EXPERIMENT - 1

1) PROBLEM - 1

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

CODE:

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

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

Output-

enter element of array : 5
enter element of array : 4
enter element of array : 5
enter element of array : 3
enter element of array : 4
enter element of array : 52

enter element of array: 5
enter element of array : 6
enter element of array: : 2
enter element of array: 1
average is 8

2) PROBLEM - 2

Problem statement - Display pattern

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

CODE:

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

Output -

*

* *

* * *

PROBLEM-3

Problem statement- Find first repeating element from array.

Code:

```
#include <stdio.h>
int main()
{
  int arr[10], flag=0;
  for(int i=0; i<10; i++)
  {
    printf("enter element of array:");
    scanf("%d", &arr[i]);
  }
  for(int i=0; i<10; i++)
  {
    for(int j=0; j<10; j++)
    {
      if(j!=i)
      {
        if(arr[j]==arr[i])
        {
          printf("%d is a repeating number\n", arr[i]);
        }
      }
    }
  }
  return 0;
}
```

Output :

*

* *

* * *

3)

PROBLEM-3

Problem statement - Find first repeating element from array.

Code:

```
#include<stdio.h>
int main()
{
  int arr[10], flag=0;
  for(int i=0; i<10; i++)
  {
    printf("enter element of array:");
    scanf("%d", &arr[i]);
  }
  for(int i=0; i<10; i++)
  {
    for(int j=0; j<10; j++)
    {
      if(j!=i)
      {
        if(arr[j]==arr[i])
        {
          printf("%d is a repeating number\n", arr[i]);
        }
      }
    }
  }
  return 0;
}
```

OUTPUT-

enter element of array : 1
enter element of array : 2
enter element of array : 2
enter element of array : 3
enter element of array : 4
enter element of array : 5
enter element of array : 6
enter element of array : 7
enter element of array : 8
enter element of array : 9
2 is a repeating number

4) PROBLEM-4

Problem Statement - Find greatest and smallest element of an array.

Code:

```
#include<stdio.h>
int main()
{
  int i;
  int arr[5];
  printf("Enter elements of array");
  for(i=0; i<5; i++)
  {
    scanf("%d", &arr[i]);
  }
  int max=arr[0];
  int min = arr[0];
  for(i=1; i<5; i++)
  {
   if(arr[i] > max)
   {
    max=arr[i];
   }
   if(arr[i]<min)
   {
    min = arr[i];
   }
}
printf("Greatest element is %d\n", max);
```

```
    printf("smallest element = %d in", min);
    return 0;
}
```

OUTPUT-

Enter elements -
1
3
2
4
5
greatest element = 5
smallest element = 1

PROBLEM-5

Problem statement - Square odd elements of an array

code:

```
#include<stdio.h>
int main(){
  int arr[5];
  printf("Enter 5 numbers");
  for (int i=0; i<5; i++)
  {
    scanf("%d", &arr[i]);
  }
  for( int i=0; i<5; i++)
  {
    if((i+1)%2!=0)
    {
      arr[i] = arr[i] * arr[i];
    }
  }
  printf("Array after squaring elements at odd position");
  for(int i=0; i<5; i++)
  {
    printf("%d", arr[i]);
  }
  printf("\n");
  return 0;
}
```

OUTPUT:

Enter 5 numbers:

1
3
4
5

Array after squaring odd positions:

1 3 0 4 25

EXTRA

Q Patterns

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

2)
*
# #
$ $ $
? ? ? ?

3)
$ $
$ $
$

4)
1
1 2
1 2 3

5)
*
* *
* * *

```
#include <stdio.h>
int main(){
  printf("Enter number of rows: ");
  int n;
  scanf("%d", &n);
  printf("Pattern 1: \n");
  for(int i=0; i<=n; i++){
    for(int j=1; j<=i; j++){
      printf("%d", i);
    }
    printf("\n");
  }
  printf("Pattern 2: \n");
  char arr = {'*', '#', '$', '@'};
  for(int i=1; i<5; i++){
    for(int j=1; j<=i; j++){
      printf("%c", arr[i-1]);
    }
    printf("\n");
  }
  printf("Pattern 3: \n");
  for(int i=n; i>=1; i--){
    for(int s=0; s<(n-i); s++){
      printf(" ");
    }
    for(j=1; j<=(2*i-1); j++){printf("*");}
    printf("\n");
  }
```

```
printf("Pattern 4 :\n");
for(int i=1;i<=n;i++)
{
  for(int j=1;j<=i;j++)
  {
    if(i%2!=0)
    {
      printf("%d",i);
    }
    else
    {
      printf("%d",i);
    }
    printf("\n");
}
printf("Pattern 5 :\n");
for(int i=1;i<=n;i++)
{
  for(int j=1;j<=n-i;j++)
  {
    printf(" ");
  }
  for(int k=1;k<=2*i-1;k++)
  {
    printf("*");
  }
  printf("\n");
}
return 0;
}
```

OUTPUT -

Enter no. of rows: 5

Pattern 1:

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

Pattern 2:

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

Pattern 3:

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

Pattern 4:

```
1
1 2
1 2 3
```

Pattern 5:

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

EXPERIMENT - 2

Q1 Search data using linear search. Consider the following list to perform linear search
56, 36, 89, 57, 1, 0, 67, 59
(i) Search item '1' from the above list and write wether item is found or nor.
(ii) Search item '55' from the above list and write wether the item is found or nor

A1
```
#include<stdio.h>
int main()
{
  int arr[] = {56, 36, 89, 57, 1, 0, 67, 59};
  int n = sizeof(arr);
  int k, f = 0;
  printf("Enter the element to search: ");
  scanf("%d", &k);
  for(int i = 0; i<n; i++)
  {
    if(arr[i] == k){
      printf("Element found at index: %d\n", i);
      f = 1;
      break;
    }
  }
  if(f == 0)
  {
    printf("Element not found in the array \n");
  }
```

```
    return 0;
}
```

O/P: 1:

Enter the element to search: 1
Element found at index : 4

O/P: 2

Enter the element to search : 55
Element not found in the array

Q2 Search data using binary search

```
#include<stdio.h>
int main()
{
  int a[100], n, m, l, h, i, found=0, k;
  printf("Enter no. of array elements : ");
  scanf("%d", &n);
  printf("Enter key to be searched. ");
  scanf("%d", &k);
  printf("\n Enter sorted array elements : \n ");
  for(i=0; i<n; i++)
  {
    scanf("%d", &a[i]);
  }
  l=0; h=(n-1);
  while(h>=l)
  {
```

```
m(int)(l+h)/2);
if(k==a[m])
{
printf("\n element found at index : %d",
break;
}
else
{
  if(k>a[m])
  {
    l=m+1;
  }
  if(k<a[m])
  {
    h=m-1;
  }
}
if(l>h)
{
printf("element not found");
return 0;
}
```

O/P 1:

```
Enter no. of elements : 4
Enter key to be searched : 3
Enter sorted array elements :
2
```

3
4
Element not found.

