Extra codes

Q Declare an array of 5 subjects. Accept subject marks from user & calculate avg %.

Q Write a program to number n from user is even or odd

Q Link a program to print prime numbers b-w 1 to 50

Q Write a ~~n~~ menu driven program.

Q Write a program to accept no. from program & check if it is palindrom or not.

Q1→

```
# include <stdio.h>

int main () {
  float marks [5];
  float total = 0, average;
  printf ("Enter marks for 5 subjects: ");
  for(int i = 0; i < 5; i++) {
  printf ("subject %d: ", i + 1);
  scanf ("%f", &marks[i]);
  total + = marks [i];
  }
  average = total / 5
  printf ("Total marks: %.2f", total);
  printf("Average percentage: %.2f%%", average)

  return 0;
}
```

92 Write a program to number n from user is even or odd

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

3 Link a program to print prime numbers by 1-50

```
→ #include <stdio.h>
  int main () {
    int num, j, i;

    Printf("prime numbers between 1
    and 50 are:");

    for (num = 2; num <= 50; num ++) {
    j = 1;
    for (i = 2; i <= num/2 ; i++) {
      if (num % i == 0) {
      j = 0;
```

```
break;
}
}

if (j) {
Printf ("%d", num);
}
}
   return 0;
~~Prin~~        }
```

Q Write a ~~program~~ menu driven program

→

```
# include <stdio.h>

int main() {
   int choice;
      printf("select a day of the week:");
      printf("1 - monday");
      printf("2 - tuesday");
      printf("3 - wednesday");
      printf("4 - thursday");
      printf("5 - Friday");
      printf("6 - saturday");
      printf("7 - sunday");
      printf("Enter your choice (1-7):");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("monday");
            break;
```

```
case 2;
 printf("Tuesday");
break;

case 3;
 printf("wednesday");
 break;

case 4;
printf("Thursday");
 break;
case 5;
printf("Friday\n");
 break;
 ~~co~~
 case 6;
   printf("Saturday");
   break;
 case 7;
 printf("sunday");
   break;
   default;
 Printf("Invalid choice, please
 enter number between 1 and
 7"); }
   return 0;
                }
```

Q5 Write a program to accept no. from program & check it is palindrom or not.

```
#(stdio.h)

int main (){
  int num, reversedNum = 0, remainder,
  orignal Num;
  printf("Enter an integer:");
  Scanf("%d", &num);
  orignalnum = num;
  while (num != 0) {
   remainder = num % 10;
   reversedNum = reversedNum * 10 + remainder;
   num /= 10;
   }
   if (orignal Num == reversedNum) {
    printf("%d is a palindrome.", orignal
      num);
   }else {
      printf("%d is not a palindrome
      orignal Num);
   }
   return 0;
}
```

Experiment: 1

Write program to implemen array operation

1 Find avg of 10 numbers using array.

```
#include <stdio.h>
int main () {
int ~~ma~~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;
}
```

off

2 ~~Find~~ Display the following pattern

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

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

9) Find first repeating element in array

```
#include <stdio.h>

int main()
{
int arr[] = {4, 7, 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", arr
   found = 1;
   break;
   }
  }
   if (found) {
     break;
   }

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

Q4 Find greatest and smallest element in 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; i++]
  {
    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;
}
```

5 Write a program squaring the odd position element

-> #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:");
for(i=0; i<n; i++)
{
printf("%d\n", arr[i]);
}
return0;
}
```

11/7/25

Q1 search data using linear search consider following list to perform linear search

例 56, 36, 89, 57, 1, 0, 67, 59
search item 1 from the list and write the item is found in procedure

Q2 search data using binary search

Q3 compair linear search and binary search

Q4 state limitations of linear search interms of time complexity

Extra codes

* Display following pattern

1)
```
*
* *
* * *
* * * *
```

→
```
#include <stdio.h>
int main ()
int i, j ;

for (i =1 ; i<= 4 ; i++) {

    for (j =1 ; <=i ; j ++) {

    printf (" **") ;

