Experiment 1:

1) Find avg of 10 number using an array.

```
# include <stdio.h>
  int main () {
    int num [10];
   int i, sum = 0;
    float avg;

    printf ("Enter 10 numbers:");

  for (i=0; i<10; i++)
{
  scanf ("%d ", & num [i]);
    sum + = num [i];
  }

avg = sum /10;

printf ("Average of 10 numbers = %.2f ", avg);

  return 0;

}
```

Output: Enter 10 element : 67	64
87	97
64	64
69	62
07	Avg of no. 58
64	

__/__/__

2) Display the following pattern:

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

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

Output

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

3) Find First 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 ;
 }
}

 if (found ) {

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

}
```

Output

First repeating element is 2.

__/__/____

4) Find greatest and smallest element in an array.

```
# include <stdio.h>
  int main ()
{
   int arr [] = {45, 3, 88, 12, 78, 34, 23};
   int n = size of(arr) / size of (arr [0]);
    int max = arr [0]
    int min = arr [0];

   for (int i = 1; i<n; ;++ );
{
   if (arr [i] > max) {
       max = arr [i];
    }
 if (arr [i] < min) {
     min = arr [i];
  }
}
  printf ("Greatest element = %d ", max);
  printf (" Smallest element = %d ", min );

   return 0;
}
```

Output:

Greatest element = 88
Smallest element = 3

5) Write a program squaring the odd position elements.

```
# include <stdio.h>
 int main ()
{
 int arr [] = { 7, 6, 9, 2, 8, 4, 3 };
 int n = sizeof (arr) / sizeof (arr[0]);
 int i;

 printf ("Original array : \n");

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

 for (i=1 ; i<n ; i+=2) {
    arr [i] = arr [i] * arr [i];
 }

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

11/1/25

Output

Original array:	New array:
7, 6, 9, 2, 8, 4, 3	7, 36, 9, 4, 8, 16, 3.

__/__/____

Extra:

1) Declare an Array of subject - Accept subject marks from user. Calculate the avg or percentage.

```
#include <stdio.h>
    int main (){

  int marks [5]
  char subject [5][20] = {"Math", "Phy", "Chem",
          "Eng", "comp"};

  int total = 0;
  float percentage;

  for(int i=0; i<5; i++){
     printf("Enter marks for %s: ", subject [i]);
  scanf("%d", &marks [i]);

  total += marks [i]
  }

  percentage = (float) total /5;

  printf("\n Total Marks = %d\n", Total );
  printf(" Percentage = %.2f %%\n", percentage );

  return 0;

  }
```

Output

```
Enter marks for each subject
Math: 44
Phy: 76
Chem: 88
Eng: 96
Com: 57
Total marks: 361
Percentage: 72.20%
```

__/__/____

2) Write a program to print accept number n from user and check it its even or odd.

```
#include <stdio.h>
    int main () {
    int n;

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

    if (n%2 == 0)
      printf ("Is even");
  else
      printf (" is odd");

      return 0;

}
```

Output

Enter a number: 7747
7747 is odd.

3) Write a program to print prime number betn 1-50.

```
# include <stdio.h>
  int main () {

  int i, j, is Prime;

printf ("Prime numbers between 1 and 50
                                are");

for (i=2 ; i<=50 ; i++) {
      is Prime = 1;

  for (j=2 ; j<i ; j++) {
      if (i % j == 0) {
          is Prime = 0

    }
}
    if (is Prime == 1)
      printf ("%d", i);

  }

    return 0;

}
```

Output

Prime numbers between 1 and 50 are:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

__/__/___

4) Write a menu driven program where u give user choices like 1-Monday to 7. sundy and if user chooses 6 the the output should give whether the day o6 iJ weekend or weekday.

```
#include <stdio.h>
 int main () {

  int choice

printf ("choose a day (1-7) ;
printf ("1. Monday");
printf ("2. Tuesday ");
printf ("3. Wednesday");
printf ("4. Thursday ");
printf ("5. Friday");
printf ("6. saturday");
printf ("7. Sunday");

printf("Enter your choices");
scanf ("%d", & choice);

   Switch (choice) {

case 1;
    printf ("Monday is a Weekday.\n");
Cas 2;
   printf ("Tuesday is a Weekday.\n ");
eas 3;
   printf ("Wednesday is a Weekday.\n");
case 4
   printf ("Thursday is a Weekday.\n");
```

```
case 5:
    printf("Friday is a Weekday.\n");
case 6:
    printf("Saturday is a Weekend.\n");
case 7:
    printf("Sunday is a Weekend.\n");
}
return 0;
}
```

Output:

Choose a day by entering a number

1. Monday
2. Tuesday
3. Wednesday
4. Thursday
5. Friday
6. Saturday
7. Sunday

Enter your choice (1-7) = 6
Saturday is a weekend.