O/P 2:

Enter no. of array elements : 5
Enter key to be searched : 5
Enter 5 elements :
1
2
3
5
4
Elements found at index : 3

Q3 Compare linear search vs Binary search

Linear search	Binary search
1) Unsorted or sorted (Data)	must be sorted (Data)
2) Sequential scanning	Divide and conquer
3) O(n) [Time complexity]	O(log n) [Time complexity
4) Less efficient	more efficient
5) simpler code	Complex code

Q84 State limitations of linear search in terms of time complexity

Ans -> 1) Insufficient for large data sets
- Linear search has a time complexity of O(n) where n is no. of elements in the list

2) NO improvement for sorted data
- Linear search does not take advantage of sorted data

3) Average and worst case are the same order
- The average case and worst case both require scanning of about half or all the elements respectively, so it doesn't perform or average

4) Not suitable for time-critical applications:
- Due to its linear time, it is not ideal for applications where fast ~~scanning~~ searching is necessary, especially with large data sets

EXTRA QUESTIONS

1) Write a program to copy the elements of one array in another array in reverse order

Code:

```
#include <stdio.h>
int main()
{
  int n;
  printf("Enter size of array : ");
  scanf("%d", &n);
  int arr1[n], arr2[n];
  for(int i=0 ; i<n ; i++)
  {
    printf("Enter element %d : ", i+1);
    scanf("%d", &arr1[i]);
  }
  for(int i=0 ; i<n; i++)
  {
    arr2[i] = arr1[n-1-i]
  }
  printf("Reversed array : \n");
  for(int i=0 ; i<n ; i++)
  {
    printf("%d \n", arr2[i]);
  }
  return 0;
}
```

OUTPUT:

Enter size of array: 5
enter element 1: 1
enter element 2: 2
enter element 3: 3
enter element 4: 4
enter element 5: 5
Reversed array:
5
4
3
2
1

2 Write a program to count total no. of repeating elements in array

Code:

```
#include<stdio.h>
int main()
{
  int n;
  printf("enter size of array: ");
  scanf("%d", &n);
  int arr[n];
  int i, j, k = 1;
  for(int i=0; i<n; i++)
  {
```

```
    printf("enter element : ");
    scanf("%d", &arr[i]);
  }
  for(i=0; i<n; i++)
  {
    for(j=i+1; j<n; j++)
    {
      if (arr[i] == arr[j])
      {
        k++;
      }
    }
  }
  printf("no. of repeating of elements are %d", k);
  return 0;
}
```

OUTPUT-

Enter no. elements in array : 5
enter element : 1
enter element : 2
Enter element : 2
enter element : 3
enter element : 4
no. of elements repeating elements are : 1

3 Write a program in C to print all unique elements in an array

```
#include<stdio.h>
int main()
{
  int n;
  printf("enter size of array : ");
  scanf("%d", &n);
  int arr[n];
  int i, j, u;
  for(i=0; i<n; i++)
  {
    printf("enter element %d : ", i+1);
    scanf("%d", &arr[i]);
  }
  printf("Unique elements are : \n");
  for(i=0; i<n; i++)
  {
    u=1;
    for(j=0; j<n; j++)
    {
      if(i!=j && arr[i]==arr[j])
      {
        u=0;
        break;
      }
    }
    if(u)
    {
      printf("%d\n", arr[i]);
```

```
    }
}
return 0;
```

OUTPUT

```
nter size of array : 5
nter element: 1
nter element : 2
nter element : 2
nter element: 3
nter element: 4
Unique elements are
1
3
```

Write a program to seperate odd and even integers in seperate arrays

Code:

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

 int n;
 print("enter the size of the array: ");
 scanf("%d", &n);
 int arr[n], even[n], odd[n];
 int i, even count = 0, odd coont = 0;
```

```
for(i=0;i<n;i++)
{
  printf("enter element %d : ", i+1);
  scanf("%d", &arr[i]);
  if(arr[i]%2 == 0){
   even[evencount++]=arr[i];
  }
  else
  {
   odd[oddcount++]=arr[i];
  }
 }
 printf("\n even element : \n");
 for(i=0;i<evencount;i++)
 {
  printf("%d", even[i]);
 }
 return 0;
}
```

OUTPUT -

```
#include <stdio.h>
int main(){
   int n;
   printf("Enter size of array : ");
   scanf("%d", &n);
   int arr[n], even[n], odd[n];
```

OUTPUT -

```
enter size of array : 5
enter element : 1
enter element : 2
enter element : 3
enter element : 4
enter element : 5
even element :
2 4
odd elements
1 3 5
```

5 WAP to find second smallest element in an array

Code:

```
#include<stdio.h>
#include<limits.h>
int main()
{
  in arr[] = {5, 1, 9, 6, 3};
  int n = sizeof(arr)/sizeof(arr[0]);
  int i, first = int_max, second = int_max;
  for(i=0; i<n; i++)
  {
    if(arr[i]<first)
    {
      second = first;
      first = arr[i];
    }
```

```
else if (arr[i] < second && arr[i] != first
{
    second = arr[i];
}
if (second == int_max
{
  printf("No smallest element:\n");
}
  else
  {
    printf("second smallest element is %d\n");
  }
return 0;
```

O/P:

second smallest element is 3:

6 Write a program to count the total no. of words in a string

Code:

```
#include<stdio.h>
int main()
{
  char str[1000];
  int i=0, wordcount=0;
  int inword=0;
  printf("enter a string:\n");
  fgets(str, sizeof(str), stdin)
  while (str[i]!= '\0')
```

```
{
  if(str[i] == ' ' || str[i] == '\t' || str[i] == '\n')
  {
     inword = 0;
  }
  else if (inword == 0)
  {
     inword = 1;
     wordcount++;
  }
  i++;
}
printf("\n Total no. of words = %d\n", wordcount);
return 0;
}
```

O/P

enter a string:

Hi Hello

Total no. of words: 2

EXPERIMENT - 3

1) Problem statement - Sort elements in ascending order using bubble sort.

Code:

```
#include<stdio.h>
int main()
{
   int n, i, j, temp;
   printf("enter no. of elements in array: ");
   scanf("%d", &n);
   int arr[n];
   for(i=0; i<n; i++)
   {
      printf("enter element of array: ");
      scanf("%d", &arr[i]);
   }
   for(i=0; i<n-1; i++)
   {
      for(j=0; j<n-i-1; j++)
      {
         if(arr[j]>arr[j+1])
         {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
      }
   }
```

2)

```
For(i=0 ; i<n; i++)
{
   prinrf("array is: ");
   prinrf("%d\n", arr[i]);
}
return 0;
```

UTPUT-

nter no. of elements : 5

nter 5 elements:

4

3

3

5

6

ascending

rray in sorted ~~descending~~ order :

~~4 6 5 4 3~~ 3 3 4 5 6

Problem statement: Sort elements in descending order using selection sort

Code:

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

   int n, i, j, temp, max_idx, temp;
```

```
printf(" enter no. of elements : ");
scanf("%d", &n);
int arr[n];
printf(" Enter %d elements : \n", n);
for(i=0; i<n; i++)
{
  scanf("%d", &arr[i]);
}
for(i=0; i<n-1; i++)
{
  max_idx = i;
  for(j=i+1; j<n; j++)
  {
    if(arr[j] > arr[max_idx])
    {
      max_idx = j;
    }
  }
  temp = arr[i];
  arr[i] = arr[max_idx];
  arr[max_idx] = temp;
}
printf(" array in sorted decending order : \n");
for(i=0; i<n; i++)
{
  print("%d", arr[i]);
}
return 0;
}
```

3)

Pass 1

Pass 2

Output:
Enter no. of elements: 5
enter 5 elements:
5
6
.
2
3
Array in sorted descending order:
6 5 3 2 1

Find the no. of comparisons required in bubble sort method of the following list having 5 number 100, 200, 300, 400, 500

The given list - 100, 200, 300, 400, 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 100 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

Pass 3 100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

Pass 4 100 200 300 400 500

100 200 300 400 500

The sorted list is -

100, 200, 300, 400, 500

No. of comparisons = 10

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

Given list - 500, -20, 30, 14, 50

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

-20 500 30 14 50

-20 30 500 14 50

-20 14 500 30 50

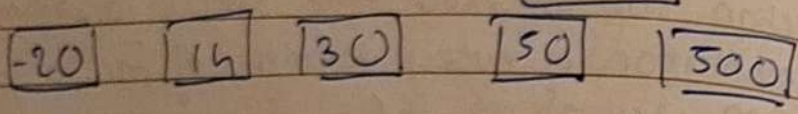
[-20] [14] 500 30 50

-20 14 500 30 50

-20 14 30 500 50

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

-20 14 30 500 50



∴ The sorted list is -
-20, 14, 30, 50, 500

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

EXPERIMENT - 4

Sort element in ascending order using insertion sort

Code:

