1. Declare an array of 5 subjects, accept marks from the user and calculate average.

```
# include <stdio.h>
int main ()
{
   int marks[5];
   int sum=0;
   int i;

 printf("Enter marks of 5 subjects:");
   for ( i = 0; i < 5 ; i++)
   {
    scanf(" %d", &marks[i]);
    }
    for ( i=0 ; i < 5 ; i++)
    {
     sum+= marks[i];
    }
    float Avg = sum / 5;

 printf(" Average marks:", Avg);
  return 0;
  }
```

~~Output~~

Output

Enter marks of 5 subjects : 98

78

98

95

74

Average Marks : 88.000

Page No.:
Date:

2. Write a program to accept no. n from user and check if it is even or odd.

```
# include <stdio.h>
int main ()
{
   int num;
   printf ("Enter Number:");
   scanf ("%d", &num);
   if (num % 2 ==0) {
    printf("Number is even.");
   }
   else
   {
   printf("Number is odd.");
   }
   return 0;
   }
```

Output
Enter Number: 25
Number is odd.

3. Write a program to print prime numbers from 1 to 50.

```
## include <stdio.h>
int main ()
{
  int i, j, count=0;
  printf("Prime Numbers from 1 to 50 are:\n");
  for(i=1; i<=50; i++)
  {
    for(j=2; j<i; j++)
    {
      if (i % j==0)
      {
        count ++;
        break;
      }
    }
    if (count==0 && i!=1)
      printf("%d\n", i);
    count=0;
  }
  return 0;
}
```

Page No.:
Date:

Output

Prime Numbers from 1 to 50 are:

2
3
5
7
11
13
17
19
23
29
31
37
41
43
47

Page No.:
Date:

4. Write a menu driven program ~~for~~ on weekly basis to know if ~~Pase~~ it is a work day or a Holiday by using Numbers.

```
# include <stdio.h>
int main(){
  int num;
printf("Enter a no. :");
scanf("%d", &num);
switch (num){
    case 1 :
    printf("Monday In Week day.");
    break;
    case 2:
    printf("Tuesday In Week day.");
    break;
    case 3:
    printf ("Wednesday In Week day.");
    break;
    case 4:
    printf (" Thursday In Week day.");
    break;
    case 5:
    printf (" Friday In Week day.");
    break;
    case 6:
    printf (" Saturday In Holiday.");
    break;
    case 7:
    printf(" Sunday In Holiday.");
```

```
    break;
}
return 0;
}
```

Output

Enter a no. : 3

Wednesday

Week day.

5. Write a program to accept number from user, if the number is palindrome or not.

```
#include <stdio.h>
int main()
{
int num, reversed_num=0, remainder,
original_num;
printf("%d", &num);
original_num = num;
while (num!=0){
remainder = num % 10;
reversed_num = reversed_num* 10+ remainder;
num /= 10;
}
if (original_num == reversed_num){
printf("%d is a palindrome.\n", original_num);
}
else{
printf("%d not a palindrome.\n", original_num);
}
return 0;
}
```

Output

Enter an integer : 67
67 is not a palindrome.

Experiment - 1

1) Find avg of 10 no.s using an array.

```
# include <stdio.h>
int main ()
{
  int numbers [10];
  int sum = 0;
  float average;
  printf(" Enter 10 number : \n" );
  for (int i=0; i<10; i++) {
  printf( " Number %d:", i+1 );
  scanf ("%d ", & numbers [i] );
  sum+= numbers [i];
  }
  average = sum / 10.0;
  printf(" The average is : %. 2f \n", average);
  return 0;
  }
```

Output

```
Enter 10 number : 0
1
2
3
4
5
6
7
8
9
The average is : 4.5
```

Page No.
Date:

Display the pattern

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

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

    printf("\n");
  }
  return 0;
}
```

Output

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

Page No.:
Date:

Find the greatest and the smallest element in the array.

```
# include <stdio.h>
int main(){
 int n, i;
 printf(" Enter the number of elements: \n", n);
 for (i=0; i<n; i++){
  scanf("%d", &arr[i]);
 }
  int max = arr[0];
  int min = arr[0];
  for (i=1; i<n; i++){
   if (arr[i] >max)
      max = arr[i];
   if (arr[i] < min)
      min = arr[i];
   }
  printf(" Greatest element: %d\n", max);
  printf("Smallest element: %d\n", min);
 return 0;
 }
```

Output

Enter the number of elements : 6

Enter elements :

12 5 89 3 45 7

Greatest element : 89

Smallest element : 3

Square the odd positioned numbers.

```
# include <stdio.h>
int main() {
 int n, i;
printf("Enter the number of elements: ");
scanf("%d", &n);
 int arr[n];
printf("Enter %d elements: \n", n);
 for(i=0; i<n; i++){
  scanf("%d", &arr[i]);
}
 for(i=1; i<n; i+=2){
 arr[i] = arr[i] * arr[i];
}
printf("Array after squaring: \n");
 for(i=0; i<n; i++){
  printf("%d", arr[i]);
 }
 printf("\n");
 return 0;
}
```

Output

1 2 9 4 25 6

Page No.:
Date.

* Find 1st repeating element of array.

```
# include <stdio.h>
int main ()
int arr[ ] = { 4, 2, 1, 3, 2, 5, 1 }
int n = sizeof (arr[]) / sizeof (arr[0]);
int found = 0;

for (i=0; i<n-1 ; i++) {
 for (j = i+1 ; j<n ; j++) {
   if ( arr[i] == arr[j] ) {
 printf(" First repeating element is : %d \n",
            arr[i] );
     found = 1;
     break ;
   }
 }

 if (found) {
    break ;
   }
 else {
 printf(" No repeating element found. ");
  }
 return 0;
}
```

Output

First repeating element is 2.

Extra Questions.

1)

```
*
# #
$ $ $
? ? ? ?
```

```
#include <stdio.h>
int main(){
 int i,j;
 for(i=1; i<=5; i++)
 { for(j=1; j<=i; j++)
 { switch (i){
    case 1:
    printf(" * ");
    break;
    case 2:
    printf(" # ");          → break;
    case 3:
    printf(" $ ");
    break;
    case 4:
    printf(" ? ");
    break;
    }
 }
 printf("\n");
}
return 0;
}
```

Output

```
*
# #
$ $ $
? ? ? ?
```

Page No.
Date

2)
```
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=1 ; i<=5 ; j++)
{ for(j =1 ; j<=5 ; j++)
 {
 printf(" i")
 }
 printf("\n");
 }
 return 0;
 }
```

output

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

Page No.:
Date:

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

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

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

Page No.:
Date:

4)

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

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

Output

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

Page No.:
Date:

5)
```
# include <stdio.h>
int main() {
 int i, j;
for ( i=1; i<=4; i++) {
 for ( j=1; j<=4-i ; j++)
 {
  printf(" _ ");
 }
 for (j=1; j<=i ; j++) {
  printf(" * _ ");
 }
 printf("\n");
}
return 0;
}
```

Output

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

Experiment - 2

1) Search data using linear search, consider following list to perform linear search:

a = 56 36 39 57 1 0 67 59

1) key element = 1, 55

```
# include <stdio.h>
int main ()
int data[] = { 56, 36, 39, 57, 1, 0, 67, 59};
int size = sizeof(data)/sizeof(data[0]);
int k, i;
int found = 0;
printf("enter the number to be searched:");
scanf("%d", &k);
for (i=0; i< size; i++){
if( data[i] == k){
printf(" element : %d found at index: %d ",k,i);
found = 1;
break;
}
}
if (k != found){
printf("element not found.");
}
return 0;
}
```

Output

I] enter the number to be searched : 1
element : 1 found at index : 4.

II] enter the number to be searched : 55
element not found.

Page No.
Date:

2) Search data using binary search.

