ADS

Programs:-

Declare int array of 5 subjects. Accept subject marks & find %.

WAP to check whether given no is even or odd.

Find prime numbers btw 1 to 50

Ito a menu driven program if the days of the week from Mon-Fri print weekday else weekend.

Write a C program to accept no from user. Check whether given number is palindrome or armstrong or not.

1
```
#include<stdio.h>
int main()
{
int arr[5];
float m;
int sum=0;
printf("Enter array elements");
for(int i=0; i<5; i++)
 {
 scanf("%d", &arr[i]);
 }
for(int i=0; i<5; i++)
 {
  sum = sum + arr[i];
 }
m = sum/5;
printf("the percentage of 5 subjects is: %f", m);
return 0;
}
```

2
```
#include<stdio.h>
int main()
{
int n;
printf("Enter a number");
scanf("%d", &n);
if(n%2==0)
printf("Even number");
else
printf("Odd number");
return 0;
}
```

③
```
#include <stdio.h>
int main (){
printf("The prime numbers between 1 to 50 a
for(int i=1, i<50, i++)
 {
   int n = i;
   int c = 0;
for(int j=1; j<=i; j++)
  {
    if(n%j==0)
    {
      c++;
    }
  }
if(c==2)
printf("the prime numbers are: \n %d ",
return 0;
}
```

④
```
#include <stdio.h>
int main (){
int n;
printf("Enter a number ");
scanf("%d",&n);
int rev = 0;
int t = n;
while (n>0)
{
int rem = n%10;
rev = rev*10 + rem;
n = n/10;
}
```

```
if (rev==t)
printf("It is a pallindrome number");
else
printf("Not a pallindrome number");
return 0;
}
```

⑤
```
#include <stdio.h>
int main(){
int n;
printf("Enter 1 for Monday");
printf("Enter 2 for Tuesday");
printf("Enter 3 for Wednesday");
printf("Enter 4 for Thursday");
printf("Enter 5 for Friday");
printf("Enter 6 for Saturday");
printf("Enter 7 for Sunday");
printf("Enter user's choice");
switch(n)
{
case 1: printf("It is a weekday");
        break;
case 2: printf("It is a weekday");
        break;
case 3: printf("It is a weekday");
        break;
case 4: printf("It is a weekday");
        break;
case 5: printf("It is a weekday");
        break;
case 6: printf("It is a weekend");
        break;
case 7: printf("It is a weekend");
```

default: printf("The number is out of limit");
break;
}
return 0;
}

EXPERIMENT - 1

1) WAP to implement array operation:-

i) Find avg of 10 no's using our array

2) Display pattern:-

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

3) Find the 1st repeating element in an ar

4) Find the largest & smallest number

5) Squaring the odd position element

Answers:

i) #include <stdio.h>
int main() {
int arr[10];
int sum = 0;
float avg;
printf("Enter 10 array elements");
for (int i = 0; i < 10; i++)
{
scanf("%d", &arr[i]);
}
for (int i = 0; i < 10; i++)
{
sum = sum + arr[i];
}
avg = avg / 10;
printf("The average of all elements: %d", avg
return 0;
}

Output :- avg :- 5.0000

ii) #include <stdio.h>
int main() {
for (int i = 1; i <= 5; i++)
{
for (int j = 1; j <= i ; j++)
{
if (i % 2 == 0)
{
printf("#");
}
else
{
printf("*");
}

Output :-
```
*
# #
* * *
# # # #
```

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

iv)
```
#include <stdio.h>
int main() {
int arr[10];
int largest=arr[0];
int smallest=arr[0];
printf("Enter array elements");
for(int i=0; i<10; i++)
 {
 scanf("%d", &arr[i]);
 }
for(int i=0; i<10; i++)
  {
    if(largest<arr[i])
     {
       largest=arr[i];
     }
    if(smallest>arr[i])
     {
       smallest=arr[i];
     }
  }
printf("The largest element: %d", largest);
printf("The smallest element: %d", smallest);
return 0;
}
```

v)
```
#include<stdio.h>
int main() {
int arr[10];
printf("Enter array elements");
for(int i=0; i<10; i++)
  {
   scanf("%d", &arr[i]);
  }
for(int i=0; i<10; i++)
{
   if(i%2==0)
   {
    printf("arr[i]");
   }
  else
   {
    printf("%d", (arr[i]*arr[i]));
   }
}
return 0;
}
```

Output:-
Enter no of elements: 5
Enter 5 elements:-
1
2
3
4
5
Array:- 1
2
9
4
25

iii)
```
#include <stdio.h>
int main (){
int arr[10];
printf("Enter array elements");
for(int i=0; i<10; i++)
{
  scanf("%d", &arr[i]);
}
for(int i=0; i<9; i++)
{
for(int j=i+1; j<10; j++)
{
  if (arr[i] == arr[j])
  {
  printf("First repeating number is: %d",
                                  arr[i]);
return 0;
}
}
}
printf("No repeating elements found");
return 0;
}
```

Output:-
Number of elements: 5
Enter 5 elements:-
5
2
5
5
2
First repeating number is 5