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

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

OUTPUT -

```
Enter number of array elements: 9
8
7
6
5
After pass 1 : 8 9 7 6 5
After pass 2 : 7 8 9 6 5
After pass 3 : 6 7 8 9 5
After pass 4 : 5 6 7 8 9
Sorted array
5
6
7
8
9
```

Sort elements in ascending order using radix sort

```
#include <stdio.h>
int main()
{
  int a[50], output[50], count[10];
int n, j, i, max = 0, place = 1;
printf("enter no. of elements: ");
scanf("%d", &n);
printf("enter %d numbers \n", n);
for(i=0; i<n; i++) {
  scanf("%d", &a[i]);
  if(a[i] > max)
    max = a[i]
}
while (max / place > 0) {
 for(i=0; i<10; i++) {
   count[i] = 0;
  }
  for(i=0; i<n; i++){
    count[(a[i] / place) % 10]++;
  }
  for(i=1; i<10; i++){
      count[i] += count[i-1];
  }
  for(i=n-1; i>=0; i--){
    int digit = (a[i] / place) % 10
    output[--count[digit]] = a[i];
  }
   for(i=0; i<n; i++){
   printf("%d", a[i]);
  return 0;
}
```

Q3 What is the output of the insertion sort after the second iteration of the following no.s

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

1st iteration:-

7 3 1 9 4 8 6

↙

3 7 1 9 4 8 6

2nd iteration :-

3 7 1 9 4 8 6

↙

1 3 7 9 4 8 6

After the second iteration, result is

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

Q4 Sort the following no.s using radix sort

i) 100, 225, 890, 4130, 956, 99, 543

0 1 2 3 4 5 6 7 8 9

0100 100
0225 225
0390 390
4130 4130
0456 0456
0099 0099
5431 5431

0 1 2 3 4 5 6 7 8 9

0100 100 390
0390
4130 4130
5431 5431
0225 225
0456 936
0099 99

0 1 2 3 4 5 6 7 8 9

0100 100
0225 225
4130 4130
5431 5431
0390 0390
0456 0456
0090 0090

d)

0	1	2	3	4	5	6	7	8	9
0090	90								
0100	100								
4130				4130					
0225	225								
0390	390								
5431							5431		
0956	956								

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

Hence, sorted

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

0	1	2	3	4	5	6	7	8	9
256						256			
99									99
145					145				
239									239
20	20								
18								18	

0	1	2	3	4	5	6	7	8	9
20		20							
145					145				
256						256			
18	18								
99									99
239									

239

0 1 2 3 4 5 6 → 8 9

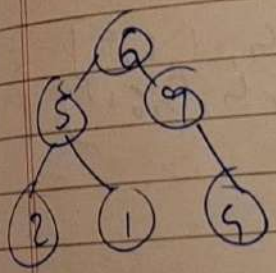
18
20 20
99 145
145
256
256
99 99

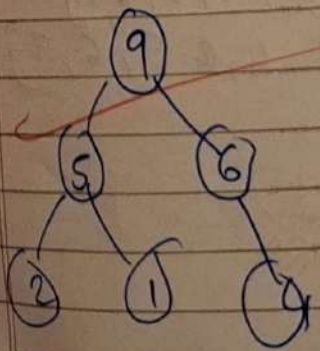
18, 20, 99, 145, 256,

Hence sorted

5) Solve the Following elements using heap sort

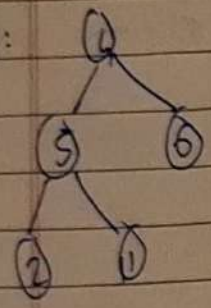
(i) 6, 5, 9, 2, 1, 4

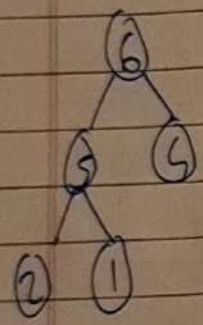




arr: [9, 5, 6, 2, 1, 4]

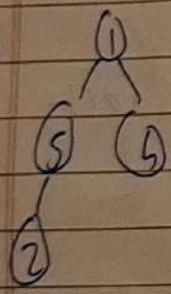
Pass 2:

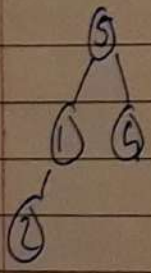




arr: [6, 5, 4, 2, 1, 9]
arr: [1, 5, 4, 2, 6, 9]

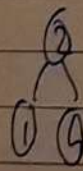
Pass 3:



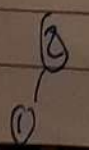


arr: [5, 2, 4, 1, 6, 9]
arr: [1, 2, 4, 5, 6, 9]

Pass: 4

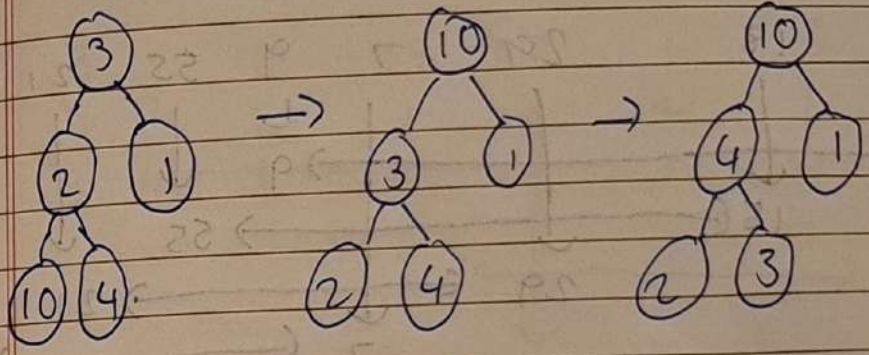


arr: [4, 2, 1, 5, 6, 9]
arr: [1, 2, 4, 5, 6, 9]



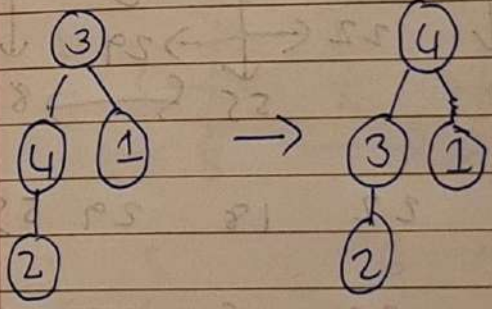
Hence sorted

2) arr: [3, 2, 1, 10, 4]



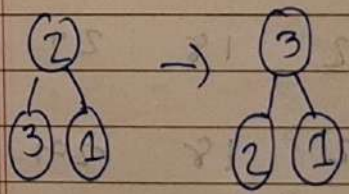
arr = [10, 4, 1, 2, 3]
arr = [3, 4, 1, 2, 10]

Pass: 2



arr = [4, 3, 1, 2, 10]
arr = [2, 3, 1, 4, 10]

Pass: 3

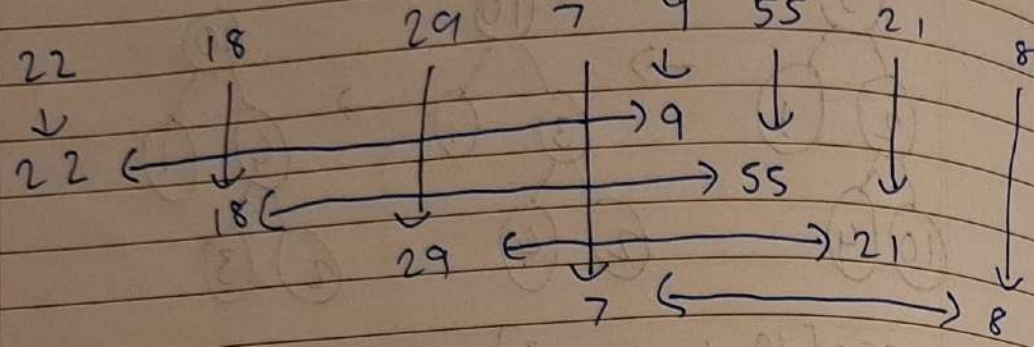


arr = [3, 2, 1, 4, 10]
arr = [1, 2, 3, 4, 10]