5) Write a program to accept numbers from user and to check it is palindrome or not.

```
#include <stdio.h>
    int main () {
    int num , originalNum , reversedNum =0,
                                remainder;

    printf ("Enter an integer: ");
    scanf ("%d", &num);

    originalNum = num;

    while (num!= 0) {
      remainder = num %. 10;
      reversed Num = reversed Num * 10 + remainder;
    }
    if (original Num = reversed Num ) {
        printf ("%d is a palindrome.\n", original
                                    Num);
    } else {
        printf ("%d is not a palindrome. \n",
                                    original Num);
    }
    return 0;
    }
```

Output.

```
Enter an integer : 67
67 is not a palindrome.
```

Exp. 2.

Q.) Search data using linear search, consider following list:

56, 36, 89, 57, 10, 67, 59

Search item 1 from the above list and write down the item found or not on the procedure

Soln :- **K = 1**

```
# include <stdio.h>
  int linearsearch (int arr[], int size, int item)
{
  for (int i = 0 ; i < size ; i++ ) {
  if (arr[i] == item) return i;
}
   return -1 ;
}
  int main ( ) {
 int number = { 56 , 36 , 89, 57, 1, 0, 67, 59};
int size = size of (number) / size of (numbers[0]);
    int searchitem = -1
int foundindex = Linear Search (numbers, size,
                                searchitem);
if (found Index != -1) printf ("Item %d found at
 Index %d \n", searchitem, found index);
else printf ("Item %. not found in the list /n",
searching
return 0;
}
```

__/__/____

OIP

Item 1 found at index 4

Q.) Search data using Binary search.

```
# include <stdio.h>
  int main () {
   int s;
printf ("Enter the number of Elements : ");
  scanf ("%d", &s);
 int k, l=0, h=s-1, f=0, a[5];
printf ("Enter sorted integers : \n");
for (int i=0; i<5; i++) { scanf ("%d", &a[i]);
                                          }
printf ("Enter element to search : ");
  scanf ("%d", &k);
  while (l <= h) {
     int m = l + h/2
if (a[m] == k { f = m; break; }
if (a[m] < k) { l = m+1; }
  else { h = m-1; }
}
  (f != 0) ? printf ("Element %d found at
index %d\n"): printf (
printf("Element not found.\n");
  return 0;
}
```

__/__/____

Q. Compare linear and binary search.

	Linear Search	Binary search
	The data can be unsorted or sorted	The data must be sorted (either ascending or descending)
	Sequential search (checks each ele. one by one).	Divide & conquer (Repeatedly divides the search spaces in half).
Time complexity	O(n) linear	O(logn) logarithmic
Bat-case	The target item is the first element	The target item is the middle element or at the end.
Efficiency	Slower, Especially for large datasets.	Significantly faster for large database.

__/__/__

Q. State limitations of linear Search of time complexity.

→ The primary limitations of linear seach regarding its time complexity are related to its performance on large dataset

→ The time complexity of linear seach is typically expressed in Big O notations.

① Worst Case time Compexity :
→ Linear search takes a long time because it has to check on average, half of the items.

② Average - Case time complexity :
→ The bigger the item, the more time it takes
(Not a Good choice for large amount of data)

③ Inefficiency of large ~~database~~ sets :
→ It has no advantage even if the list is already sorted, unlike more efficient method.

④ No. Advantage from sorted data
→ Worst - Case scenario (when the item is at very end or not in the list), it has to checks every single item.

O/P (1)

Enter no. of element : 5

Eter sorted int :

1 2 3 4 5

Enter the element to search : 3

Element 3 found at index 2

O/P => (2)

Enter no. of element : 5

Enter sorted int : 1 2 3 4 5

Enter the element to search : 8

Element not found.