```
#include <stdio.h>

int main (){
 int m, k, l, h, i;
 int arr [9];

 for (i=0; i<9; i++){
 printf("Enter element : %d", i);
  scanf(" %d", &arr[i]);
 }
 printf("\nEnter the element to be searched: ");
  scanf(" %d", &k);
 l=0, h=8;
 while (l<h){
    m = (l+h)/2;
    if (arr[m] == k){
    printf(" element %d found at index %d", k, m);
    break;
    }
    else {
       if (arr[m] < k){
          l = m+1;
       }
       if (arr[m] > k){
          h = m-1;
       }
    }
 }
```

```
if( l>h){
    printf("Element not found.");
  }
return 0;
}
```

Output

```
enter element 1 : 1
    "         2 : 2
    "         3 : 3
    "         4 : 4
    "         5 : 5
    "         6 : 6
    "         7 : 7
    "         8 : 8
    "         9 : 9

enter the element to be found: 5
element 5 found at index: 5
```

Page No.
Date

3) Compare linear & Binary search

Linear Search	Binary Search
It works on both sorted and unsorted array.	It works only on sorted array.
Its algorithm type is sequential.	Its algorithm type is divide and conquer.
It is simple to implement.	It is more complex to implement.
It is used when data is small or unsorted.	It is used when data is large & ~~un~~sorted.

4) State limitations of linear search in terms of time complexity.

→ Linear search has limitations in term of time complexity, especially for large datasets. Its worst case and avg. case time complexity is O(n) meaning it may need to scan every element in the array to determine if it present or not. It makes it slowers than binary search which operates in O(logn) time on sorted array. In linear data doesn't need sorted arrays leading to poor performance as data size grows.

7/7/25

Page No.:
Date:

Extra questions

1) WAP to copy elements of one array into another array in a reverse order.

```
#include <stdio.h>
int main () {
    int arr[100], rev[100];
    int n, i;

printf(" Enter number of elements: ");
scanf(" %d", &n);

printf(" Enter array elements: \n" );
 for( i=0; i<n; i++){
  scanf(" %d", arr[i]);
 }

// Copy in reverse order
 for (i=0; i<n; i++){
    rev[i] = arr[n-1-i];
 }

printf("\nOriginal array: ");
 for( i=0; i<n; i++){
  printf(" %d", arr[i]);
 }

printf("\nReversed array: ");
 for( i=0; i<n; i++ ){
  printf(" %d", rev[i]);
 }

 return 0;
```

Output
Enter number of elements: 5
Enter array elements:
10 20 30 40 50

Original array: 10 20 30 40 50
Reversed array: 50 40 30 20 10

2) WAP to count total numbers of duplicate elements in an array.

```
#include <stdio.h>
int main (){
    int arr[100],n,i,j, count=0;

printf("Enter no. of elements: ");
scanf("%d", &n);

printf("Enter array elements:");
for(i=0; i<n; i++){
    scanf("%d", arr[i]);
}

for(i=0; i<n; i++){
    for(j=i+1; j<n; j++){
        if (arr[i] == arr[j]){
            count++;
            break;
        }
    }
}
```

```
printf("Total number of duplicate elements:%d",
return 0;
}
```

Output

Enter no. of elements : 5
Enter array elements:
2, 3 5 5 3

Total number of duplicate elements : 2

3) WAP to seperate odd and even integers into seperate arrays.

```
#include <stdio.h>
int main (){
int arr[] = {10, 21, 4, 45, 93, 66, 1};
int even[10], odd[10];
int i, evencount = 0, Oddcount = 0;
int n = sizeof(arr) / sizeof(arr[0])

for (i=0; i<n; i++){
  if (arr[i] % 2 == 0){
  even[ evencount++] = arr[i];
} else{
  odd [ Oddcount++] == arr[i];
  }
}
```

Page No.
Date

```
print("Even numbers:\n");
[~~for(i=0; i<n; i++){~~]
    ~~print~~ for(i=0; i< evencount; i++){
        printf("%d", even[i]);
    }

    printf("\n Odd number:\n");
        for(i=0; i<oddcount; i++){
            printf("%d", odd[i]);
    }
    return 0;
}
```

Output
Enter numbers:
2 3 5 6 7 8 9

Even numbers: 2 6 8

Odd numbers: 3 5 7 9

4) ~~DAP~~ to print all unique elements in an array.

```
#include <stdio.h>
int main(){
    int arr[] = {1, 2, 3, 4, 5, 6, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    int i, j, count;
```

```
printf(" Unique elements in the array: \n");
  for (i=0; i<n; i++){
      count =0;
    for (j=0; j<n; j++){
    if(arr[i] == arr[j] && i != j ){
      count ++;
      break;
    }
  }
    if( count ==0){
    printf(" %d", arr[i]);
    }
}
return 0;
}
```

Output

Unique elements in the array: 5

5b) WAP to find second smallest element in the array.

```
#include <stdio.h>
int main (){
    int arr[] ={ 10, 5, 8, 20, 2 };
    int n = sizeof (arr) / sizeof (arr[0])
```

```
int i, j, temp;

for(i=0; i<n; i++){
  for(j=i+1; j<n; j++){
      if(arr[i]> arr[j]){
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
      }
  }
}

for(i=1; i<n; i++){
  if(arr[i] =arr[0]){
  printf("Second Smallest element is= %d \n", arr[i]);
  return 0;
  }
}

printf("All elements are equal. No second smallest. \n");
return 0;
}
```

Output

Second smallest element is : 5

6) WAP to count total no. of words in a string

```
#include <stdio.h>
#include <string.h>
int main(){
    char str[100];
    int i, words = 1;

printf(" Enter a string : ");
scanf("%[^\n]", str);//[^\n] -> reads space

for (i=0; str[i]!='\0', i++){ //'\0' -> used for null character
  if (str[i] == ' '){
    words ++;
  }
}

printf(" Total number of words : %d \n", words
return 0;
}
```

Output

Enter a string : She is a tennis player.
Total number of words : 5

Page No.
Date

Experiment - 3

1) Write 'C' program to sort data from given array using Bubble sort.

→
```
#include <stdio.h>
int main(){
int a[50], i, j, n, temp;
printf("\n Enter the number of elements: ");
scanf("%d", &n);

printf("\n Enter elements:", 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++){
if (a[j] > a[j+1])
{
temp = a[j];
a[j] = a[j+1];
a[j+1] = temp;
}
}
}
printf("\n Sorted array in ascending order:");
for (i=0; i<n; i++){
printf("\n%d", a[i]);
}
return 0;
}
```

Output
Enter no. of elements : 3
enter elements : 58
98
12
sorted array in ascending order : 12
58
98

2) Write 'c' program to sort data from given array using selection sort.

→
```
#include <stdio.h>
int main (){
 int a[50], i, j, n, max, temp;
printf("Enter no. of elements:");
scanf("%d", &n);

printf("Enter %d elements:", n);
for(i=0; i<n; i++){
    scanf("%d", &a[i]);
}

for(i=0; i<n-1; i++){
    max = i;
    for(j=i+1; j<n; j++){
      if(a[j] > a[max]){
        max = j;
    }
}
```