PATTERNS

i)
```
*
* *
* * *
* * * *
```

ii)
```
1
2 2
3 3 3
4 4 4 4
```

iii)
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

iv)
```
*
# #
$ $ $
? ? ? ?
```

v)
```
* * * * *
  * * *
    * *
      *
```

vi)
```
1
1 2
3 3 3
1 2 3 4
5 5 5 5 5
```

viii)
```
    *
  * *
* * *
* * * *
```

ANSWERS

i)
```
for Cinte
#include <stdio.h>
int main (){
for(int i=01; i<=4; i++)
{
for(int j=01; j<=i; j++)
{
    printf("*");
}
printf("\n");
}
return 0;
}
```

ii)
```
#include <stdio.h>
int main() {
for (int i = 1 ; i <= 4 ; i++)
{
for (int j = 1; j <= i ; j++)
{
    printf ("%d", i);
}
printf ("\n");
}
```

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

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

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

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

EXPERIMENT - 2

1) Search data using Linear search
Consider the following list to perform linear search.
(56, 36, 89, 57, 01, 00, 67, 8

i) Search the item 01 from the above list & write the item is found or not with procedure.

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

Program

```
#include <stdio.h>
int main ()
{ int k;
 int arr[] = {56, 36, 89, 57, 01, 00, 67,
printf ("Enter the element to be found
scanf ("%d", &k);
 int c = 0;
for (int i = 0; i<10 ; i++)
{
  if (k == arr[i])
{
printf ("Element found at position %d
c++;
break;
}
}
if (c == 0)
{
printf ("Element not found");
}
return 0;
}
```

Output:-

i) Enter element to be found
01
Element found at position 4

ii) Enter element to be found
55
Element not found.

2) Search data using Binary Search

```
#include <stdio.h>
int main ()
{
  int n, h, l, k, mid ;
printf ("Enter the limit of array \n"
scanf ("%d", &n);
int arr[n];
printf ("Enter the sorted elements in
        the array");
for (int i = 0; i<n; i++)
{
   scanf ("%d", &arr[i]);
}
printf ("Enter element to be found"
scanf ("%d", &k);
 l = 0;
 h = n-1;
```

```
while (l <= h)
{
mid = (l+h)/2;
if (arr[mid] == k)
{
printf ("Element found at the element");

break;
}
else
if (arr[mid] > k)
{
  h = mid - 1;
}
else
{
  l = mid + 1;
}
}
if (l > h)
{
printf ("Element not found");
}
return 0;
}
```

Output:-

Enter the limit of the array

10

Enter sorted element in an array:

Enter element to be found

2

Enter found at the element

1

3) Compare Linear & Binary search.

Linear Search	Binary Search
i) Unsorted or sorted data	i) Must be sorted
ii) Sequential Scanning	ii) Divide & Conquer
iii) O(n) [Time Complexity]	iii) O(log n) [Time Complexity]
iv) Less efficient	iv) More efficient
v) Simpler code	v) More complex code

4) Limitations of Linear Search in terms of time complexity as follows;

1) Not efficient for large data sets: Linear Search has a time complexity O(n), where n is the number of elements in the list.

2) No Improvement for Sorted data: Linear Search does not take advantages of sorted data

3) Avg & worst case are the same order: The avg case & worst case both require scanning about half or all the elements respectively so it doesn't perform an avg.

4) Not suitable for times: Critical Applications Due to its linear time, it is not ideal for application where fast searching is necessary, especially when large datasets.

PRACTICE QUESTIONS

WAP to copy the elements of one array into another array in reverse order.

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

WAP in C to print all unique elements in an array (non-repeating elements).

WAP in C to separate odd & even integers into separate arrays.

WAP to find the 2nd largest element in the array.

Total no. of words in a string.

20)
```
#include <stdio.h>
int main()
{
 int arr[10];
 printf("Enter elements of an array");
 for(int i=0; i<10; i++)
 {
  scanf("%d", &arr[i]);
 }
 int c=0;
 for(int i=0; i<10; i++)
 {
 for(int j=i+1; j<10; j++)
  {
   if(arr[i]==arr[j])
     c++;
  }
 }
 printf("The total number of duplicate
        elements in our array\n");
 printf("%d", c);
 return 0;
}
```

Output :-

Enter elements of an array :-
1
2
2
3
1
5
6
7
8
4

The total number of duplicate elements in an array:
2

39) #include <stdio.h>
int main()
{
int arr[10];
printf("Enter elements of an array");
for(int i=0 ; i<10; i++)
{
scanf("%d", &arr[i]);
}
int c=0;
printf("The unique elements are\n");
for(int i=0 ; i<10; i++)
{
int ch=0;
for(int j=0 ; j<10; j++)
{
if(arr[i]==arr[j] && i!=j)
{
ch=1;
break;
}
}
if(ch==0)
printf("%d\n", arr[i]);
}
return 0;
}
Output:-
Enter elements of an array
1
3 3
3 3
2
4

40) #include<stdio.h>
int main()
{
int arr[100], e[100], o[100], n, i, c1, c2;
printf("Enter number of array elements");
scanf("%d", &n);
printf("Enter array elements");
for(i=0 ; i<n; i++)
{
scanf("%d", &arr[i]);
} c1=0; c2=0;
for(i=0 ; i<n; i++)
{
if(arr[i]%2==0)
even[c1] = arr[i];
++c1;
}
else
{
odd[c2] = arr[i]
++c2;
}
printf("\n Even array :-\n");
for(i=0 ; i<c1; i++)
{
printf("%d", even[i]);
}
printf("\n Odd array :-\n");
for(i=0; i<c2; i++)
{
printf("%d", odd[i]);
}
return 0;
}

