2 2
3 3 3
4 4 4 4
5 5 5 5 5

⇒
```
# include <stdio.h>
int main
{
  int i, j;
 for (i=0; i<=5; j++)
 {
   for (j=1; j<=i; j++) {
       printf ("%d", i);
 }
 return 0;
```

ii)

1
1 2
3 3 3
1 2 3 4
5 5 5 5 5

```
# Include <stdio.h>
int main ()
{
int i,j ;
for (i=1 ; i<= 5 ; i++)
{
if (i% 2 = = 0)
{
for (j=1 ; j<= i ; j++)
{
printf ("%d", ;);
}
}
else {
for (j=1 ; j<=i , j++)
{
printf ("%d" ;);
}
printf ("\n");
}
return 0;
}
```

iii) *
# #
$ $ $
? ? ? ?

```
#include <stdio.h>
int main ()
{
int i, j ;
for (i=0 ; i<=4; i++)
{
switch (i)
{
Case 1 :
printf ("*\n");
break;
Case 2:
Printf ("##\n");
break ;
Case 3:
printf ("$$$\n");
break;
default :
break ;
}
}
return 0;
}
```

iv)

```
   *
  * *
 * * *
* * * *
```

```
# include <stdio.h>
int main ()
{
  int i, j, space ;
  int rows = 4;
 for (i=1 ; i<= rows ; i++)
 {
  for (space =1; space <= rows i ; space ++)
 {
    printf (" ");
 }
  for (j= 1 ; j<=i ; j++) {
    printf ("*");
 }
 return 0;
}
```

v)

```
* * * *
 * * *
  * *
   *
```

```
#include <stdio.h>
int main () {
int i,j; space
int rows = 4;
for (i=0 ; i<rows ; i++) {
for (space=0 ; space <i ; space++) {
printf (" ");
}
printf (" ");
}
for (j=0 ; j<rows-i ; j++)
{
printf ("*");
}
for (j=0 ; j<rows -i ; j++) {
{
printf ("*");
}
printf("\n")
}
return 0;
}
```

Experiment: 2

*) ① Search data using linear search.
Consider the following list to perform linear search.
(56, 36, 89, 57, 01, 00, 67, 59)

```
# include <stdio.h>
int main () {
  int arr[] = {56, 36, 89, 57, 01, 67, 59};
  int n = sizeof (arr) / sizeof (arr[0]);
  int key, i, found = 0;
  printf ("Enter number to search :");
  scanf ("%d", &key);
for (i=0; i<n; i++) {
  if (arr[i] == key) {
printf ("Element found at position %d (index %d) \n", i+1, i);
  found = 1;
  break;
  }
}
if (!found) {
  printf ("Element not found in the array \n");
}
return 0;
}
```

Output: Enter number to search : 01
Element found in the array.

*) ② Search the item 55 from the above list & write the item is found or not with procedure.

```
→ # Include <stdio.h>
int main () {
  int arr [] = {56, 36, 89, 57, 01, 00, 67, 53};
  int n = size of (arr) / size of (arr [0]);
  int key, i, found = 0;
printf ("Enter number to search : ");
scanf ("%d", & key);
  for (i = 0; i<n ; i++) {
    if (arr[i] == key) {
      printf (" Element found at position : %d (index : %d)\n",
                                                  i+1, i);
      found = 1;
      break;
    }
  }
  if (! found) {
    Printf (" Element not found in the array.\n");
  }
return 0;
}
```

Output :- Enter number to search : 55
Element not found in the array.

3) Difference between linear search & binary search

→ Linear Search	Binary Search
1) Time complexity low	Time complexity is $O(\log n)$
2) Works on both sorted & unsorted.	Only works on sorted data.
3) Less efficient, especially for large data sets.	More efficient especially for large data sets.
4) code is simpler.	Code is more complex.
5) It uses sequential searching approach.	It uses divide & conquer approach.

4) State limitations of linear search interms of time complexity.

→ A linear search runs in an worst linear time & makes at most comparison where n is the length of the list.

Linear search is rarely practical because other search algorithm & schemes, such as the binary search algorithm, flashtable allow significantly faster searching for all but short list.

extra questions

1) Write a c program to copy the elements of one array into another array in a reverse order.

```
# include <stdio.h>
int main () {
int original [100], reversed [100] ;
int n, i;
Printf ("Enter the number of elements :");
scanf ("%d", & n );
printf (Enter %d elements :\n", n);
for (i=0 ; i<n ; i++) {
  Scanf ("%d", & original[i] );
}
for(i=0 ; i<n ; i++) {
  reversed [i] = original [n-1-i];
}
printf ("Array in reverse order :\n");
for(i=0 ; i<n ; i++) {
  Printf ("%d", reversed [i] );
}
return 0;
}
```

Output :- Enter the number of elements : 5
Enter ~~the~~ 5 elements :
10 20 30 40 50
Array in reversed order :
50 40 30 20 10

(2) Write a C Program to count total number of duplicate elements in an array

```
# include <stdio.h>
int main () {
  int a [100], i, j, n, count = 0;
printf (" Enter number of elements : ");
  scanf ("%d", & n);
Printf (" Enter %d elements : \n", n);
  for (i = 0 ; i<n; i++) {
  scanf ("%d", & a[i]);
}
  for (i=0 ; i<n ; i++){
    for (j=i+1; j<n; j++){
    if (a[i] == a[j]) {
    count ++ ;
    break ;
  }
  }
}
printf (" Total number of duplicate elements = %d\n", count);
return 0;
}
```

Output

```
Enter number of elements : 6
Enter 6 elements :
1 2 3 4 2 5
Total number of duplicate elements = 2.
```

③ WA C Program to print all unique elements in an array.

```
#include <stdio.h>
int main () {
int a [100], i, j, n, count ;
Printf ("Enter the number of elements :");
Scanf ("%d", &n) ;
Printf ("Enter %d elements :\n", n);
for (i =0 ; i<n ; i++) {
scanf ("%d", & a [i] );
}
Printf ("unique elements are :\n");
for (i =0; i<n ; i++) {
count =0;
for (j =0 ; j<n ; j++) {
if (a[i] == a [j] && i != j) {
count ++ ;
}
}
if (count == 0) {
Printf ("%d", a[i]);
}
}
return 0;
}
```

Output :- Enter the number of elements : 6
Enter 6 elements :
1 2 2 3 4 1
Unique elements are :
3 4

(4) WAP to ~~find~~ seperate odd & even intergers into separate arrays.

```
# include <stdio.h>
int main () {
int a[5], even[5], odd[5]
int i, e= 0, o=0
printf (" enter 5 elements :\n");
for (i=0; i<n; i++) {
scanf ("%d", & a[i] );
}
for (i=0; i<5; i++) {
if (a[i] % 2==0) {
even [e]=a [i]
e++;
} else {
odd[o]= a[i];
o++;
}
}
printf ("Even elements :");
for (i=0 ; i<e; i++) {
printf ("%d", odd[i]);
}
return 0;
}
```

Output:- Enter 5 elements : 1 2 3 4 5 :
Even elements : 24
Odd elements : 135

(5) WAP to find second smallest element in an array.

→
```
# include <stdio.h>
int main () {
int a [5] , i, min, second;
printf ("Enter 5 elements :\n");
for (i=0 ; i<5 ; i++) {
scanf ("%d", & a [i]);
}
min = a [0]
second = a [1];
if (second < min) {
int temp = min ;
min = second;
second = temp ;
}
for (i=2 ; i<n ; i++){
if (a [i] < min ) {
second = min ;
min = a[i] ;
} else if (a [i] < second && a[i] != min ) {
second = a [i] ;
}
}
printf ("Second smallest = %d\n", second );
return 0;
}
```

Output :- Enter 5 elements :
5 3 1 2 4
Second smallest = 2

(6) Q.AP to count the total no. of words in a string.

```
#include <stdio.h>
int main () {
char str [100] ;
int i , count = 1 ;
printf (" Enter a String : ");
fgets (str, sizeof (str), stdin);
for (i=0 ; str[i] != '\0' ; i++) {
if (str [i] == ' ')
count ++ ;
}
printf (" Total numbers of words = %d\n", Count) ;
return 0;
}
```

Output :- Enter a String : Atharva Patil
Total Number of words = 2.