```
temp = a[i];
a[i] = a[max];
a[max] = temp;
}

printf("\n Sorted array in descending order: ");
for (i=0; i<n; i++){
  printf("%d", a[i]);
}
printf("\n");
return 0;
}
```

Output

Enter no. of elements: 3

Enter 3 elements: 24

96

58

Sorted array in descending order: 24 96

58

24

8) Find no. of comparison required in bubble sort method for the following numbers

100, 200, 300, 400, 500.

→ 100 200 300 400 500

pass 1 100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500 Pass 2

100 200 300 400 500

100 200 300 400 500

Pass 3 100 200 300 400 500

100 200 300 400 500

100 200 300 400 500 Pass 4

Pass 5 100 200 300 400 500

Total comparisons =
4 + 3 + 2 + 1 = 10

Page No.:
Date:

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

~~100~~ 500, -20, 30, 14, 50

→ 500 -20 30 14 50

Pass 1 -20 500 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 30 500 50

-20 14 30 500 50

Pass 3 -20 14 30 500 50

-20 14 30 500 50

-20 14 30 500 50

Pass 4 -20 14 30 500 50

-20 14 30 50 500

Null
119

Page No.:
Date:

Experiment - 4

Sort elements in ascending order using Insertion Sort.

```
#include <stdio.h>
int main(){
  int i,n, a[50], temp;
printf(" Enter no. of array elements: ");
 scanf(" %d", &n);
printf(" Enter elements: \n", n);
 for(i=0; i<n; i++)
 {
   scanf("%d", &a[i]);
 }
 for(i=1; i<n; i++)
 {
    for(int j=0; j<i; j++)
    {
      if (a[i]< a[j])
      {
        temp = a[i];
          for(int k=i; k>=j; k--)
          {
            a[k+1] = a[k];
          }
          a[j] = temp;
          break;
        }
      }
    printf("After pass: %d", i);
     for(int m=0; m<n; m++){
```

```
    printf("%d", a[m]);
  }
  printf("\n");
}
printf("\n Sorted array in ascending order
using insertion sort:\n");
  for (i=0; i<n; i++)
  {
    printf("%d\n", a[i]);
  }
  return 0;
}
```

Output

Page No.:
Date:

Sort elements in ascending order using Radix Sort.

```
#include <stdio.h>
  int main () {
int a[50], output [50], count[10], int n, j, i,
 max = 0, place = 1;
printf(" Enter number of elements: ");
scanf("%d", &n);
printf(" Enter %d number : \n", n);
for ( i=0; i<n; i++)
{
 scanf ("%d", &a[i]);
 if (a[i] > max)
      max = a[i];
 }
 while (max / place > 0)
 {
  for (i=0; i<10; i++) {
     count [i] = 0;
  for ( i=0; i<n; i++){
  count [a[i] / place ) % 10] ++;
  { for (i=1; i<10; i++)
  count [i] + = count [i-1];
  {
   for (i=n-1; i>=0; i--)
  {
  int digit = (a[i] / place) % 10;
  output [-- count [digit]] - a [i];
```

