ABES ENGINEERING COLLEGE

NAME – ABHAY PANDEY

ROLL NO. – 2100320120004

BRANCH – CS

SECTION – A

DATE-21-02-2023

PROGRAM 1A - Program for Insertion in any array

ALGORITHM Insertion(A[], N, i, key)
BEGIN:

```
FOR j=N TO i STEP-1 DO
        A[j+1]=A[j]
        A[i]=key
        N=N+1
```

END;

Time Complexity:ϴ(N)
Space Complexity:ϴ(1)

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

{
  printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
  int arr[100];
  int n;
  printf("Enter the size of array\n");
  scanf("%d",&n);
  printf("Enter %d elements\n",n);
  for(int i=0;i<n;i++)
  scanf("%d",&arr[i]);
  int posi;
  printf("position:");
  scanf("%d",&posi);
  int ele;
  printf("Enter element\n");
  scanf("%d",&ele);
  for(int i=n;i>=posi;i--)
    arr[i]=arr[i-1];
    arr[posi-1]=ele;
    n++;
  for(int i=0;i<n;i++)
    printf("%d",arr[i]);

  return 0 ;}
```

OUTPUT:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the size of array-6
Enter 6 elements-2 4 5 7 3 9
Enter the position:3
Enter element you want to insert-89
Resultant array is:2 4 89 5 7 3 9
```

PROGRAM 22 - Transpose without using second matrix

ALGORITHM: Matrixtranspose(A[][], M,N)

BEGIN:

```
    FOR i=1 TO M DO
        FOR j=1 TO i DO
            temp=A[i][j]
            A[i][j]=A[j][i]
            A[j][i]=temp
    RETURN A
```

END;

Time Complexity: ϴ(N^2)

Space Complexity:ϴ(1)

```
#include<stdio.h>
int main()
{
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n,m;
   printf("Enter the rows and columns of matrix:\n");
   scanf("%d%d",&n,&m);
   int arr[n][m];
   printf("Enter the elements of matrix:\n");
    for(int i=0;i<n;i++)
    {
      for(int j=0;j<m;j++)
         scanf("%d",&arr[i][j]);
    }
   for(int i=0;i<n;i++)
   {
      for(int j=i;j<m;j++)
      {
         int temp=arr[i][j];
         arr[i][j]=arr[j][i];
         arr[j][i]=temp;
      }
   }
   printf("Transpose of the matrix is:\n");
    for(int i=0;i<m;i++)
    {
```

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

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the rows and columns of matrix:
3 3
Enter the elements of matrix:
1 2 3 4 5 6 8 8 9
Elements of matrix:
1 2 3
4 5 6
8 8 9
Transpose of the matrix is:
1 4 8
2 5 8
3 6 9
```

PROGRAM 1C - Program for Traversing of array

ALGORITHM Traverse(A[], N)
BEGIN:

```
        FOR i=1 TO N DO
              WRITE(A[i])
```

END;

Time Complexity:ϴ(N)
Space Complexity:ϴ(1)

```c
#include<stdio.h>

int main()

{
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   printf("Enter the size of array:");
   scanf("%d",&n);
   int arr[n];
   printf("Enter the elements of array:");
   for(int i=0;i<n;i++)
   scanf("%d",&arr[i]);

   printf("Elements of array are-->\n");
   for(int i=0;i<n;i++)
   printf("%d element of array is: %d\n",i+1,arr[i]);
   return 0 ;
}
```

OUTPUT:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of array:6
Enter the elements of array:5 6 9 2 4 0
Elements of array are-->
1 element of array is: 5
2 element of array is: 6
3 element of array is: 9
4 element of array is: 2
5 element of array is: 4
6 element of array is: 0
```

PROGRAM 1B - Program for Deletion of elements in array

ALGORITHM Deletion(A[], N, i)
BEGIN:

```
X=A[i]
  FOR j=i+1 TO N DO
        A[j-1]=A[i]
        N=N-1
RETURN x
```

END;

Time Complexity:Ɵ(N)
Space Complexity:Ɵ(1)

```
#include <stdio.h>


int main()
{    printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int array[100], position, c, n;

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

   printf("Enter %d elements\n", n);

   for ( c = 0 ; c < n ; c++ )
   scanf("%d", &array[c]);

   printf("Enter the location where you wish to delete element\n");
   scanf("%d", &position);

   if ( position >= n+1 )
   printf("Deletion not possible.\n");

   else
   {
      for ( c = position - 1 ; c < n - 1 ; c++ )
      array[c] = array[c+1];
```

```
    printf("Resultant array is\n");

    for( c = 0 ; c < n - 1 ; c++ )
    printf("%d\n", array[c]);
  }
  return 0; }
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter number of elements in array

6
Enter 6 elements
1 2 3 4 5 6
Enter the location where you wish to delete element
4
Resultant array is
1
2
3
5
6
```

PROGRAM 3 - Program to Find the number, which is not repeated in Array of integers, others are present for two times

ALGORITHM: Arr_func(A[], N)
BEGIN:

```
    K=0,c,B[20]
    FOR i=0 TO N DO
        c=0
        FOR j=0 TO N DO
            IF A[j]==A[i] THEN
                c=c+1
            IF c==1 THEN
                B[k++]=A[i]
    FOR i=0 TO k DO
        WRITE(B[i])
```

END;