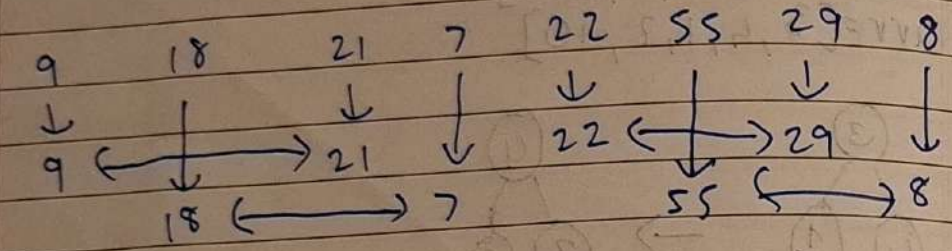
PAGE No.
DATE / /

Qs Sort following elements using shell sort

Pass:1 22 18 29 7 9 55 21 8



Pass:2 9 18 21 7 22 55 29 8



Pass:3 9 7 21 8 22 18 29 55

7 9 21 8 22 18 29 55

7 9 21 8 22 18 29 55

7 8 9 21 22 18 29 55

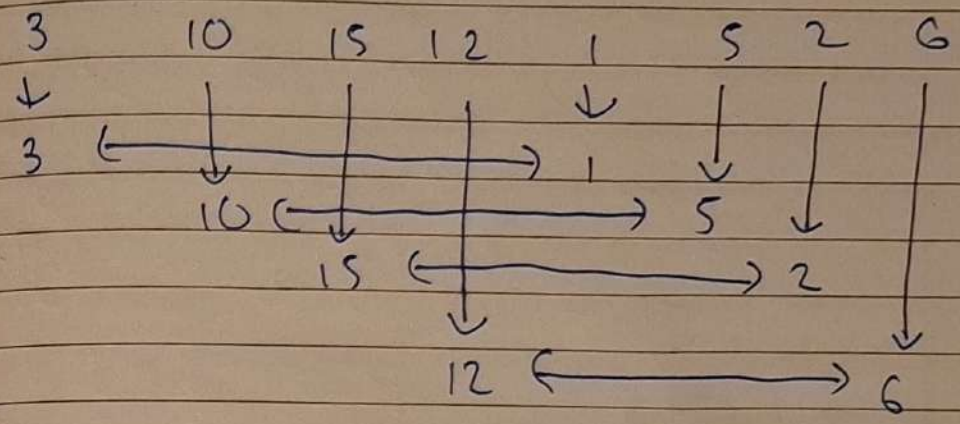
7 8 9 21 22 18 29 55

7 8 9 18 21 22 29 55

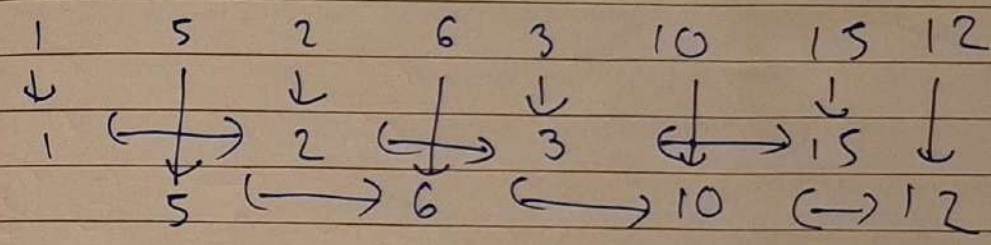
Sorted arr = [7, 8, 9, 18, 21, 22, 29, 55]

2) arr = [3, 10, 15, 12, 1, 5, 2, 6]

Pass: 1



Pass: 2



Pass: 3

1 5 2 6 3 10 15 12

1 5 2 6 3 10 15 12

1 2 5 6 3 10 15 12

1 2 5 6 3 10 15 12

1 2 3 5 6 10 15 12

1 2 3 5 6 10 15 12

1 2 3 5 6 10 15 12

1 2 3 5 6 10 12 15

Final array: 1, 2, 3, 5, 6, 10, 12, 15

EXPERIMENT-5

(1) Write a menu driven program that implements Singly linked list for the following operations - Create, display, insert begin, insert end, insert end, insert mid, delete begin, delete end, delete position.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int info;
 struct node *next;
};
struct node *start=NULL;
void create()
{
 struct node *start=NULL;
 temp=(struct node*)malloc(sizeof(struct node));
 printf("\n enter data\t");
 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;
  }
 }
}
void insert_begin()
{
 struct node *temp;
 temp = (struct node*) malloc (sizeof (struct node));
 printf("\n Enter data : ");
 scanf ("%d", &temp -> info);
 temp -> next = start;
```

```
    start = temp;
}
void insert_mid()
{
  struct node *temp, *ptr;
  int pos, i;
  printf("\n Enter position to insert :");
  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("\n position out of range\n");
  free(temp);
}
else
{
  temp->next=ptr->next;
  ptr->next=temp;
}
```

```
void insert_end()
{
struct node *temp, *ptr;
temp = (struct node *)malloc(sizeof(struct node));
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 delete_begin()
{
 struct node *temp;
 if (start == NULL)
 {
  printf("\n List is empty \n");
  return;
 }
 temp = start;
```

```
    start = start -> next;
    printf("\n deleted element : %d\n", temp -> info);
    free(temp);
}
void delete_end()
{
    struct node *temp, *ptr;
    if (start == NULL)
    {
        printf("List is empty");
        return;
    }
    if (start -> next == NULL)
    {
        print("\n deleted element : \n", start -> info);
        free(start);
        start = NULL;
        return;
    }
    ptr = start;
    while (ptr -> next -> != NULL)
    {
        ptr = ptr = ptr -> next;
    }
    temp = ptr -> next;
    printf("Deleted element : \n", temp -> info);
    free(temp);
    ptr -> next = NULL
}
```

```
void delete_pos()
{
struct node *temp, *ptr;
int pos, i;
if (start == NULL)
{
  printf("List is empty");
  return;
}
printf("enter position to delete:");
scanf("%d", &pos);
if(pos == 1)
{
  temp = start;
  start = start->next;
  printf("\n deleted element: %d\n", temp->info);
  free(temp)
  return;
}
ptr = start;
for(i=1; i<pos-1 && ptr!=NULL; i++)
{
  ptr = ptr->next;
}
if(ptr==NULL || ptr->next==NULL)
{
  printf("position out of range");
  return;
}
temp = ptr->next;
ptr->next = temp->next;
```

```
    printf("\n deleted element : %d \n", temp->info);
    free(temp);
}
int main()
{
    int choice;
    while(1){
    {
        printf("\n MENU :\n 1:Create \n 2. Display \n
        3. insert begin \n 4. Insert end \n 5. Insert mid \n
        6. Delete begin \n 7. Delete end \n 8. Delete position
        \n 9. exit ");
        printf("\n enter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                create();
                break;
            case 2:
                display();
                break;
            case 3:
                insert_begin();
                break;
            case 4:
                insert_end();
                break;
            case 5:
                insert_mid();
                break;
```

```
case 6:
    delete_begin();
    break;
case 7:
    delete_end();
    break;
case 8:
    delete_pos();
    break;
case 9:
    exit(0);
default:
    printf("invalid choice");
}
}
return 0;
}
```

List and explain basic terminologies of linked list with example

(i) Node:

- It is the basic building block of a linked list.
- Node has two parts, data and pointer (next)
- Example:

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

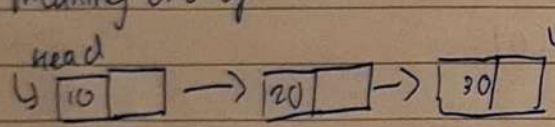
(ii) Head -

- Pointer that stores address of first node in the list
- Without head, we cannot access the list
- Example:

```
struct Node* head = NULL;
```

(iii) Tail node

- The last node in the list points to NULL, meaning end of list



(iv) Traversal

- Process of visiting each node in the list (starting from head. (to access or display data)
- Example -

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

(v) Insertion

- Adding new node to the linked list
- Types - At beginning
  - At end
  - At specific position
- Example -

```
struct Node* newnode = (struct Node*) malloc(
sizeof(struct node));
newNode->data = 5;
```

```
newNode->next = head;
head = NewNode
```

(vi) Deletion -

- Removing a node from the ~~user~~ linked list
- Types:
  - From the beginning
  - From the end
  - From a specific position
- Example -

```
struct Node* temp = head;
head = head -> next;
free(temp);
```

i) Differentiate between linked list and array

Linked list	Array
1) Dynamic size	1) Fixed size
2) Complex to implement	2) Easy to implement
3) Needs less memory	3) Needs more memory
4) Elements can be of different type	4) All elements must be of same type

EXPERIMENT - 6

(i) Write a menu driven program that implements doubly linked list for insertion deletion, and creation

```
#include <stdio.h>
#include <stdlib.h>
Struct node node
{
  int info;
  struct node * next ;
  Struct node * prev;
};
Struct node * start = NULL;
void create ()
{
  struct node * temp, * ptr;
  temp = (struct node*) malloc (sizeof (struct node));
  printf (" \n enter data : ");
  scanf ("%d " ) & temp -> info);
  temp -> next = NULL;
  temp -> prev = NULL;
  if (start == NULL)
  {
    Start = temp;
  }
  else
  {
    ptr = start ;
    while ptr -> next != NULL)
```

```
{
    ptr = ptr -> next;
}
ptr -> next = temp;
temp -> prev = ptr;
}
void display()
{
struct node *ptr;
if (start == NULL)
{
printf("empty list\n");
return;
}
else
{
ptr = start;
printf("\n list elements are : \n");
while (ptr != NULL)
{
    printf("%d ", ptr -> info);
    ptr = ptr -> next;
}
}
}
void insert_begin()
{
struct node* temp;
temp = (struct node*) malloc(sizeof (struct node));
printf("enter data\n");
```

```
    scanf("%d", &temp->info);
    temp->next = start;
    temp->prev = NULL;
    if (start != NULL)
        start->prev = temp;
    start = temp;
}
void insert_mid()
{
    struct node *temp, *ptr;
    int pos, i;
    printf("\n enter position to insert: ");
    scanf("%d", &pos);
    temp = (struct node *) malloc (sizeof (struct node));
    printf("enter data: ");
    scanf("%d", &temp->info);
    if (pos == 1)
    {
        temp->next = start;
        temp->prev = NULL;
        if (start != NULL)
            start->prev = temp;
        start = temp;
        return;
    }
    ptr = start;
    for(i=2; i< pos-1 && ptr != NULL; i++)
    {
        ptr = ptr->next;
    }
```

```
if (ptr == NULL)
{
  printf("\nposition out of range");
  free(temp);
}
else
{
  temp->next = ptr->next;
  temp->prev = ptr;
  if (ptr->next != NULL)
    ptr->next->prev = temp;
  ptr->next = temp;
}
}
void insert_end()
{
struct node *temp, *ptr;
temp = (struct node*)malloc(sizeof(struct node));
print("\nenter data:");
scanf("%d", &temp->info);
temp->next = NULL;
temp->prev = NULL;
if (start == NULL)
{
  start = temp;
}
else
{
  ptr = start;
  while(ptr->next != NULL)
  {
```

```
        ptr = ptr->next;
      }
      ptr->next = temp;
      temp->prev = ptr;
    }
}
void delete_begin()
{
  struct node *temp;
  if(start==NULL)
  {
    printf("list is empty");
    return;
  }
  temp = start;
  start = start->next;
  if(start != NULL)
    start->prev = NULL;
  printf("Deleted element : %d\n", temp->info);
  free(temp);
}
void delete_beginandend()
{
  struct node *temp
  if (start == NULL)
  {
    printf("List is empty");
    return;
  }
  if (start->next == NULL)
```

h(25)

```
{
  printf("Deleted element :%d\n", start->info);
  free(start);
  start = NULL;
  return;
}
temp = start;
while(temp->next != NULL)
{
  temp = temp->next;
}
printf("\n Deleted element %d\n", temp->info);
temp->prev->next = NULL;
free(temp);
}
void delete_pos()
{
 struct node *temp, *ptr;
 int pos, i;
 if(start == NULL)
 {
   printf("List is empty");
   return;
 }
 printf("\n enter position to delete :");
 scanf("%d", &pos);
 if(pos == 1)
 {
   temp = start;
   start = start->next;
   if(start != NULL)
```

```
    start -> prev = NULL;
    printf ("\n deleted element %d \n", temp -> info);
    free (temp);
    return;
  }
  ptr = start;
  for(i=1; i<pos && ptr != NULL; i++)
  {
    ptr = ptr -> next;
  }
  if(ptr == NULL)
  {
    printf("\n position out of range \n");
    return;
  }
  ptr -> prev -> next = ptr -> next;
  if(ptr -> next != NULL)
    ptr -> next -> prev = ptr -> prev;
  printf("deleted element : %d", ptr -> info);
  free(ptr);
}
int main()
{
  int choice;
  while (1)
  {
    printf("\n MENU : \n 1. Create \n 2. Display \n
        3. Display \n 4. Insert begin \n 5. Insert
        mid \n 6. Delete begin \n 7. Delete end
        \n 8. Delete position \n 9. Exit \n ");
  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_mid();
  break;
case 6:
  delete_begin()
  break;
case 7:
  delete_end();
  break;
case 8:
  delete_pos();
  break;
case 9:
  exit(0);
default:
  printf("invalid choice");
}
```

```
    }
    return 0;
}
```

OUTPUT -

MENU:
1) Create
2) Display
3) Insert begin
4) Insert end
5) Insert mid
6) Delete begin
7) Delete end
8) Delete position
9) Exit
Enter your choice : 1
Enter data : 4

Enter your choice : 2
List elements are : 4

Enter your choice : 9
Exiting ...

2) Explain circular linked list

ISA (25)

A circular linked list is a variation of a linked list in which the last node is connected back to the first node, forming a circular structure.
Circular linked list doesn't have null pointer at end.
Two types of circular linked list are -
(i) Singly circular linked list :
- Each node has one pointer to the next node, and the last node points back to the first node

(ii) Doubly circular linked list:
- Each node has two pointers, one to the next node and one two the previous node, with the first and last nodes linked to each other.

EXPERIMENT-7

(i) Write a C program to perform primitive operations on stack.

Code:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 5
int top=-1;
int S[max];
void push()
{
  int element;
  if (top==MAX-1)
  {
    printf("\n overflow");
  }
  else
  {
    printf("\n enter element to enter in stack:");
    scanf("%d", &element);
    top=top+1;
    S[top]=element;
  }
}
void pop()
{
  int item;
```

```
if(top == -1)
{
  printf("\n underflow");
}
else
{
  item = s[top];
  top--;
  printf("\n deleted element is %d", item);
}
}
void display()
{
int i;
printf("\n array: \n");
for(i=0; i<=top; i++)
{
  printf("%d", s[i]);
}
}
int main()
{
  int choice;
  do
  {
    printf("\n enter your choice: \n 1: push \n 2: Pop
    \n 3: display \n 4 exit \n");
    scanf("%d", &choice);
    switch(choice)
    {
      case 1:
```