Output:-
Enter no of array elements: 5

Enter Array Elements
1
22
339
56
9

Even Array:
22
56

Odd Array:
1
339
9

5) #include <stdio.h>
int main()
{
int arr[100], i, min, min2, n;
printf("Enter array elements");
for (i=0; i<10; i++)
{
scanf("%d", &arr[i]);
}
min = arr[0];
for (i=0; i<10; i++)
{
if (arr[i] < min)
min = arr[i];
}
for (i=0; i<05; i++)
{
if (arr[i] != min)
min2 = arr[i];
}
for (i=0; i<05; i++)
{
if (arr[i] < min2 && arr[i] != min)
min2 = arr[i];
}
printf("Second Smallest element in array:
%d", min2);
return 0;
}
Output:
Enter Array Elements
1
2
3
4
5

Second Smallest element : 2

⑥
```
#include<stdio.h>
int main()
{
int i=0, c=0;
char str[100], c;
printf("Enter a string:");
scanf("%[^\n]%c", str, &c);
while(str[i]!='\0')
{
if(i==0 || str[i]==' ')
{
++c;
}
++i;
}
printf("\n No of words in string:%d", c);
return 0;
}
```

Output:
Enter a string: Boy says bye
No. of words in string: 3

①
```
#include<stdio.h>
int main()
{
int arr[100], b[100], i, c;
printf("Enter array elements");
for(i=0; i<5; i++)
{
scanf("%d", &arr[i]);
}
c=5-1;
for(i=0; i<5; i++)
{
b[c]=arr[i];
--c;
}
printf("New Array");
for(i=0; i<5; i++)
{
printf("%d\n", b[i]);
}
return 0;
}
```

Output:
Enter Array elements:-
1
2
3
4
5
New Array:
5
4
3
2
1

Experiment : 3

Q Sort element in ascending order using bubble sort?

Q Sort elements in descending order using selection sort?

Q Find the no of comparisons required in bubble sort having 5 no's (100, 200, 300, 400, 500)

Q Sort the given array in ascending order using selection sort & diagrammatic representation of for loop (500, -20, 30, 14, 50)

ANSWERS:

Q1)
```
#include <stdio.h>
int main()
{
int arr[100], i, j, c, temp;
printf("Enter array elements");
for (i = 0 ; i<5 ; i++)
{
scanf ("%d", &arr[i]);
}
for (i = 0; i<5-1 ; i++)
{ c = 0;
for (j = 0; j<5-i-1; j++)
if (arr[j] > arr[j+1])
{
temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
c = 1;
}
if (c == 0)
{
printf ("The number of passes %d", (i+1));
break;
}
}
printf ("The sorted array:");
for (i = 0 ; i<5 ; i++)
{
printf ("%d", arr[i]);
}
}
```

Output:-

```
Enter array elements:
100
200
300
400
500

Number of passes : 1
Sorted array:
100
200
300
400
500
```

Q
```
#include<stdio.h>
int main()
{
int arr[100], i, j, temp;
printf("Enter array elements");
for(i=0; i<10; i++)
{
 scanf("%d", &arr[i]);
}
for(i=0; i<10-1; i++)
{
for(j=i+1; j<10; j++)
{
   if(arr[i] < arr[j])
   {
      temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }
}
}
printf("The sorted array");
for(i=0; i<10; i++)
{
 printf("%d", arr[i]);
}
return 0;
}
```

Output :-

```
Enter array elements :
10
1
4
3
6
8
2
5
7
9

Sorted array :-
10
9
8
7
6
5
4
3
2
1
```

Q

100, 200, 300, 400, 500

PASS - 1

100, 200, 300, 400, 500

100, 200, 300, 400, 500

100, 200, 300, 400, 600

100, 200, 300, 400, 500

100, 200, 300, 400, [500]

∴ Therefore only 1 comparison is required for the following numbers.

Q 500, -20, 30, 14, 50

PASS 1

500, -20, 30, 14, 50

-20, 500, 30, 14, 50

-20, 500, 30, 14, 50

-20, 500, 30, 14, 50

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

PASS 2

-20, 500, 30, 14, 50

-20, 30, 500, 14, 50

-20, 14, 500, 30, 50

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

PASS 3

-20, 14, 500, 30, 50

-20, 14, 30, 500, 50

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

PASS 4

-20, 14, 30, 500, 50

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

ASCENDING ORDER:

-20, 14, 30, 50, 500

EXPERIMENT -4

1) WAP to sort data using insertion and radix sort.
2) Sort elements in ascending order using insertion sort.
3) Sort elements in ascending order using radix sort.
4) What is the output of the insertion sort after the second iteration following numbers :- 7, 3, 1, 9, 4, 8, 6
5) Sort the following no's using radix sort
   i) 100, 225, 390, 4130, 956, 99, 643
   ii) 85, 6, 99, 145, 239, 20, 18.