```
}
for( i=0; i<n; i++)
    a[i] = output [i];
    place* = 10;
}
printf(" sorted array :\n" );
for(i=0; i<n; i++)
{
printf (" %d" , a[i]);
}
return 0;
}
```

Page No.:
Date:

[~~Algorithms for Sorting~~]

What is the output of the insertion sort after the 2nd iteration for the following:

7, 3, 1, 9, 4, 8, 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

Output after 2nd iteration:-

1, 3, 7, 9, 4, 8, 6

Sort the following using radix sort.

i) 100, 225, 390, 4130, 956, 99, 5431.

ii) 256, 99, 145, 239, 20, 18.

i)

Elements	0	1	2	3	4	5	6	7
0100	100					225		
0225								
0390	390						956	
4130	4130							
0956								
0099								
5431		5431						

Elements	0	1	2	3	4	5	6	7
0100	100							
0225			225					
0390								
4130				4130				
0956						956		
0099								
5431				5431				

Elements	0	1	2	3	4	5	6	7
0100		100						
0225			225					
0390				390				
4130		4130						
0956								
0099	0099							
5431					5431			

Page No.:
Date:

	Elements	0	1	2	3	4	5	6	7	8
9	0099	99								
	0100	100								
	4130					4130				
	0225	225								
	0390	390								
99	5431						5431			
	0956	0956								

=> 99, 100, 225, 390, 956, 4130, 5431.

9

390

99

9

956

ii)

Elements	0	1	2	3	4	5	6	7	8	9
256							256			
099						145				
145										
239										
020	020								018	
018										

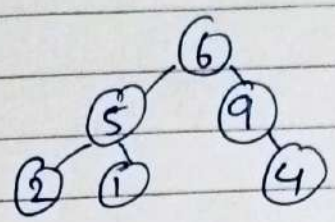
Elements	0	1	2	3	4	5	6	7	8	9
256						256				
099					145					
145				239						
239			20							
020		18								
018										

Elements	0	1	2	3	4	5	6	7	8	9
18 ~~256~~	018									
20 ~~099~~	020									
99	099									
145		145								
239			239							
256			256							

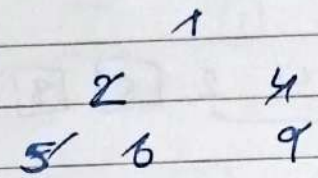
Sorted Array → 18, 20, 99, 145, 239, 256

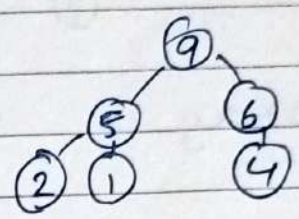
Sort the following using heap sort.

1) 6, 5, 9, 2, 1, 4.



⇒

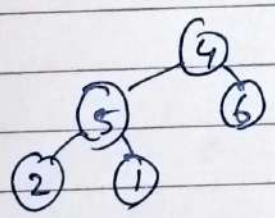


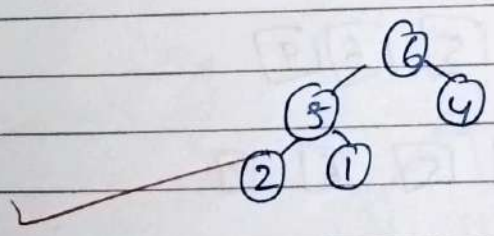


Array - $\boxed{9}$ 5 6 2 1 4

4 5 6 2 1 $\boxed{9}$

⇒

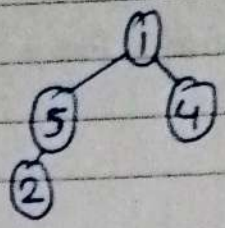


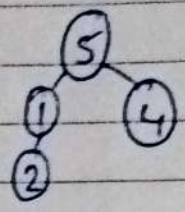


Array - 6 5 4 2 1 $\boxed{9}$

1 5 4 2 $\boxed{6}$ $\boxed{9}$

=>

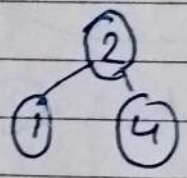


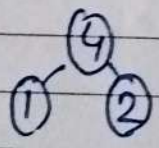


Array - 5 1 4 2 6 9

2 1 4 5 6 9

=>





Array - 4 1 2 5 6 9

2 1 4 5 6 9

=>

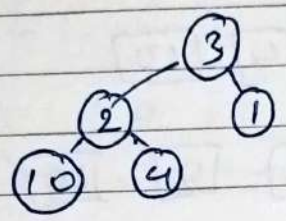
Array 2 1 4 5 6 9

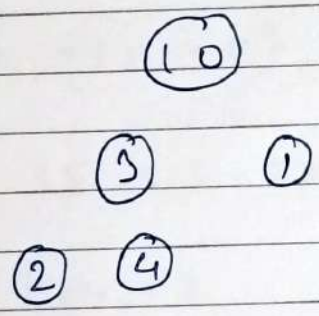
1 2 4 5 6 9

Sorted Array → 1, 2, 4, 5, 6, 9

ii) 3, 2, 1, 10, 4

⇒

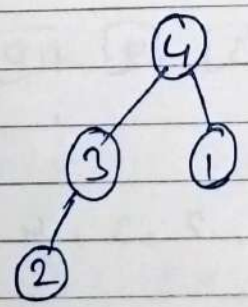




Array - 10 3 1 2 4

4 3 1 2 $\boxed{10}$

⇒

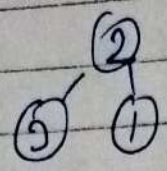


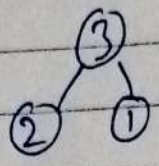
Array - 4 3 1 2 $\boxed{10}$

2 3 1 $\boxed{4}$ $\boxed{10}$

Page No.
Date.

$\Rightarrow$

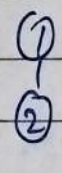


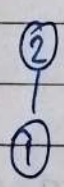


Array - 3 2 1 $\boxed{4}$ $\boxed{10}$

1 2 $\boxed{3}$ $\boxed{4}$ $\boxed{10}$

$\Rightarrow$





Array - 2 1 $\boxed{3}$ $\boxed{4}$ $\boxed{10}$

1 $\boxed{2}$ $\boxed{3}$ $\boxed{4}$ $\boxed{10}$

Sorted array - 1, 2, 3, 4, 10

Page No.:
Date:

Sort the following using shell sort.

i) 22, 18, 29, 7, 9, 55, 21, 8

→ $n = 8$,

$n/2 = \underline{4}$

22 18 29 7 9 55 21 8

{22 9}, {18, 55}, {29, 21}, {7, 8}

⇒ 9 22 18 55 21 29 7 8

$n/4 = 8/4 = 2$

{9, 22, 18, 55}, {21, 29, 7, 8}

~~{9, 18, 22, 55}~~

⇒ 9 18 22 55 7 8 21 29

$n/8 = 8/8 = 1$.

{9, 18, 22, 55, 7, 8, 21, 29}

Sorted array → 7, 8, 9, 18, 21, 22, 29, 55.

ii) 3, 10, 15, 12, 1, 5, 2, 6

→ $n = 8$

$n/2 = 8/2 = 4.$

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

{3, 1}, {10, 5}, {15, 2}, {12, 6}

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

$n/4 = 8/4 = 2$

{1, 3, 5, 10}, {2, 15, 6, 12}

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

$n/8 = 8/8 = 1$

{1, 3, 5, 10, 2, 6, 12, 15}

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

sorted array - 1, 2, 3, 5, 6, 10, 12, 15

Page No.
Date

Experiment - 5

Write code for singly linked list. Use concepts of create, search, display, insert at end; beginning: At Position, delete last; first: At Position, reverse.

```
#include <stdio.h>
#include <stdio.h>
struct Node {
  int data;
  struct Node* next;
};

struct Node* head = NULL;
void ~~insertion~~ insertAt Beginning ();
void insertAt End();
void insert At Position ();
void delete First();
void delete Last();
void delete At Position();
void display ();
void create list();
void search();
void reverse();

int main (){
  int choice;

  While (1) {
  printf(" \n -- Singly linked list - - \n");
  printf(" \n 1. Create List \n");
  printf(" 2. Insert at Beginning \n");
```

```
printf(" 3. Insert at End \n");
printf(" 4. Insert at Position. \n");
printf(" 5. Delete First Node. \n");
printf(" 6. Delete Last Node. \n");
printf(" 7. Delete at Any Position. \n");
printf(" 8. Display List. \n");
printf(" 9. Reverse List. \n");
printf(" 10. Search List. \n");
printf(" 11. Exit \n");
printf(" Enter your choice :");
scanf(" %d", &choice);

switch (choice) {
    case 1:
        createList();
        break;
    case 2:
        insertAtBeginning();
        break;
    case 3:
        insertAtEnd();
        break;
    case 4:
        insertAtPosition();
        break;
    case 5:
        deleteFirst();
        break;
    case 6:
        deleteLastEnd();
        break;
```

```
case 7:
    deleteAtPosition ();
    break;
  case 8:
    display();
    break;
    case 9:
      reverse();
      break;
      case 10:
        search();
        break;
      case 11:
        printf(" Exit.. \n");
        exit(0);
        default:
        printf(" Invalid choice! \n");
    }
}
  return 0;
}

void  createlist() {
    if (head != NULL){
      printf (" List  already  exists! \n");
      return;
    }

struct Node* newNode = (struct Node *) malloc (
                                sizeof (struct Node));
```

```
printf("Enter data for first node:");
scanf("%d", &newNode->data);
newNode->next = NULL;
head = newNode;
printf("List created with node: %d\n", newNode->data);
}

void insertAt~~End~~Beginning(){
struct Node* newNode = (struct Node*) malloc(sizeof
                        (struct Node));
printf("Enter data to insert at beginning:\n");
scanf("%d", &newNode->data);
newNode->next = head;
head = newNode;
printf("Inserted %d at the beginning.\n",
                        newNode->data);
}

void insertAtEnd(){
struct Node* newNode = (struct Node*) malloc(sizeof
                        (struct Node));
printf("Enter data to insert at end:");
scanf("%d", &newNode->data);
newNode->next = NULL;

if (head==NULL){
    head = newNode;
    printf("Inserted %d as first node.\n", newNode->
                        data);
} else{
```

```
    struct Node * temp = head;
    while (temp -> next != NULL ){
        temp = temp -> next;
    }
    temp -> next = newNode;
    printf(" Inserted %d at the end.\n", newNode->data);
  }
}

void insertAtPosition (){
  int pos;
printf(" Enter position to insert at (1-based): ");
 scanf("%d", &pos);
 if (pos <1) {
   printf(" Position must be >=1 !\n");
   return;
 }
struct Node* newNode = (struct Node*) malloc (
                    sizeof (struct Node));
printf(" Enter data for new node: ");
 scanf(" %d", &newNode->data);

  if (pos ==1){
   newNode -> next= head;
   head = newNode;
   printf(" Inserted %d at position 1.\n" );
   return;
  }
```