Time Complexity:Θ(N^2)
Space Complexity:Θ(1)

```c
#include<stdio.h>
void unique(int arr[],int n)
{
   int count=1,i,j;
   for( i=0;i<n;i++)
   {
      for( j=0;j<n;j++)
      {
         if(arr[i]==arr[j]&& i!=j)
         break;
      }
         if(j==n)
         {
            printf("Unique element %d is:%d\n",count,arr[i]);
            count++;
          }
   }

}


int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   printf("Enter size of array:\n");
   scanf("%d",&n);
```

```c
    int arr[n];
    printf("Enter array elements:\n");
    for(int i=0;i<n;i++)
    scanf("%d",&arr[i]);

    unique(arr,n);
    return 0 ;
}
```

OUTPUT:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter size of array-5
Enter array elements-2 3 4 5 2
Unique element 1 is:3
Unique element 2 is:4
Unique element 3 is:5
```

PROGRAM 63 - Program for finding nth Fibonacci number using Recursion and improving its run time to save stack operations

ALGORITHM Fibo(a)

```
BEGIN:
        IF a==1 THEN
                RETURN 0
         ELSE
                IF a==2 THEN
                        RETURN 1
                 ELSE
                        RETURN Fibo(a-1)+Fibo(a-2)
END;
```

Time Complexity: $\Theta(2^N)$
Space Complexity: $\Theta(N)$

```c
#include<stdio.h>

int fibo(int n){
   if(n<=1)
   return n;

   return fibo(n-1)+fibo(n-2);
}
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   printf("Enter the number:");
   scanf("%d",&n);

   printf("%dth fibonacci number is:%d",n,fibo(n-1));
   return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the number:6
6th fibonacci number is:5
```

PROGRAM 59 - Program for factorial of a given number using recursion

ALGORITHM FACTORIAL(a)

```
BEGIN :
IF a==0
        RETURN(1)
ELSE
        IF(a>0)
                RETURN(a*FACTORIAL(a-1))
END;
```

Time Complexity: $\Theta(n)$
Space Complexity: $\Theta(n)$

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


int fact(int n){
   if (n==0)
   {
      return 1;
   }
   else
   {
      return n * fact(n-1);
   }


}
int main(){
   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   printf("Enter the number  : \n");
   scanf("%d",&n);

   printf("Factorial of the number is : ");
   printf("%d",fact(n));
   return 0;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the number  :
7
Factorial of the number is : 5040
```

PROGRAM 64 - Program for finding the GCD of two numbers using Recursion

```
ALGORITHM HCF(a,b)
BEGIN:
        IF a==b THEN
                RETURN a
        ELSE IF a>b THEN
                RETURN HCF(a-b,b)
                ELSE
                RETURN HCF (a,b-a)
END;
```

Time Complexity: O(log n)
Space Complexity: Θ(1)

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

int gcd(int a, int b)
{
  if (a == b)
  {
    return a;
  }
  else
  {
    if (a > b)
    {
      return gcd(a - b, b);
    }
    else
    {
      return gcd(a, b - a);
    }
  }
}
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
  int a, b;
  printf("Enter the numbers  : \n");
  scanf("%d %d", &a, &b);
    printf("GCD of the numbers is : ");
  printf("%d", gcd(a, b));
return 0;
}
```

Output:

```
Enter the numbers  :
48
24
GCD of the numbers is : 24
```

EXPERIMENT 61 - Program for Computing A raised to power n using Recursion

```
ALGORITHM POWER(a,b)
BEGIN:
        IF b == 0 THEN
                RETURN 1
         ELSE
                IF b%2 == 0 THEN
                        RETURN  POWER(a,b/2) * POWER(a,b/2)
                ELSE
                        RETURN  a+ POWER(a,b/2) * POWER(a,b/2)

END;
```

Time Complexity: O(log b)
Space Complexity: Ө(log b)

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

int power(int a, int b)
{
   if (b == 0)
   {
      return 1;
   }
   else
   {
      return a * power(a, b - 1);
   }
}
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int a, b;
   printf("Enter the numbers  : \n");
   scanf("%d %d", &a, &b);

   printf("Power of the number is : ");
   printf("%d", power(a, b));
   return 0;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the numbers  :
7 3
Power of the number is : 343
```

PROGRAM 65 - Program to reverse the given number using Recursion

ALGORITHM REV (a,len)
BEGIN:
    IF len ==1
        RETURN a
    ELSE
        RETURN((a%10)*pow(10,len-1))+REV(a/10,len-1)
END;

Time Complexity: Θ (log n)
Space Complexity: Θ (log n)

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


int reverse(int n,int temp,int sum)
{
   if (n > 0)
   {
      temp = n % 10;
      sum = sum * 10 + temp;
      reverse(n / 10 , temp,sum);
   }
   else
   {
      return sum;
   }
}
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   int temp = 0, sum = 0;
   printf("Enter the number  : ");
   scanf("%d",&n);
   printf("Reverse of the number is : ");
   printf("%d", reverse(n,temp,sum));
   return 0;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the number  : 489215
Reverse of the number is : 512984
```

PROGRAM 60 - Program for Towers of Hanoi for n disk (user defined)

ALGORITHM TOH(N,S,M,D)

```
BEGIN:
IF N==1 THEN
        Transfer disk from S to D
ELSE
        TOH(N-1,S,M,D)
        Transfer Disk From S to D
        TOH(N-1M,S,D)
End;
```

Time Complexity: Θ (2^n)
Space Complexity: Θ (n)

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

void tower_of_hanoi(int n,int s,int m,int d){
 if (n>0)
   {
      tower_of_hanoi(n-1,s,d,m);
      printf("Move from %d -> %d \n",s,d);
      tower_of_hanoi(n-1,m,s,d);
   }
}
int main(){
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   printf("Enter the number of discs  : ");
   scanf("%d",&n);
 printf("Process to transfer discs are :");
   tower_of_hanoi(n,1,2,3);

   return 0;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the number of discs  : 3
Process to transfer discs are :Disc from 1 -> 3
Disc from 1 -> 2
Disc from 3 -> 2
Disc from 1 -> 3
Disc from 2 -> 1
Disc from 2 -> 3
Disc from 1 -> 3
```