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

Output :-

```
Enter the no. of array elements
5
Enter array elements
6
2
6
3
2
Insertion sorted array:
2
2
3
6
6
```

2) #include <stdio.h>
int main() {
int n, i;
printf("Enter the number of array elements:");
scanf("%d", &n);
int arr[n];
printf("Enter array elements:");
for(i=0 ; i<n ; i++)
{
  scanf("%d", &arr[i]);
}
int max = arr[0];
for(i=1; i<n; i++)
{
 if(arr[i] > max)
 {
   max = arr[i];
 }
}
for(int exp=1; max/exp > 0; exp *= 10)
{
 int output[n], count[10] = {0};
 for(i=0; i<n ; i++)
 {
  count[(arr[i]/exp) % 10]++;
 }
for(i=1 ; i<10 ; i++){
  count[i] += count[i-1];
}
for(i=n-1 ; i>=0; i--)
{
 output[--count[(arr[i]/exp) % 10]] =
}

for(i=0 ; i<n ; i++) arr[i] = output[i];
}
printf("Sorted Array:");
for(i=0 ; i<n; i++)
{
printf("%d", arr[i]);
}
return 0;
}

Output:
Enter number of array elements: 5
Enter the array elements:
85
45
205
25
1

Sorted array: 1 25 45 85 205

a)

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

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

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

Output

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

b) 100, 225, 390, 4130, 956, 99, 5431

	0	1	2	3	4	5	6	7	8	9
0100	100									
0225						225				
0390	390									
4130	4130									
0956							956			
0099										99
5431		5431								

	0	1	2	3	4	5	6	7	8	9
100	100									
390										390
4130				4130						
5431				5431						
225			225							
956						956				
99										

	0	1	2	3	4	5	6	7	8	9
100		100								
225			225							
4130		4130								
5431					5431					
956										
390				390						
99	099									

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

Sorted :-

0099, 0100, 0225, 0390, 0956, 4130, 5431

ii) 25, 6, 99, 145, 239, 20, 18

	0	1	2	3	4	5	6	7	8	9
25						25				
6							6			
99										99
145						145				
239										239
20	20									
18									18	

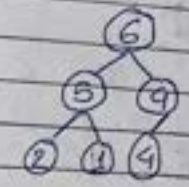
	0	1	2	3	4	5	6	7	8	9
20			20							
25			25							
145					145					
6	06									
18		18								
99										99
239				239						

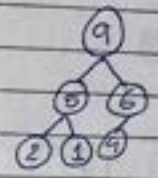
	0	1	2	3	4	5	6	7	8	9
006	006									
018	018									
020	020									
025	025									
239			239							
145		145								
099	099									

Sorted:-
6, 18, 20, 25, 99, 145, 239

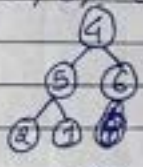
5) Sort the following elements using heap sort.

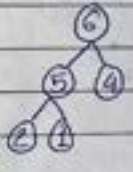
i) 6, 5, 9, 2, 1, 4





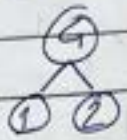
9, 5, 6, 2, 1, 4
4, 5, 6, 2, 1, 9





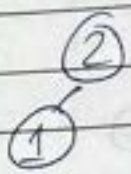
6, 5, 4, 2, 1, 9
1, 5, 4, 2, 6, 9

1, 2, 4, 5, 6, 9



4, 1, 2, 5, 6, 9

2, 1, 4, 5, 6, 9



2, 1, 4, 5, 6, 9

1, 2, 4, 5, 6, 9

EXPERIMENT - 05

```
#include <stdio.h>
#include <stdlib.h>
void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_node();
void count_nodes();
void search();
struct node {
 int info;
 struct node *next;
};
struct node *start = NULL;
int main() {
int choice;
while (1) {
printf("\n Menu \n");
printf("1. Create first node \n");
printf("2. Display \n");
printf("3. Insert at the beginning \n");
printf("4. Insert at the end \n");
printf("5. Insert at the position \n");
printf("6. Exit \n");
printf("7. Delete node by value");
printf("8. Count nodes");
printf("9. Search for a value");
printf("Enter your choice");
scanf("%d", &choice);
```

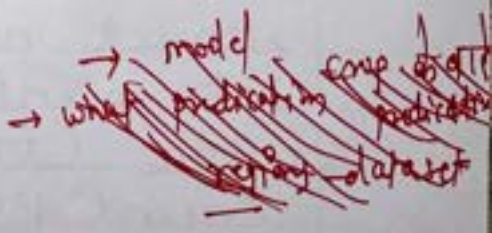
```
switch (choice){
case 1 : create();
         break;
case 2 : display();
         break;
case 3 : insert_begin();
         break;
case 4 : insert_end();
         break;
case 5 : insert_pos();
         break;
case 6 : exit(0);
case 7 : delete_node();
         break;
case 8 : count_nodes();
         break;
case 9 : search();
         break;
default : printf("Invalid Choice \n");
  }
}
return 0;
}
void create(){
  struct node* temp *ptr;
  temp = (struct node*) malloc (sizeof(struct node));
  if (temp == NULL){
  printf("Memory allocation failed \n");
   return;
  }
  printf("\n Enter data:");
  scanf("%d", &temp -> info);
  temp -> next = NULL;
  if (start == NULL){
    start = temp;
  }
  else {
    ptr = start;
    while (ptr -> next != NULL){
      ptr = ptr -> next;
    }
    ptr -> next = temp;
    }
}
```