Experiment : 3

1) Sort elements in ascending order using bubble sort.

⇒
```
# include <Stdio.h>
int main () {
    int arr [100], n, i, j, temp;
    printf ("Enter the no. of elements :");
    scanf ("%d", &n);
    Printf ("Enter %d elements :\n", n);
  for (i=0; i<n; i++) {
    scanf ("%d", & arr[i]);
  }
    for (i=0; i<n -1; i++) {
    for (j=0 ; j<n-1; j++) {
     if (arr[j] > arr [j+1]) {
       temp = arr[j];
       arr[j] = arr [j+1];
       arr [j+1] = temp;
     }
    }
  }
printf (" Sorted array in ascending order :\n");
  for (i=0 ; i<n ; i++) {
    printf ("%d", arr[i]);
  }
return 0;
}
```

Output :- Enter the no. of elements : 3
Enter 3 elements :
12 54 36
Sorted array in ascending order :
12 36 54

2) Sort elements in descending order using selection sort.

```
# include <stdio.h>
int main () {
    int a [100], n, i, j, temp;
    printf ("Enter no. of elements :");
    scanf ("%d", &n) ;
    Printf ("Enter %d elements :\n", n);
    for (i=0 ; i<n ; i++) {
        scanf ("%d", & a[i]);
    }
    for (i=0; i<n-1; i++) {
        for (j=i+1 ; j<n ; j++) {
            if (a[i] <a[j]) {
                temp = a[i] ;
                a[i] = a[j];
                a[j] = temp ;
            }
        }
    }
    printf ("Sorted array in descending order :\n");
    for (i=0 ; i<n ; i++) {
        printf ("%d", a[i]);
    }
    return 0;
}
```

Output :- Enter no. of elements : 3
Enter 3 elements :
37 49 21
Sorted array in descending order :
49 37 21

Extra

3) Find the no. of comparision required in bubble sort method of the following list having 5 no. (100, 200, 300, 400, 500)

```
# include <stdio.h>
int main () {
    int a[100], n, i, j, temp, count = 0;
    printf ("Enter number of elements :");
    scanf ("%d", &n);
    printf ("Enter %d elements :\n", n);
    for (i=0 ; i<n ; i++) {
        scanf ("%d", & a[i]);
    }
    for (i=0; i<n-1; i++){
        for (j=0 ; j<n-i-1 ; j++) {
            count ++ ;
            if (a[j] > a [j+1]) {
                temp = a[j];
                a[j] = a [j+1] ;
                a [j+1] = temp ;
            }
        }
    }
    printf ("Total number of comparision : %d\n", count);
    return 0;
}
```

Output:- Enter number of elements : 5
Enter 5 elements :
100 200 300 400 500
Total number of comparisions : 10.

extra

4) Sort the given array in ascending order using selection sort method & show diagramitic representation

```
#include <stdio.h>
int main () {
    int a [100], n, i, j, min, temp;
printf ("Enter number of elements :");
scanf ("%d", &n) ;
printf (" Enter %d elements :\n", n);
for (i=0 ; i<n ; i++) {
    scanf ("%d", & a[i]);
}
for (i=0 ; i<n-1 ; i++) {
  min = i;
for (j=i+1 ; j<n ; j++) {
  if (a[j] < a [min]) {
    min = j ;
  }
}
temp = a[i] ;
a[i] = a[min] ;
a [min] = temp ;
printf ("Iteration %d :", i+1);
for (j=0 ; j<n ; j++) {
    printf ("%d", a[j]) ;
}
  printf ("\n") ;
}
return 0 ;
}
```

output :-

Enter number of elements : 5
Enter 5 elements :
500 -20 30 14 50 30
Iteration 1 : -20 500 30 14 50
Iteration 2 : -20 14 30 500 50
Iteration 3 : -20 14 30 500 50
Iteration 4 : -20 14 30 50 500

3) Find the no. of comparison required in bubble sort method of the following list having 5 no. (100, 200, 300, 400, 500)

⇒ Given numbers - 100, 200, 300, 400, 500

100	200	300	400	500
100	200	300	400	500
100	200	300	400	500
100	200	300	400	500

1] Start with the list of numbers
2] Compare each pair of adjacent elements & swap if the first element is greater than second element. Continue comparing & sorting.
3] After each full pass the largest unsorted element moves to the end of array.
4] Repeat the process of 2 step on the unsorted array.
5] Stop when all elements are in the correct order (after n-1 passes).

4) Sort the given array in ascending order using selection sort method & show diagramitic representation of every iteration of for loop:
(500, -20, 30, 14, 50)

⇒ 5 elements : 500, -20, 30, 14, 50

Pass 1)

500 -20 30 14 50 ✓

-20 500 30 14 50 ✓

-20 500 30 14 50 ✗

-20 500 30 14 50 ✗

Pass 2)

-20 500 30 14 50 ✓

-20 30 500 14 50 ✓

-20 14 500 30 50 ✗

Pass 3)

-20 14 500 30 50 ✓

-20 14 30 500 50 ✗

Pass 4)

-20 14 30 500 30 ✓

-20 14 30 50 500

*) Algorithm :-

1) Start main to 0.
2) Search min. element of the array.
3) Swap value at the selection of main.
4) Increment min. to point the next element location.
5) Repeat till sorted
6) Stop.

29/7/25

Experiment : 4

Q.D→ Sort elements in ascending order using Insertion Sort?

⇒
```
# include <stdio.h>
int main ()
{
int a[100], n, temp, i, j, k;
printf ("Enter no. of elements in array :\n");
scanf ("%d", &n);
printf ("\n Enter elements of array :\n");
for (i=0 ; i<n ; i++)
{
scanf ("%d", & a[i]);
}
for (i=1 ; i<n ; i++)
{
for (j=0 ; j<i ; j++)
{
if [a[i] < a[j])
{
temp = a[i]
for (k=i-1 , k>=j ; k--)
{
a [k+1] = a [k];
}
a [j] = temp ;
}
}
printf ("\n Array after Pass %d:", p);
for (ii=0 ; ii<n ; ii++)
{
printf ("%d", a[ii]);
}
++p ;
}
```

```
printf("\n\n the sorted elements are :");
for (i=0; i<n; i++)
  {
    printf("\n%d", a[i]);
  }
return 0;
```

Output :-

Enter number of elements in array :
5
Enter elements of array :
9000
569
6
8589
3456

Array after	Pass	1 :	569	9000	6	8589	3456
"	"	"	2 :	6	569	9000	8589	3456
"	"	"	3 :	6	569	8589	9000	3456
"	"	"	4 :	6	569	3456	8589	9000

The sorted elements are :
6
569
3456
8589
9000

Q.2) What is the output of insertion sort after the 2nd iteration given the following sequence of no: 7,3,5,1,9,8,4,6

→ 7 3 5 9 1 8 4 6

1st iteration:- 7 3 1 9 4 8 6
3 7 1 9 4 8 6

2nd iteration:- 3 7 1 9 4 8 6
1 3 7 9 4 8 6

After 2nd iteration the result is:-
→ 1,3,7,9,4,8,6

Q.3 Sort the following numbers using Radix Sort :-

① 100 225 390 4130 956 99 5431

Pass 1)	0	1	2	3	4	5	6	7	8	9
0100	0100									
0225						0225				
0390	0390									
4130	4130									
956							0956			
0099										
5431		5431								0099

Pass 2)	0	1	2	3	4	5	6	7	8	9
0100	0100									
0390										0390
4130				4130						
5431				5431						
0225			0225							
0956						0956				
0099										0099

	0	1	2	3	4	5	6	7	8	9
Pass 3)		~~0100~~								
0100		0100								
0225			0225							
4130		4130								
5431					5431					
0956						~~0956~~				0956 ~~0956~~
0390 ~~0099~~				0390						~~0099~~
0099	0099									

Pass 4)	0	1	2	3	4	5	6	7	8	9
0099	0099									
0100	0100									
4130					4130					
0225	0225									
0390	0390									
5431	~~5431~~					5431				
0956	0956									

Sorted arr = [0099, 0100, 0225, 0390, 0956, 4130, 5431]

(2)