    }
  }
  return 0;
}
```

2
```
  *
 * *
* * *
* * * *
```

→
```
# include <stdio.h>
int main () {
int i, j ;

for (i =1 ; i <= 4; i + +) {

  for (j=1 ; j <=4-1 ; j++) {
```

```
~~for~~
printf("__ ");
}
for (j = 1; j <= i; j++) {
printf("*__ ");
}
}
return 0;
}
```

3

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

→

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

int i, j ;
~~for (j = 1 ; j <= 4 ; j ++)~~ {
for (j = 1 ; j <= 4-i ; j ++) {
printf("__ ");
}
for (j = 1 ; j <= i ; j ++) {
printf("* ");
}
}
return 0; }
```

4) * * * *
* * *
* *
*

→
```
# include <stdio.h>
int main ()
{
 int i, j;

 for (i=4; i>=1; i--) {

 for (j=1; j<=i ;j ++) {

 print ("* ");
 }
  print ("\n");
  }
   return 0;
  }
```

5 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; j <= i; j++) {
  printf ("%d", i);
}
printf ("\n");
}
return 0;
}
```

Experiment - 2

Q1 search data using linear search
56, 36, 89, 52, 1, 0, 62, 59
search item 1 from the list
and write the item is found
in the procedure

```
#include <stdio.h>
int main()
{
    int m, k, l, h, i;
    int arr[7];
    printf("enter array element:");
    for (i = 0; i<7; i++)
    {
     scanf ("%d", & arr [i]);
    }
    printf ("enter search key:");
    scanf ("%d", & k);

    l = 0
    h = 6;
     while (l < h)
     {
       m = (l + h)
       {
         m = (l + h) /2;
         if (arr[m] == k)
         {
          printf("element ~~found~~ position : %d", m);
```

```
break;
}
else
{
 if( arr[m] < ks)
  {
    l = m + 1;
  }
  else
  {
   h = m-1
  }
 }
}
if (l > h)
{
 printf(" no element found");
}
return 0;
}
```

* Output :-
Enter ~~array~~ array elements :- 56, 36, 89, 52, 1, 0, 62, 5
Enter search key : 1
element found

b

```
# include <stdio.h>
int main ()
{
int a [8], k;
printf ("Enter elements: ");
for(int i = 0; i <8 ; i++)
{
  scanf ("%d", & a [i]);
}
printf ("Enter the search key: ");
scanf ("%d", & k);
for (int i = 0 ; i < 3 ; i++)
{
  if (a [i] == k)
  {
    printf (" Element found");
    return 0;
  }
}
printf (" Element not found. ");
return 0;
}
```

* Output:

Enter elements: 56 36 84 52 1 0 67 52

Enter the search key :- 55.

Element not found.

2. search data using binary search

→
```
# include <stdio.h>
int main()
{
int a[10], l, h, m, k;
printf("Enter 10 elements (sorted),")
for (int i=0; i<10; i++)
{
scanf("%d", &a[i]);
}
printf("Enter the search key:");
scanf("%d", &k);

l=0, h=9
while (l<=h) {
m=(l+h)/2;
if (a[m] == k)
{
printf(" Element found");
break;
}
else
{
if(k < a[m])
{
h = m-1;
}
else
{
l = m+1;
}
```

```
}
}
printf("\n Element not found");
return 0;
}
```

* output

```
enter elements of the array in
sorted order :- 56
36
89
57
1
0
67
59
enter element you want to search :- 55
element not found
```

3 compare linear search and binary search.

Linear search	Binary search
i) it works on both unsorted and sorted arrays	i) it works only on sorted arrays.
ii) its algorithm type is sequential	ii) its algorithm type is divide and conquer
iii) it is simple to implement.	iii) it is more complex to implement.
iv) Time complexity is (n)	iv) time complexity is (log n).
v) it is used when data is smaller unsorted	v) it is used when data is large and unsorted.

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 its present or not. it makes it slower than binary search which operates

* Extra questions

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 elements: ");
    scanf("%d", &n);

    printf("Enter the elements: ");
    for (i=0; i<n; i++) {
    scanf("%d", &arr[i]);
    }
    for (i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
    }
    for(i = 0; i < n; i++) {
    rev[i] = arr[n-1 - i];
    }
    printf("original array: ");
    for (i=0; i<n; i++) {
    printf("%d", arr[i]);
    }

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