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

```
printf("Enter data:");
scanf("%d", &temp->info);
temp->next = NULL;
if(Start == NULL){
    Start = temp;
}else{
    ptr = Start;
    while(ptr->next != NULL){
        ptr = ptr->next;
    }
    ptr->next = temp;
}
void insert_pos(){
    struct node *temp, *ptr;
    int pos, i;
printf("Enter position to insert at (starting from 1):");
scanf("%d", &pos);
if(pos < 1){
    printf("Invalid position!\n");
    return 0;
}
temp = (struct node *)malloc(sizeof(struct node));
if(temp == NULL){
printf("Memory allocatiocation failed");
return;
}
printf("Enter data:");
scanf("%d", &temp->info);
if(pos == 1){
    temp->next = Start;
    Start = temp;
    return;
}
ptr = Start;
for(i = 1; i < pos-1 && ptr != NULL; i++){
    ptr = ptr->next;
}
if(ptr == NULL){
printf("Position not found\n");
free(temp);
return;
}
temp->next = ptr->next;
ptr->next = temp;
}
```

```
void delete_node() {
int key;
struct node *temp = Start, *prev = NULL;
if (Start == NULL) {
printf ("List is empty \n");
return ; }
if (Start != NULL){
printf ("List is empty \n");
printf ("Enter the value to delete: ");
scanf ("%d", &key);
if (temp != NULL && temp->info == key) {
  Start = temp -> next;
printf ("%d deleted from the list", key);
return;
}
while (temp != NULL && temp -> info != key)
{ prev = temp;
   temp = temp -> next;
}
if (temp == NULL){
printf ("Value %d not found in the list", key);
return;
prev -> next = temp -> next;
free (temp);
printf ("%d deleted from the list.", key);
}
void count_nodes() {
 struct node* temp = Start;
 int count = 0;
while (temp != NULL) {
 count++;
 temp = temp -> next;
}
```

```
printf("Number of nodes in the list : %d\n",
                                    count);
}
void search(){
  struct node * temp = Start;
  int key, pos = 1, found = 0;
  if (Start == NULL){
  printf("List is empty\n");
  return;
  }
  printf("Enter the value to search : ");
  scanf("%d", &key);
  while (temp != NULL){
    if (temp -> info == key){
  printf("Value %d found at position
          %d\n", key, pos);
  found = 1;
  break;
  }
  temp = temp -> next;
  pos++;
  }
  if (!found){
  printf("Value %d not found in the list\n",
          key);
  }
}
```



EXPERIMENT - 06

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

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

struct node* head = NULL;
struct node* tail = NULL;

void create();
void display_forward();
void display_backward();
void insert_begin();
void insert_end();
void insert_pos();
void delete_node();
void count_nodes();
void search();
```

```
int main() {
    int choice;
    while (1) {
        printf("\nMenu\n");
        printf("1. Create first node\n");
        printf("2. Display forward\n");
        printf("3. Display backward\n");
        printf("4. Insert at the beginning\n");
        printf("5. Insert at the end\n");
        printf("6. Insert at position\n");
        printf("7. Delete node by value\n");
        printf("8. Count nodes\n");
        printf("9. Search for a value\n");
        printf("10. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: create(); break;
            case 2: display_forward(); break;
```

```
            case 3: display_backward(); break;
            case 4: insert_begin(); break;
            case 5: insert_end(); break;
            case 6: insert_pos(); break;
            case 7: delete_node(); break;
            case 8: count_nodes(); break;
            case 9: search(); break;
            case 10: exit(0);
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}

// Create (insert at end)
void create() {
    struct node* temp = (struct
node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("Memory allocation failed\n");
        return;
```

```
    }
    printf("\nEnter data: ");
    scanf("%d", &temp->info);
    temp->next = NULL;
    temp->prev = tail;

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

void display_forward() {
    struct node* ptr = head;
    if (head == NULL) {
        printf("\nList is empty\n");
        return;
    }
    printf("\nList (forward):\n");
```

```
    while (ptr != NULL) {
        printf("%d <-> ", ptr->info);
        ptr = ptr->next;
    }
    printf("NULL\n");
}

void display_backward() {
    struct node* ptr = tail;
    if (tail == NULL) {
        printf("\nList is empty\n");
        return;
    }
    printf("\nList (backward):\n");
    while (ptr != NULL) {
        printf("%d <-> ", ptr->info);
        ptr = ptr->prev;
    }
    printf("NULL\n");
}
```

```
void insert_begin() {
    struct node* temp = (struct
node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("\nEnter data: ");
    scanf("%d", &temp->info);
    temp->prev = NULL;
    temp->next = head;

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

void insert_end() {
```

```
    struct node* temp = (struct
node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("\nEnter data: ");
    scanf("%d", &temp->info);
    temp->next = NULL;
    temp->prev = tail;

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

void insert_pos() {
    int pos, i;
```

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

    struct node* temp = (struct
node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter data: ");
    scanf("%d", &temp->info);

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

```
    struct node* ptr = head;
    for (i = 1; i < pos - 1 && ptr != NULL; i++)
{
        ptr = ptr->next;
    }

    if (ptr == NULL) {
        printf("Invalid position\n");
        free(temp);
        return;
    }

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

    if (ptr->next != NULL)
        ptr->next->prev = temp;
    else
        tail = temp; // inserted at end
```

