PAGE No.
DATE / /

~~8.6~~

Experiment - 1

1) PROBLEM - 1

Write a program to find average of 10 numbers in an array

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

int arr[10], avg", sum ;
for (int i =0 ; i<10; i++)
{
printf("Enter element of array:");
scanf ("%d", &arr[i]);

sum=sum + arr[i];
}
avg= sum/10;
printf("average is %d", avg );

return 0;
}
```

2) PROBLEM-2

Display pattern

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

```
#include <stdio.h>
int main ()
{
 int rows : 3;
for (i=1; i<=rows; i++)
for ( j=1; j<=rows; j++)
{
if (i % 2 == 0)
{
printf (" #");
}
else
{
printf (" * ");
}
}
 printf (" \n");
}
return 0;
}
```

3) Problem - 3

Find the first repeating Element in the Array

⇒

```
#include <stdio.h>

int main (){
int arr[] = [5,3,4,5,2,3,1];
 int n = size of (arr) /size of (arr[0]);
 int i,j, found = 0;

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

printf (" First repeating element is :%.d\n", arr[i])
 found = 1;
break ;
}
}
if (found)
 break;
}
if(!found)
printf("No ~~reap~~ repeating elements found\n");

return 0;
}

// Output....
1st repeating elements is: 5
```

4) Problem - 4

Find the Greatest & the smallest Element in the Array

⇒
```
#include <Stdio.h>

int main (){
 int arr [] = (15, 22, 8, 19, 31, 7);
 int n = size of (arr) / size of (arre [0]);
 int max = arr [0], min = arr [0];

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

 return 0;
 }

 //output....

 Greatest element : 31
 Smallest element : 7
```

5 Problem - 5

Squaring the odd positioned Elements

⇒

```
#include <stdio.h>

int main (){
int arr[] = {2,4,6,8,10,12};
int n = size of (arr)/size of (arr[0]);

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

for (int i=0; i<n ;i++){
if (i %2 != 0){
arr [i] = arr[i] * arr(i);
}
}
printf("\n Modified array (odd- postioned squared):");
for (int i=0; i<n; i++)
printf("%d", arr[i]);

return 0;
}
```

//Output ....

Original array: 2 4 6 8 10 12
Modified array: (odd -postioned squared): 2 16 6 64 10 144

PAGE No.
DATE / /

Experiment - 2

Q. WAP to search particular Data from given array using linear search.

→
```
#include <stdio.h>
int main(){
printf("Enter 8 Elements :\n");
for(int i=0;i<8;i++){
  scanf("%d",&a[i]);
}
printf("Enter search Key :");
scanf("%d",&k);
for(int i=0;i<8;i++)
 {
  if(a[i]==k)
  {
   printf("Element %d Found at %d",a[i],i+1);
   return 0;
  }
}
printf("Element not Found.");
return 0;
}
```

1] O/P:

Enter 8 elements:

36

89

57

1

0

67

56

Enter Search Key : 1

Element 1 Found at 4.

2] O/P:

Enter 8 elements:

36

89

57

1

0

67

56

Enter Search Key : 55

Element not Found.

PAGE No.
DATE / /

Q Search Using For ~~Linear~~ Binary Search.

→
```
#include <stdio.h>
int main() {
int a[100], n, k, l, m, h;
printf("Enter Number of Array: ");
scanf("%d", &n);
printf("Enter %d Sorted Elements: \n", n);
   for(i=0; i<n; i++){
     scanf("%d", &a[i]);
}
printf("Enter the key to search: ");
scanf("%d", &k);
 l=0;
 h=n-1;
 while(l<=h){
   m=(l+h)/2;
if(a[m]==k){
 printf("key found at index %d \n", m+1);
 return 0; }
else if (a[m]<k){
    l=m+1;
 }else{
 h=m-1;}}
printf("key not found in Array \n");
 return 0;
}
```

1] O/P:

Enter Number of Array: 8
Enter 8 ~~elements~~ Sorted Elements:
0
1
36
56
57
59
67
89.
Enter the key to search: 1
Element found to index 2.

2] O/P:

Enter Number of Array: 8
Enter 8 Sorted Elements:
0
1
36
56
57
59
67
89.
Enter the key to search: 55
Element not found.

Q3] Compare Linear & Binary Search:

→

Linear	Binary
• It works on both sorted & unsorted array.	• it only works on sorted array.
• Time complexity in O(n).	• Time complexity in O(log(n))
• It is used for small & unsorted data set.	• it is used for large & sorted dataset.
• Simple to Implement.	• Complex to Implement.
• Algorithm type is sequential.	• Algorithm type is divide & conquer.

Q4] State limitations of Linear Search in terms of time complexity.

→ it has limitations in terms of time complexity especially for large data every element in array to determine if it is present or not which makes it slower. The binary data is sorted.

PAGE NO.
DATE / /

Experiment - 03

Q.3)

i)

```
#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-1; j++){

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

temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
    }
  }
 }
}
void printArray(int arr[], int n) {
  int i;
for (i=0; i<n; i++)
 printf("%d ", arr[i]);
 printf("\n");
}
int main (){
int arr[] = {64, 34, 25, 12, 22, 11, 90};
int n = sizeof(arr)/sizeof(arr[0]);
```