```
struct Node* temp = head;
  for (int i=1; i<pos-1; i++){
      temp = temp -> next;
      if (temp == NULL) {
  printf(" Position %d is out of bounds.\n", pos);
   free (newNode);
    return;
  }
}


if (temp == NULL){
printf(" Position %d is out of bounds.\n", pos);
  free(newNode);
  return ;
}

  newNode -> next = temp -> next;
    temp -> next = newNode;
printf(" Inserted %d at position %d.\n", newNode ->
                                          pos);
  }


void deleteFirst() {
  if (head == NULL) {
  printf(" List is empty!");
  return;
}

struct Node* temp = head;
    head = head -> next;
printf(" Deleted first node.\n", temp -> data);
  free (temp);
}
```

Page No.:
Date:

```
void deleteLast(){
  if (head == NULL){
  printf(" List is empty !\n");
  return;
  }
  if (head->next == NULL){
printf(" Deleted last node.", head->data);
  free(head);
  head = NULL;
  return;
  }

Struct Node* temp = head;
while( temp->next->next != NULL){
        temp = temp->next;
  }
  printf(" Deleted last node with data %d.", temp->
                                next->data);
  free ( temp->next);
  temp->next = NULL;
}

void deleteAtPosition () {
  int pos;
  printf (" Enter position to delete from (1-based)");
  scanf ("%d", &pos);

  if ( pos < 1 || head == NULL){
   printf(" Invalid position. !\n");
   return;
  }
```

```
    if (pos == 1) {
        deleteFirst();
        return;
    }
    struct Node* temp = head;
    for(int i=1; i < pos-1; i++) {
        temp = temp -> next;
    if(temp == NULL || temp -> next == NULL) {
    printf(" Position %d is out of bound.\n", pos);
        return;
    }
}
    if ( temp -> next == NULL){
    printf(" Position %d is out of bounds!\n", pos);
        return;
    }

    struct Node* nodeToDelete = temp->next;
    temp->next = nodeToDelete ->next;
    printf(" Deleted node at position.\n", pos,
    nodeToDelete -> ~~next~~ data);
    free (nodeToDelete);
    }

void display (){
    if (head == NULL){
    printf(" List is empty.\n");
    return;
}
```

Page No.:
Date:

```
struct Node* temp = head;
  printf("List:");
  while( temp != NULL){
  printf("%d ->", temp->data);
   temp = temp->next;
 }
 printf("NULL\n");
}

void reverse(){
   if(head == NULL){
  printf("List is empty!\n");
  return;
 }
  struct Node* prev = NULL;
  struct Node* current = head;
  struct Node* next;
   while( current != NULL){
       next = current->next;
       current->next = prev;
          prev = current;
          current = next;
       }
       head = prev;
     printf("list reversed successfully.\n");
 }

 void search(){
    if( head == NULL){
    printf("list is empty!\n");
    return;
 }
```

```
int value;
printf(" Enter value to be searched: ");
scanf("%d",&value);
struct Node * temp = head;
int pos = 1;
while( temp != NULL){
if(temp -> data == value){
printf(" Value %d found at position %d.\n",
                    value, pos);

return;
}
temp = temp -> next;
pos++;
}
printf(" Value %d not found.",value);
}
```

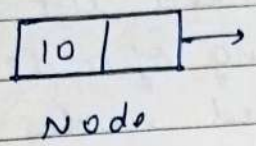
8] Difference between Array & Linked List.

Array	Linked List
• Resizing not possible.	Resizing is possible.
• Memory allocation is static.	memory can change
• Stored in contiguous memory.	Stored in non-contiguous memory.
• Memory usage is efficient.	Uses extra memory for pointers.
• O(1)	O(n)

Page No.:
Date:

8] Basic Terminology of linked list:

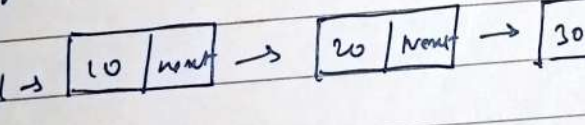
a) Node: Basic building block of linked list. It contains data & next (address of next node).



b) Head: The head is a reference or pointer to first node of list. Null or none.

c) Tail: Last node where next = NULL.

d) Next: Each node has a next reference to the following node.



e) Null Pointer: Special value that indicates that pointer does not have object/memory.

f) Empty linked list: Singly linked list with no nodes in it.

Head → null

Nyay
1/9

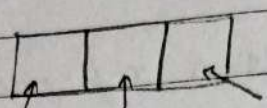
Experiment - 6

Doubly linked List

Explain circular ~~doubly~~ doubly linked lists.

In a circular ~~singly~~ doubly linked list is a type of data structure consisting of a set of sequent linked records called nodes.

Node →



Operations performed are :-

→ Insertion -
- at beginning
- at end
- at position

→ Deletion -
- delete first node
- delete last node
- delete middle node

→ Searching

→ Traversing.

Doubly Linked List.

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

void create();
void display();
void insert_begin();
void insert_end();
void insert_position();

struct node {
   int info;
   struct node* next;
   struct node* prev;
};

struct node* start = NULL;

int main() {
   int choice;
   while (1) {
      printf("\n MENU ");
      printf("\n 1. CREATE");
      printf("\n 2. DISPLAY");
      printf("\n 3. INSERT_BEGIN");
      printf("\n 4. INSERT_END");
      printf("\n 5. INSERT_POSITION");
      printf("\n 6. EXIT");
      printf("\n 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_position();
            break;
        case 6:
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice, please try again.\n");
    }
}
return 0;
}

void create() {
    struct node* temp;
    struct node* ptr;

    temp = (struct node*)malloc(sizeof(struct node));
    if (temp == NULL) {
        printf("Memory allocation failed\n");
        return;
    }

    printf("\n Enter data: ");
    scanf("%d", &temp->info);
    temp->next = NULL;
    temp->prev = NULL;

    if (start == NULL) {
        start = temp;
    } else {
        ptr = start;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        ptr->next = temp;
        temp->prev = ptr;
    }
    printf("Node created and added at the end.\n");
}
```

```
d display() {
struct node* ptr;
f (start == NULL) {
  printf("\n Empty List\n");
  return;

}
ptr = start;
printf("\n List elements are: ");
while (ptr != NULL) {
  printf("%d ", ptr->info);
  ptr = ptr->next;

}
printf("\n");

oid insert_begin() {
struct node* temp;
temp = (struct node*)malloc(sizeof(struct node));
if (temp == NULL) {
  printf("Memory allocation failed\n");
  return;

}
printf("\n Enter data to insert at beginning: ");
scanf("%d", &temp->info);

temp->next = start;
temp->prev = NULL;

if (start != NULL)
  start->prev = temp;

start = temp;
printf("Node inserted at the beginning.\n");
}

void insert_end() {
  struct node* temp;
  struct node* ptr;

  temp = (struct node*)malloc(sizeof(struct node));
  if (temp =:= NULL) {
    printf("Memory allocation failed\n");
    return;
  }
```

```
printf("\n Enter data to insert at end: ");
scanf("%d", &temp->info);
temp->next = NULL;
temp->prev = NULL;

if (start == NULL) {
  start = temp;
} else {
  ptr = start;
  while (ptr->next != NULL) {
    ptr = ptr->next;

  }
  ptr->next = temp;
  temp->prev = ptr;

}
printf("Node inserted at the end.\n");
}

void insert_position() {
  struct node* temp;
  struct node* ptr;
  int pos, i;

  temp = (struct node*)malloc(sizeof(struct node));
  if (temp == NULL) {
    printf("Memory allocation failed\n");
    return;
  }

  printf("\n Enter data to insert: ");
  scanf("%d", &temp->info);
  temp->next = NULL;
  temp->prev = NULL;

  printf(" Enter position to insert node (starting from 1): ");
  scanf("%d", &pos);

  if (pos == 1) {
    temp->next = start;
    temp->prev = NULL;
    if (start != NULL)
      start->prev = temp;
    start = temp;
    printf("Node inserted at position 1.\n");
```

```
    return;
}

ptr = start;
for (i = 1; i < pos - 1 && ptr != NULL; i++) {
    ptr = ptr->next;
}

if (ptr == NULL) {
    printf("Position out of bounds.\n");
    free(temp);
} else {
    temp->next = ptr->next;
    temp->prev = ptr;

    if (ptr->next != NULL)
        ptr->next->prev = temp;

    ptr->next = temp;
    printf("Node inserted at position %d.\n", pos);
}
}
```