24/7/25

__/__/__

Extra:

1) Write a program to copy the 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 element : ");
  scanf ("%d", &n);

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

 for (i=0; i<n; i++) {
   rev [i] = arr [n-1-i]
 }

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

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

   return 0;
}
```

__/__/__

O/P

Enter number of elements : 7
Enter the elements :
8
9
6
5
3
4
0
Original array: 8 9 6 5 3 4 0
Reversed array: 0 4 3 5 6 9 8

2) WAP in C to count total number of duplicate element in an array.

```
#include <stdio.h>

int main () {
    int arr[100], n, i, j, count = 0;

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

    printf("Enter array element: \n");
    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\n",
                                        count);

return 0;

}
```

O/P

```
Enter number of elements : 8
Enter array elements:
5
4
6
7
6
6
0
0
```

Total number of duplicate elements = 3.

Q.3)

```
# include <stdio.h>
int main() {
  int arr[] = {1, 2, 3, 2, 4, 5, 1, 6};
  int n = size of (arr) / sise of (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;
      }
    }
  }

  return 0;
}
```

O/P

Unique elements ~~are~~ in the array are:
3 4 5 6.

Q.4)

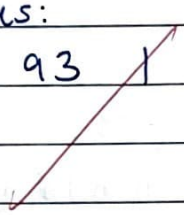
```
#include <stdio.h>

int main () {
    int arr[] = {10, 21, 4, 45, 66, 93, 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];
        }
    }

    printf("Even numbers : \n");
    for (i=0; i< evenCount; i++) {
        printf("%d ", even[i]);

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

__/__/____

O/P

Even numbers:
10 4 66

Odd numbers:
21 45 93



Q.5)

```
#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;
}
```

__/__/____

O/P

Second smallest element is : 5.

Q.6)

```
#include <stdio.h>
#include <string.h>
int main() {
    printf("\n Word Count \n");
    printf(" Enter a string : ");
        char[100];
    scanf("%.s ", s);
    int wc=0 ; iwf=0;
    for(i=0 ; s[i] != "\0" ; i++) {
        if(s[i]== ' ' || s[i]== '\t' || s[i]=='\n'
                                                    //
        s[i]=='r' ) { iwf =0; }
    else if(iwf == 0) { wc++ ; iwf =1; }
}   printf("words : %d\n", wc);
    return 0;
}
```

Exp : 3

1) Sort elements in ascending order using Bubble sort.

→

```
# include <stdio.h>
void bubblesort (int arr [] , int n) {
        int i, j , temp;
for (i=0 ; i<n-1 ; i++ ) {

for (j=0 ; j<n-i-1 ; j++) {

   if (arr [j] > arr [j+1] {

      temp =arr [j] ;
      arr [j] = arr [j+1];
      arr [j+1] = temp;
    }
  }
}

void print array (int arr [], int size)
   int i;
   for (i=0; i<size ; i++ ) {
      printf ("%d ", arr [i]);
   }
      printf ("\n");
   }
```

```
int main () {
    int arr [] = { 64, 34, 25, 12, 22, 11, 90};
    int n = size of (arr) / size of arr [0]);

    printf(" original array: \n");
    print array (arr, n);

    bubblesort (arr, n);
    printf (" sorted array in ascending order: \n");
    print array (arr, n);

    return 0;
}
```

O/P

ascending order = (bubble sort): 1 2 3 5 8

2) Sort elements in descending order using selection sort ?

```
# include <stdio.h>
void selectionsortDesc(int a[], int n){
for (int i = 0 ; i<n-1 ; i++ ){
    int max = i;
for (int j = j+1 ; j <n ; j++) {
    if (arr [j] >a [max] )
            max = j;
}
  int temp = a [i];
   a[i] = a[max];
   a[max]= temp;
  }
}
int main () {
  int a[] = { 5, 2, 4, 3, 1 };
    int n =5;
selectionsort Desc (a, n).
printf (" Descending order (selection sort : ");
  for (int i=0 ; i<n ; i++)
     printf ("%d ", a [i] );
      return 0;
    }
```

OIP

Descending order (selection sort) = 8 5 3 2 1

3) Find the number of comparisons required in bubble sort method of the following list having 5 numbers : 100, 200, 300, 400, 500.