PROGRAM 2 - Program for Insertion in sorted array

ALGORITHM Sorted(A[], N, key)
BEGIN:

```
    i=0
    WHILE A[i]<key DO
        i=i+1
        RETURN i
```

END;

Time Complexity:Ɵ(N)
Space Complexity:Ɵ(1)

ALGORITHM: INS_sorted(A[], N ,i, key)
BEGIN:

```
    FOR j=N-1 TO i STEP-1 DO
        A[j+1]=A[j]
    A[i]=key
    N=N+1
```

END;

Time Complexity:Ɵ(N)
Space Complexity:Ɵ(1)

```
#include<stdio.h>
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   printf("Enter the size of array:\n");
   scanf("%d",&n);
   int arr[n];
   printf("Enter the array elements:");
      for(int i=0;i<n;i++)
      {
      scanf("%d",&arr[i]);
   }
  int ele;
  printf("Enter the element that you wants to enter:");
  scanf("%d",&ele);

  int pos=0;
  for(int i=0;i<n;i++)
  {
   if(arr[i]<ele)
      pos++;
   else
```

```c
        break;
    }

  for(int i=n;i>=pos;i--)
        arr[i]=arr[i-1];

        arr[pos]=ele;
        n++;

    printf("Array after the insertion is:\n");
    for(int i=0;i<n;i++){
        printf("%d",arr[i]);
    }
    return 0;

}
```

OUTPUT:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the size of array:6
Enter the array elements:2 4 7 8 9 11
Enter the element that you wants to enter:6
Array after the insertion is:2 4 6 7 8 9 11
```

PROGRAM 15 - Program for Intersection of two Sets

ALGORITHM: SetIntersection(A[],m,B[],n)
BEGIN:

```
    C[m+n]
    i=1, j=1, k=1
    WHILE i<=m AND j<=n DO
        IF A[i]<B[j] THEN
            i=i+1
        ELSE
            IF A[i]==B[j] THEN
                C[k]=B[j]
            i=i+1
            j=j+1
            k=k+1
            ELSE
                j=j+1
    RETURN C
```

END;

Time Complexity:ϴ(N)
Space Complexity:ϴ(N)

```
#include<stdio.h>

void intersection(int arr[],int brr[],int n,int m)
{
   int i=0,j=0;
   printf("Instersection of first and second set is:");
   while(i<n and j<m)
   {
      if(arr[i]<brr[j])
         i++;

      else if(arr[i]>brr[j])
         j++;

      else
      {
         printf("%d ",arr[i]);
         i++;
         j++;
      }
   }
}

int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
```

```c
    int n,m;
    printf("Enter the size of first and second set :");
    scanf("%d%d",&n,&m);

    int arr[n],brr[m];

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

    printf("Enter the second set elements:");
    for(int j=0;j<m;j++)
    scanf("%d",&brr[j]);

    // sort(arr,arr+n);
    // sort(brr,brr+m);
    intersection(arr,brr,n,m);
    return 0 ;
}
```

output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the size of first and second set :5 5
Enter the first set elements:1 2 3 4 5
Enter the second set elements:3 4 5 6 7
Instersection of first and second set is:3 4 5
```

PROGRAM 11 - Program for Merging of two Sorted arrays

ALGORITHM: MergeArr(A[],m,B[],n)
BEGIN:

```
    C[m+n]
    i=1, j=1, k=1
    WHILE i<=m AND j<=n DO
        IF A[i]<B[j] THEN
            C[k]=A[i]
            i=i+1
            k=k+1
        ELSE
            C[k]=B[j]
            J=j+1
            k=k+1
    WHILE i<=m DO
        C[k]=A[i]
        i=i+1
        k=k+1
    WHILE j<=n DO
        C[k]=B[j]
        J=j+1
        k=k+1
    RETURN C
```

END;

Time Complexity: Θ(N)
Space Complexity: Θ(N)

```c
#include<stdio.h>
void merge(int arr[],int brr[],int n,int m,int ans[])
{
  int i=0,j=0,k=0;
  printf("Sets after the merging is:");
  while(i<n&&j<m)
  {
   if(arr[i]<brr[j])
   ans[k++]=arr[i++];

   else
   ans[k++]=brr[j++];
  }
   while(i<n)
   ans[k++]=arr[i++];

   while(j<m)
```

```
    ans[k++]=brr[j++];

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

int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
    int n,m;
    printf("Enter the size of first and second set:");
    scanf("%d%d",&n,&m);

    int arr[n],brr[m];

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

    printf("Enter the second set elements:");
    for(int j=0;j<m;j++)
    scanf("%d",&brr[j]);

    int ans[n+m];

    merge(arr,brr,n,m,ans);
    return 0 ;
}
```

OUTPUT:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of first and second set:5 5
Enter the first set elements:2 3 4 5 6
Enter the second set elements:5 6 7 8 9
Sets after the merging is:2 3 4 5 5 6 6 7 8 9
```

PROGRAM 16 - Program for Set Difference

ALGORITHM: SetDIFference(A[],m,B[],n)
BEGIN:

```
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        i=i+1
                ELSE
                         IF A[i]==B[j] THEN
                        i=i+1
                        j=j+1
                ELSE
                        C[k]=B[j]
                        j=j+1
                        k=k+1
        WHILE j<=n DO
                C[k]=B[j]
                J=j+1
                k=k+1
        RETURN C
```

END;

Time Complexity:ϴ(N)
Space Complexity:ϴ(N)