17/10

Experiment - 7

Stack

```
#include <stdio.h>
#include <stdlib.h>
# define max 5

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

int S[max];
int top = -1;

int main (){
   int choice;
   printf(" \n Menu : ");
   printf(" \n 1.Push : ");
   printf(" \n 2. Pop : ");
   printf(" \n 3. Display : ");
   printf(" \n 4. Exit. ");
   do {
   printf(" \n Enter your choice : ");
   scanf(" %d", &choice);

switch(choice){
      case 1: push();
      break;
      case 2: pop();
      break;
      case 3: display();
      break;
```

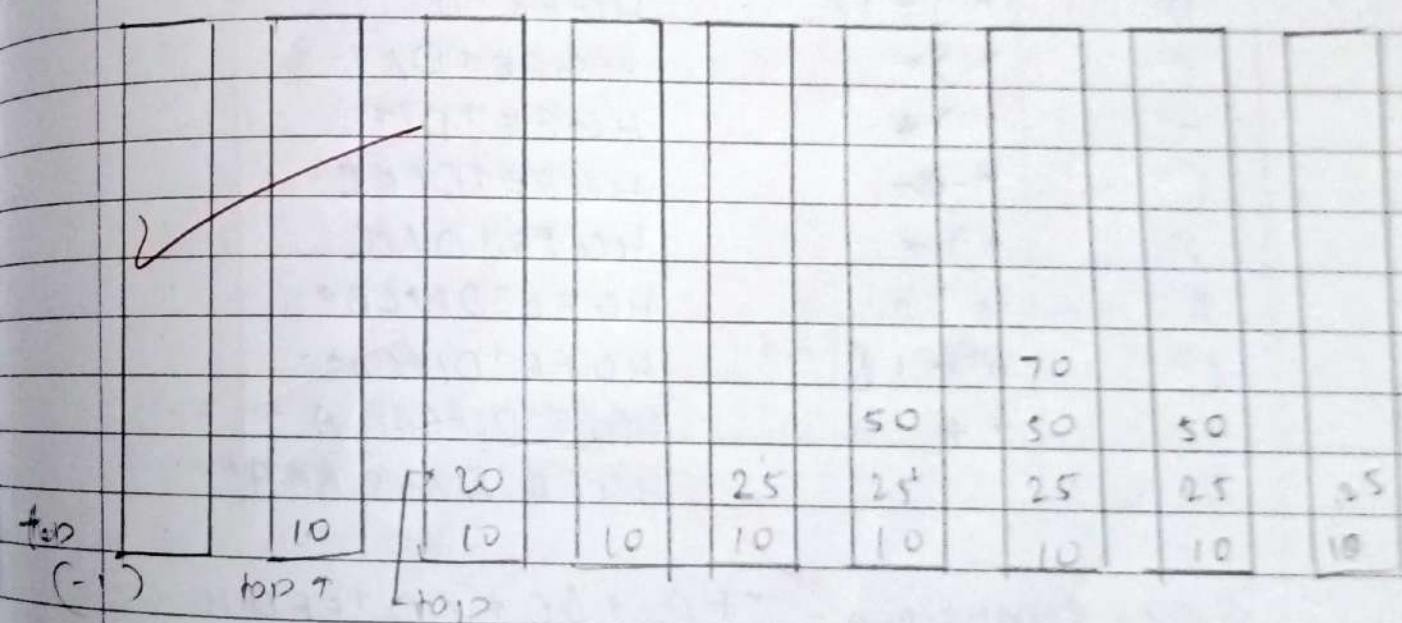
```
    case 4: exit(0);
    break;
  }
} while (choice != 4);
 return 0;
}

void push(){
 int element;
  if(top == max-1){
   printf(" Overflow.");
  }
  else{
   printf("Enter value :");
   scanf("%d", &element);
    top += 1;
   S[top] = element;
  }
}

void pop(){
 int item;
 if (top == -1){
  printf(" Underflow.");
 }
 else{
    item = S[top];
    top--;
  printf(" The deleted element is "%d", item);
 }
}
```

```
void display(){
  if (top == -1){
    printf("Empty Stack.");
  }
  else{
  printf("Stack elements are:");
  for (int i=0; i<=top; i++){ // max is capacity
  printf("\t %d", S[i]);
  }
}}
```

Q] Stack size 8, push(10), push(20), pop, push(25), push(50), push(70), pop, pop, push(100), pop.



17/10

Experiment - 8

Convert infix expression into prefix expression using stack:

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

I/P	Stack	O/P
H		H
*	*	H
)	*)	H
G	*)	HG
*	*)*	HG
)	*)*)	HG
F	*)*)	HGF
^	*)*)^	HGF
E	*)*)^	HGFE
/	*)*)/	HGFE^
D	*)*)/	HGFE^D
C	*)*	HGFE^D/
-	*)*-	HGFE^D/*
C	*-*	HGFE^D/*C
*	*)-*	HGFE^D/*C
B	*)*	HGFE^D/*CB*
C	*)* + C -out	HGFE^D/*CB*-
+	*+	HGFE^D/*CB*-A
A		HGFE^D/*CB*-A+*