P1
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

4) Sort the gi
order using
show diagram
iteration of fo

P1
500 , -20
-20 , 500 , 3
-20 , 500 , 3
-20 , 500 ,
-20 , 500 , 3

P2
-20 , 500 , 3
-20 , 30 , 500
-20 , 14 , 500
-20 , 14 , 500

P3
-20 , 14 , 500
-20 , 14 , 30 ,
-20 , 14 , 30 ,

P4
-20 , 14 , 30 ,
-20 , 14 , 30 ,

4) Sort the given arrays in ascending order using selection sort method and show diagrammatic representation of every iteration of for loop : 500, -20, 30, 14, 50

500, -20, 30, 14, 50

-20, 500, 30, 14, 50

P_1 -20, 500, 30, 14, 50

-20, 500, 30, 14, 50

[-20], 500, 30, 14, 50

-20, 500, 30, 14, 50

-20, 30, 500, 14, 50

P_2 -20, ~~30~~ 14, 500, 30, 50

[-20], [14], 500, 30, 50

-20, 14, 500, 30, 50

P_3 -20, 14, 30, 500, 50

[-20], [14], [30], 500, 50

-20, 14, 30, 500, 50

P_4 -20, 14, 30, 50, 500

Exp: 4

1) Sort element in ascending order using insertion sort.

```
# include <stdio.h>
    int main ()
{
 int a[50], n, i, j, temp;
printf("\n enter the number of elements: ");
 scanf("%d", &n);
 printf("\n ener %d elements: " 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 (int k=i-1; k>=j; k--)
  {
   a[k+1] = a[k];
  }
   a[j] = temp;
 Break;
  }
}
}
```

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

O/P

enter the number of elements: 5
enter 5 elements: 3 5 9 1 2

After pass 1: 3 5 9 1 2
After pass 2: 3 5 9 1 2
After pass 3: 1 3 5 9 2
After pass 4: 1 2 3 5 9

Sorted array: 1 2 3 5 9

__/__/__

2) Sort element in ascending order using radix sort:

```
#include <stdio.h>
  int main ()
{
 int a [50] , output [50] , cout [10];
  int n, j, i, max=0, place =1;
  printf("enter number of elements :");
  scanf("%d", &n);

  printf("enter %d numbers \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++)
    cout [i]=0 ;
  for (i=0 ; i<n ; i++)
    cout [a(i) / Place; %. \n] ++ ;
   for (i=1 ; i<10 ; i++)
    cout [i] += cout [i-1];
  for (i=n-1 ; i>=0; i--){
    int digit= (a [i] / Place ) %. 10;
   output [-- cout [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;
}
```

Extra Q.

1) Algorithm for selection sort:

→
- Start the program.
- Start with the first element.
- Find the smallest element in the unsorted part of the array.
- Swap it with the first element of the unsorted array
- Swap it with the first element of the unsorted part.
- Move the bound array of the sorted / unsorted part by one element.
- Reapeat until the array is sorted.

2) Algorithm for insertion sort.

→
- Start the program with the second element of the list (because the first element is already 'sorted').
- Pick the current element and call it the "key".
- Compare the key with the elements before it (i.e the sorted part of the list)

- Shift all the elements that are greater than key on position to the right.
- Insert the key into the correct position where it's greater than the element after than the element before it and less than element after it.
- Repeat this process for each element from left to right until the entire list is sorted.

3) Algorith for radix sort:

→
- Stat the program.
- Define 10 queues each represent a bucket for each digit from 0 to 9.
- Considered the least significant digit of each number in the list which is to be sorted.
- Insert each number into their respective queue based on the least significant digit.
- Group all the number from queue 0, to queue 9 in the order they have is sorted into their respective queues.
- Repeat from step 3 based on the next least significant digit.
- Repeat from step 2 until all the grouped based on the most significant digit.

4) Algorithm for heap sort:

→
- Start the program
- Build a max Heap from the input data.
  - A max Heap is a binary tree, where the parent is always greater than or equal to its children.
  - This step rearranges the array so that the largest value is at the root (first index).
- Swap the root (largest element) with the last element in the Heap.
- This puts the largest value at the end of the array where, it belongs in sorted order.
- Reduce the size of the Heap.
- Heapify the root again to restore the max Heap property.
- This ensures the next largest element moves to the root.
- Repeat step 2-4 until the Heap size is 1.