Pass 1	0	1	2	3	4	5	6	7	8	9
025						025				
006							006			
099										099
145						145				
239										239
020	020									
018									018	

Pass 2)	0	1	2	3	4	5	6	7	8	9
020			020							
025			025							
145					145					
006	006									
018		018								
099										099
239				239						

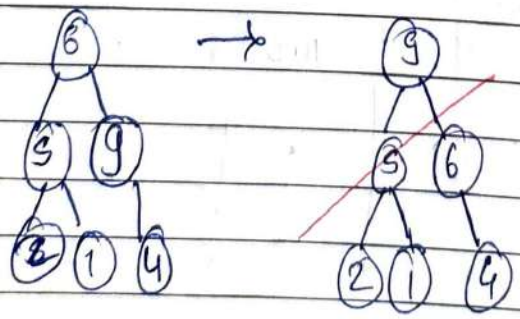
Pass 3)	0	1	2	3	4	5	6	7	8	9
006	006									
018	018									
020	020									
025	025									
239			239							
145		145								
099	099									

Sorted arr = [006, 018, 020, 025, 099, 145, 239]

Q.4) Sort the following elements using heap sort :-

① arr [6, 5, 9, 2, 1, 4]

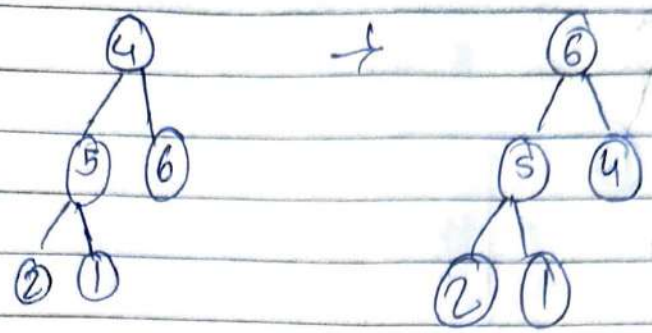
Pass 1



→ arr = [9, 5, 6, 2, 1, 4]

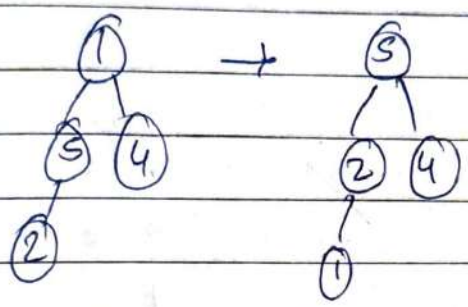
arr = [4, 5, 6, 2, 1, 9]

Pass 2)



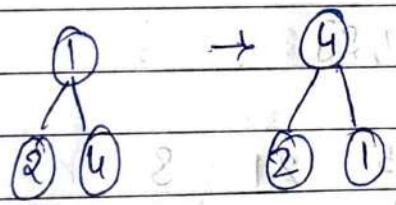
arr = [6, 5, 4, 2, 1, 9]
arr = [1, 5, 4, 2, 6, 9]

Pass 3)



arr = [5, 2, 4, 1, 6, 9]
arr = [1, 2, 4, 5, 6, 9]

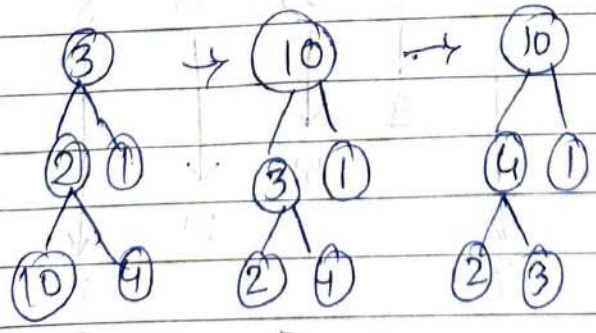
Pass 4)



arr = [4, 2, 1, 5, 6, 9]
arr = [1, 2, 4, 5, 6, 9]

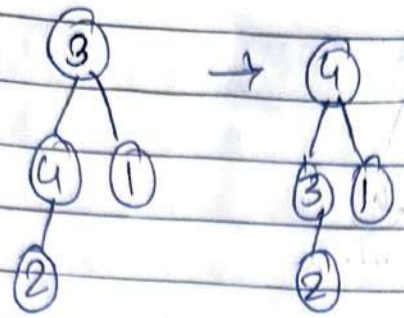
(2) arr = [3, 2, 1, 10, 4]

Pass 1)



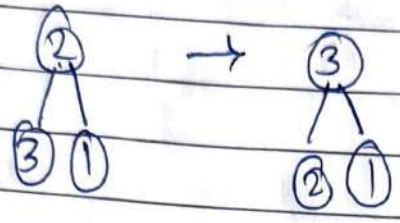
arr = [10, 4, 1, 2, 3]
arr = [3, 4, 1, 2, 10]

Pass 2)



arr = [4, 3, 1, 2, 10]
arr = [2, 3, 1, 4, 10]

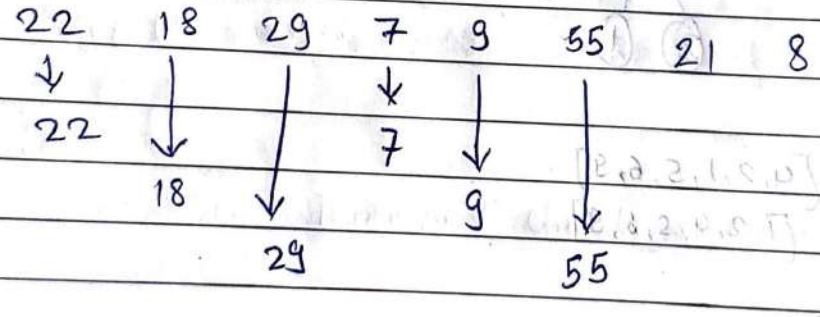
Pass 3)



arr = [3, 2, 1, 4, 10]
arr = [1, 2, 3, 4, 10]

Q.5] Sort following elements using shell sort.
① arr = [22, 18, 29, 7, 9, 55, 21, 8]

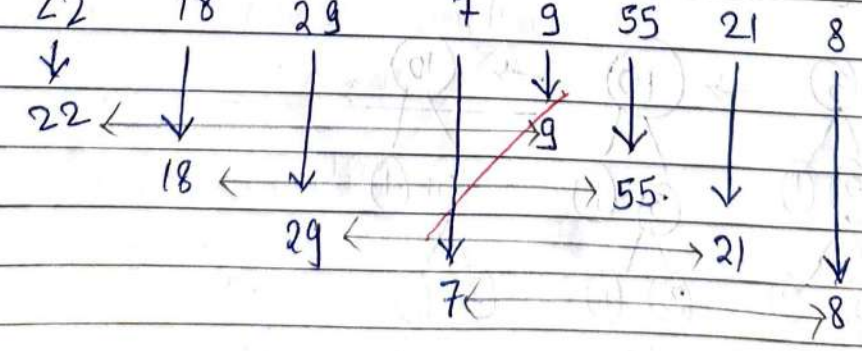
Pass 1



Pass 1]

$\frac{n}{4} = 2$

$n = 8$



Pass 2) 9 18 21 7 22 55 29 8

9 ← → 21 ← → 22 ← → 29

18 ← → 7 ← → 55 ← → 8

Pass 3) 9 7 21 8 22 18 29 55

9 7 21 8 22 18 29 55

7 9 21 8 22 18 29 55

7 9 21 8 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 18 21 22 29 55

sorted arr = 7 8 9 18 21 22 29 55

② ~~arr~~ arr = [3, 10, 15, 12, 1, 5, 2, 6]

Pass 1) 3 10 15 12 1 5 2 6

3 ← → 1

10

15

12

5

2

6

Pass 2) 1 5 2 6 3 10 15 12

1 ← → 2 ← → 3 ← → 15

5 ← → 6 ← → 10 ← → 12

Pass 3)

1	5	2	6	3	10	15	12
1	5	2	6	3	10	15	12
1	2	5	6	3	10	15	12
1	2	5	6	3	10	15	12
1	2	5	6	3	10	15	12
1	2	3	5	6	10	15	12
1	2	3	5	6	10	15	12
1	2	3	5	6	10	15	12
1	2	3	5	6	10	12	15

Q.6) Sort elements in ascending order using Radix Sort ?