```
    ptr->next = temp;
}

void delete_node() {
    int key;
    printf("Enter value to delete: ");
    scanf("%d", &key);

    struct node* temp = head;
    while (temp != NULL && temp->info !=
key) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Value %d not found\n", key);
        return;
    }

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

```
    else
        head = temp->next;

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

    free(temp);
    printf("%d deleted\n", key);
}

void count_nodes() {
    struct node* temp = head;
    int count = 0;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    printf("Total nodes: %d\n", count);
}
```

```
void search() {
    int key, pos = 1, found = 0;
    printf("Enter value to search: ");
    scanf("%d", &key);

    struct node* temp = head;
    while (temp != NULL) {
        if (temp->info == key) {
            printf("Value %d found at position
%d\n", key, pos);
            found = 1;
            break;
        }
        temp = temp->next;
        pos++;
    }
    if (!found)
        printf("Value %d not found\n", key);
}
```

Experiment - 7

i) Primitive operation on stack

```
#include <stdio.h>
#include <stdlib.h>
#define max 10
void push();
void pop();
void display();

int s[Max];
int top = -1;
int main() {
int choice;
    printf("\n Menu:");
    printf("\n 1. Push");
    printf("\n 2. Pop");
    printf("\n 3. Display");
    printf("\n 4. Exit");
  do {
  printf("\n Enter your choice:");
  scanf("%d", &choice);
  switch(choice) {
     case 1: push();
             break;
     case 2: pop();
             break;
     case 3: display();
             break;
     case 4: exit(0);
  default:
    printf("Invalid Choice!");
} }
```

```
while (choice != 4);
return 0;
}
void push ()
{
  int element;
if (top == max-1)
{
  printf ("Overflow");
}
else
{
 printf ("Enter Value");
 scanf ("%d", &element);
 top = top + 1;
 s[top] = element;
}
}
void pop ()
{
  int item;
if (top == -1)
{
 printf ("Underflow");
}
else {
    item = s[top];
    top = top - 1;
 printf ("The deleted element is %d", item);
}
}
void display ()
{
   if (top == -1)
   {
     printf ("Stack is Empty");
   }
else
{
  printf ("Stack elements are :");
 for (int i = 0; i <= top; i++)
{
  printf ("\t %d", s[i]);
}
}
}
```

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

EXPERIMENT - 08

① Convert infix to postfix expression

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

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

o/p

2 Evaluate postfix expression

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

```
case '-':
    result = a - b;
    break;
case '/' :
    result = a / b;
    break;
case '*':
    result = a * b;
    break;
case '%':
    result = a % b;
    break;
}
push(result);
}
}
pEval = pop();
printf("\n The postfix evaluation is : %d\n",
                                    pEval);
return 0;
}
void push(int n)
{
if (top < SIZE - 1)
{
    stack[++top] = n;
}
else {
 printf("stack is full!\n");
 exit(-1);
}
int pop()
{
int n;
 if (top > -1)
{
    n = stack[top]
    stack[top--] = -1;
    return n;
}
else {
 printf("stack is empty!\n");
  exit(-1);
}
}
```

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

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

+A*-*BC-/D^EFGH

Q Evaluate prefix expression using stack

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

I/P	Op 1	Op 2	Result	Output
2				2
4				2, 4
$	4	^ 2		16
1				16, 1
2				16, 1, 2
4				16, 1, 2, 4
/	4	/ 2		16, 1, 2
2				16, 1, 2, 2
1				16, 1, 2, 2, 1
+	1	+ 2		16, 1, 2, 3
*	3	* 2		16, 1, 6
-	6	- 1		16, 5
+	5	+ 16		21

Algorithm

1) Scan the infix expression from left to right
2) If the scanned character is an operand, output
3) Else,
(a) If the precedance of the scanned operator is greater than the precedance of the operator in the stack.
(b) Else, pop all the operators from the stack which are greater or equal to in the ~~procedure~~ precedance than that of the scanned operator. After doing that push the scanned operator to the stack.
4) If the scanned character is an '(', push it to the stack.
5) If the scanned character is an ')', pop the stack & output it until a '(' is encountered & discard both parenthesis.
6) Repeat steps 2-6 until infix exp is scanned.
7) Print the output
8) Pop & output from the stack until it is not empty

(2) Algorithm

(1) Scan the expression from right to left or reverse the expression.

2) Scan the char one by one.

3) If the char is an operand, copy it to prefix notation output.

~~(3) If the char is an operand, copy it to prefix~~

4) If the char is a closing parenthesis, then push it to the stack.

5) If the char is a opening parenthesis, pop the elements in the stack until we ~~need~~ find the corresponding closing parenthesis.

6) If the char scanned is an operator -

(a) If the operator has precedence lesser than the top of the stack, output it to the prefix notation output & then check the above condition again with the new top of the stack.

(b) If the operator has precedence greater than or equal to the top of the stack, push operator to the stack.

(7) After all the char are scanned, reverse the prefix notation output.

Algorithm

i) Initialize empty stack S
ii) Scan postfix expression from left to right.
iii) If operand → push into S.
iv) If operator → pop two operands from S, apply operation, push result back to S
v) After scanning → Top of the stack is result.