12/11

Exp 5: singly linked list.

1) WA menu driven C program that implements singly linked list for the following: operation: create, insert node at beginning at a, insert node at in middle, insert node at end.

→

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

Struct Node {
   int data;
   Struct Node * Next;
};

Struct Node * Head = Null;
void create (int value) {
Struct Node * Newnode = (Struct Node*)
malloc (sizeof (Struct Node));
Struct Node * temp = Head;
New Node → data = value;
New Node → Next = Null;
if (Head == Null) {
    Head = Newnode;
} else{
    while ( temp → Next != Null)
   temp = temp → Next;
   temp → Next = Newnode;
  }
}
```

__/__/___

```
Void insert at Beginning (int value){
 Struct Node * NewNode = (Struct Node *)
malloc (size of (Struct Node));
   New Node → data = value;
   New Node → Next = Head;
        Head = New Node;
 }
Void insert At end (int value) {
        create (value);
 }

void insert At Middle (int value, int position
                                          )


{ Struct Node * New node = (struct Node *)
malloc (size of (struct Node));
 Struct Node * temp = Head;
   int i = 1;
if (position == 10) {
   insert At Beginning (value);
  return;
}
  while (i < position - 1 && temp != Null) {
        temp = temp → Next;
}  i++

   if (temp == Null)
      printf ("position out of range. \n");
  return 0;
}
```

```
NewNode -> data = value;
NewNode -> Next = temp -> Next;
    temp -> Next = NewNode;
}
 void display () {

 Struct Node * temp = Head;
 if (Head == = Null) {
 printf ("List is empty. \n");

  return 0;
}
  printf (" linked List: ");

while (temp = Null) {
printf (" %d -> ", temp -> data );

   temp = temp -> Next ;
}
  printf (" Null \n");
}
  int main () {
  int choice, values, pos;

while (1) {
  printf (" menu \n");
  printf (" 1. create (add Node at beginning/ end \n");
  printf (" 2. Insert
```

__/__/____

```
printf("1. create (add Node at end\n");
printf("2. Insert at Beginning\n");
printf("3. Insert at middle (after position.)\n");
printf("4. Insert at end\n");
printf("5. Display\n");
printf("6. Exit\n");

printf("Enter your choice:");
scanf("%d", &choice);

switch (choice) {

Case 1:
    printf("enter value to add:");
    scanf("%d", &value);
     create (value);
     Break;

Case 2;
    printf("enter value to insert at
     beginning");
    scanf("%d", &value);
  insert At Beginning (value);
Break;

case 3;
    printf("Enter position to insert after:");
      scanf("%d", &position);
    printf("Enter value to insert:");
    scanf("%d", &value);
      insert At middle, value, pos);
     Break;
```

__/__/___

```
Case 4;
    printf("Enter value to insert at end: ");
    scanf("%d", &value);
    insertAtEnd(value);
    Break;

Case 5:
    display();
    Beak;

Case 6;
    exit(0)
default:
    printf("Invalid choice; try again\n");
    }
}
    return 0;
}
```

O/P menu:

1. Creat (add Node at end)
   insert at Beginning
   insert at middle (your position)
   insert at end.
   Display
   exit.

enter your choice: beginning
invalid choice try again:

(ii)

```
# include <stdio.h>

struct Node {
    int data;
    struct Node * Next;
};
struct Node * Head = Null;
void Insert At end (int value)
Struct Node * New Node = (Struct Node *)
malloc(size of (struct Node));
 Struct Node * temp = head;
   New Node → data = value;
   New Node → Next = Null;

   if (Head = = Null) {
       Head = New node;
   return ;
}
while (temp → Next ! = Null)
temp → Next = New Node;
}
  void delete first Node () {
    if ( Head = = Null) {
printf ("List is empty. \n");
return;
}
  Struct Node * temp = Head;
    Head = Head → Next;
    free (temp);
printf(" first node deleted. \n ");
}
    void deleted At middle (int position) {
```

```
if (Head == Null) {
    printf("List ~~sym~~ isempty \n");
    return 0;
}
    if (position = 0) {
        delete first Node();
    return ;
    }