```
printf("Original array: \n");
printArray(arr, n);

bubbleSort(arr, n);

printf("Sorted array in ascending order using
        Bubble sort: \n");
printArray(arr, n);

return 0;
}
```

ii)

```
#include <stdio.h>

void selectionSortDescending(int arr[], int n) {
int i, j, MaxIndex, temp;

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

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

```
    }
  }
}
void printArray(int arr[], int n){
  int i;
for(i=0; i<n; i++)
printf("%d", arr[i]);
printf("\n");
}
int main(){
int arr[] = {29,10,14,37,13};
int n = sizeof(arr)/sizeof(arr[0]);

printf("Original array:\n");
printArray(arr, n);

SelectionSortDescending(arr, n);

printf("Sorted array in descending order using
        selection sort:\n");
printArray(arr, n);

return 0;
}
```

iii)

```
#include <stdio.h>

int main (){

int arr[] = {100,200,300,400,500};
int n = sizeof(arr) / sizeof(arr[0]);
int i, j, temp;
int comparison = 0;

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

 temp = arr[j];
 arr[j] = arr[j+1];
 arr[j+1] = temp;
  }
 }
}
printf("Total number of comparisons: %d\n", compar-
        isons);

return 0;
}
```

iv)

```
#include <stdio.h>

void printArray(int arr[], int n, int step){
 printf("After iteration %d: ", step);

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

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

 for(j=i+1; j<n; j++){
 if(arr[j] < arr[minIndex]){
    minIndex = j;
    }
 }
 if(minIndex != i){
  temp = arr[i];
  arr[i] = arr[minIndex];
  arr[minIndex] = temp;
 }
 printArray(arr, n, i+1);
  }
}

int main(){
int arr[] = {500, -20, 30, 14, 50};
int n = sizeof(arr) / sizeof(arr[0]);
```

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

selectionSort(arr, n);

printf("\nFinal sorted array (Ascending Order): \n");

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

return 0;
}
```

PAGE No.
DATE / /

Experiment - 01

★ Extra Question:

a) Bubble Sort:

1. Compare 0th element (j) with next/first element (j+1)
2. If (j+1) is smaller than j swap the two elements.
3. Move to the next set or elements (1st & 2nd) & repeat step 2.
4. Continue this process till you reach the end of the list/array.
5. At end of first iteration, largest element is found at end, exclude that element & repeat from step 1.
6. For n number of elements n-1 iterations will occur.

b) Selection Sort:

1. Start from the 0th element of the list (i) and compare with each element.
2. If arr[i] is greater than any element swap the two.
3. At end of first iteration smallest element is found at begining Exclude that & repeat step 1 & 2.
4. Repeat step 3 for each position/element till only one element remains unsorted (largest)

5. Hence, Sorted list is ~~object~~ obtained.

c) Insertion Sort :

1. Start with 1st index (2nd position)
2. Compare it with element before i.e. 0th element.
3. If it is smaller, insert in position of 0th element by sniffing all others to the right.
4. The move to next element compare with 0th & 1st, insert at its appropriate position by making space.
5. Repeat with all elements until list is sorted.

d) Radix Sort:

1. Define 10 queus each with buckets from digit 0 to 9.
2. Consider least significant digit of each no. in the list to besorted. And insert each number in repective bucket based on least-significant digit.
3. Group all numbers from queue from 0 to 9 in order they have been inserted (top to bottom)
4. Repeat the process with next least significant digit.
5. Continue till all elements are grouped based on most significant digit.

PAGE No.
DATE / /

Experiment - 04

i)

```
#include <stdio.h>

void insertionsort (int arr[], int n) {
  int i, key, j;

for (i=1 ; i<n; i++){
  key= arr[i];
  j = i-1;

while (j >=0 && arr [j] > key ) {
  arr [j+1] =arr[j];
    j--;
}
arr [j+1] = key;
  }
}
void printArray (int arr[], int n) {
  for (int i=0; i<n; i++)
  printf("%d", arr[i]);
  printf("\n");
}
int main (){
int arr [] = {45, 23, 53, 12, 78, 34};
int n = sizeof (arr) /sizeof (arr[0]);

printf ("Original array: \n");
printArray (arr, n);

insertionSort (arr, n);
```

```
printf("Sorted array in ascending order using
        Insertion Sort :\n");

printArray(arr, n);

return 0;
}
```

ii)

```
#include<stdio.h>

int getMax(int arr[], int n) {
int max = arr[0];
for (int i=1; i<n; i++)
 if (arr[i] > max)
  max = arr[i];
return max;
}
void countingSort(int arr[], int n, int exp){
int output[n];
int count[10] = {0};

for (int i=0; i<n; i++)
 count[(arr[i]/exp)%10]++;

for (int i=1; i<10; i++)
 count[i] += count[i-1];

for (int i=n-1; i>=0; i--){
int digit = (arr[i]/exp)%10;
output[count[digit]-1] = arr[i];
count[digit]--;
}
```