```
return 0;
}
```

→ output

```
Enter ~~number~~ number of elements:- 5
Enter the elements:
1
2
3
4
5

Original array: 1 2 3 4 5
Reversed array: 5 4 3 2 1
```

2 WAP in C to count total number of duplicate elements in an array

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

output:-

```
Enter number of elements: 5
Enter array elemnts:
34
65
75
34
57
Total number of duplicate elements=1
```

3) WAP in C to print all unique elements in an array

```
# include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 2, 4, 5, 1, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    int i, j, count;
    printf("unique elements in the array
    are: ");
    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 are:
3 4 5 6

4) WAP to seperate odd and even integer into seperate arrays.

```
#include <stdio.h>
int main() {
    int arr[] = {10, 21, 4, 45, 66, 93, 1};
    int even[10], odd[10];
    int i, even count = 0, odd count = 0;
    int n = size of (arr) / size of (arr[0]);
    for (i = 0; i < n; i++) {
    if (arr[i] % 2 == 0) {
    even [even count ++] = arr[i];
    } else {
    odd [odd count ++] = arr[i];
    }
    }
    printf ("Even numbers: ");
    for (i = 0; i < even count; i++) {
    printf ("%d", even[i]);
    }
    printf ("odd numbers:");
    for (i = 0; i < odd count; i++) {
    printf ("%d", odd[i]);
    }
    return 0;
}
```

output

Even numbers:
10 4 66
odd numbers:-
21 45 93 1

5. WAP to find second smallest element in an 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", arr[i];
      return 0;
      }
      }
      printf("All elements are equal.
         No second smallest.");
      return 0;
```

output:-

second smaltest 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);

for (i = 0; str[i] != '\0', i++) {
if (str[i] == ' ') {
words ++;
}
}
printf (" Total number of words
: %d \n",
return 0;
}

output
enter a string: I am kartik
Total number of words = 5.

Experiment 3

Q:)

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 = size of (arr) / size of (arr[0]);
```

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

bubble sort (arr, n);

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

return 0;
}
```

(ii)

```
#include <stdio.h>
void selection sort descending [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 [max Index])
{
   maxIndex = j;
   }
}
 if (maxindex != i) {
 temp = arr [i];
 arr [i] = arr [max Index];
```

```
        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);

    SelectionSort decending (arr, n);

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

    return 0;
}
```

PAGE No.
DATE / /

iii)
```
# include <stdio.h>

int main () {

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

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

   temp = arr [j];
   arr[j] = arr [j+1];
   arr [j+1] = temp;
   }
  }
}
 printf(" Total no. of comparison :%d\n", comparison
 );

 return 0;
}
```

iv) # include <stdio. h>

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

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

void selection sort (int arr[] ,int n) {
int i, j, minIndex, temp;

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

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

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

```
int n = size of (arr) / size of (arr[0]);

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

Selection sort (arr, n);

printf ("\n final sorted array (ascending
        order) : \n");

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

return 0;
}
```

* 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 ele is found at end, exclude that element & repeat from step 1.

6. For n number of elements n-1 itera will occur

b) Selection Sort:

1) Start from the 0th element fro list (i) and compare with ea element

2) If arr[i] is greater than any elem swap the two.

3) At end of first iteration smallest element is found at beginning exclude that and repeat step

4) Repeat step 3 for each position /element till only one element remains unsorted (largest).

5) Hence, Sorted list is obtained

c) Insertion Sort:

1) Sort with 1st index (2nd position)
2) compare it with element befor i.e. 0th element.
3) If it is smaller, insert in position of 0th element by sniffing all others to the right.

4) To move to next element compare with 0th & 1st, insert at it's appropriate position by making space.

5) Repeat with all elements until list is sorted.

d) Radix Sort:

1) Define 10 queues each with buckets from digit 0 to 9.

2) Consider least significant digit of each no. in the list to be sorted. And insert each number in respective 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.

13/11

PAGE No.
DATE / /

Experiment - 04

1)
```
# include <stdio.h>

void insertion sort (in 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 print Array (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 = size of (arr) / size of (arr [0]);