Struct Node * temp = Head;
Struct Node * prev = Null;
    int i = 0;

while (temp != Null && i < position) {
        prev = temp;
    temp = temp -> Next;
    i++;
}

if (temp == Null) {
    printf("position out of range \n");

return;
}

prev -> Next = temp -> Next;
 free (temp);
printf("Node at position %. deleted \n"
    position);
}

void delete last Node () {
    if (Head == Null) {
     printf(" list is empty. \n");
```

```
return n;
}

If (Head -> Next == Null) {
free (Head);

Head = Null;
printf ("last Node deleted. \n");

return 0;
}

struct Node * temp = Head;
Struct Node * prev = Null;

while (temp -> Next != Null) {
    prev = temp;
    temp = temp -> Next;
}

prev -> Next = Null;
free (temp);
print ("last Node deleted. \n");
}

void display () {
    struct Node * temp = Head;
if (Head == Null) {
    printf ("List is empty. \n");
return;
}

printf ("Linked List: ");
while (temp != Null) {
    printf ("%d -> ", temp -> data);
    temp = temp -> Next;
}
```

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

int main() {
    int choice, value, pos;

while (1) {
    printf("menu:");
    printf("1. inser at end (for testing)\n");
    printf("2. delete first Node\n");
    printf("3. delete Node at middle (position)\n");
    printf("4. delete last Node\n");
    printf("5. display list\n");
    printf("6. exit\n");
    printf("enter your choice");
    scanf("%d", &choice);

switch (choice) {

case 1:
    printf("enter value to insert:");
    scanf("%d", &value);
    inserted At end (value);
    Break;

case 2:
    delete first Node();
    Break;

Case 3:
    printf("enter position to delete (0-based)");
    scanf("%d", &pos);
```

```
    delete At
    Break;

Case 4:
    delete las
    Break;

case 5:
    display (
    Break;

case 6:
    exit(0);

default:
    printf("
    }
}
    return 0;
}
```

O/P

Menu:
- insert at end
- deleting first N
- deleting Node
- delete last No
- display list
- exit.

enter your
invalid

```
        deleteAtmiddle(pos);
        Break;

    Case 4:
        deleteLastNode();
        Break;

    case 5:
        display()
        Break;

    case 6:
        exit(0);

    default:
            printf("invalid choice try again");
        }
    }
        return 0;
    }
```

O/P

Menu:

- insert at end (for testing)
- deleting first Node
- deleting Node at middle (position)
- delete last Node
- display list
- exit.

enter your choice: insert at end
invalid choice. Try again.

__/__/____

Exp: 7

(i) primitive operation on stack - Push, pop, Is empty, is full.