```c
#include<stdio.h>

void AminusB(int arr[],int brr[],int n,int m){
   int k=0;
   int ans[100];
   int i,j;
   printf("Difference of both sets(i.e, A-B) is:");
   for(i=0;i<n;i++)
   {
      for(j=0;j<m;j++)
      {
         if(arr[i]==brr[j])
         break;
      }
         if(j==m)
         ans[k++]=arr[i];
   }

   for(int i=0;i<k;i++)
   printf("%d ",ans[i]);
}
```

```c
void BminusA(int arr[],int brr[],int n,int m){
   int k=0;
   int ans[100];
   int i,j;
   printf("Difference of both sets(i.e, B-A) is:");
   for(i=0;i<m;i++){
      for(j=0;j<n;j++)
      {
         if(brr[i]==arr[j])
         break;
      }

      if(j==n)
       ans[k++]=brr[i];
   }

   for(int i=0;i<k;i++)
   printf("%d ",ans[i]);
}

int main()
{  printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n,m;
   printf("Enter the size of A and B set:");
   scanf("%d%d",&n,&m);

   int arr[n],brr[m];

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

   printf("Enter the set B elements:");
   for(int j=0;j<m;j++)
   scanf("%d",&brr[j]);

   int i=0;
   int j=0;

   int c;
   printf("Enter the choice-\n1 for A-B\n2 for B-A\n");
   scanf("%d",&c);

   if(c==1)
   AminusB(arr,brr,n,m);

   if(c==2)
   BminusA(arr,brr,n,m);
   return 0 ;
```

}

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of A and B set:5 5
Enter the set A elements:2 3 4 5 6
Enter the set B elements:5 6 7  8 9
Enter the choice-
1 for A-B
2 for B-A
1
Your Choice is 1
Difference of both sets(i.e, A-B) is:2 3 4
```

PROGRAM 14 - Program for Union of two sets

ALGORITHM: SetUnion(A[],m,B[],n)
BEGIN:

```
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        C[k]=A[i]
                        i=i+1
                        k=k+1
                 ELSE
                        IF A[i]==B[j] THEN
                                C[k]=B[j]
                                i=i+1
                                j=j+1
                                k=k+1
                         ELSE
                                C[k]=B[j]
                                j=j+1
                                k=k+1
        WHILE i<=m DO
                C[k]=A[i]
                i=i+1
                k=k+1
        WHILE j<=n DO
                C[k]=B[j]
                J=j+1
                k=k+1
RETURN C
```

END;

Time Complexity:ϴ(N)
Space Complexity:ϴ(N)

```c
#include<stdio.h>

void unionArr(int arr[],int brr[],int n,int m,int ans[])
{
  int i=0,j=0,k=0;

  while(i<n&&j<m)
  {

   if(arr[i]<brr[j])
   ans[k++]=arr[i++];
```

```c
   else if(arr[i]=brr[j])
   {
     ans[k++]=arr[i++];
     j++;
         }
   else
   ans[k++]=brr[j++];

  }
   while(i<n)
   ans[k++]=arr[i++];

   while(j<m)
   ans[k++]=brr[j++];

   printf("Union of the first and second set is:");
   for(int i=0;i<k;i++)
   printf("%d ",ans[i]);

}
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n,m;
   printf("Enter the size of first and second set :");
   scanf("%d%d",&n,&m);

   int arr[n],brr[m];

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

   printf("Enter the second set elements:");
   for(int j=0;j<m;j++)
   scanf("%d",&brr[j]);

   int ans[n+m];

   unionArr(arr,brr,n,m,ans);
   return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of first and second set :5 5
Enter the first set elements:2 3 4 5 6
Enter the second set elements:5 6 7 8 9
Union of the first and second set is:2 3 4 5 6 7 8 9
```

PROGRAM 5 - Program for Binary Search in an array

ALGORITHM Binary_search(A[], N, key)

BEGIN:

```
    HIGH=N-1
    LOW=0
    WHILE LOW<=HIGH DO
        MID=(LOW+HIGH)/2
        IF A[MID]==key THEN
            RETURN MID
        ELSE
            IF key<A[MID] THEN
                HIGH=MID-1
            ELSE
                LOW=MID+1
    RETURN -1
```

END;

Worst Case Time Complexity: O(logN)
Best Case Time Complexity: Ω(1)
Space Complexity: Θ(1)

```c
#include<stdio.h>

int binarySearch(int arr[],int n,int key){
   int s=0;
   int l=n;

    while(s<=l)
    {
      int mid=(s+l)/2;
      if(arr[mid]>key)
      l=mid-1;

      else if(arr[mid]<key)
      s=mid+1;

      else
      return mid;
   }
   return -1;

}

int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
```

```
    printf("Enter the size of array:");
    scanf("%d",&n);
    int arr[n];
    printf("Enter the elements of array:");
    for(int i=0;i<n;i++)
    scanf("%d",&arr[i]);
    int key;
    printf("Enter the element to search:");
    scanf("%d",&key);
    printf("Key is present at %d index",binarySearch(arr,n,key));

    return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the size of array:6
Enter the elements of array:1 3 5 7 9 13
Enter the element to search:7
Key is present at 3 index
```

PROGRAM 4 - Program for Linear Search

ALGORITHM Linear_search(A[], N, key)
BEGIN:

```
        FOR i=1 TO N DO
                IF A[i]==key THEN
                        RETURN i
        RETURN -1
```