```
for (int i=0; i<n; i++)
arr[i] = output[i];

void radixsort(int arr[], int n){
int max = getMax(arr, n);

for(int exp = 1; max/exp >0; exp *= 10)
countingSort(arr, n, exp);
}
void printArray(int arr[], int n){
for(int i=0; i<n; i++)
printf("%d", arr[i]);
printf("\n");
}
int main(){
int arr[] = {170, 45, 75, 90, 802, 24, 2, 66};
int n = sizeof(arr)/sizeof(arr[0]);

printf("original Array:\n");
printArray(arr, n);

radixSort(arr, n);

printf("Sorted array in ascending ordere using
Radix sorted);
printfArray(arr, n);

return 0;
}
```

iii)

```
#include<stdio.h>

void insertionSort(int arr[], int n){
 int i, key, j;

printf("\nInsertion Sort Steps:\n");
for(i=1; i<n; i++) {
 key = arr[i];
j = i-1;

while (j>=0 && arr[j]>key){
  arr[j+1] = arr[j];
  j--;
}
arr[j+1] = key;

printf("After iteration %d:", i);
for(int k=0; k<n; k++)
printf("%d", arr[k]);
 printf("\n");

 if(i==2){
printf("\n Output after 2nd iteration of Insertion
        sort:\n");
 for(int k=0; k<n; k++)
printf("%d", arr[k]);
printf("\n\n");
   }
  }
}
int getMax(int arr[], int n){
```

```
int max = arr[0];
for (int i = 1; i<n; i++)
 if (arr[i] > max)
 max = arr[i];

return max;
}
void countingSort (int arr[], int n, int exp){
int output [n];
int count [10] = {0};

for(int i=0; i<n; i++)
count (arr[i] /exp) % 10]++;

for (int i=1; i<10; i++)
count [i] += count [i-1];

for (int i=n-1; i>=0; i--) {
int digit = (arr[i] /exp) % 10;
output [count [digit] - 1] = arr[i];
count [digit] --;
}
for (int i=0; i<n; i++)
arr[i] = output [i];

void radixSort (int arr[], int n) {
int max = getmax (arr, n);
for (int exp = 1; max/exp > 0; exp *= 10)
countingSort [arr, n, exp);
}
Void PrintArray (int arr[], int n) {
```

```
for(int i=0; i<n; i++)
printf("%d", arr[i]);
printf("\n");
}
int main(){
int arr1[] = {7,3,5,1,9,8,4,6};
int arr2[]= {7,3,5,1,9,8,4,6};
int n= sizeof(arr1)/sizeof(arr1[0]);

printf("Original Array:\n");
printArray(arr1,n);

insertionSort(arr1,n);

radixSort(arr2,n);
printf("Sorted array using radix sort:\n");
printArray(arr2,n);

return 0;
}
```

PAGE No.
DATE / /

Experiment - 05

Q. Singly linked list

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

struct node {
    int data;
    struct node* next;
};

Struct node* head = NULL;

void create_list();
Void display_list();
void add_at_beginning ();
Void add_at_end ();
void add_at_position ();
void search_node ();
void delete_node ();
Void reverse_list ();

int main() {
    int choice;

while (1) {
printf ("\n Menu:\n");
printf ("1. Create list\n");
printf ("2. Display list \n");
printf ("3. Add at beginning \n");
printf ("4. Add at end \n");
```

```
Printf("5. Add at position\n");
printf("6. Search node\n");
printf("7. Delete node \n");
printf("8. Reverse list \n");
printf("9. Exit \n");
printf("Enterchoice: ");
scanf("%d", &choice);

switch (choice) {

    case 1: create_list (); break;
    Case 2: display_list (); break;
    Case 3: add_at_beginning (); break;
    Case 4: add_at_end (); break;
    Case 5; add_at_position (); break;
    Case 6: search_node(); break;
    Case 7: delete_node (); break;
    Case 8: reverse_list (); break;
    case 9: exit (0);
    default: printf("Invalid choice, try agai
                                        .\n")

    }
}
return 0;
}


Void create_list () {
    int n, i;
    Struct node * temp, *tail;
```

```
printf("How many nodes? ");
scanf("%d", &n);
head = Null;

for (i=0; i<n; i++) {
    temp = (struct node*) malloc(size of(struct node));
    printf("Enter data for node %d; ", i+1);
    scanf("%d", &temp->data);
    temp -> next = NULL;

    if (head == NULL) {
        head = temp;
        tail = temp;
    } else {
        tail -> next = temp;
        tail = temp;
    }
}
Printf("List created with %d nodes. \n", n);
}

Void display_list() {

    struct node * temp = head;

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

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

void add_at_beginning () {

    struct node * temp = (struct node*) malloc (size of
                                    (struct node));
    printf ("Enter data to add at beginning: ");
    scanf ("%d", & temp -> data);

    temp -> next = head;
    head = temp;
    printf ("Node added at beginning.\n");
}


void add_at_end () {
    struct node * temp = (struct node*) malloc (sizeof
                                    (struct node));
    Struct node * current = head;
    printf ("Enter data to add at end: ");
    scanf ("%d", & temp -> data);
    temp -> next = NULL;
```

```
if (head == NULL) {
    head = temp;
} else {
    while (current -> next != NULL)
    current = current -> next;
    current -> next = temp;
}
printf ("Node added at end. \n");
}

void add_at_position () {
    int pos, i;
    struct node *temp, *current = head;
    printf ("Enter position to add node
            (starting from 1): ");
    scanf ("%d", &pos);

    if (pos <= 0) {
    printf ("Invalid position! \n");
    return;
}

temp = (struct node *) malloc (size of (struct node));
printf ("Enter data to add: ");
scanf ("%d", &temp -> data);

if (pos == 1) {
    temp -> next = head;
    head = temp;
    printf ("Node added at position 1. \n");
    return;
}
```

```
for (i=1, i < pos -1 && current != NULL, i++)
current = current -> next;

if (current == NULL) {
  printf("Position out of range. \n");
  free (temp);
  return;
}

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

void search_node () {
  int value, pos = 1;
  struct node * current = head;

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

print f ("Enter value to search: ");
Scanf ("%d", &value);

While (current != NULL) {
   if (current -> data == value) {
   printf ("Value %d found at position %d. \n",
                                value pos);
```

```
        return;
    }
    current = current -> next;
    pos ++;
}

printf("Value %d not found in the list.\n", value);
}


void delete_node() {
    int pos, i;
    struct node * current = head, *temp;

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

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

    if (pos <= 0) {
        printf("Invalid position!\n");
        return;
    }
```

```
if (pos == 1) {
    temp = head;
    head = head -> next;
    free (temp);
    printf ("Node at position 1 deleted. \n");
    return;
}

for (i = 1; i < pos - 1 && current != NULL; i++)
    current = current -> next;

if (current == NULL || current -> next == NULL) {
    printf ("position out of range. \n");
    return;
}

temp = current -> next;
current -> next = temp -> next;
free (temp);
printf ("Node at position %d deleted. \n", pos);
}

Void reverse_list () {
    struct node * prev = NULL, *current = head, *next = NULL;

    if (head == NULL) {
        printf ("List is empty, cannot reverse. \n");
        return;
    }
```

```
while (current != NULL) {
    next = current -> next;
    current -> next = prev;
    prev = current;
    current = next;
}

head = prev;
printf("List reversed successfully.\n");
}
```