→
```
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
int main ()
{
int a[20][10], b[20], temp[10], len, i, j, k, n, p, pos, max;
printf ("Enter the no. of elements :\n");
scanf ("%d", &n);
if (n > 20)
{
printf ("Error : Maximum 20 elements allowed. \n");
return 1;
}
printf ("\n Enter the elements :\n");
for (i=0 ; i<n ; i++)
{
scanf ("%d", & b[i]);
}
max = b[0];
for (i=1 ; i<n ; i++)
{
```

```
    if (b[i] > max)
        max = b[i];
}
len = (int) log10 (max) + 1;
for (i=0; i<len; i++)
{
    for (j=0; j<10; j++)
    {
        top [j] = -1;
    }
    for (j=0; j<n; j++)
    {
        pos = (int)(b[j]/ pow (10, i)) %10;
        a [pos] [++ top [pos]] = b[j];
    }
    for (j=0, p=-1; j<10; j++)
    {
        for (k=0; k<= top[j]; k++)
        {
            b[++ p] = a [j][k];
        }
    }
}
printf ("\n");
return 0;
}
```

Output: Enter the no. of elements : 5
Enter the elements : 45
35
64
99
78

After sorting the elements :
35 45 64 78 99

Exp 5

*) Theory Questions

→ Array vs Linked List

Feature	Array	Linked List
• Memory Allocation	Contagious, static	Non-contagious, dynamic
• Size	fixed	flexible
• Access	direct	sequential
• Memory usage	No extra pointer needed	Extra memory for pointers.

→ Basic terminologies in linked list.

1) Node [10|] →
2) Head
3) Tail
4) Next – Head → [10 | Null] → [20 | Null] ← Tail
5) Next pointer.
6) Empty linked list. Head → Null.

① Node – Basic building blocks of linked list. It contains data & next (address of next node).

② Head – The head is a reference to the following node.

③ Tail – last node where NEXT = NULL

④ Next – Each node has a next reference to the following node.

⑤ Null pointer – special value that indicates that ptr does not have object or memory.

⑥ Empty linked list – Singly linked list with no nodes in it.

*) singly linked list code

```
# include <stdio.h>
# include <stdlib.h>
void create ();
void display ();
void insert_begin ();
void insert_end ();
void insert_pos ();
void delete_all ();
void delete_pos ();
int count_elems ();
void reverse_ll ();
void search ();
struct node {
int info;
struct node * next;
}
struct node * start = NULL;
int main ()
{
int choice;
while (1) {
printf ("1. create \n");
printf ("2. display \n");
printf ("3. insert at begin \n");
printf ("4. insert at end \n");
printf ("5. insert at any position \n");
printf ("6. delete at given position \n");
printf ("7. delete all \n");
printf ("8. count elements \n");
printf ("9. reverse linked list \n");
printf ("10. Search for an element \n");
printf ("11. exit \n");
```

```
printf ("Enter your choice :");
scanf ("%d", & choice");
switch (choice)
{
 Case 1 : create () ; break ;
 Case 2 : display (); break ;
 Case 3 : insert_begin () ; break ;
 Case 4 : insert_end () ; break ;
 Case 5 : insert_pos () ; break ;
 Case 6 : delete_pos () ; break ;
 Case 7 : delete_all () ; break ;
 Case 8 : count_elems () ; break ;
 Case 9 : reverse_ll () ; break ;
 Case 10 : search () ; break ;
 Case 11 : exit (0) ;
  default : printf ("Incorrect choice, \n") ;
 }
};
return 0;
}
void create ()
{
struct node * temp * ptr ;
temp = (struct node *) malloc (size of (struct node)) ;
printf ("Enter data :");
scanf ("%d" & temp -> info);
temp -> next = NULL ;
if (start == NULL)
{
start = temp ;
}
else
{
```

```
ptr = start;
while (ptr -> next != NULL)
{
  ptr = ptr -> next;
}
}
void display ()
{
struct node * ptr;
 if (start == NULL)
{
printf ("\n Empty list \n");
 return;
}
ptr = start;
printf ("\n List elements :");
 while (ptr != NULL)
{
 printf ("%d", ptr -> info);
  ptr = ptr -> next;
}
printf ("\n");
}
void insert_begin ()
{
struct node * temp * ptr;
temp = (struct node*) malloc (size of (struct node));
printf ("Enter data :");
scanf ("%d & temp -> info);
 temp -> next = NULL;
  if (start == NULL)
  {
```

```
start = temp;
}
else
{
  ptr = start;
  while (ptr -> next != NULL)
{
  ptr = ptr -> next;
}
ptr -> next = temp;
}
}
void insert_end ()
{
struct node * temp * ptr;
temp = (struct node *) malloc (size of (struct node));
printf ("Enter data:");
scanf ("%d", & temp -> info);
temp -> next = NULL;
if (start = NULL)
{
  start = temp;
}
else
{
  ptr = start;
  while (ptr -> next != NULL)
{
  ptr = ptr -> next;
}
  ptr -> next = temp;
}
}
```

```
void Insert-pos ()
{
 struct node * temp, * ptr ;
 int i, pos ;
  temp = (struct node *) malloc (size of (struct node));
 Printf ("Enter data:");
 scanf ("%d", & temp -> info);
 temp -> next = NULL
 printf ("\n Enter position:");
  scanf ("%d", & pos);
  if (pos == 1)
 {
   temp -> next = start ;
   start = temp ;
   return ;
 }
  ptr = start ;
 for (i=1 ; i<pos -1 && ptr != NULL, i++ )
 {
     ptr = ptr -> next ;
 }
  if (ptr == NULL)
 {
   Printf ("Position not found\n");
    free (temp);
 }
 else
 {
  temp -> next = ptr -> next ;
   ptr -> next = temp ;
 }
}
```

```
void delete all ()
{
struct node * temp ;
while (start != NULL)
{
 temp = start ;
 start = start -> next
 free (temp) ;
}
printf ("All values deleted !\n ");
}

void delete pos ()
{
 int i, pos ;
struct node * temp * ptr ;
  if (start == NULL)
{
 printf (" Enter list ~~position~~ :");
  return ;
}
 printf (" Enter position :");
 scanf ("%d" & pos);
 if (pos == 1)
{
 temp = start ;
 start = start -> next ;
free (temp);
 return ;
}
 ptr = start ;
  for (i=1; i<pos-1 && ptr != NULL ; i++)
```

```
{
  ptr = ptr -> next;
}
if(ptr == NULL || ptr -> [illegible] == NULL)
{
  printf("Position not found\n");
}
else {
  temp = ptr -> next;
  ptr -> next = temp -> next;
  free(temp);
  printf("Node deleted\n");
  }
}
int count_elemns () {
  int count = 0;
struct node * ptr = start;
while (ptr) NULL).
{
  count ++;
  ptr = ptr -> next;
}
printf("Number of elements : %d\n", count);
return 0;
}
void reverse_dll() {
struct node * ptr;
int keypos = 1= found = 0;
printf("Enter element to search :");
for (int i = len-1; i >= 0; i--)
{
  printf( ) %d", arr[i]);
}
```

```
printf ("\n");
}

struct node * ptr ;
int key , pos =1, found = 0;
printf (" Enter element to search : ");
scanf ("%d, & key);
 ptr = start ;
 while (ptr != NULL)
{
 if (ptr -> info == key)
{
 Printf (" element found at position %d\n", pos);
 found = 1;
 break ;
}

 ptr = ptr -> next ;
 pos ++ ;
}
if (!found) {
 printf (" Element not found \n");
}
}.
```

Null
7/11

Exp: 6

Q) Doubly linked list write a menu driven C program that implements doubly linked list for the following operations :- Create, display, insert a node, delete a node, search a node, count no. of nodes, reverse a node ?