(4) Algorithm

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

Nandu 19/9

Experiment - 09

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

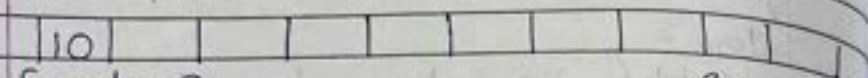
```
void enqueue(){
int element;
if(rear == SIZE-1){
printf("\n Queue is full....");
}
else{
printf("\n Enter element to insert:");
scanf("%d", &element);
 if(front == -1)
  front = 0;
  rear++;
  queue[rear] = element;
}
}

void dequeue(){
 if(front == -1 || front > rear){
   printf("\n Queue is empty...");
}
else{
printf("\n Deleted element: %d",
                        queue[front]);
 front++;
}
}

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

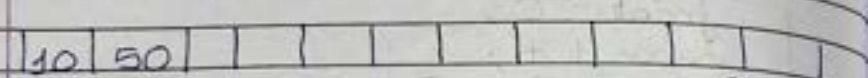
Q Perform following operation on linear queue in array size (10).

(i) insert (10)



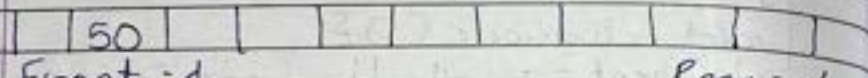
Front = 0 Rear = 0

(ii) insert (50)



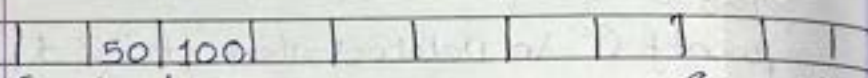
Rear = 1

(iii) Delete



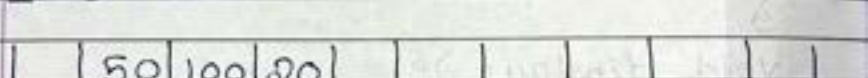
Front = 1 Rear = 1

(iv) Insert 100



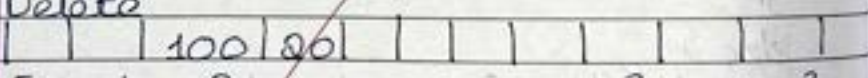
Front = 1 Rear = 2

(v) Insert (20)



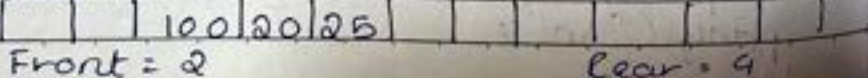
Front = 1 Rear = 3

(vi) Delete



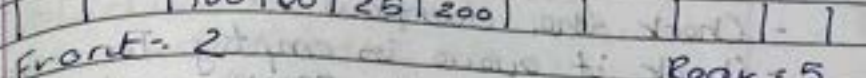
Front = 2 Rear = 3

(vii) Insert (25)



Front = 2 Rear = 4

(viii) Insert (200)



Front = 2 Rear = 5

Q Explain concept of priority queue in detail?

A priority queue is a type of data structure similar to a regular queue but each element is assigned a priority. Instead of following the FIFO rule. The element with highest priority is served first, if two elements have the same priority, the one that entered first is usually served first.

Q Applications of Queue

- Serving requests on a single share resource, like printer.
- In real life scenario, call centre phone system uses queues to hold people calling them in an order, until a service representative is free.
- FCFS job scheduling.
- Online banking fund transfer requests

Q Algorithm for insertion in linear queue

- Check the overflow.
- Check if queue is empty. In case it is, then both FRONT and REAR are set to zero. So that new value can be stored at the 0th location.
- Otherwise, if queue already has some values, then REAR is incremented such that it points to the next location in array.
- The value is stored in the queue at the location pointed by REAR.
- Exit

Deletion

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

EXPERIMENT - 10

```
#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("\n 1. Enqueue \n 2. Dequeue
        \n 3. Display \n 4. Exit \n");
Do{
  printf("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("Invalid Choice");
  }
}
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");
scanf ("%d", &value);
if (FRONT = = -1)
  Front = 0;
rear = [rear + 1] / size;
queue [rear] = value;
}
void dequeue () {
 if (Front = = -1)
{
printf ("Queue is Empty");
}
else {
printf ("\n Deleted %d from the queue",
          queue [front]);
if (front = = rear)
{
  Front = rear - 1;
}
else {
  Front = (front + 1) / size;
}
}
void display () {
if (Front = = -1)
{
printf ("\n Queue is empty!");
}
else {
int i = Front;
printf ("\n Queue elements are:");
while (i) {
printf ("%d", queue [i]);
 if (i = = rear)
    break;
 i = (i + 1) / SIZE;
}
printf ("\n");
}
}
```

Q

Insertion Algorithm

- If FRONT = 0 and rear = max - 1 the queue will overflow
- If FRONT is rear + 1 it will also overflow queue
- If FRONT = -1 and rear = -1, FRONT = REAR = 0
- Else if rear = max - 1 and FRONT != 0
  Rear = 0
- Else, rear = rear + 1
- End
- queue [rear] = value
- Exit