ii)

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

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

 struct Node* head = NULL;

 struct Node * create Node (int data) {
   Struct Node = (Struct Node*) malloc (sizeof (struct Node));
   new Node -> data = data;
   newNode -> next = NULL;
   return newNode;
 }

 void insert At End (int data) {
   struct Node* newNode = create Node (data);
   if (head == NULL) {
      head = newNode;
      return;
   }

   struct Node * temp = head;
   while (temp -> next != NULL)
       temp = temp -> next;
       temp -> next = newNode;
   }
```

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

void deletefirst () {

if (head == NULL) {
    printf ("List is empty, nothing to delete.\n");
    return;
}
    struct Node * temp = head;
    head = head -> next;
    free (temp);
    printf ("first node deleted.\n");
}

    void deleteLast () {
        if (head == NULL) {
        printf ("List is empty, nothing to delete.\n");
        return;
    }
```

```
if (head -> next == NULL) {
    free (head);
    head = NULL;
    printf ("Last node deleted. \n");
    return;
}

Struct Node * temp = head;
while (temp -> next -> next != NULL)
    temp = temp -> next;
free (temp -> next);
temp -> next = NULL;
printf ("Last node deleted. \n");
}

Void deleteAtPosition (int pos) {
if (head == NULL) {
    printf ("List is empty, nothing to delete. \n");
    return;
}

if (pos == 1) {
delete first ();
return;
}

struct Node * temp = head;
for (int = i = 1; i < pos - 1 && temp -> next != NULL;
                                        i++); {

temp = temp -> next;
}
```

```
if (temp -> next == NULL) {
    printf ("Invalid position ! \n");
    return;
}

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

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

  while (1) {
  printf ("\n Menu: \n");
  printf ("1. Insert node at end | n");
  printf ("2. Display list \n");
  printf ("3. Delete first node \n");
  printf ("4. Delete node at position (middle) \n");
  printf ("5. Delete last node \n");
  printf ("6. Exit \n");
  printf (" Enter your choice: \n");
  scanf ("%d", & choice);
```

```
switch (choice) {
    case 1:
        printf("Enter data: ");
        scanf("%d", &data);
        insertAtEnd(data);
        break;
    case 2:
        displayList();
        break;
    case 3:
        deleteFirst();
        break;
    case 4:
        printf("Enter position to delete: ");
        scanf("%d", &pos);
        deleteAtPosition(pos);
        break;
    case 5:
        deleteLast();
        break;
    case 6:
        exit(0);
    default:
        printf("Invalid choice!\n");
  }
}
  return 0;
}
```

iii)

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

Struct Node {
 int data;
 struct Node* next;
};

struct Node* createNode (int data) {
  struct Node* newnode = (struct Node*) malloc (sizeof
                          (struct Node));
  newNode -> data = data;
  newNode -> next = NULL;
  return newNode;
}

Void insertAtEnd (struct Node** head, int data) {
  struct Node* = createNode (data);
  if (*head == NULL) {
     * head = newNode;
     return;
  }
  struct Node* temp = * head;
  while (temp -> next != NULL)
     temp = temp -> next;
     temp -> next = newNode;
  }
```