→ Code :-

```
#include <stdio.h>
#include <stdlib.h>
type def struct node {
  int data ;
  struct node * prev;
  struct node * next;
} node ;
node * head = NULL ;
node * create Node ~~= (Node *) malloc~~ (int data) {
      node * newNode = (Node*) malloc (size of (Node)) ;
      new node → data = data ;
      newNode → prev = newNode → next = NULL ;
      return newNode ;
    }
void create list () {
  int data, n, i ;
  printf ("Enter no. of nodes :") ;
  scanf ("%d", n) ;
  head = NULL ;
  node * temp = NULL ;
for (i=0 ; i<n ; i++) {
  printf ("Enter no. for node %d :", i+1) ;
  scanf ("%d", &data) ;
  node * newNode = create Node (data) ;
    if (head == NULL) {
      head = newNode ;
      temp = head ;
  } else {
```

```
temp -> next = newNode ;
newNode -> prev = temp;
temp = newNode ;
}
}
printf (" list created successfully .\n");
}
void display list () {
if (head == NULL ) {
printf ("list is empty .\n");
return ;
}
node * temp = head ;
printf ("list : ");
while (temp) {
printf ("%d", temp -> data);
temp = temp -> next;
}
printf ("\n");
}
void insert Node () {
int pos, data ;
printf ("Enter position to insert : ");
scanf ("%d", & pos);
printf ("Enter data : ");
scanf ("%d", & data);
Node * newNode = create Node (data);
if (pos == 1) {
newNode -> next = head ;
if (head) head -> prev = newNode ;
head = newNode ;
} else {
```

```
node * temp = head ;
int i ;
for (i=1 ; i< pos -1 && temp ; i++)
    temp = temp -> next ;
if (!temp) {
    printf ("Invalid position .\n");
    free (newNode);
    return ;
}
newNode -> next = temp -> next ;
if (temp -> next)
    temp -> next -> prev = newnode ;
temp -> next = newNode ;
newNode -> prev = temp;
}
printf ("Node inserted. \n");
}
void deleteNode () {
int pos ;
printf ("Enter position to delete :");
scanf ("%d", &pos);
if (head == NULL) {
printf ("List is empty. \n");
return 0;
}
node * temp = head ;
int i ;
if (pos == 1) {
head = head -> next ;
if (head)
    head -> prev = NULL ;
    free (temp);
```

```
} else {
  for (i=1 ; i<pos ++ temp ; i++)
      temp = temp -> next ;
  if (! temp) {
    printf (" Invalid position .\n");
      return ;
  }
if (temp -> prev)
  temp -> prev -> next = temp -> next ;
  if (temp -> next)
    temp -> next -> prev = temp -> prev ;
  free (temp) ;
}
printf (" Node deleted . \n");
}
void search node () {
int key , pos = 1 ;
Printf ("Enter value to search :");
Scanf ("%d", & key ) ;
Node * temp = head ;
while (temp) {
    if (temp -> data == key ) {
        Printf ("Value %d found at position %d\n", key, pos) ;
    return ;
  }
  temp = temp -> next ;
    pos ++ ;
}
printf ("Value not found in the list . \n ") ;
}
```

```
void countNodes () {
    int count = 0;
  Node * temp = head;
  while (temp) {
    count ++;
  temp = temp -> next;
}
  printf ("Total nodes : %d\n" ;count);
}
void reverse list () {
    if (!head) {
  printf ("List is empty.\n");
  return ;
}
Node * temp = NULL , * current = head;
while (current) {
    temp = current -> prev;
  current -> prev = current -> next;
  current -> next = temp;
  current = current -> prev;
}
if (temp)
    head = temp -> prev;
  printf ("List reversed.\n");
}

int main () {
    int choice;
while (1) {
    printf ("\n --- Doublylinked list menu ---\n");
```

```
printf ("1. Create list \n");
printf ("2. Display list \n");
printf ("3. Insert node \n");
printf ("4. Delete node \n");
printf ("5. Search node \n");
printf ("6. count node \n");
printf ("7. Reverse list \n");
printf ("8. Exit \n");
printf ("Enter your choice (1-8) :");
scanf ("%d", &choice);

switch (choice) {
case 1 : Create list ();  break ;
Case 2 : display list ();  break ;
Case 3 : insert node ();  break ;
Case 4 : delete node ();  break ;
Case 5 : search node ();  break ;
Case 6 : count node ();  break ;
Case 7 : reverse list ();  break ;
Case 8 : Exit (0);
default : printf ("Invalid choice ! Try again . \n");
```

Exp. 7

1) Primitive operations on stack : POP, PUSH, isEmpty, isFull ;

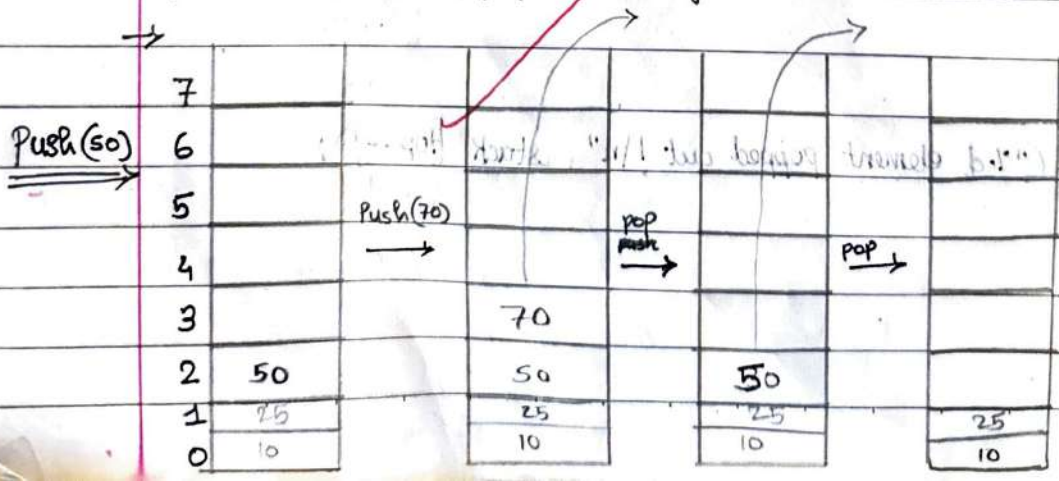
→
```
# Include <stdio.h>
# define MAX 100
int stack [MAX]
int top = -1;
void push (int n);
void pop ();
void is Empty ();
void is Full ();
int main ()
{
int value, choice ;
printf ("Enter '1' to push :\n");
printf ("Enter '2' to pop :\n");
printf ("Enter '3' to check if stack is empty :\n");
printf ("Enter '4' to check if stack is full :\n);
printf ("Enter '5' to exit :\n");
while (1)
{
printf ("Enter your choice :");
scanf ("%d", & choice) ;
switch (choice)
{
Case 1 : printf ("Enter value :");
        scanf ("%d", & value );
        push (value);
        break ;
Case 2 : pop ();
        break ;
Case 3 : is Empty ();
        break ;
```

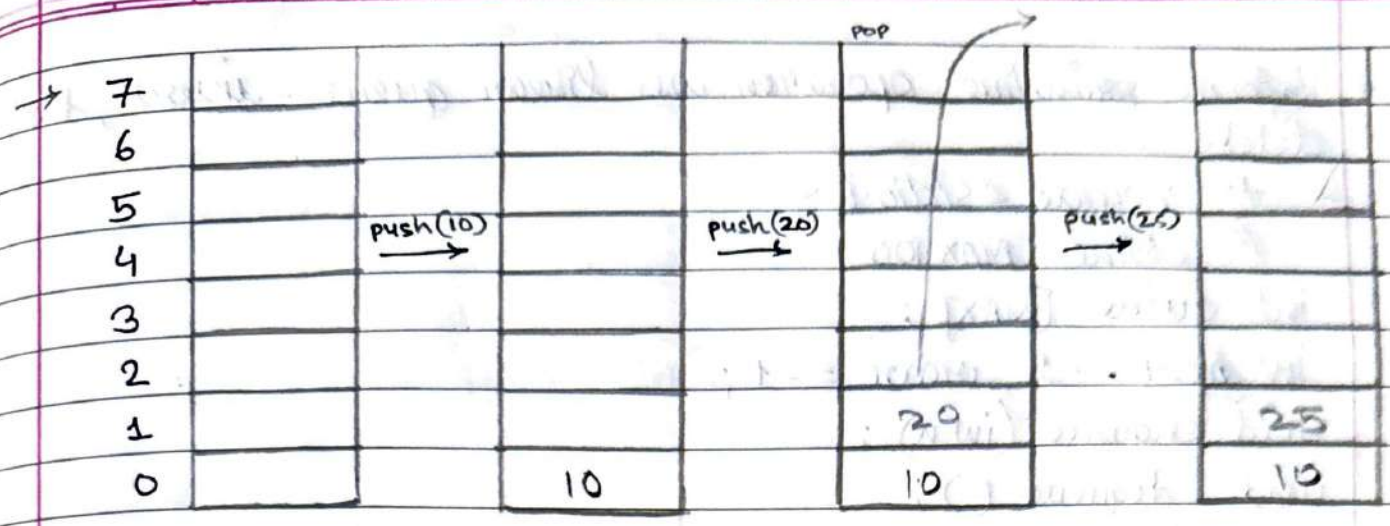
```
Case 4 : isfull ();
        break ;
Case 5 : printf ("Exiting !----\n");
        return 0;
 default : Printf ("Invalid input :\n");
}
return 0;
}
void push (int n)
{
  if (top == MAX-1)
  {
     printf ("Stack overflow !\n");
  }
 else
  {
    stack [++ top]= n ;
    printf ("%d added to stack \n\n" n);
  }
}
void pop ()
{
  if (top== -1)
  {
     printf ("Stack underflow \n");
  }
else
  {
   printf ("%d element popped out !\n", stack [top--]);
  }
}
```