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

```
insertionSort(arr, n);

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

printArray(arr, n);

return 0;
}
```

```
#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 countrySort (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)
counting sort (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 = size of (arr) / size of (arr[0]);

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

radix sort (arr, n);
printf ("Sorted array in ascending order
        using Radix sorted);
printArray (arr, n);

return 0;
}
```

5 # include <stdio.h>

```
void insentionsort (int arr[], int n){
int i , key , J ;

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

if ( i = = 2) {
printf ("\nOutput after 2nd iteration of Insention
          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 Print Array (int arr [] ; int n) {
for (int i=0; i<n; i++)
 printf ("%d", arr [i]);
 printf ("\n");
}
 int main () {
 int arr 1 [] = { 7, 3, 5, 1, 9, 8, 4, 6 };
 int arr 2 [] = { 7, 3, 5, 1, 9, 8, 4, 6 };
 int n = size of (arr 1) / size of (arr 1[0

printf ("Original Array : \n");
 print Array (arr 1, n);

insertion sort (arr1, n);

radix sort (arr 2, n);
 printf ("Sorted array using radix sort:
 print Array (arr 2, n);

 return 0;
}
```

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("Enter choice : ");
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_~~beginning~~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 again\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
                                  (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 (size of
                                  (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 = (structnode*) malloc(size of (structnode));
  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
                                  , pos);
}

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

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

printf ("Enter value to search: ");
scanf ("%d", &value);

while (current != NULL) {
  if (current -> data == value) {
  print f ("value %d found at position %d
                                  \n", value, p
  return;
}
current = current -> next
```

```
}postt
~~current~~
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 rang \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 <stib.h>

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

Struct node * head = NULL

Structnode* create Node (int data) {
 struct node = (struct node*) malloc (size of (structno
                                    -e));
 New node -> data = data;
 new node -> next = NULL;
 return new node;
}

void insert_at_end (int data) {
 struct node* new node = create node (data);
 if (head == NULL) {
     head = new node;
     return;
}
```

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

void display list () {
  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 delete first () {

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 delete last () {
 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 deleted at position (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* node to delete = temp->next;
temp->next = node to delete->next;
free (node to delete);
print f(" 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);
        insert at end (data);
        break;
  Case 2:
        display list();
        break;
  Case 3:
        delete first();
        break;
  Case 4:
        printf ("Enter position to delete: ");
        scanf ("%d", &pos);
        delete at position (pos);
        break;
  Case 5:
        delete last();
        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 * create node (int data) {
struct node * new node = (struct node*) malloc
                                (size of (struct node
new node -> data = data;
new node -> next = NULL;
return new node;
}

void insert at End (struct node** head, int data
struct node* = create node (data);
if (*head == NULL) {
   *head = new node;
   return;
}
  struct node* temp = *head;
  while (temp -> next) = NULL)
     temp = temp -> next;
     temp -> next = new node;
}

void display list (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 count nodes (struct node* head) {
  int count = 0
struct node* temp = head;
while (temp != NULL) {
    count ++;
    temp = temp->next;
  }
```

```
    return count;
}

Struct node* reverse list (struct node* h
    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 search node (struct node* head, int key
    int pos = 1;
    struct node* temp = head;
    while (temp! = NULL) {
    if (temp -> data = = key) {
    pointf ("node with value %d found a
            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, list choice;

while (1) {
    printf(" \n menu: \n");
    printf(" 1. create (insert at end) \n");
    printf(" 2. Display \n");
    printf(" 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);
        printf(" Enter data: ");
        scanf(" %d", & data);
        if (list choice == 1)
```

```
            insert at end(&head 2, data);
        else
            printf("Invalid list choice!\n");
        break;

    case 2:
        printf("which list? (1,2 or 3): ");
        scanf("%d", &list choice);
        if (list choice == 1)
            display list(head 1);
        else if (list choice == 2)
            display list (head 2);
        else if (list choice == 3)
            display list (~~head3~~ result);
        else
            printf("Invalid choice!\n");
        break;

    Case 3:
        result = concatenate (head1, head2);
        printf = ("list concatenated into list3\n
        break;

    Case 4:
        printf("which list? (1,2 or 3): ");
        scanf("%d", &list choice);
        if (list choice == 1)
        printf("node count: %d\n", countnodes(head
        else if (list choice == 2)
        printf("node count: %d\n", countnodes(head2
        else if (list choice == 3)
        printf("node count: %\n", count nodes(result));
```

```
else
    printf("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)
    result = reverse list(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", &list choice);
  if (list choice ==1)
      search node (head1, key);
  elese if (listchoice ==2)
      Searchnode (head2, key);
  else is (list choice ==3)
      search node (result, key);
  else
      printf("Invalid choice!\n");
    break;
```

```
    case 7:
        exit(0);

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

6!!!