Final Expression - *+A-*BC*/D^EFGH

Page No.
Date

Write a C program to evaluate infix or Postfix 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;
return 0;
}

int main () {
char exp[100];
char *e, x;
```

```
printf("enter the expression:");
scanf("%s", &exp);
printf("\n");
e = exp;

while (*e != '\0')
{
 if (isalnum(*e))
 printf("%c", *e);

 else if (*e == '(')
 push(*e);
 else if(*e == ')')
 {
 while ((x=pop()) != '(')
 printf("%c", x);
 }
 else
 {
 while (priority(stack[top]) >= priority(*e))
 printf("%c", pop());
 push(*e);
 }
 e++;
}

while (top != -1)
{
 printf("%c", pop());
}
return 0;
}
```

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

Page No.
Date:

Evaluate prefix expression using Stack:

Expression:- + - * + 12 / 4 2 1 $ 4 2

I/P	Operand1	Operand 2	Result	Stack
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
+	5	16	22	22

Algorithm

1) Infix to postfix

→ Read expression from left to right.
→ If an operand (A-Z, 0-9) is encountered add it directly to output.
→ Pop operator to stack and opperand to output.
→ Check the priority of operators and pop it to

output accordingly.

- When parenthesis occurs the operators present are moved to stack & then output.
- After scanning the entire expression, pop any remaining operators from stack to output.

2) Infix to prefix.

- Read the expression from right to left.
- Check the priority of the operators and act accordingly.
- Prefix is the reverse expression of the postfix.

3) Evaluation of postfix.

- Scan the elements from left to right.
- If operand push to stack & if operator push to output.
- Perform arithmetic operations & push to the result & then stack.
- When input expression is completely traversed pop operand stack & return the value.

Page No.:
Date:

Evaluation of prefix.

- Start from right to left.
- It is the reverse of postfix evaluation
- The same method is used as postfix evaluation.

17/10

Experiment - 9

Queue

1) Perform primitive operations on Linear Queue - Insert, Delete, Display with array size [10]. Insert (10), Insert(50), Delete, Insert (100), Delete, Insert(25), Insert (200).

```
#include <stdio.h>
# include <stdlib.h>
# define max 10
void insert();
void delete();
void display();
int queue [max];
int front = -1;
int rear = -1;
int main(){
    int choice;
    do {
      printf("\n menu: ");
      printf("\n1. Insert ");
      printf("\n2. Delete ");
      printf("\n3. Display ");
      printf("\n4. Exit);
      printf("\n Enter your choice: ");
      scanf("%d", &choice);
      switch (choice) {
      case 1 : insert();
       break;
```

```
        case 2: delete();
        break;
        case 3: display();
        break;
        case 4: Exit(0);
        default : printf("\n Invalid choice!\n");
    }
} while (choice != 4);
return 0;
}
void insert(){
    int element;
    if( rear == max-1){
    printf("\n Queue Overflow");
    }
    else {
    printf("\n Enter value to insert:");
    scanf("%d", &element);
    if( front == -1) {
        front = 0;
    }
    rear = rear + 1;
    queue[rear] = element;
    printf("\n Element %d inserted!\n", element
    }
}
void delete(){
    int item;
    if ( front == -1 || front > rear){
    printf("\n Queue is Overflow!");
```

```
}else {
  item = queue[front];
    front ++;
  printf("\n The deleted element is :", item);
  }
}
void display () {
  if (front == -1 || front > rear) {
  printf("Queue is empty!\n");
} else {
  printf("\nQueue elements (front to rear):");
  for (int i = front; i <= rear; i++){
      printf("%d", queue[i]);
  }
  printf("\n");
  }
}
```

Explain the concept of priority queue.

→ A priority queue is a data structure where each element has a priority, and elements with higher priority are served before others.

Unlike regular queues (FIFO), priority queues process elements based on priority.

Types -

Ascending Priority Queue.

Descending Priority Queue.

Q] Write algorithm for insertion & deletion for linear queue.

→ Insertion:

1. Check for overflow condition.
2. Check if the queue is empty. In case the queue is empty, then both FRONT & REAR are set to zero, so that the new value can be stored at the 0th location, if the queue already has some values, then REAR is incremented so that it points to next location in the array.
3. The value is stored at the location pointed by REAR.
4. Exit.

Deletion:

1. We check for underflow condition. An underflow condition occurs if FRONT = -1 or FRONT > REAR. However, if queue has some values, then FRONT is incremented so that it now points to the next value in the queue.
2. Exit.

Q] Write algorithm for insertion & deletion for circular queue.

→ ~~Insertion~~:

Deletion:

1. Check if queue is ~~full~~ empty. [FRONT == -1] - Underflow.
2. Store the elements in front. Queue [FRONT]
3. If FRONT == REAR, set FRONT = REAR = -1.
4. Exit.

Insertion:
1. Check if queue is full. [If (REAR + 1) % size = FRONT.] - Overflow.
2. If queue is empty. [FRONT == -1] set [FRONT = REAR = 0].
3. Go for REAR.
   [REAR = (REAR + 1) % size].
4. Insert the new element at Queue [REAR].

8] Write applications of Queue.
→ a) Serving requests on a single shared resource, like a printer, CPU task scheduling.
b) Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come First served.
c) FCFS job scheduling.
d) Online banking fund transfer requests.

Q] array size 10 : Insert(10), Insert (50), Delete, Insert (100), Insert (20), Delete, Insert (25), Insert (200).

0	1	2	3	4	5	6	7	8	9
10									
10	50								
~~10~~	50								
~~10~~	50	100							
~~10~~	50	100	200						
~~10~~	~~50~~	100	200						
~~10~~	~~50~~	~~100~~	20						
~~10~~	~~50~~	~~100~~	20	25					
~~10~~	~~50~~	~~100~~	20	25	200				

Experiment - 10

Write code for circular queue.

```
#include <stdio.h>
#include <stdlib.h>
# define SIZE 5

void enqueue();
void dequeue();
void display();
int queue[SIZE];
int front = -1 , rear = -1;
int main(){
   int ch;
printf(" 1. ENQUEUE \n 2. DEQUEUE \n 3. DISPLAY
        \n 4. EXIT ");

do {
 printf(" \n Enter your choice : ");
 scanf(" %d", &ch);
 switch (ch){
      case 1: enqueue();
       break;
      case 2: dequeue();
       break;
      case 3: display();
       break;
      case 4: exit(0);
       break;
      default :
      printf(" \n Invalid choice! \n");
 }
```

```
} while (ch != 4);
  return 0;
}
void enqueue (){
  int value;
if ((front == 0 & rear = SIZE - 1) ||
      (rear + 1 == front))
  {
  printf("\n Queue is full \n");
  }
  else
  {
  printf("\n Enter value to insert:");
  scanf("%d", &value);
  if (front == -1)
      front = 0;

  rear = (rear + 1) % SIZE ;
  queue [rear] = value;
  printf("%d insert into queue", value);
  }
}


void dequeue (){
  if (front = = -1)
  {
  printf("\n Queue is Empty");
  }
  else
  {
```