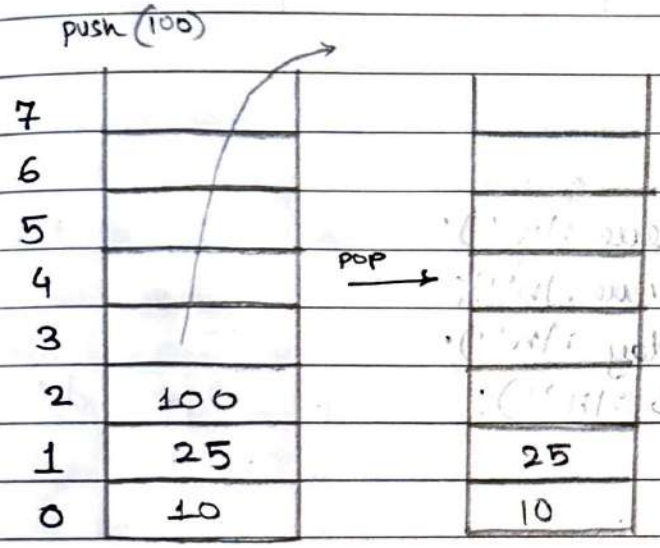
```
void isEmpty ()
{
  if (top == -1)
{
  printf ("stack is empty!\n");
 }
else
 {
  printf ("stack is not empty!\n");
 }
}

void isFull ()
{
 if (top == MAX-1)
{
  printf ("Stack is full !\n");
 }
else
 {
  printf ("stack is not full!\n");
 }
}
```

ii) Stack size - 8 for push (10), push (20), push (25), push (50), push (70), pop, pop, push (100), pop & draw final output.







stack [8] → (size, 8)
push(10) → stack [10]
push (20) → stack [10, 20]
pop → stack [10]
push (25) → stack [10, 25]
push (50) → stack [10, 25, 50]
push (70) → stack [10, 25, 50, 70]
pop → stack [10, 25, 50]
pop → stack [10, 25]
push (100) → stack [10, 25, 100]
pop → stack [10, 25]

Final output : stack [10, 25]

Experiment: 8

Write a 'C' program for expression conversion using stack

1)→ Convert Infix to Postfix or Prefix expression.

```
⇒ # Include <stdio.h>
  # Include <ctype.h>
  char stack [100]
  int top = -1;
  void push (char x) {
      stack [++ top] = x;
  }
  char pop () {
    if (top == -1)
      return -1;
  else
    return stack [top --];
  }
  int priority (char x) {
    if (x == '(')
      return 0;
    if (x == '+' || x == '-')
      return 1;
    if (x == '*' || x == '/')
      return 2;
    if (x == '^')
      return 3;
    return -1;
  }
  int main () {
    char exp [100];
    char *e, x;
  printf ("Enter infix expression :");
    scanf ("%s", exp);
      e = exp;
    printf ("Postfix expression :");
```

```
while (*e != '\0') {
    if (isalnum(*e)) {
    print("%c", *e);
}
  else if (*e == ')') {
    while ((x = pop()) != '(')
      printf("%c", x);
  }
  else {
    while(top != -1 && priority (stack[top]) >= priority (*e))
    printf("%c", pop());
    push(*e);
  }
  e++;
  while (top != -1) {
      Printf("%c", pop());
  }
  return 0;
}
```

Output:- Enter infix expression:-
A+(B*C-(D/E^F)*G)*H

Postfix expression :- ABC*DEF^/G*-H*+

Algorithm :-

ii) Convert infix expression into prefix using Stack: Infix expression

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

→ # Include <stdio.h>

Input	Stack	Output
H	*	H
*	*	H
)	*)	H
G	*)*	HG
*	*)*	HG
)	*)*)	HG
F	*)*)	HGF
^	*)*)^	HGF
E	*)*)^	HGFE
/	*)*)/	HGFE^
D	*)*)/	HGFE^D
C	*)*	HGFE^D/
-	*)-	HGFH^D/*
C	*)-	HGFH^D/*C
*	*)-*	HGFH^D/*C
B	*)-*	HGFH^D/*CB
C	*	HGFH^D/*CB*-
+	+	HGFH^D/*CB*-
A	+	HGFH^D/*CB*-*A

~~SA~~ HGFH^D/*CB*-*A+

iii) Write a C Program to evaluate infix or postfix expression.

→
```
# Include <stdio.h>
# Include <ctype.h>
# include <stdlib.h>
# define SIZE 40
int pop ();
void push (int);
char postfix [SIZE] ;
int stack [SIZE], top = -1 ;
int main ()
{
 int i, a, b, result , pEval ;
 char ch ;
for (i=0 ; i< size ; i++)
{
  stack [i] = -1 ;
}
printf ("\n Enter a postfix expression :") ;
scanf (" %s", postfix) ;
 for (i=0 ; postfix [i] != '\0' ; i++)
{
   ch= postfix [i] ;
  if (isdigit (ch))
 {
   Push (ch -'0') ;
}
else if (ch == '+' || ch == '-' || ch == '*' || ch == '/')
{
     b= pop () ;
     a= pop () ;
```

```
    switch (ch)
    {
     case '+' ;
        result = a+b;
        break ;
     case '-' ;
        result = a-b ;
        break ;
     case '*' ;
        result = a*b ;
        break ;
     case '/' ;
        result = a/b;
        break;
    }

    push (result);
  }
}

pEval = pop () ;
printf ("\n The postfix evaluation is %d\n", pEval);
return 0;
}

void push (int n)
{
 if (top < SIZE-1)
 {
    stack[++top] = n ;
 }
 else
 {
```

```
printf ("Stack is full.\n");
    exit (-1);
  }
}
int pop ()
{
  int n;
  if (top > -1)
  {
    n= stack[top--];
    return n;
  }
  else
  {
    printf ("Stack is empty !\n");
    exit (-1);
  }
};
```

Output :- Enter a postfix expression :- 12+42/*1-42$+
The postfix evaluation is :- 6.

iv) Evaluate prefix expression using stack : Prefix expression.

+ - * + 1 2 / 4 2 1 $ 4 2

Input	opr 1	opr 2	Result	Output
2				2
4				2, 4
$	4	2	16	16
1				16, 1
2				16, 1, 2
4				16, 1, 2, 4
/	4	2	2	16, 1, 2
2				16, 1, 2, 2
1				16, 1, 2, 2, 1
+	1	2	3	16, 1, 2, 3
*	3	2	6	16, 1, 6
-	6	1	5	16, 5
+	16	5	21	21

*) Algorithm

1) Scan the Infix expression from left to right
2) If the scanned character is an operant, output it.
3) else;
(a) If the precedence of the scanned operator is greater than the precedence of the operator in the stack.
(b) Else, pop all the operators from the stack which are greater or equal to in the precedence than that of the scanned operator. After doing that push the scanned operator to the stack.