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

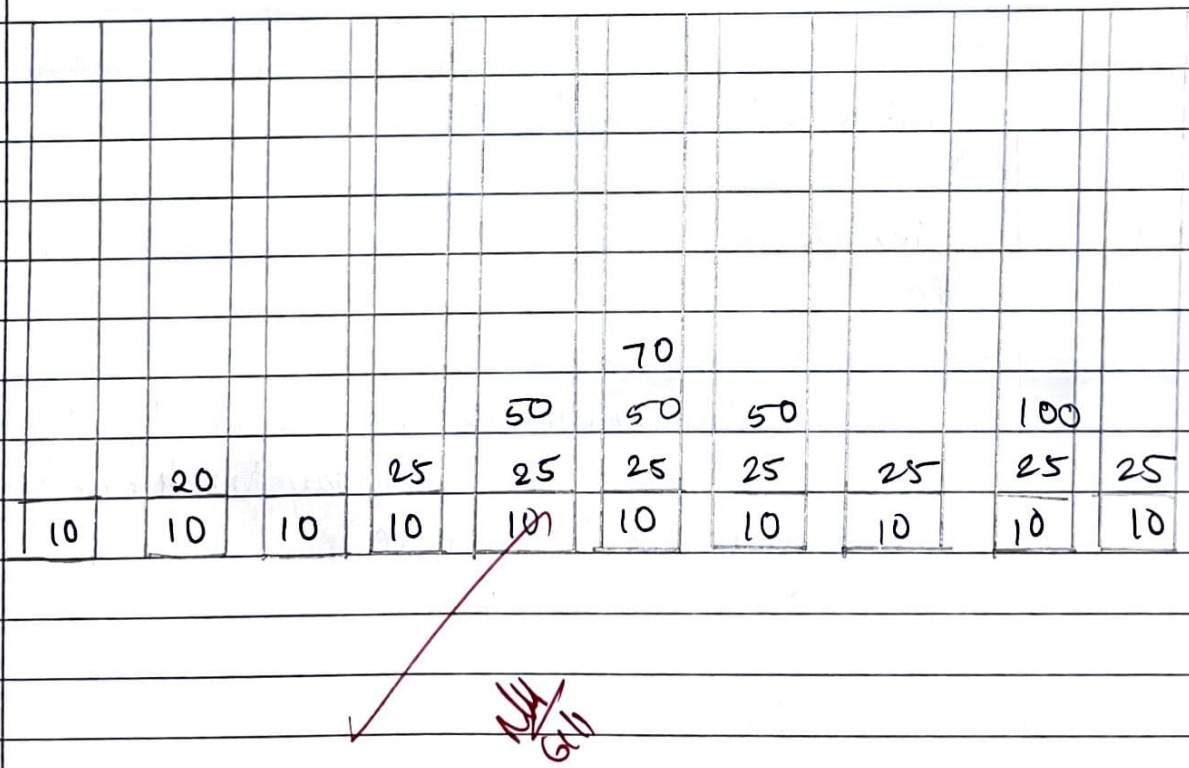
→

```
#include <stdio.h>
#include <Stdlib.h>
#define Max 5

int top = -1;
   int s [MAX];
   void push()
{
  int element ;
    if (top == max-1);
{
  printf("\n overflow");
}
  else
{
  printf("\n enter element to enter in stack");
  scanf("%d", & elements);
       top = top+1;
       s[top] = element;

void pop()
{
  int item;
  if (top == -1)
```

```
{
  printf("\n underflow");
}
else
{
  item = s[top];
  top --;
  printf ("\n elected elements is :%d", item);
}
}
Void display ()
{
  int i;
  printf("\n array is: \n");
    for (i=0; i<= top; i++)
  {
    printf("%d ", s[i]);
  }
}
int main()
{
  int choice;
do
{
  printf("\n enter ur choice: \n 1.push \n
        2. pop \n 3.display \n 4.exit \n");
  scanf ("%d", &choice);
    switch (choice)
  {
  case 1:
    push();
    break;
```

```
case 2:
    pop();
    break;

case 3:
    display();
    break;

case 4:
    printf("\nexiting");
        break;
    }
} while (choice!=4)
    return 0;
}
```

O/P



__/__/____

* Experiment 8:

(i) Convert infix to postfix or pretix 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 (charx)
{
  if (x == '(')
      retur 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;
}
```

(ii) Convert infix expression into prefix using stack : infix expression : A+(B*C-CD/E^F)*G)*H

H		H
*	*	H
)	*)	H
G	*)	HG
*	*)*	HG
)	*)*)	HG
F	*)*)	HGF
^	*)*)^	HGF
E	*)*)^	HGFE
/	*)*)/	HGFE^
D	*)*)/	HGFE^D
(	*)*)/	HGFE^D/
-	*)-	HGFE^D/*
C	*)-	HGFE^D/*C
*	*)-*	HGFE^D/*C
B	*)-*	HGFE^D/*CB
(	*)-*	HGFE^D/*CB*-
+	+	HGFE^D/*CB*-
A	+	HGFE^D/*CB*-A+

+A-*BC*/D^EFGH.

(ii) evaluate 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);
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);
   }
}
```

Exp 9:

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 <stdio.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("\n 1. Insert");
 printf ("\n 2. Display");
 printf("\n 3. exit);
 printf ("\n Enter your 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) {

print("\n Queue is overflow!");

}
```

```
else {
    item = queue [front];
    front ++;
    printf ("\n the delete element is ");
  }
}

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

__/__/____

Extra Q:

1) Explain the concept of primary queue

→ A priority queue is a data structure where each elements with higher priority and elements with higher priority are served before others. Unlik regular Queue (FIFO) priority Queues process elements based on priority.

2) Write applications of Queue

→ (a) Serving requests on a single shared resources, like a printer.