```
printf("\n Deleted %d from queue :"
    queue[front]);
  if (front == rear)
  {
    front = rear - 1;
  }
  else
  {
    front = (front + 1) % SIZE;
  }
}
}

void display(){
  if (front == -1)
  {
  printf("\n Queue is empty ! \n");
  }
  else
  {
    int i = front;
    printf("\n Queue elements are:");
    while (1) {
    printf ("%d", queue[i]);
      if (i == rear)
        break;
      i = (i+1) % SIZE;
    }
    printf("\n");
  }
}
```

Page No.:
Date:

Write & explain applications of priority queue.

a. Traffic Light control:
Manages signal priorities based on real-time traffic sensor data.

b. Operating System Algorithms:
Used for scheduling process to execute high priority first.

c. Huffman codes:
Priority queue helps build Huffman trees for efficient data compression.

17/10

Experiment - 11

Create binary tree & traverse Inorder, Preorder & Postorder.

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

struct node {
  int item;
  struct node * right;
  struct node * left;
};

struct node * createnode (int item){
struct node* newNode = (struct node*) malloc
        (sizeof (struct node));
      newnode -> item = item;
      newnode -> left = NULL;
      newnode -> right = NULL;
      return newNode;
}

struct node* insertAtLeft (struct node * root,
                    int item){
      root-> left = CreateNode (item);
      return root -> left;
}

struct node * insertAtRight (struct node* root,
                    int item){
```

```
    root -> right = createNode (item);
    return root -> right;
}

void inorder (struct node * root){
if(root == NULL)
    return;
inorder (root -> left);
printf ("%d", root -> item);
    inorder (root -> right);
}

void preorder (struct node * root){
if (root == NULL)
    return;
printf ("%d", root -> item);
preorder (root -> left);
preorder (root -> right);
}

void postorder (struct node * root){
if (root == NULL)
    return;
postorder (root -> left);
postorder (root -> right);
printf ("%d", root -> item);
}
```

```
int main() {
  struct node* root = createNode (40);

  insertAtLeft (root, 28);
  insertAtRight (root, 87);
  [~~insertAtleft (root, 60);~~]
  insertAtleft (root->left, 60);
  insertAtRight (root->left, 30);
  insertAtleft (root->right, 50);
  insertAtRight (root->right, 95);

  printf(" Inorder Traversal:");
  inorder (root);

  printf(" In Preorder Traversal:");
  preorder (root);

  printf(" In Postorder Traversal:");
  postorder (root);

  return 0;
}
```

Inorder : 60 28 30 40 50 87 95

Preorder : 40 28 60 30 87 50 95

Postorder : 60 30 28 50 95 87 40

Page No.:
Date:

Q Creation of Tree.

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

    struct node{
        int item;
        struct node * left;
        struct node * right;
    };

Struct node * Createnode (item)
{
Structnode * newNode = malloc (sizeof (struct node)
        newnode -> item = item;
        newnode -> left = NULL;
        newnode -> right = NULL;
        return newNode;
}

struct node * insertAtleft (struct node* root, int item
{
    root -> left = Createnode (item);
    return root -> left;
}

struct node * insertAtRight (struct node * root, int item
{
    root -> right = Createnode (item);
    return root -> right;
}
```

6/11

Experiment - 12

Page No.
Date:

8.1) Graph

```
#include <stdio.h>
#define V 5

void init (int arr[V][V]){
    int i, j;
    for (i=0; i<V; i++)
      for (j=0; j<V; j++)
        arr[i][j]=0;
  }

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

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

int main(){
  int adjMatrix[V][V];
  init (adjMatrix);
```

```
insertEdge (adjMatrix, 0, 1);
           (adjMatrix, 0, 2);
           (adjMatrix, 1, 2);
           (adjMatrix, 2, 0);
           (adjMatrix, 2, 3);
           (adjMatrix, 0, 3);
           (adjMatrix, 0, 0);
           (adjMatrix, 1, 3);
           (adjMatrix, 2, 4);
           (adjMatrix, 4, 3);

printAdjMatrix(adjMatrix);
return 0;
}
```

Page No.:
Date:

8] Using linked list.

```
#include <stdio.h>
#include <stdlib.h>
#define V 5

struct Node {
    int dest;
    struct Node * next;
};

struct ~~Node~~ graph {
    struct Node * adj[V];
};

struct Node * newNode (int dest) {
  struct Node * temp = (struct Node*) malloc (sizeof
                                    (struct Node));

    temp -> dest = dest;
    temp -> next = NULL;
    return temp;
}

struct graph * create Graph() {
  struct Graph * graph = (struct Graph*) malloc
                        (sizeof (struct graph));

  for(int i = 0; i < V; i++)
    graph -> adj[i] = NULL;
  return graph;
}
```

```
void addEdge(struct Graph* graph, int src,
                                  int dest){
    struct Node* temp = newNode (dest);
    temp -> next = graph -> adj[src];
    graph -> adj[dest] = temp;
}


void printGraph (struct Graph* graph){
    printf ("Adjacent List: \n");
    for (int i=0; i<v; i++) {
      printf ("%d ->", i);
    struct Node* temp = graph -> adj[i];
    while (temp){
      printf ("%d", temp -> dest);
      temp = temp -> next;
    }

    printf ("\n");
  }
}


int main(){
  struct Graph* graph = createGraph (5);

    addEdge (graph, 0, 1);
            (graph, 0, 2);
            (graph, 0, 3);
            (graph, 0, 4);
            (graph, 1, 3);
            (graph, 2, 3);
            (graph, 2, 4);
            (graph, 3, 4);
```

```
printGraph( graph);
return 0;
}
```

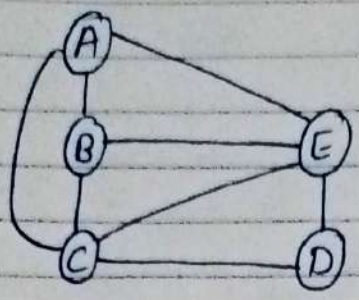
Explain Terminologies

1. vertex - Each node at graph is represented as a vertex. In the examples, labeled circle is known as vertex.

2. Edge - Edge represents a path between 2 vertices or a line between two vertices.

3. Adjacency - Two nodes or vertices are adjacent if they are connected to each other through an edge.

4. Path - Path represents a sequence of edges between the two vertices.



Matrix :

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	0	1
C	1	1	0	1	1
D	0	0	1	0	1
E	1	1	1	1	0

list :

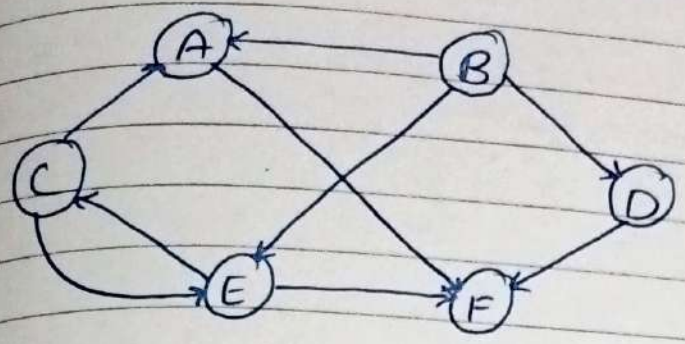
A → B → C → E

B → C → A → E

C → B → A → D → E

D → E → C

E → A → B → C → D



Matrix:

	A	B	C	D	E	F
A	0	0	0	0	0	1
B	1	0	0	1	1	0
C	1	0	0	0	1	0
D	0	0	0	0	0	1
E	0	0	1	0	0	1
F	0	0	0	0	0	0

List

A → F

B → A → E → D

C → A → E

D → F

E → C → F

F → NULL

Nil 6/11