```
Void displayList(struct Node* head) {
  if (head == NULL) {
    printf("List is empty!\n");
    return;
  }
  struct Node* temp = head;
  while (temp != NULL) {
    printf("%d -> ", temp -> data);
    temp = temp -> next;
  }
  printf("NULL\n");
}

Struct Node* concatenate(struct Node* head1, struct Node*
                                                head2) {
  if (head1 == NULL) return head2;
  if (head2 == NULL) return head1;

  struct Node* temp = head1;
  while (temp -> next != NULL)
    temp = temp -> next;
    temp -> next = head2;
    return head1;
  }

int countNodes(struct Node* head) {
    int count = 0;
    struct Node* temp = head;
    while (temp != NULL) {
      count++;
      temp = temp -> next;
    }
```

```
    return count;
}

Struct Node* reverseList(struct Node* head) {
    Struct Node* prev = NULL;
    Struct Node* curr = head;
    Struct Node* next = NULL;

    while (curr != NULL) {
      next = curr -> next;
      curr -> next = prev;
      prev = curr;
      curr = next;
    }
    return prev;
}

Void searchNode (struct Node* head, int key) {
    int pos = 1;
    struct Node* temp = head;
    while (temp != NULL) {
      if (temp -> data == key) {
      printf("Node with value %d found at
                position %d \n", key, pos);
      return;
    }
    temp = temp -> next;
    pos++;
}
printf ("Node with value %d not found in the list.\n"
                                        key);
}
```

```
int main () {
    struct Node * head1 = NULL;
    struct Node * head2 = NULL;
    struct Node * result = NULL;
    int choice, data, key, listchoice;

    while (1) {
        printf ("\n Menu: \n");
        print f(" 1. Create (Insert at end) \n");
        printf ("2. Display \n");
        print f("3. Concatenate two lists \n");
        printf ("4. Count number of nodes \n");
        printf ("5. Reverse List \n");
        printf ("6. Search a node \n");
        printf ("7. Exit \n");
        printf (" Enter your choice: ");
        scanf ("%d", & choice);

    switch (choice) {

    Case 1:
        printf ("Which list? (1 or 2): ");
        Scanf ("%d", &list choice);
        Print f ("Enter data: ");
        Scanf ("%d", & data);
        if (list choice == 1)
            insert At End (& head 1, data);
        else if (list choice == 2)
            insert At End (& head 2, data);
        else
            printf ("Invalid list choice ! \n");
        break;
```

```
case 2:
    printf ("which list? (1,2 or 3[concatenated]): ");
    scanf ("%d", &listchoice);
    if (list choice == 1)
      display list (head 1);
    else if (list choice == 2)
      display list (head2);
    else if (list choice == 3)
      display list (result);
    else
      print f ("Invalid choice! \n");
      break;

case 3:
    result = concantenate (head 1, head 2);
    print f = ("List concatenated into list 3. \n");
    break;

case 4:
    printf ("which list? (1,2 or 3); ");
    scanf ("%d", & listChoice);
    if (listChoice == 1)
      printf("Node count: %d \n", count Nodes(head1));
    else if ( list choice == 2)
      printf("Node count: %d \n", count Nodes(head2));
    else if (list choice == 3)
      print f ("Node count: %d \n", Count Nodes(result));
    else
      print f ("Invalid choice! \n");
    break;
```

```
case 5:
    printf("which list to reverse? (1,2 or 3): ");
    scanf("%d", &listchoice);
    if (list choice == 1)
        head1 = reverse list (head1);
    else if (list choice == 2)
        head2 = reverse list (head2);
    else if (list choice == 3)
        head result = reverseList (result);
    else
        printf("Invalid choice ! \n");
    break;

case 6;
    printf("Enter value to search: ");
    scanf("%d", &key);
    printf("In which list? (1, 2, or 3): ");
    scanf("%d", &listChoice);
    if (listchoice == 1)
        SearchNode (head1, key);
    else if (list choice == 2)
        searchNode (head2, key);
    else if (list choice == 3)
        searchNode (result, key);
    else
        printf("Invalid choice ! \n");
    break;
```

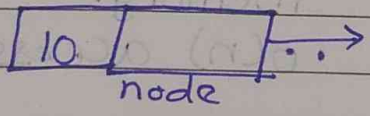
```
        case 7:
            exit(0);

        default:
            printf("Invalid choice!\n");

    }
}
    return 0;
}
```

PAGE No.
DATE / /

iv) Basic Terminology of linked list:-

a) NODE : Basic building block of linked list. It contains data (value) & next (pointer) to next node)

eg.



b) head: The head is a reference or pointer to first node of list. NULL or nun.

c) Tail:
last node where next == NULL

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

e) NULL pointer:
Special value that indices that pointer does not have object memory.

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

head → NULL

v)

Array	Linked List
• Resizing not possible	• Can grow or shrink
• Memory allocation is fixed size.	• Size of array thus memory can change
• Stored in contiguous memory	• Stored in non-contiguous memory.
• O(1) access time.	• O(n) access time.

DATE / /

Experiment - 06

Q1)

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


struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* head = NULL;


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}


void createList(int n) {
    int data, i;
    struct Node *newNode, *temp;

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

    for (i = 0; i < n; i++) {
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &data);
        newNode = createNode(data);

        if (head == NULL) {
            head = newNode;
        } else {
            temp->next = newNode;
            newNode->prev = temp;
        }
        temp = newNode;
    }
}


void displayList() {
    struct Node* temp = head;

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

    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


void insertNode(int data, int pos) {
    struct Node* newNode = createNode(data);
    struct Node* temp = head;
    int i;

    if (pos == 1) {
        newNode->next = head;
        if (head != NULL)
            head->prev = newNode;
        head = newNode;
        return;
    }

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