```
#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;
}
```

```
#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) {
```

```
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;
}
```

```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;
}
```

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

// Structure for doubly linked list node
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* head = NULL;

// Function to create a new node
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;
}

// Function to insert node at the end
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;
}

// Function to display list
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");
}

// Function to search a node
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);
    }

    // Function to count number of nodes
    void countNodes() {
        struct Node* temp = head;
        int count = 0;

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

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

    // Function to reverse the doubly linked list
    void reverseList() {
        struct Node *current = head, *temp = NULL;

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

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

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

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

    // Main menu
    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;
}
```

13/11

PAGE NO.
DATE / /

Experiment 7

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

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

int is full () {
return (top == max - 1);
}
int is Empty () {
return (top == -1);
}
void push (int value) {
 if ( is full (1) {
 printf ("stack overflow! Cannot push %d
                \n", value);

} else {
stack [++top] = value;
 printf (" %d pushed \n", value);
 }
}
void pop () {
if (is empty ()) {
    printf (" stack underflow! Cannot pop \n");
} else {
printf ("%d popped \n", stack [top--]);
 }
}
```

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

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

  if ( is Empty ())
    printf ("stack is Empty now\n")

  return 0;
}
```

ii)

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

int stack
 int top = -1;

int is full () {
 return (top == MAX -1);
 }
 int is Empty () {
 return (top == -1);
}
 void push(int value) {
```

```
if (is full ()) {
printf("Stack overflow! cannot push %d\n",
                                value);
 } else {
  stack [++top] = value;
 printf ("pushed %d \n", value);
    }
 }
void pop ()
 if (is Empty ()) {
    printf("stack underflow! cannot pop\n
                                  ");
 } else {
 printf ("popped %d \n", stack [top--]);
   }
}
 void display () {
if (is Empty ()) {
 print f ("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) ;
  ~~push~~
```

```
pop();
push(25);
push(50);
push(70);
pop();
push(100);
pop();

display();

return 0;
}
```

13/11

Experiment 8

i)
```
#include <iostream>
#include <e.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 (top != -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	*)	H G
Y	*)*	H G
)	*)*)	H G F
F	*)*)	H G F
n	*)*)^	H G F E
E	*)*)^	H G F E ^
/	*)*)^\	H G F E ^ D
D	*)*)/	H G F E ^ D
C	*)*)/c	H G F E ^ D /
-	*)-	H G F E ^ 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: ");
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;
}
```

13/11

Experiment 9

Q1)
```
# include <stdio.h>
# include <stdib.h>

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

  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)
{
 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 enque()
{
int element();
if (rear = SIZE -1)
{
Printf ("\n queue is full ...");
}
else
{
printf("\n Enter 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+++;
   }
}
void display ();
{
printf("\n queue is empty...");
}
 else
 {
  printf("\n queue element are: \n")
  for (int i= front , i<= near ; i++)
{
 printf ("%d \n", queue [i])};
 printf ("\n");
 }
}
```

b) WAP to count digits & spaces using f
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 occurred while opening!"
        << end ;

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

while (new_file.get (ch)) {
 if (is digit (ch)) {

   digit_count ++;
 }
 if (is space (ch)) {
   space_count ++;
 }
 new_file . close (ch)} {
   space_count ++;
 }
 new_file . close ();
 cout << " No. of digits : " << digits_count << en
                                                  ;

 cout << " No of space: " << spaces_count << end;

return 0;
}
```

NM
13/11

PAGE No.
DATE / /

Experiment 9 [extra questions]