END;

Worst Case Time Complexity: O(N)
Best Case Time Complexity: $\boldsymbol{\Omega}$(1)
Space Complexity: Θ(1)

```c
#include<stdio.h>int main()
{  printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n;
   printf("Enter the size of array: ");
   scanf("%d",&n);
    int arr[n];
    printf("Enter the elements of array :");
    for(int i=0;i<n;i++)
    scanf("%d",&arr[i]);
    int key;
   printf("Enter the element to be search:");
   scanf("%d",&key);
   int flag=0;
   for(int i=0;i<n;i++)
   {
      if (arr[i]==key)
      {
         printf("Elements is present at %d place.",i+1);
         flag=1;
         break;
      }
   }
    if(flag==0)
    printf("Element is not present in array !!!");
    return 0 ;
}
```

OUTPUT:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of array: 6
Enter the elements of array :6 4 3 7 8 1
Enter the element to be search:3
Elements is present at 3 place.
```

PROGRAM 19 - Program for Matrix Addition

ALGORITHM: Matrixadd(A[][], B[][], M,N)
BEGIN:C[M][N]

    FOR i=1 TO M DO
        FOR j=1 TO N DO
            C[i][j]=A[i][j]+B[i][j]
    RETURN C

END;
Time Complexity: $\Theta(N^2)$
Space Complexity: $\Theta(N^2)$
Source Code :

```
#include <stdio.h>
#include <math.h>
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
    int m, n, o, p;
    printf("Enter the row and column of first matrix  : \n");
    scanf("%d %d", &m, &n);
    int a[m][n];
    printf("Enter elements of first matrix  : \n");
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf("Enter the row and column of second matrix  : \n");
    scanf("%d %d", &o, &p);
    int b[o][p];
    printf("Enter elements of second matrix  : \n");
    for (int i = 0; i < o; i++)
    {
        for (int j = 0; j < p; j++)
        {
            scanf("%d", &b[i][j]);
        }
    }
    if (n == o)
    {
        printf("Addition of matrix is : \n");
        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                printf("%d ", (a[i][j] + b[i][j]));
            }
            printf("\n");
        }
```

```
    }

    return 0;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the row and column of first matrix  :
3 3
Enter elements of first matrix  :
1 2 3 4 5 6 7 8 9
Elements of first matrix :
1 2 3
4 5 6
7 8 9
Enter the row and column of second matrix  :
3 3
Enter elements of second matrix  :
1 2 3 4 5 6 7 8 9
Elements of second matrix :
1 2 3
4 5 6
7 8 9
Addition of matrix is :
2 4 6
8 10 12
14 16 18
```

PROGRAM 20 - Program for Matrix Multiplication

ALGORITHM: Matrixmultiply(A[][], M,N, B[][], P,Q)
BEGIN:

```
    C[M][Q]
    IF N!=P THEN
    FOR i=1 TO M DO
        FOR j=1 T0 Q DO
            C[i][j]=0
            FOR k=1 TO N DO
                C[i][j]=C[i][j]+A[i][k]*B[k][j]
    RETURN C
```

END;

Time Complexity: $\Theta(N^3)$
Space Complexity:$\Theta(N^2)$

```c
#include<stdio.h>

int main()

{

   int n,m,p,q;
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   printf("Enter the rows and columns of matrix A and B-");
   scanf("%d%d%d%d",&n,&m,&p,&q);
   if(m==p){
   int arr[n][m];
   int brr[m][q];
   int ans[n][q];
   printf("Enter the elements of matrix A-");
    for(int i=0;i<n;i++){
      for(int j=0;j<m;j++)
         scanf("%d",&arr[i][j]);
    }

    printf("Enter the elements of matrix B-");

     for(int i=0;i<m;i++){

      for(int j=0;j<q;j++)

         scanf("%d",&brr[i][j]);

    }

    for(int i=0;i<n;i++){

      for(int j=0;j<q;j++)

         ans[i][j]=0;
```

```
    }


    for(int i=0;i<n;i++){
      for(int j=0;j<q;j++){
          for(int k=0;k<m;k++)
           ans[i][j]+=arr[i][k]*brr[k][j];
      }
    }
   printf("Multiplication of matrix A and B is-");
    for(int i=0;i<n;i++){
      for(int j=0;j<q;j++)
          printf("%d ",ans[i][j]);
         printf("\n");
    }}
   return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the rows and columns of matrix A and B-3 3 3 3
Enter the elements of matrix A-1 2 3 4 1 0 5 2 1
Enter the elements of matrix B-5 3 2 1 0 5 2 1 8
 Elements of matrix A-
1 2 3
4 1 0
5 2 1
 Elements of matrix B-
5 3 2
1 0 5
2 1 8
Multiplication of matrix A and B is-
13 6 36
21 12 13
29 16 28
```

PROGRAM 21 - Program for Transpose of matrix using second matrix

ALGORITHM: Matrix_transpose (A[][], M,N)
BEGIN:

```
        B[N][M]
        FOR I =1 TO M DO
                FOR j=1 TO N DO
                        B[j][i]=A[i][j]
        RETURN B
```

END;

Time Complexity: Ɵ(N^2)
Space Complexity:Ɵ(N^2

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

int main()
{  printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
   int n, m;
    printf("Enter the row and column of matrix  : \n");
   scanf("%d %d", &m, &n);
   int a[n][m];
   int t[m][n];
   printf("Enter the elements of matrix  : \n");
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < m; j++)
      {
         scanf("%d", &a[i][j]);
      }
   }

   printf("The input matrix is  \n");
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < m; j++)
      {
         printf("%d ", a[i][j]);
      }
      printf("\n");
   }

   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < m; j++)
      {
         t[i][j] = a[j][i];
      }
   }
```

```c
    printf("Transpose of matrix is : \n");
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", t[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the row and column of matrix  :
3 3
Enter the elements of matrix  :
1 2 3 4 5 6 7 8 9
The input matrix is
1 2 3
4 5 6
7 8 9
Transpose of matrix is :
1 4 7
2 5 8
3 6 9
```