4) If the scanned character is an '(', push it to stack.
5) If the scanned character is an ')', pop the stack & output it until a '(' is encountered & discard both parenthesis.
6) Repeat steps 2-6 until infix exp is scanned.
7) Print the output.
8) Pop & output from the stack until it is not empty.

---

27 Algorithm :-

1) Scan the expression from right to left or reverse the infix expression.
2) Scan the char one by one.
3) If the char is an operand, copy it to prefix notation output.
4) If the char is a closing parenthesis, then push it to the stack.
5) If the char is a opening parenthesis, pop the elements in the stack until we find the corresponding closing ~~Patterns~~ parenthesis.
6) If the char scanned is an operator -
   (a) If the operator has precedence greater than or equal to the top of the stack, push the operators to the stack.
   (b) If the operator has precedence lesser than the top of the stack, output it to the prefix notation output & then check the above condition again with the new top of the stack.
7) After all the char are scanned, reverse the prefix notation output.

3]→

1) Initialize an empty stack S.
2) Scan postfix expression from left to right.
3) If operand → Push into S.
4) If operator → pop two operands from S, apply operation, push result back to S.
5) After scanning → Top of the stack is result.

4]→ 1) Initialize an empty stack S.
2) Scan postfix expression from right to left.
3) If operand → push into S.
4) If operator → pop two operands from S, apply operation, push result back to S.
5) After scanning → Top of the stack is result.

Exp 9

1) Perform primitive operation on linear queue - insert, display, delete.

→
```
# include <stdio.h>
# define MAX 100
int queue [MAX];
int front = -1, rear = -1;
void enqueue (int n);
void dequeue ();
void display ();
int main ()
{
int choice value;
Printf ("Enter '1' to enqueue :\n");
printf ("Enter '2' to dequeue :\n");
printf ("Enter '3' to display :\n");
printf ("Enter '4' to exit :\n");
while (1)
{
printf ("Enter choice :");
scanf ("%d", & choice);
switch (choice)
{
case 1: printf ("Enter value :");
        scanf ("%d" & value);
      enqueue (value);
        break;
Case 2 : dequeue ();
        break;
Case 3 : display ()
        break;
case 4 : printf ("exiting !--\n");
      return ()
```

```
default : Printf (" Invalid input \n");
  }
}

void enqueue (int n)
{
  if (rear = MAX-1)
   printf (" Queue overflow \n");
  }
else {
  if (front ==-1)
    {
      front =0;
    }
rear ++;
queue [rear] = n;
printf ("%d" enqueued ! \n", n);
}
}

void dequeue ()
{
  if (front ==-1 || front > rear)
    {
      printf ("Queue underflow ! \n");
    }
else {
  printf(" %d dequeued from Queue ! \n", queue [front]);
  front ++;
  }
}

void display ()
{
  if (front == 1 || front > rear)
    {
```

```
    printf ("Queue is empty !\n"); ~~queue[front]~~);
  } ~~front++~~;
else :
  {
for int (i= front ; i<rear, i++)
  {
    printf ("%d", queue[i]);
  }
  printf ("\n")
}
}
```

2] → Perform following operations on linear queue in array size 10 with this operations insert (10), insert (50), delete, insert (100), insert (20), delete, insert (25), insert (200)

→ insert 10

insert →

									10	← delete

Rear = 0 Front = 0

insert(50)

insert →

								50	10	← delete

Rear = 1 Front = 0

Delete

insert →

								50	~~10~~	← delete

insert (100)

insert →

							100	50	~~10~~	← delete

insert(20)

insert →

						20	100	50	~~10~~	← delete

Rear=2 Front = 0

Delete

insert →

						20	100	~~50~~	~~10~~	← delete

Rear = 1 Front = 0

insert(25)

insert →

					25	20	100	~~50~~	~~10~~	← delete

insert(200)

insert →

				200	25	20	100	~~50~~	~~10~~

rear = 2 front = 0

insert(10) if empty front = 0, rear = -1

[-,-,-,-,-,-,-,-,-,10],

insert (50), front = 0, rear = 0

[-,-,-,-,-,-,-,-,50,~~10~~]

front = 0, rear = 1

insert(100)

[-,-,-,-,-,-,-,100,50,~~10~~]

front = 0, rear = 1

insert (20)

[-,-,-,-,-,-,20,100,50,~~10~~]

front = 0, rear = 2

delete

[-,-,-,-,-,-,20,100,~~50~~,~~10~~]

front = 0, rear = 1

Insert (25)

[–, –, –, –, –, 25, 20, 100, ~~50~~, ~~20~~]

front = 0, rear = 2

Insert (200)

[–, –, –, –, 200, 25, 20, 100, ~~50~~, ~~10~~]

front = 0, rear = 3

*) Theory Questions

1) Explain the concept of priority queue.

→ A priority queue is a special type of queue where each element is assigned a priority, & elements are served based on their priority - not just their arrival order.

2) Write algorithm of insertion & deletion of linear queue

→ Insertion :-

1) Check if the queue is in overflow condition.
2) If no overflow condition check if front is equal to -1 & change it to 0.
3) Increase rear by 1, equate the rear to the value entered.
4) Print the value enqueued.

Exp 10

Q) Primitive operation on Circular Queue. Insert, Delete, Display.

→
```
# include <stdio.h>
# Include <stdlib.h>
# define size
# int queue (size);
int f = -1, rear = -1;
void insert ()
{
int value;
if (f == 0 && r == size-1) || (rear +1) % size == f)
}
printf ("Queue is full \n");
}
else
{
printf ("enter value to insert :");
scanf ("%d, & value);
if (f == -1)
{
f=0;
r=0;
}
else
{
r= (r+1) % size;
}
Queue (r) = value;
printf ("Inserted %d\n", value;
}
}
void delete ()
{
```

```
if (f == -1)
{
  printf ("Queue is empty \n");
}
else
{
  printf ("Deleted %d\n", queue (f));
   if (f==r)
{
   f = -1;
   r = -1;
}
else
{
   f = (f+1) % size;
}
}
}
void display ()
{
 if (f == -1)
 {
  printf ("Queue is empty \n");
  int i = f;
  while (1)
  {
   printf ("%d", queue (i));
  if (i == r)
    break;
  i = (i+1) % size;
}
printf ("\n");
}
}
```

```
int main ()
{
  int choice ;
  while (1)
{
  printf ("\n 1. Insert . \n2. Delete \n3. 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 () break;
  }
  default : printf ("Invalid choice :");
  }
}
return 0 ;
}
```

Output

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

Enter your choice : 1
Enter element to insert : 10

2) what are advantages of circular queue show diagramatic representation.

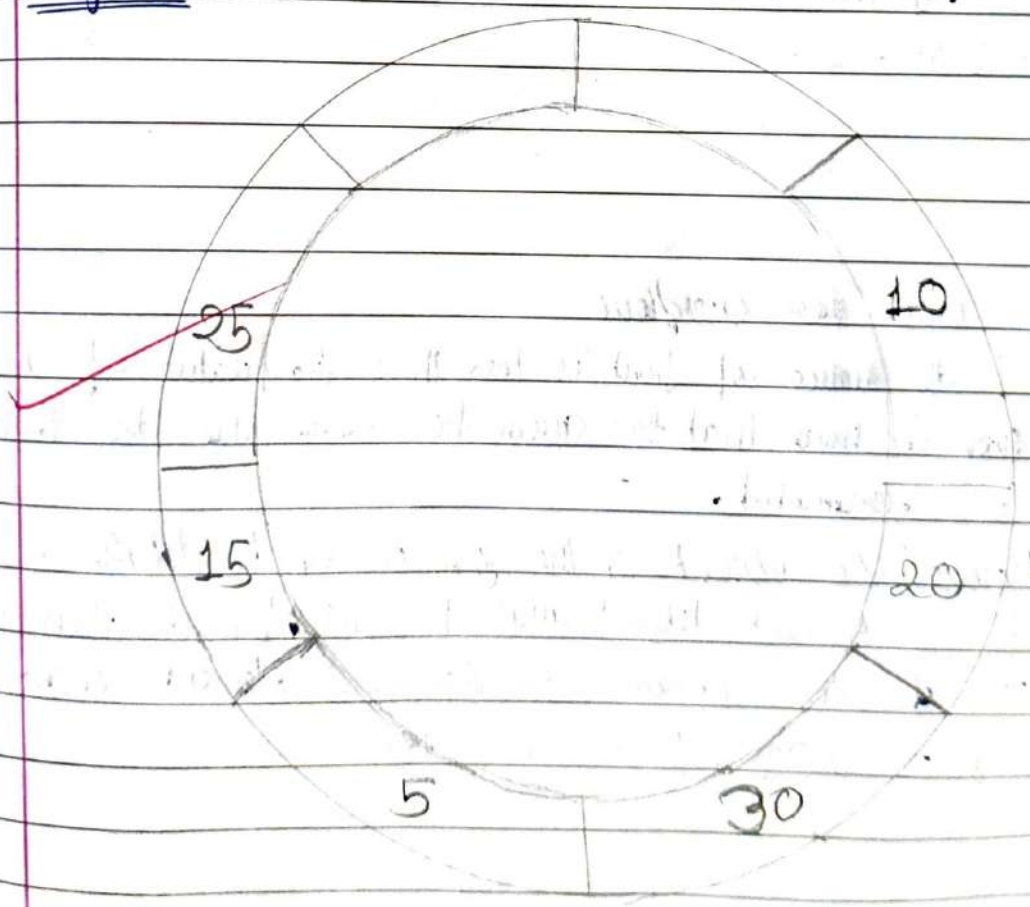
→ 1) Efficient memory utilization :-
Unlike linear Queue, space is required after deletion.