```
void enqueue()
{
  int value;
  if ((FRONT = = 0 && REAR = = SIZE - 1) ||
      (rear + 1 = = FRONT))
  {
    printf ("\n Queue is Full
```

* Deletion

- If Front = -1 write underflow.
- Go to step 4 at end of if.
- Set val = queue [front]
- If Front = rear, set Front = rear = -1
- Else, if Front = max - 1, set Front = 0
- Else, set Front = Front + 1.
- Exit

Q) Application of priority queue.

- Traffic Light control.
  Manages signal priorities based on real-time traffic sensor data.
- Operating system algorithms.
  Used for scheduling process to execute high priority first.
- Huffman codes -
  Priority queue helps build Huffman trees for efficient data compression.

EXPERIMENT - 11

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int item;
  struct node* left;
  struct node* right;
void inOrder (struct node * root){
  if (root = = NULL){
    return;
inOrder (root → left);
printf ("%d", root → item);
inOrder (root → right);
void preorder (struct node * root ){
  if (root = = NULL){
    return; }
printf ("%d", root → item);
preorder (root → left);
preorder (root → right);
}
void postorder (struct node * root ){
  if (root = = NULL){
    return; }
postOrder (root → left);
postOrder (root → right);
printf ("%d", root → item);
}
struct node* createNode (int item){
struct node* newnode = (struct node*)
malloc (sizeof (struct node));
newnode → item = item;
```

```
newnode -> left = Null;
newNode -> data = value;
newNode -> left = newNode -> right = NULL;
return newNode; }
struct node * insert (struct Node * root,
int = value);
if (root == NULL) [return 0 create node (value)
if (value < root -> data)
root -> left = insert (root -> left, value);
else if (value > root -> data);
root -> right = insert (root -> right, value);
return root;
struct Node * delete Node (Struct node*
                    root, int key ) {
if (root == NULL) {
  return root; }
if (key < root -> data )
  root -> right . delete Node (root -> right, key);
else
  if (root -> left == NULL) {
struct node * temp . root -> right;
  free (root);
  return temp;
else if (root -> right NULL) {
struct node* temp = root -> right;
free (root);
return temp; }
int main () {
root = insert (root, 50),
insert (root, 30);
insert (root, 70);
insert (root, 60);
root = delete Node (root, 30);
printf ("After deleting:");
return 0; }
new node -> right = NULL;
return newnode; }
struct node * insert left (struct node*
root int item) {
  root -> left = create Node (item);
  return root -> left;
struct node * insert Right (struct node
* root int item) {
struct node * newnode = create Note
                    (item);
return root -> right;
int main () {
struct node * root = create Node (1);
insert left (root, 2);
insert Right (root, 3);
insert left (root -> left, 4);
printf ("Inorder Traversal");
inorder (root);
printf ("\n Post Order traversal:");
postorder (root);
}
```

EXPERIMENT - 12

Q)

```
#include <stdio.h>
#define V 5
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 insertedge (int arr[v][v], int i, int j)
{
   arr[i][j] = 1;
   arr[j][i] = 1;
}
void printAdjMatrix (int arr[v][v])
{
  int i, j;
 for (i=0; i<v; i++)
 {
 for (j=0 ; j<v; j++)
 {
 printf ("%d", arr[i][j]);
 }
 printf ("\n");
 }
}
```

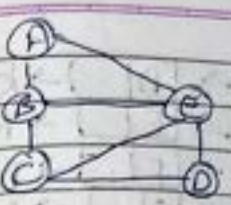
```
int main()
{
int adjMatrix[v][v];
init (adjMatrix)
insertEdge (adjMatrix, 0, 1);
insertEdge (adjMatrix, 0, 2);
insertEdge (adjMatrix, 1, 2);
insertEdge (adjMatrix, 2, 0);
insertEdge (adjMatrix, 2, 3);
insertEdge (adjMatrix, 0, 3);
insertEdge (adjMatrix, 0, 0);
insertEdge (adjMatrix, 1, 3);
insertEdge (adjMatrix, 4, 3);
insertEdge (adjMatrix, 2, 4);
printAdjMatrix (adjMatrix);
return 0;
}
```

Output :-

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

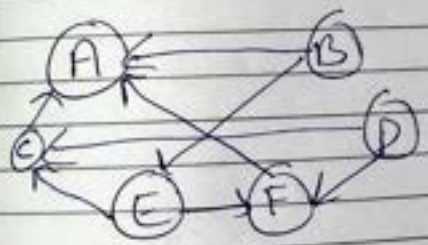
② Basic Terminologies

i) Vertex (Node) - Point in a graph
ii) Edge - Line connecting 2 vertices
iii) Degree - No of edges connected to a vertex
iv) Path - Sequence of connected vertices
v) Cycle: Path that starts & ends at the same vertex.
vi) Loop: Edge from a vertex to itself
vii) Connected Graph: Every vertex is reachable
viii) Disconnected Graph:- Some vertices aren't connected
ix) Directed Graph:- Edges have direction
x) Undirected Graph:- Edges have no direction
xi) Weighted Graph:- Edges have values
xii) Simple Graph:- No loops or multiple edges
xiii) Complete Graph:- Every vertex connects to each other.
xiv) Subgraph - Part of larger graph
xv) Tree - Connected graph with no cycles.
xvi) Spanning Tree - Including all vertices
xvii) Adjacency:- 2 vertices connected by an edge
xviii) Adjacy matrix - Table showing connections



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

Adjacency List



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


A
B
C
D
E
F
Null
A
D
E
Null
A
E
Null
C
F
Null
C
F
Null
A
Null