PROGRAM 6 - Program for Index Sequential Search

ALGORITHM: INDsearch(data[N],KEY,index[M][2])
BEGIN:

```
FOR i=0 TO M-1 DO
    IF KEY==index[i][1] THEN
      RETURN index[i][0]
 ELSE
     IF KEY <index[i][1] THEN
          high=index[i][0]-1
          Low =index[i-1][0]+1
           BREAK
 FOR i=low TO high DO
          IF KEY ==data[i] THEN
            RETURN i
 RETURN -1
```

END;

Worst Case Time Complexity: O(N/K+K)
Best Case Time Complexity: Ω(1)
Space Complexity: Θ(1)

```c
#include<stdio.h>

int index_search(int arr[],int n,int key)
{
   int m=0,start,end,flag=0;
   int index[n/3],indexEle[n/3];

   for(int i=0;i<n;i+3)
  {
      indexEle[m]=arr[i];
      index[m]=i;
      m++;
   }

   if(key<indexEle[0])
      return -1;

   else
  {
      for(int i=1;i<m;i++)
       {
         if(key<indexEle[i])
        {
           start=index[i=1];
```

```
            end=index[i];
            flag=1;
            break;
          }

          if(flag==0)
          {
            start=index[i-1];
            end=n-1;
          }
        }
    }

    for(int i=start;i<end;i++)
   {
       if(arr[i]==key)
       return i;
    }

    return -1;
}
int main()
{   printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
    int n;
    printf("Enter the size of array:");
    scanf("%d",&n);
    int arr[n];
    printf("Enter the elements of array:");
    for(int i=0;i<n;i++)
    scanf("%d",&arr[i]);
    int key;
    printf("Enter the element to be search:");
    scanf("%d",&key);

    int ans=index_search(arr,n,key);
    if(ans==-1)
    printf("Element not found!!");
    else
    printf("Element is present at %d place.", ans+1);
    return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of array: 6
Enter the elements of array :6 4 3 7 8 1
Enter the element to be search:3
Elements is present at 3 place.
```

PROGRAM 18 - Program for Radix Sort

ALGORITHM: RadixSort(A[],N,d)
BEGIN:

```
        FOR i=1 TO d DO
                Apply counting Sort on A[] at radix i
```

END;

Time Complexity: Θ(N)
Space Complexity:Θ(N)

```
#include <stdio.h>

int getMax(int a[], int n) {
  int max = a[0];
  for(int i = 1; i<n; i++) {
    if(a[i] > max)
      max = a[i];
  }
  return max;
}

void countingSort(int a[], int n, int place)
{
 int output[n + 1];
 int count[10] = {0};

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

 for (int i = 1; i < 10; i++)
   count[i] += count[i - 1];
   for (int i = n - 1; i >= 0; i--) {

   count[(a[i] / place) % 10]--;
 }

 for (int i = 0; i < n; i++)
```

```
    a[i] = output[i];
}
void radixsort(int a[], int n) {

  int max = getMax(a, n);

  for (int place = 1; max / place > 0; place *= 10)
    countingSort(a, n, place);
}

void printArray(int a[], int n) {
printf(“Sorted array”);
  for (int i = 0; i < n; ++i) {
    printf("%d  ", a[i]);
  }
  printf("\n");
}

int main() {
  int a[] = {181, 289, 390, 121, 145, 736, 514, 888, 122};
  int n = sizeof(a) / sizeof(a[0]);
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
  radixsort(a, n);
  printArray(a, n);
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the size of array 6
Enter the elements of array 5 4 7 8 2 1
Sorted array 1 2 4 5 7 8
```

PROGRAM 17 - Program for Counting Sort

ALGORITHM: CountingSort(A[],k,n)

BEGIN:

```
    FOR i = 0 TO k DO
        c[i] = 0
    FOR j = 0 TO n DO
        c[A[j]] = c[A[j]] + 1
    FOR i = 1 TO k DO
        c[i] = c[i] + c[i-1]
    FOR j = n-1 TO 0 STEP-1 DO
        B[ c[A[j]]-1 ] = A[j]
        c[A[j]] = c[A[j]] - 1
    RETURN B
```

END;

Time Complexity: Omega(N)

Space Complexity:Θ(N)

```c
#include <stdio.h>

void countingSort(int array[], int size) {
  int output[10];

  int max = array[0];
  for (int i = 1; i < size; i++) {
    if (array[i] > max)
      max = array[i];
  }
  int count[10];

  for (int i = 0; i <= max; ++i) {
    count[i] = 0;
  }

  for (int i = 0; i < size; i++) {
    count[array[i]]++;
```

```
  }

  for (int i = 1; i <= max; i++) {
    count[i] += count[i - 1];
  }

  for (int i = size - 1; i >= 0; i--) {
    output[count[array[i]] - 1] = array[i];
    count[array[i]]--;
  }

  for (int i = 0; i < size; i++) {
    array[i] = output[i];
  }
}

void printArray(int array[], int size) {
printf("Sorted array ");
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}
int main() {
int n;
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
  printf("Enter the size of array ");
  scanf("%d",&n);
  int array[n];
  printf("Enter the elements of array ");
  for (int i = 0; i < n; i++)
  {
    scanf("%d",&array[i]);
  }
countingSort(array, n);
  printArray(array, n);
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter the size of array 6
Enter the elements of array 5 4 7 8 2 1
Sorted array 1 2 4 5 7 8
```