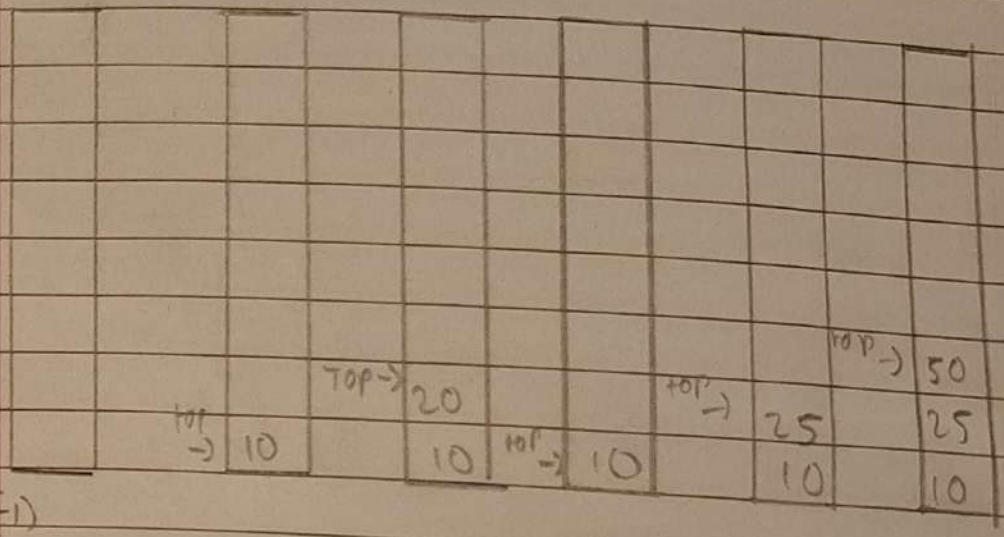
push(25)

```
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
            display();
            break;
        case 4:
            printf("exiting \n");
            break;
    }
} while choice != 4);
return 0;
}
```

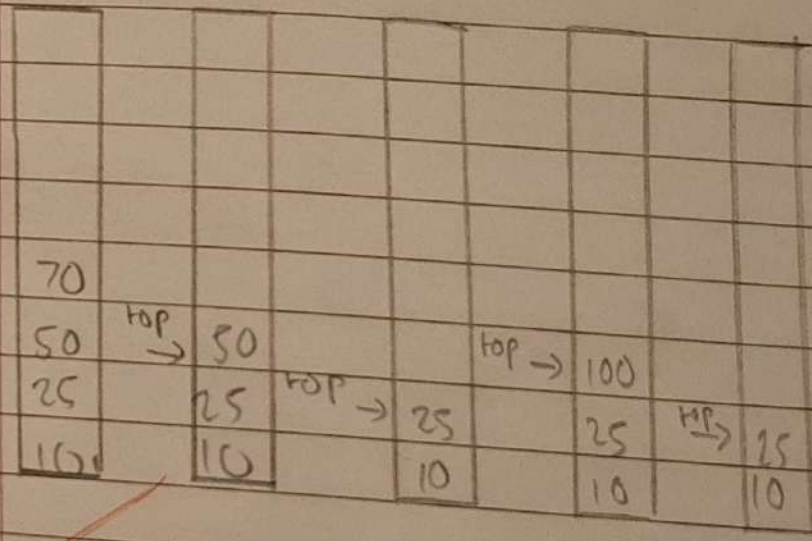
O/P:

```
enter your choice:
1. push
2. pop
3. display
4. Exit

1
enter element to enter in stack: 4

enter your choice: 3
array is: 4
enter your choice: 4
exiting
```

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



top(-1)



PAGE No.
DATE / /

EXPERIMENT - 8

(1) WAP to convert infix to postfix

Code:

```
#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("\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+

-

(ii) 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
*	*)*	H G
)	*)*)	H G
F	*)*)	H G F
^	*)*)^	H G F
E	*)*)^	H G F E
/	*)*)/	H G F E ^
D	*)*)/	H G F E ^ D
(	*)*)/(	H G F E ^ D /
-	*)*)/(-	H G F E ^ D / *

Input	Stack	Output
C	*) -	HGFE ^D/* C
*	*) - *	HGFE^D/*C
B	*) - *	HGFE^D/*CB
C	*) - *C	HGFE ^D/*CB* -
+	+	HGFE^D/*CB * -*
A	+	HGF E^D/*CB * -*A+

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

3) Evaluate prefix expression using stack
+ - * + 1 2 / 4 2 1 $ 4 2

Input	OP1	OP2	Result	Stack
2				2
4				2,4
$	4	2	16	16
1				16, 1
2				16, 1, 2
4				16, 1, 2, 4
/	4	2	2	16, 1, 2
2				16, 1, 2, 2
1				(6, 1, 2, 2, 1)
+	2	1	3	16, 1, 2, 3
*	3	2	6	16, 1, 6
-	6	1	5	16, 5
+	5	16	21	21

21

4) Write a C program to evaluate infix or postfix expression

Code:

```
#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 *c, x;
  printf("enter 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 -

enter expression : 1-+*765

1-765*+

EXPERIMENT - 9

1) Perform primitive operations on linear queue - insert, display

Code:

```
#include <stdio.h>
#include<stdlib.h>
#define MAX 5;
int queue [MAX]
int front = -1; rear = -1;

void insert ()
{
  int element ;
  if (rear = = MAX - 1)
  {
    printf("Queue overflow");
  }
  else
  {
    printf(" \n enter element to insert in queue :");
    scanf("%d", &element);
    if (front = = -1)
    {
      front = 0;
    }
    rear
    }
    rear ++;
    queue [rear ] = element;
```

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

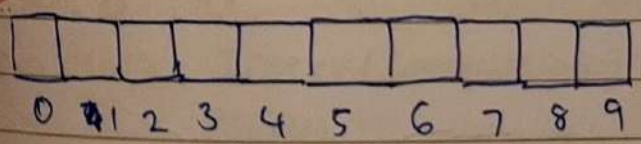
```
int main()
{
    int choice;
    do
    {
        printf("enter your choice : \n 1: Insert, \n2 : Delete; \n3 : Disp
        \n 4 : Exit \n");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                insert();
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("exiting");
                break;
            default:
                printf("invalid choice");
        } while (choice != 4);

        return 0;
}
```

classmate
Date
Page

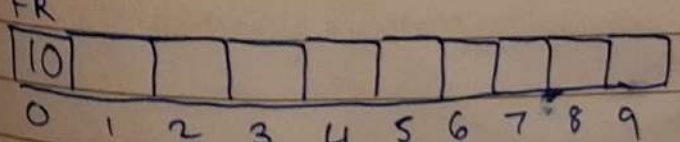
eg. O/P:
Enter your choice :
1) Insert
2) Delete
3) Display
4) Exit

1
Enter element to insert in queue: 5
Enter your choice: 3
Queue elements are : 5
Enter your choice: 4
exiting..

2) (i) Insert (10)
(ii) Insert (50)
(iii) Delete
(iv) Insert (100)
(v) Insert (20)
(vi) Delete
(vii) Insert (25)
(viii) Insert (200)

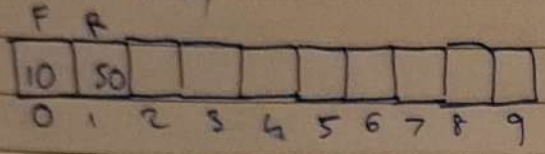


(i) Insert 10



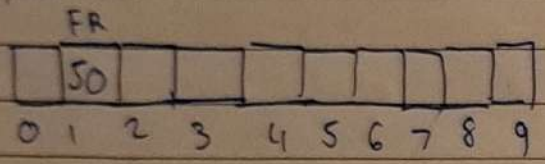
F=0
R=0

(ii) Insert (50)



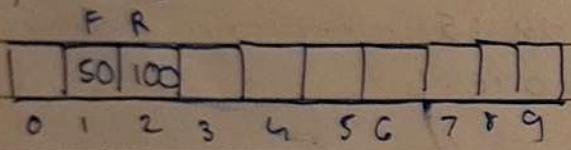
F = 0
R = ~~500~~ 1

(iii) Delete



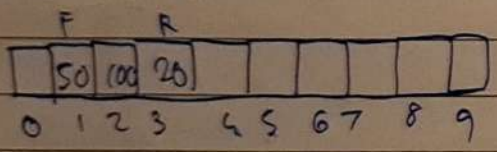
F = 1
R = 1

(iv) Insert(100)