```
    if (temp == NULL) {
        printf("Position out of range!\n");
        free(newNode);
        return;
    }

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


void deleteNode(int pos) {
    struct Node* temp = head;
    int i;

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

    if (pos == 1) {
        head = head->next;
        if (head != NULL)
            head->prev = NULL;
        free(temp);
        return;
    }

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

    if (temp == NULL) {
        printf("Position out of range!\n");
        return;
    }

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

    free(temp);
}


int main() {
    int choice, n, data, pos;

    while (1) {
        printf("\n--- Doubly Linked 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. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            if (head == NULL) {
                printf("Enter number of nodes: ");
                scanf("%d", &n);
                createList(n);
            } else {
                printf("List already exists!\n");
            }
            break;

        case 2:
            displayList();
            break;

        case 3:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position: ");
            scanf("%d", &pos);
```

```
            insertNode(data, pos);
            break;

        case 4:
            printf("Enter position to delete: ");
            scanf("%d", &pos);
            deleteNode(pos);
            break;

        case 5:
            printf("Exiting...\n");
            exit(0);

        default:
            printf("Invalid choice!\n");
        }
    }

    return 0;
}
```

PAGE No.
DATE / /

(Q.2)

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


struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* head = NULL;


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}


void insertEnd(int data) {
    struct Node* newNode = createNode(data);
    if (head == NULL) {
        head = newNode;
        return;
    }
    struct Node* temp = head;
    while (temp->next != NULL)
        temp = temp->next;

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


void displayList() {
    struct Node* temp = head;

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

    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}


void searchNode(int key) {
    struct Node* temp = head;
    int pos = 1, found = 0;

    while (temp != NULL) {
        if (temp->data == key) {
```

```
            printf("Node %d found at position %d\n", key, pos);
            found = 1;
            break;
        }
        temp = temp->next;
        pos++;
    }

    if (!found)
        printf("Node %d not found in the list.\n", key);
}


void countNodes() {
    struct Node* temp = head;
    int count = 0;

    while (temp != NULL) {
        count++;
        temp = temp->next;
    }

    printf("Total number of nodes = %d\n", count);
}


void reverseList() {
    struct Node *current = head, *temp = NULL;

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


    while (current != NULL) {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev; // move to next node (previous in original list)
    }


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

    printf("List reversed successfully!\n");
}


int main() {
    int choice, data, key;

    while (1) {
        printf("\n--- Doubly Linked List Menu ---\n");
        printf("1. Insert Node at End\n");
        printf("2. Display List\n");
        printf("3. Search Node\n");
        printf("4. Count Nodes\n");
        printf("5. Reverse List\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            insertEnd(data);
            break;

        case 2:
            displayList();
            break;

        case 3:
            printf("Enter data to search: ");
            scanf("%d", &key);
            searchNode(key);
            break;

        case 4:
            countNodes();
```

```
            break;

        case 5:
            reverseList();
            break;

        case 6:
            printf("Exiting...\n");
            exit(0);

        default:
            printf("Invalid choice!\n");
        }
    }
    return 0;
}
```

PAGE No.
DATE / /

Experiment -07

1)
```
# include <stdio.h>
# define MAX 5

int stack [MAX];
int top = -1;

int isFull () {
return (top == MAX - 1);
}
int isEmpty () {
 return (top == -1);
}
void push (int value) {
    if (isFull ()) {
    printf ("stack Overflow ! Cannot push %d
            \n", value);
} ~~ete~~ else {
Stack [++top] = value;
 printf ("%d pushed \n", value);
    }
}
void pop () {
if (isEmpty ()) {
  printf ("Stack Underflow ! Cannot pop\n");
} else {
printf ("%d popped \n", stack [top--]);
   }
}
```

DATE / /

```
int main (){
    push (10);
    push(20);
    push(30);

    pop();
    pop();
    pop();
    pop();

 if (isEmpty())
 printf("Stack is Empty now\n");

 return 0;
}
```

ii)

```
#include<stdio.h>
# define MAX 8

int stack
int top = -1;

int isFull (){
return (top == MAX - 1);
}
int isEmpty (){
return (top == -1);
}
Void push (int value){
 if (isFull ()){
   printf ("Stack Overflow! Cannot push %d\n",
          value);
 } else{
    stack [++top] = value;
    printf ("Pushed %d\n", value);
     }
}
void pop ()
if (isEmpty ()){
  printf ("Stack Underflow! Cannot Pop\n");
}else{
 printf ("Popped %. d\n", stack [top--]);
   }
}
void display (){
if (isEmpty ()){
printf ("Stack is empty \n");
```

```
}else{
printf("Final stack (bottom -> top): ");
 for(int i = 0; i <= top; i++){
  printf("%d ", stack[i]);
 }
printf("\n");
  }
}
int main(){
push(10);
push(20);
pop();
push(25);
push(50);
push(70);
pop();
push(100);
pop();

display();

return 0;
}
```