PROGRAM 7B - Program For Selection sort

ALGORITHM: SelectionSort(A[], N)
BEGIN:

```
FOR i=1 TO N-1 DO
        min=i
        FOR  j=i+1 TO N DO
                IF A[j]<A[min] THEN
                min=j
        Exchange(A[min], A[i])
```

END;
Time Complexity: $\Theta(N^2)$
Space Complexity:$\Theta(1)$

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

{
   int n;
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");

   printf("Enter the size of array-");
   scanf("%d",&n);
   int arr[n];
   printf("Enter the array:");
   for (int i=0;i<n;i++){
      scanf("%d",&arr[i]);
   }

   for (int i=0;i<n-1;i++)
   {
      for(int j=i+1;j<n;j++)
    {  if(arr[j]<arr[i])
      {
         int temp=arr[j];
         arr[j]=arr[i];
         arr[i]=temp;
      }
    }
   }
   printf("Sorted array is :");
   for(int i=0;i<n;i++)
   {
      printf("%d ",arr[i]);
   }
```

return 0 ;}

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of array-6
Enter the array:7 4 2 9 1 0
Sorted array is :0 1 2 4 7 9
```

PROGRAM 9 - Program for Quick sort

```
ALGORITHM: QuickSort(A[],low,high)
BEGIN:
        IF low<high THEN
                j=Partition(A[],low,high)
                QuickSort(A[],low,j-1)
                QuickSort(A[],j+1,high)
END;

ALGORITHM: Partition(A[],low,high)
BEGIN:
        i=low, j=high+1,pivot=A[low]
        DO
                DO
                        i=i+1
                WHILE(A[i]<pivot)
                DO
                        J=j-1
                WHILE(A[j]>pivot)

                IF i<j THEN
                        Exchange(A[i],A[j])
        WHILE(i<j)

          Exchange(A[j],A[low])
         RETURN j
END;
```

Worst Case Time Complexity: $O(N^2)$
Best Case Time Complexity: $\Omega(N\log_2 N)$
Space Complexity: $\Theta(\log_2 N)$

```
#include<stdio.h>
 void swap(int arr[],int i,int j){
   int temp=arr[i];
   arr[i]=arr[j];
   arr[j]=temp;
}

 int partition(int arr[],int l,int r){
   int pivot= arr[r];
   int i=l-1;
   for(int j=l;j<r;j++){
      if(arr[j]<pivot)
      {
```

```
            i++;
            swap(arr,i,j);
        }
    }
    swap(arr,i+1,r);
    return i+1;
}

void quickSort(int arr[],int l,int r){
    if(l<r){
        int pi=partition(arr,l,r);
        quickSort(arr,l,pi-1);
        quickSort(arr,pi+1,r);
          }
}
int main()

{
    int n;
    printf("Enter size of array:");
    scanf("%d",&n);
    int arr[n];
    printf("Enter array elements:");
    for(int i=0;i<n;i++)
        scanf("%d",&arr[i]);

    quickSort(arr,0,n-1);
    printf("Sorted array is: ");
    for(int i=0;i<n;i++)
        printf("%d ",arr[i]);
    return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter size of array:6
Enter array elements:8 7 6 5 4 3
Sorted array is: 3 4 5 6 7 8
```

PROGRAM 10 - Program for Merge sort

ALGORITHM: MergeSort(A[],low,high)
BEGIN:

```
    IF low<high DO
        Mid=(low+high)/2
        MergeSort(A[],low,mid)
        MergeSort(A[],mid+1, high)
        Merge(A, low,mid,high)
```

END;

ALGORITHM: Merge(A[], low,mid,high)
BEGIN:

```
    i=low,j=mid+1,k=high
    WHILE i<=mid AND j<=high DO
        IF A[i]<A[j] THEN
            C[k]=A[i]
            i=i+1
            k=k+1
        ELSE
            C[k]=A[j]
            j=j+1
            k=k+1
    WHILE i<=mid DO
        C[k]=A[i]
        i=i+1
        k=k+1
    WHILE j<=high DO
        C[k]=A[j]
        J=j+1
       k=k+1
    FOR i=low TO high DO
        A[i]=C[i]
```

END;

Time Complexity: $O(N\log_2 N)$
Space Complexity: $\Theta(N)$

```
#include<stdio.h>

void merge (int arr[],int l,int mid,int r)
{
   int n1=mid-l+1;
   int n2=r-mid;

   int a[n1];
   int b[n2];
```

```
    for (int i=0;i<n1;i++)
        a[i]=arr[l+i];

     for (int i=0;i<n2;i++)
        b[i]=arr[mid+1+i];

     int i=0;
     int j=0;
     int k=l;

     while(i<n1 && j<n2)
     {
        if(a[i]<b[j])
        {
            arr[k]=a[i];
            k++;
            i++;
        }
        else
        {
           arr[k]=b[j];
           k++;
           j++;
        }
     }
     while(i<n1){
        arr[k]=a[i];
            k++;
            i++;
     }

     while(j<n2)
     {
         arr[k]=b[j];
           k++;
           j++;
     }
}

void mergeSort(int arr[],int l,int r)
{
    if(l<r){
```

```
        int mid=(l+r)/2;
        mergeSort(arr,l,mid);
        mergeSort(arr,mid+1,r);

        merge(arr,l,mid,r);
    }
}

int main()
{   int n;
    printf("Enter size of array:");
    scanf("%d",&n);
    int arr[n];
    printf("Enter array elements:");
     for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }

    mergeSort(arr,0,n-1);//l=0  r=n-1
    printf("Sorted array is:");
    for(int i=0;i<n;i++)
        printf("%d ",arr[i]);

    return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter size of array:6
Enter array elements:8 7 6 5 4 3
Sorted array is: 3 4 5 6 7 8
```