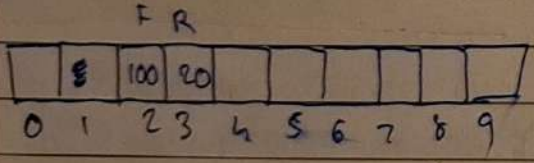
F = 1
R = 2

(v) Insert(20)



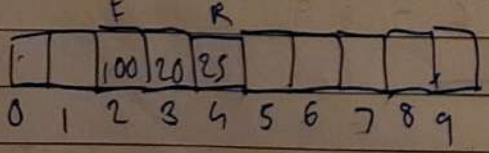
F = 1
R = 3

(vi) Delete



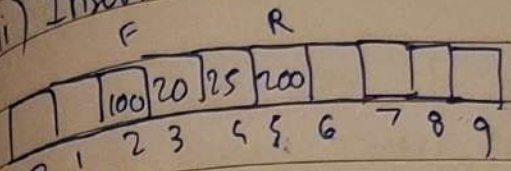
F = 2
R = 3

(vii) Insert 25



F = 2
R = 4

(viii) Insert 200



F = 2

R = 5

3) Explain concept of priority queue in detail

(3) Priority queue is a type of queue in which each element has a priority associated with it and elements are deleted according to their priority rather than the order they were inserted in

- In a normal queue it is First in First out
- In a priority queue it is highest priority first out
- Types -
  (i) Ascending priority queue
  - In APQ, smallest value gets highest priority.
  - Eg if 30, 10, 20 inserted, deletion order will be 10, 20, 30
  (ii) Descending priority queue
  - In DPQ, largest value gets highest priority
  - Eg if 30, 10, 20 inserted, deletion order be 30, 20, 10
- Operations - Insert (enqueue), Delete (Dequeue), Display
- Eg: 1) Hospitals
  2) Operating systems.

4) Algorithm for insertion and deletion on linear queue

(i) Insert/enqueue

Step 1: Check for overflow condition
Step 2: Check if queue is empty, then front and rear set to 0, so that new value can be stored at 0th location. Otherwise queue already has values then rear is incremented so that it points to the next

Step 3: value stored in queue at the location pointed by rear

Step 4: Exit

(ii) Delete/dequeue

~~Step 1: If front = -1 or f~~
1) Check for underflow condition, an underflow occurs if front = -1 or front > rear however if queue has some value then front is incremented so that it now points to the next value in the queue

2) Exit

EXPERIMENT - 10

Problem statement: WAP to perform primitive operations on circular queue - Insert, delete display

Code:-

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
int queue [size];
int f=-1, rear=-1;
void insert ()
{
 int value;
 if ((f==0 && r==size-1) || (rear+1) % size==f)
 {
  printf("queue is full \n");
 }
 else
 {
  printf("enter value to insert ");
  scanf("%d", &value);
  if (f==-1)
  {
   f=0
   r=0;
  }
  else
  {
   r=(r+1) % size;
  }
```

```
queue [r] = value;
printf("inserted %d \n", value);
}
}
void delete()
{
  if (f == -1)
  {
    printf("Queue is empty \n");
  }
  else
  {
    printf("deleted %d \n", queue[f]);
    if (f == r)
    {
      f = -1;
      r = -1;
    }
    else
    {
      f = (f+1) % size;
    }
  }
}
void display()
{
  if(f == -1)
  {
    printf("queue is empty");
    int i=f;
    while(i)
```

```
{
printf("%d", queue[i]);
if(i==r)
break;
i=(i+1)%size;
}
printf("\n");
}
int main()
{
int choice;
while(1)
{
printf("\n 1.Insert \n 2.Delete \n 3.Display
\n 4.Exit \n");
printf("enter your choice :");
scanf("%d", &choice);
switch(choice)
{
case 1: insert(); break;
case 2: delete(); break;
case 3: display(); break;
case 4: Exit(); break;
default: printf("invalid choice !");
}
return 0;
}
```

O/P:
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter element to insert: 10
Enter your choice: 4

2) What are advantages of circular queue show diagramatic representation

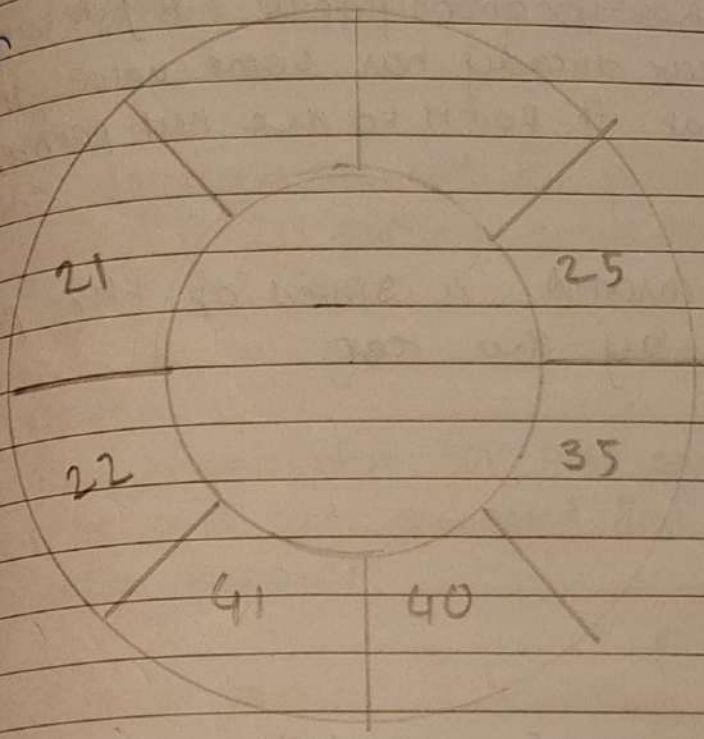
A2) Efficient memory utilization:
→ Unlike linear queue, space is reused after deletion.

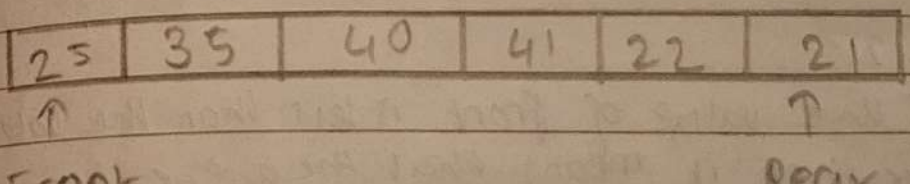
Overflow prevention:
→ In linear queue, once rear reaches the end, no more insertion are possible even if there are empty slots at the front

Fixed size but optimized
→ works well for fixed size memory buffers.

Faster operation
→ Insertion and deletion are O(1)

Diagram





Front = 0 Rear = 5

3) Algorithm - insert, delete for circular queue

Insertion -

Step 1 -

Check for overflow

Step 2 -

Check if queue is empty if f == -1, incase, the queue is empty set the values of front and rear to 0 so that the new element can be

Stored at the location pointed by the rear, if the rear already has some value, increment it so that it points to the next position.

Step 3 -
The value inserted is stored at the position pointed by the rear

Step 4 -
Exit

Deletion -
Step 1 -
Check for ~~overflow~~ underflow

Step 2 -
If the value of front is less than the value of rear then it means that the queue has some value the front is incremented

Step 3 -
Remove the element at the front and report deletion. If the element deleted was the last element then front = rear advances for the next element front = front + 1 and set front = -1 and rear = -1

Step 4 -
Exit

2) Write and explain application of queue

1) Printer:
When documents are sent for printing, they all get placed in a printing queue and get printed one after the other.

2) Call center:
The first person that calls gets ~~connect~~ entered in a queue and gets connected first to the next available operator

3) Amuzement park:
People wait in a queue to get on riders in a amuzement park.

4) Traffic managment
Traffic is a physical queue, cars leave in the order they arived

5) Differentiate between Stack and queue

Stack	Queue
1) It works on LIFO	It works on FIFO
2) Insertion happens at top	Insertion happens at rear
3) Deletion happens at top	Deletion happens at front

4)	Only top element is accessible	Both front and rear element are accessible
5)	Traversal is not posible	Traversal possible
6)	Can be implemented using array or linked list	Can be implemented using array, linked list or circular array.

Expt 11

Operations on Binary tree