2) Prevents overflow :-
In linear queue, once rear reaches the end, no more insertion are possible even if there are empty slot's of the front.

3) Fixed size for optimized :
work's well for fixed size memory buffers.

4) Faster Operation :
Insertion & deletion are O(1)

Diagram →



10	20	30	5	15	25

↑ Front = 0 ↑ rear = 5

3) Algorithm

→ insertion -

Step 1 - Check for overflow

Step 2 - Check if the queue is correctly empty if f == -1 in case the queue is empty set the value of front & rear to 0 so that the new elements can be stored at the location, if the rear already has some value increment it so that it point to the next position.

Step 3 - The value inserted is stored at the position pointed by the rear.

Step 4 - Exit.

→ Deletion -

Step 1 :- Check for overflow

Step 2 :- If the value of front is less than the value of rear then it mean that the queue has some value the front is incremented.

Step 3 :- Remove the element on the front & report deletion. If the element deleted was the last element then front & rear if not advance for the next element front = front + 1 & get front = -1 & rear = -1.

Step 4 :- Exit.

4) Write & explain the application of queue.

→ Application of queue are

1) Data transmission -

In network's, queue are used in router & switches. to prevent data loss.

2) Breath first search in graph -

BFS algorithm are a queue to explore nodes at level by level.

3) Call Center's

The first calls who entered the queue get connected to the next available operator.

4) Amusement Park -

People are waiting for rides in amusement park or for a doctor's appointment are saved in queue.

5) Differentiate between stack & queue.

Stack	Queue
1) It work on LIFO	1) It works on FIFO
2) Insertion happens at top.	Insertion happens at front.
3) Deletion happens at top.	Deletions happen at front.
4) Only top element is accessible.	Both front & rear elements are accessible.
5) Transversal is not possible.	Transversal is possible.
6) Can be implemented using array or linked list.	Can be implemented using array linked list or circular array.

Exp 11

Q.1) Operations on Binary tree

```
#include <stdio.h>
#include <stdlib>
struct node {
    int data;
    struct node * left;
    struct node * right;
}
struct node * create (int val);
struct node * insert left (struct node * root, int val);
struct node * insert right (struct node * root, int val);
void delete subtree (struct node * root);
void delete left (struct node * root);
void delete right (struct node * root);
void display (struct node * root);

int main () {
    struct node root = create (1);
    insert left (root, 2);
    insert right (root, 3);
    insert left (root -> left, 4);
    insert right (root -> right, 5);
    insert left (root -> right, 6);
    insert right (root -> right, 7);
    insert left (root -> left -> left, 8);
printf ("Original tree :\n");
    display (root);
    printf ("\n\n");
delete left (root)
printf ("Tree after deleting left subtree of root :\n");
display (root);
return 0;
}
```

```
struct node * create (int val) {
    struct node * new = (struct node *) malloc (sizeof (struct node));
  if (new == NULL) {
      printf ("memory alloc failed");
      exit (1);
  }
  new -> data = val;
  new -> left = NULL;
  new -> right = NULL;
  return new;
}
struct node * insert right (struct root, int val) {
  if (root == NULL) {
  printf ("can't insert in NULL root");
    return NULL;
  }
  root -> right = create (val);
    return root -> right;
}
  struct node * insertleft (struct node * root, int val);
{
  if (root == NULL) {
        printf ("can't insert in NULL");
          return NULL;
  }
  root -> left = create (val);
  return root -> left;
}
```

```
void delete subtree (struct node * node) {
    if (node == NULL) {
        return ;
    }
    delete subtree (node -> left);
    delete subtree (node -> right);
    free (node);
}
void delete left (struct node * root) {
    if (root == NULL || root -> left == NULL) {
        printf ("Nothing to delete");
        return;
    }
    delete subtree (root -> left);
    root -> left = NULL;
    printf ("\n left subtree deleted");
}
void delete right (struct node * root) {
    if (root == NULL || root -> right == NULL) {
        printf ("Not to delete");
        return ; }
    delete subtree (root -> right);
    root -> right = NULL;
}
void display (struct node * root) {
    if (root == NULL) {
        return ;
    }
    printf ("%d", root -> data);
    display (root -> left);
    display root -> right);
}
```

Output :-

original tree : 1 2 4 8 5 3 6 7
left subtree deleted
Tree after deletion : 1 3 6 7

Q.27 Pre Order, Post Order, In Order, Transversal.

```
# Include <stdio.h>
# include <stdlib.h>
struct node {
    int data ;
    struct node * left ;
    struct node * right ;
} ;
// declare all FNS in last Program
void preorder ( struct node * root);
void postorder ( struct node * root) ;
void inorder ( struct node * root) ;
int main () {
    // make same tree as last program
    int choice ;
  while (1) {
  printf (" --- Menu --- \n ");
    printf (" 1 Pre-order traversal \n ");
    printf (" 2 Post-order traversal \n);
    printf (" 3 In-order traversal \n);
    printf (" 4. Exit ");
    printf ("\n Choose option:");
    scanf ("%d", & choice);
      switch (choice) {
```

```
Case 1;
    preorder (root);
    break;
Case 2;
    postorder (root);
    break;
Case 3:
    inorder (root);
    break;
Case 4:
    exit (0);
    break;

default :
    printf (" Invalid output ");
  }
}

return 0;
}

// same function definations as last program
void preorder (struct node *root){
    if (root == NULL) {
        return ;
    }
    printf ("%d, root -> data);
    preorder (root -> left);
    preorder (root -> right);
}

void postorder (struct node * root) {
    if (root == NULL) {
        return ;
    }
```

Exp 12

```
# include <stdio.h>
# define V5
void init (int arr [ ][V]) {
    for (int i=0; i<V; i++) {
        for (int j=0; j<V; j++) {
            arr [i][j] = 0;
        }
    }
}

void insertEdge (int arr [ ][V], int i, int j) {
    arr [i][j] = 1;
    arr [j][i] = 1;
}

void printAdjMatrix (int arr [ ][V]) ~~{int i, int j}~~ {
    printf ("Adjacency matrix: \n");
    for (int i=0; i<V; i++) {
        for (int j=0; j<V; j++) {
            printf ("%d", arr [i][j]);
        }
        printf ("\n");
    }
}

int main () {
    int adjMatrix [V][V];
    int (adjMatrix);
    insertEdge (adjMatrix, 0,1);
    insertEdge (adjMatrix, 0,4);
    insertEdge (adjMatrix, 1,2);
    insertEdge (adjMatrix, 1,3);
    insertEdge (adjMatrix, 1,4);
    insertEdge (adjMatrix, 2,3);
    insertEdge (adjMatrix 3,4));
    return 0;
```

Output

Adjacency Matrix

```
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 1 0 1
1 1 0 1 0
```