(b) Handling of Interrupts in real-time system. the interrupts are handled in the same order as they arrive i.e first come first served.

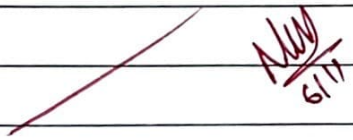
(c) FCFS job scheduling

(d) online banking found transfer requests.

3) Array size 10 inser (10) insert(50). delete, insert (100), insert(20), delete, insert (25), insert(200),

__/__/__

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



__/__/__

Exp 10:

- Circular Queue

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

   void enqueue();
   void dequeues ();
    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 ();
         Breaks 3;
         case 3: display ();
         Breaks;
          case 4: exit (0);
           default:
   } printf ("\n invalid choice !\n ");
```

```
} while (ch1 = 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 inser into queue",
              value);
  }
}

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

```
printf("\n Deleted %d form 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");
}
}
```

__/__/____

Extra q.

1) Write and explain application of priority Queue.

(a) Traffic light control:

(b) Operating system algorithm. used for scheduling process to execute high priority first.

(c) Huffman codes
priority queue helps build Huffman trees for efficient date compression.

6/11

__/__/___

Experiment - 11

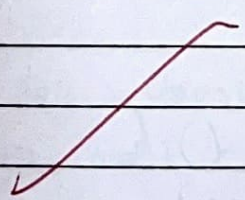
1) Create binary tree & traverse inorder, preorder & postorder

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

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

struct node * create node (int item) {
struct node * new Node = (struct node*) malloc
        (size of (struct node ));
    newnode → item = item;
    new node → left = Null
    new node → right = Null;
        return = new node;
}

struct node * insert At left (struct node * root,
                    int item ){
        root → left = Create Node (item);
        return root → left;
}

struct node * insert At right (struct node *
root, int item ){
```

```
root -> right = create node (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 = create node (40);
```

```
insertAtleft(root, 28);
 insertAtright(root, 27)
insert Atleft(root→left, 60);
insert AtRight(root→left, 30);
insert Atleft(root→right, 50);
 insert AtRight(root→right, 95);

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

printf("\n Preorder Traversal:");
    preorder(root);

printf("\n 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



2) Creation of Tree

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

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

struct node * createnode (item)
{
    struct node * 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 = create node (item);
    return root -> left;
}

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

Exp - 12

1) Graph

```
#include <stdio.h>
#define V5

void int (int arr[V][V]) {
    int i, j;
    for (i=0; j<V; j++)
    arr(i)(j)=0;
}
void insert Edge (int arr[V][V], int i,
int j) {
    arr[i][j]=1
    arr[j][i]=1;
}
    void print Adjmatrix (int arr[V][V]){
        int i, j;
        for (i=0; i<v; i++) {
        printf("%d", ;);
        for (j=0; j<v; j++) {
        printf("%d", arr(i)(j));
    }
        printf("\n");
    }
}

int main () {
    int adjmatrix [V>[V];
    int (adjmatrix);
```

~~void addEdge (struct Graph * graph, int src, int dest){~~

~~struct Node * temp = newNode (dest);~~

```
insertedge (adjmatrix, 0, 1);
           (adjmatrix, 0, 2);
           (adjmatrix, 0, 2);
           (adjmatrix, 2, 3);
           (adjmatrix 0, 3);
           (adjmatrix 0, 0);
           (adjmatrix 1, 3);
           (adjmatrix 2, 4);
           (adjmatrix 4, 3);

print adjmatrix (adjmatrix);
  return 0;
}
```

2)

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

struct Node {

    int Node {
        int test;
        struct Node * Next;
    };

    struct graph {
        struct Node * adj [V];
    };
        Struct Node * NewNode (int dest){
    Struct Node * temp = (Struct Node)
                        malloc (size of
                        (Struct Node));

    temp → dest = dest;
    temp → Next = Null;
        return graph temp;
    }

        struct graph * create graph (){
    struct graph * graph = (struct graph*)
                        malloc
                    (size of (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 -> dest);
      temp = temp -> Next;
  }

   print ("\n");
  }
}

 int main (){
struct graph * graph = creat graph();

  add edge (graph, 0, 1);

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

__/__/____

```
printGraph(graph);

return 0;
}
```