PAGE No.
DATE / /

Experiment - 08

i) #~~code~~ include <~~iostream~~>
using namespace std;

ii) #include <iostream>
#include <c.type.h>

Char stack [100];
int top = -1;
void push (char x)
{
stack [++top] = x;
}
if (top == -1)
{
return -1;
}
else;
{
return stack [top--];
}
}
int priority (Char x)
{
if (x == '(')
{
return 0;
}
if (x == '+' || x == '-')
{

```
return 2;
}
return 0;
}
int main()
{
char exp[100];
char *e, x;
printf("\n");
e = exp;
while (*e != '\0')
{
 if(isalnum(*e))
{
 printf ("%c",*e);
}
else if (*e == ')')
{
while((x=pop()) != '(')
{
printf("%c", pop());
}
push(*e);
}
 e++;
}
while (top1 = -1)
{
printf("%c", pop());
}
return 0;
}
```

O/P:

enter expression : a+b

ab+

Convert infix expression into prefix using stack

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

Input	Stack	Output
H		H
*	*(	H
)		H
G	*(	HG
Y	*(*	HG
)		HGF
F	*(*(	HGF
n	*(*(^	HGFE
E	*(*(^	HGFE^
/	*(*(^/	HGFE^D
D	*(*(/	HGFE^D
C	*(*(/	HGFE^D/
-	*(*(-	HGFE^D/*

iii)

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

int stack[100];
int top = -1;

void push(int val){
    stack[++top] = val;
}
int pop(){
    return stack[top--];
}
int main(){
 char expr[100];
 int i,a,b, result;

printf("Enter postfix expression (e.g. 23+5*): ");
scanf("%s", expr);

for(i=0; expr[i] != '\0'; i++){
    char ch = expr[i];

if(isdigit(ch)){
push(ch-'0');
}else{
 b= pop();
a= pop();

switch(ch){
case '+': push(a+b); break;
```

```
        case '-': push(a-b); break;
        case '*': push(a*b); break;
        case '/': push(a/b); break;
        }
    }
}
result = pop();
printf("Result = %d\n", result);

return 0;
}
```

Experiment-09

Q1)

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

#define SIZE 5
void enqueue();
void dequeue();
void display();
int queue[size5];

int front = -1, rear = -1;

int main()
{
int ch;
printf("1. ENQUEUE\n
        2. DEQUEUE\n
        3. DISPLAY\n
        4. EXIT\n");

{
printf("\n Enter your choice: ");
Scanf("%d", &ch);
switch(ch)
{
case1: enqueue();
break;
case 2: dequeue();
break;
Case 3: display();
```

```
break;
case 4 : exit (0);

default : printf ("\n Invalid choice \n");
  }
}
while (ch != 4)
return 0;
}
void enqueue ()
{
int element;
if (rear = SIZE - 1)
{
Printf("\n queue is full...");
}
else
{
printf ("\nEnter element to insert");
Scanf("%d", &element);
if (front == -1)
front = 0;
rear ++
queue (rear) = element;
}
}
Void dequeue ()
{
printf ("\n Queue is Empty:");
}
else
{
```

```
printf("\n Deleted element %d", queue[front]);
 front++;
 }
};
void display();
{
printf("\n Queue is empty---");
 }
 else
{
printf("\n Queue element are:\n");
for(int i = front; i<= rear; i++)
{
printf(" %d\n", queue[i]);
 }
 printf("\n");
  }
};
```

b) WAP to count digits & spaces using file handeling

```
#include <iostream>
# include <fstream>
#include <ctype>

using namespace std;

int main (){

fstream new_file
new_file.open("first.txt", ios::in);
if (!new_file){
cout<<"Error occured while opening!"
    <<endl;

return 1;
}
int digits_count=0;
int spaces_count=0;
char ch;

while (newfile.get(ch)){
 if (isdigit(ch)){

  digit_count++;
}
if (isspace(ch)){
  space_count++;
}
```

```
new_file.close(en)){
    space_count++;
}
new_file.close();
Cout<<"No. of digits:"<<digits_count
    <<endl;
cout<<"No. of space:"<<spaces-count
    <<endl;

return 0;
}
```

PAGE No.
DATE / /

Experiment - 10

Q.

```
#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("\nEnter your choice :");
scanf(" %d ", &ch);
switch (ch){

Case 1: enqueue ();
break;
Case 2: dequeue ();
break;
case 3: display()
 break
 case 4: exit (0)
 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 Delete %d from queue. " queue [front])
  if (front == rear)
{
 front = rear -1;
```

```
    }
    else{
    front = (front+1) % SIZE;
    }
    }
}
Void display (){
if (FRONT == -1)
{ printf("\nQueue is Empty!\n");
}
else{
 int i = front;
printf ("\nQueue element are:");
while (1){
printf("%d", queue[i]);
 if (i == rear)
   break;
 i = (i+1) % SIZE;
}
printf ("\n");
}
```

write & Explain application of priority of queue

a. Traffic Light control:

Manages signal priorities based on real time traffic sensor data.

b. Operating System algorithm:
used for scheduling process to execute high priority first.

c. Huffman Code:
Priority queue helps builds Huffman trees for efficient data compression.