1] Time complexity of sorting algorithms

Algorithm	Best	Average	Worst
linear search	$O(1)$	$O(n)$	$O(n)$
Binary search	$O(1)$	$O(\log n)$	$O(\log n)$
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Heap sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Radix Sort	$O(nk)$	$O(nk)$	$O(nk)$
Shell sort	& $O(n)$	$O((n\log n)^2)$	$O((n\log n)^2)$

2] Perform following operation on linear queue in array size (10)

i) insert (10)

10									

Front = 0 Rear = 0

ii) insert (50)

10	50							

Front = 0 Rear = 1

iii) Delete

~~10~~	50							

Front =

	50							

Rear

iv) insert (100)

	50	100						

front = 1 Rear = 2

v) insert (20)

	50	100	20					

front = 1 Rear = 3

vi) Delete ~~(50)~~

	~~50~~	100	20					

		100	20					

front = 2 Rear = 3

vii) Insert (25)

		100	20	25					

front = 2 Rear = 4

viii) insert (200)

		100	20	25	200				

front = 2 Rear = 5

3] Explain priority queue in detail

→ Priority queue is assigned a priority and the element with highest priority is delete befor other elements regardless of order they were inserted. It does not follow [FIFO]

It has 2 types:

- Ascending priority queue -
  The smallest priority number mean highest priority. Element's with larger priority number are removed firsts

4] Write applications at Queue.

→ a) serving request on a single shared resources, like a printer.

b) Handling of intrupts in real-time system. The interrupts are handled in the same order as they arrive. ie. first come

first served.

(c) FCFS Job scheduling.

(d) Online banking found transfer request

PAGE No.
DATE / /

Experiment 10

- Circular queue:

```
#include <stdio.h>
# include <stdio.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 \3. Display
\4. exit");
eto {{
printf ("\n enter your choice");
scanf ("%d", &ch);
Switch (ch) {
case 1: enqueue();
Break;
case 2: dequeue();
Break3;
case 3: display ();
Break;
case 4: exit (0);
defaut {:
printf ("\n invalid choice!\n");
}
```

```
} while (cul = 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)
    {
      print("\n queue is empty");
    }
    else
    {
      printf("\n deleted %d from queue", queue[fr
        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
     print f (" \n queue elements are "),
     while (i) {
    print f (" %d ", Queue [i]);
        if ( i == rear)
         Break ;

     i = (i+1) % size ;
    }
    printf (" \n");
  }
}
```

NA
13/1

Experiment 11

1) Creak binary tree and traverse inorder, preorder and post order.

```
#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 = items;
        new node -> left = NULL;
        new node -> right = NULL;
            return = new node;
    }
    Struct node* insert at left (struct node * right,
                                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 roots -> right;
}

void inorder (struct node + roo) {
    it (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 post order (struct node * root) {
        if (root = = null)
        return;
    postorder ( root -> right);
    post order (root -> left);
    printf ("%d", root -> item);
}

int main () {
    struct node* root = create node (40);
    insert at left (root, 28);
    insert at right (root, 27);
    insert at left (root -> left, 60);
    insert at right (root -> left, 30);
```

```
insert at left(root → right, 50);
insert at right (root → right, 95);

printf(" \n order Transversal:");
inorder(root);

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

printf(" \n postorder Traversal:");
post order (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 at tree

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

struct node {
   int item;
  struct node * left;
  struct node * right;
}
struct node* create node (item) {
{
 struct node * 'new node* = malloc (size of
                    (struct node)))
      new node -> 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;
}
```

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

 int main () {
    int adjmatrix [v][v] ;
      int (adjmatrix],
```

```
insert edge (adjmatrix, 0, 1);
            (adjmatrix, 0, 2);
            (adjmatrix, 1, 2);
            (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 * New Node (int dest) {
    Struct Node * temp = (structnode)
                (size of (struct node))

    temp -> dest = dest;
    temp -> Next = Null;
        return 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 add edge (struct graph* graph, int src,
                                int dest) {
Struct node* temp = New Node (dest);
temp-> Next = graph-> adj [src];
    graph-> adj [dest] = temp;
}

void print graph [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;
  }
    printf ("\n");
  }
}
int main () {
Struct graph* graph = great 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, 3, 4);
print graph (graph);

  return 0;
}
```