```
#include<stdio.h>
#include<stdlib.h>
struct node {
  int data;
  struct node * left;
  struct node * right;
};
struct node * create (int val);
struct node * insertleft(struct node *root, int val);
struct node * insertright(struct node *root, int val);
void deletesubtree(struct node * root);
void deleteleft(struct node *root);
void deleteright(struct node *root);
void display(struct node* root);

int main(){
  struct node* root = create(1);
  insertleft(root, 2);
  insertright(root, 3);
  insertleft(root->left, 4);
  insertright(root->left, 5);
  insertleft(root->right, 6);
  insertright(root->right, 7);
  insertleft(root->left->left, 8);
  printf("Original tree:");
  display(root);
  printf("\n");
```

```
    deleteleft(root);
    printf("Tree after deleting left subtree of root: ");
    display(root);
    return 0;
}
struct node* create(int val){
    struct node *new = (struct node*) malloc(sizeof
    (struct node));
    if(new==NULL){
    printf("memory allocation failed");
    exit(1);
    }
    new->data = val;
    new->left=NULL;
    new->right=NULL;
    return new;
}
Struct node * insertright(struct node* root,
int val){
    if(root==NULL){
    printf("Cant insert in NULL root");
    return NULL;
    }
    root->right = create(val);
    return root->right;
}
struct node* insertleft(struct node* root, int val)
{
    if(root==NULL){
    printf("Cant insert in NULL");
    return NULL;
}
```

```
root->left = create(val);
return root->left;
}
void deletesubtree(struct node *node){
  if(node==NULL){
    return;
  }
  deletesubtree(node->left);
  deletesubtree(node->right);
  free(node);
}
void deleteleft(struct node *root){
  if(root==NULL||root->left==NULL){
    printf("nothing to delete");
    return;
  }
  deletesubtree(root->left);
  root->left=NULL;
  printf("left subtree deleted");
}
void deleteright(struct node *root){
  if(root==NULL||root->right==NULL){
    printf("nothing to delete");
    return;
  }
  deletesubtree(root->right);
  root->right=NULL;
}
void display(struct node *
```

```
printf("%d", root->data);
display(root->left);
display(root->right);
}
```

OUTPUT -

Original tree: 1 2 4 8 5 3 6 7
Left subtree deleted.
Tree after deletion: 1 3 6 7

Q2 Preorder, post order, inorder traversal

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node* left;
    struct node* right;
};
//declare all functions from last program
void preorder(struct node* root);
void postorder(struct node* root);
void inorder(struct node* root);
int main(){
    //make same tree from last program
    int choice;
    while(1){
    printf("Menu:");
    printf("1 preorder traversal");
    printf("2 post order traversal");
    printf("3 Inorder traversal");
```

```
printf("4. Exit");
printf("\n choose option:");
scanf("%d", &choice);
switch(choice){
case 1:
preorder(root);
break;
case 2:
postorder(root);
break;
case 3:
inorder(root);
break;
case 4:
exit(0);
break;
default:
printf("invalid input");
}
}
return 0;
}
// same function definitions as last program
void preorder(struct node *root){
if(root == NULL){
return;
}
printf("%d", root->data);
preorder(root->left);
preorder(root->right);
}
```

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

OUTPUT:

Menu:

1) preorder traversal
2) Postorder traversal
3) inorder traversal

Choose option: 1

1 2 4 8 5 3 6 7

Menu:

Choose option: 2

8 4 5 2 6 7 3 1

Menu:

Choose option: 3

8 4 2 5 1 6 3 7

Exp 12

Graphs

1) Graph using adjacency matrix

```
#include<stdio.h>
#define V 5
void init (int arr[V][V]){
  int i, j;
  for(i=0;i<V;i++){
    for(j=0;j<V;j++){
      arr[i][j]=0;
    }
  }
}
void insert edge(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;
  printf("\n adjacency matrix");
  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, 2, 0);
  insertedge(Adjmatrix, 2, 3);
  insertedge(AdjMatrix, 1, 3);
  insertedge(Adjmatrix, 4, 3);
  insertedge(Adjmatrix, 2, 4);
  printAdjMatrix(AdjMawix);
  return 0;
}
```

OUTPUT:

Adjacency Matrix:

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

Adjacency List

```
#include <stdio.h>
#include <stdlib.h>
#define V 5
struct node{
  int vertex;
  struct node * next;
};
struct node * adjlist[V];
struct node * createnode(int v){
  struct node * new = malloc (sizeof (struct node));
  new -> vertex = v;
  new -> next = NULL;
  return new;
}
void init (){
  for(int i=0; i<V; i++){
    adjList[i] = NULL;
  }
}
void insertedge(int src, int dest){
  struct node * new = create node (dest);
  new -> next = adjlist [src];
  adj list [src] = new;
  new = createnode(src);
  new -> next = adj list [dest];
  adjList [dest = New;
}
void print adjlist(){
```

```
    for(int i = 0; i<v; i++){
        struct node* temp = adjlist[i];
        printf("%d: ", i);
        while(temp){
            printf("%d -> ", temp -> vertex);
            temp = temp -> next;
        }
        printf("NULL");
    }
}
int main(){
    init();
    insertedge(0,1);
    insertedge(0,2);
    insertedge(2,0);
    insertedge(1,3);
    insertedge(4,3);
    insertedge(2,4);
    printAdjlist();
    return 0;
}
```

OUTPUT:

0: 2 -> 2 -> 1 -> NULL
1: 3 -> 0 -> NULL
2: 4 -> 3 -> 0 -> 0 -> NULL
3: 4 -> 1 -> 2 -> NULL
4: 2 -> 3 -> NULL

Theory Questions:

01) Write all basic terminologies of graph and explain them

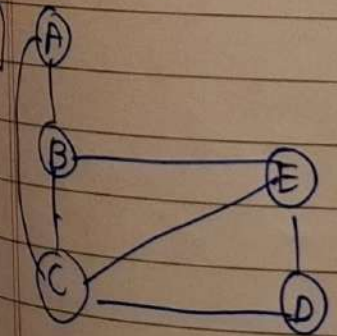
I] Vertex - Referred to as node
Fundamental unit of graph
Used to represent entity in a graph.

II] Edges - These are the lines of connections between vertexes in a graph

III] Degree of vertex - It is the number of edges incident on (connected to) that vertex

IV] path: It is the sequence of vertices where each adjacent pair is connected by an edge.

02) Adjacency matrix and adjacency list representation:

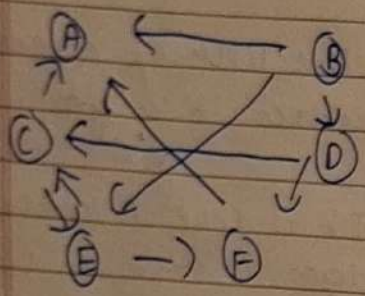


Adj. List

Vertex	Adjacent vertices
A	B, C
B	A, C, E
C	A, B, D, E
D	C, E
E	B, C, D

Adjacency Matrix

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

2) 

Adjacency list

Vertex	Adjacent vertices
A	B, C, F
B	A, D, E
C	A, D, E
D	B, C, F
E	B, C, F
F	A, E, D

Adjacency matrix

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

6/11