PAGE No.

DATE / /

PAGE No.
DATE / /

Experiment - 11

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

struct node {
int data;
struct node *left, *right;
};
struct node* newNode (int val) {
    struct node* n = (struct node*) malloc
     (size of (struct node));
    n -> data = val;
    n -> left = n -> right = NULL;

   return n;
}

struct node* insert (struct node* root, int
val) {
if (! root) return newNode (val);
if (val < root -> data) root -> left = insert
   (root -> left, val);
else root -> right = insert (root -> right, val);
return root;
}
void inorder (struct node* root) {
    if (root) {
     inorder (root -> left);
     printf ("%d", root -> data);
      inorder (root -> right);
   }
}
```

```
struct node* search (struct node* root, int
key) {
if (!root || root -> data)? search (root ->
   left, key) : search (root -> right, key);
}
struct node* minNode (struct node* r){
 while (r && r -> left) r = r -> left;
  return r;
}
struct node* delete (struct node* root, int key)
{
if (!root) return root;
if (key > root -> data) root -> right = delete
 (root -> right, key);
else {
 if (!root -> left){struct node* t = root -> right;
  free (root); return t;
  }
else if (!root -> right){struct node* t = root
        -> left; free (root);
 return t; }
struct node* t = minNode (root -> right);
 root -> data = t -> data;
 root -> right = delete (root -> right, t -> data);
}
return root;
}
int main (){
 struct node* root = NULL;
 int ch, val;
 while (1){
```

```
printf("\n1. Insert 2. Display 3. Search 4. Del
        5. Exit : ");

scanf ("%d", &ch);
switch (ch) {

case 1: printf ("Value : "); scanf ("%d", &val);
 root = insert (root, val); break;
case 2: inorder (root); printf ("\n"); break;
case 3: printf ("search : "); scanf ("%d", &val
        printf (search (root, val) ? "found \n" : "Not
         Found \n");
        break;
case 4: printf ("Delete : "); scanf ("%d", &val);
         root = delete (root, val); break;
case 5: exit (0);
      }
   }
}
```

PAGE No.
DATE / /

b.

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

struct node {
    int data;
    struct node *left, *right;
};
struct node* newNode (int val) {
  struct node* n = (struct node*) malloc (sizeof
  (struct node));
  n -> data = val;
  n -> left = n -> right = NULL;
  return n;
}
struct node* insert (struct node* root, int
                     val) {
 if (root == NULL) return newNode (val);
 if (val < root -> data) root -> left = insert
    (root -> left, val);
 else root -> right = insert (root -> right, val);
 return root;
}
void inorder (struct node* root) {
 if (root) {
  inorder (root -> left);
 printf ("%d", root -> data);
  inorder (root -> right);
  }
}
void preorder (struct node* root) {
 if (root) {
```

```
    printf(" %d", root -> data);
    preorder (root -> left);
    preorder (root -> right);
  }
}
void postorder (struct node* root){
  if (root) {
    postorder (root -> left);
    postorder (root -> right);
    printf(" %d", root -> data);
  }
}
int main () {
    struct node * root = NULL;
int ch, val;
while (1) {
printf("\n 1. Insert 2. Inorder 3. Preorder
      4. Postorder 5. Exit \nEnter choice: ");

scanf (" %d", &ch);
switch (ch){

case 1: printf ("Enter value : ");
        scanf (" %d", & val);
        root= insert(root, val);
break;

case 2: printf("Inorder: "); inorder (root);
        printf("\n");
break;
```

PAGE No.
DATE / /

```
case 3: printf ("preorder : "); preorder (root);
        printf ("\n");
break;

case 4: printf ("Postorder: "); postorder (root);
        printf ("\n");
break;

case 5: exis exit (0);
default : printF("Invalid choice!\n");
```

}
}
}

DATE / /

Experiment - 12

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

void int(int arr[V][V]) {
    int i,j;

for(i=0, i<v; i++)
for (j=0, j<v ; j++)
arr[i][j]=0;
}
void insert edge (int arr[v][v] int:
        int j>h;
        arr[j][i]=1;
}
void print adj Matrix (int arr[v][v]){
            int i,j;
printf("Adjacency Matrix: \n);
   for(i=0; i<v; i++){
   for (j=0; j<v; j++) {
   printf("%d", arr[i][j]);
}
 printf ("\n");
  int main () {
  int adj matrix [v][v];

     int (adj Matrix);

insert (adj matrix 0,1);
insert Edge (adj matrix 0,4);
```

```
insert Edge(adj matrix 1,2);
insert Edge(adj matri x 1,3);
insert Edge(adj matrix 1,4);
insert Edge(adj Matrix 2,3);
insert Edge (adj Matrix 3,4);

print Adj Matrix (adj Matrix);

return 0;

}
```

→ O/P →

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