PROGRAM 7C - Program for Insertion sort

ALGORITHM: InsertionSort(A[], N)
BEGIN:

```
FOR i=2 TO N DO
    key=A[i]
    j=i-1
    WHILE j>=1  AND A[j]>key DO
        A[j+1]=A[j]
        j=j-1
    A[j+1]=key
```

END;

Worst Case Time Complexity:O(N^2)
Best Case Time Complexity: Omega(N)
Space Complexity:Θ(1)

```c
#include<stdio.h>

int main()

{
   int n;
   printf("Enter size of array:");
   scanf("%d",&n);
   int arr[n];
   printf("Enter array elements:");
    for(int i=0;i<n;i++)
   {
      scanf("%d",&arr[i]);
   }
    for(int i=1;i<n;i++)
   {
      int current=arr[i];
      int j=i-1;

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

      }
    arr[j+1]=current;
```

```
    }

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

    }

    return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004
Enter size of array:6
Enter array elements:8 7 6 5 4 3
Sorted array is: 3 4 5 6 7 8
```

PROGRAM 7A - Program for Bubble sort

ALGORITHM: BubbleSort(A[], N)
BEGIN:

```
        FOR i=1 TO N-1 DO
                FOR j=1 TO N-i DO
                        IF A[j]>A[j+1]
                                k=A[j]
                                A[j]=A[j+1]
                                A[j+1]=k
```

END;

Worst Case Time Complexity:O(N^2)
Best Case Time Complexity: Omega(N)
Space Complexity:Θ(1)

```
#include<stdio.h>

int main()

{
  int n;
printf("This code is written by ABHAY PANDEY //CS-A//2100320120004 \n");
  printf("Enter the size of array-");
  scanf("%d",&n);
  int arr[n];
  printf("Enter the array:");
  for(int i=0;i<n;i++){
    scanf("%d",&arr[i]);

  }
  int count=1;
  while(count<n){
    for(int i=0;i<n-count;i++){
      if(arr[i]>arr[i+1])
      {int temp=arr[i];
      arr[i]=arr[i+1];
      arr[i+1]=temp;}
    }
    count++;
  }
  printf("Sorted array is :");
  for(int i=0;i<n;i++){
```

```
        printf("%d ",arr[i]);
    }
    return 0 ;
}
```

Output:

```
This code is written by ABHAY PANDEY //CS-A//2100320120004

Enter the size of array-6
Enter the array:7 4 2 9 1 0
Sorted array is :0 1 2 4 7 9
```

LAB-12

// Program for Stack Primitive Operations

```
ALGORITHM          Initialize stack(Stack S)
Begin:
        S.TOP=-1
End;
```

Time Complexity- θ(1)

Space Complexity- θ(1)

```
ALGORITHM          Push(Stack S,key)
Begin:
      IF S.TOP==SIZE-1 THEN
          WRITE("Stack Overflows")
          EXIT (1)
      S.TOP++
      S.ITEM[S.TOP]=key
End;
```

Time Complexity- θ(N), Space Complexity- θ(1)

```
ALGORITHM          Empty (Stack S)
Begin:
      IF S.TOP==-1 THEN
          RETURN TRUE
      ELSE
          RETURN FALSE
End;
```

Time Complexity- θ(1) , Space Complexity- θ(1)

```
ALGORITHM          Pop  (Stack S,key)
 Begin:
      IF EMPTY(S) THEN
```

WRITE (“Stack underflows”)

EXIT(1)

X=S.ITEM[S.TOP]

S.TOP- -

RETURN X

End;

Time Complexity- θ(1), Space Complexity- θ(1)

ALGORITHM STACKTOP (Stack S)

Begin:

RETURN (S.ITEM[S.TOP])

End;

Time Complexity- θ(1), Space Complexity- θ(1)

```
#include <stdio.h>
int stack[100],i,j,choice=0,n,top=-1;
void push();
void pop();
void show();
int main ()
{
    printf("ABHAY PANDEY  2100320120004\n");
   printf("Enter the number of elements in the stack ");
   scanf("%d",&n);
   printf("*********Stack operations using array*********");

printf("\n----------------------------------------------\n");
   while(choice != 4)
   {
      printf("Chose one from the below options...\n");
      printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
```

```
        printf("\n Enter your choice \n");
        scanf("%d",&choice);
        switch(choice)
        {
          case 1:
          {
            push();
            break;
          }
          case 2:
          {
            pop();
            break;
          }
          case 3:
          {
            show();
            break;
          }
          case 4:
          {
            printf("Exiting....");
            break;
          }
          default:
          {
            printf("Please Enter valid choice ");
          }
        };
    }
                        return 0;
}
```

```
void push ()
{
   int val;
   if (top == n )
   printf("\n Overflow");
   else
   {
      printf("Enter the value?");
      scanf("%d",&val);
      top = top +1;
      stack[top] = val;
   }
}

void pop ()
{
   if(top == -1)
   printf("Underflow");
   else
   top = top -1;
}
void show()
{
   for (i=top;i>=0;i--)
   {
      printf("%d\n",stack[i]);
   }
   if(top == -1)
   {
      printf("Stack is empty");
   }
```

}

OUTPUT:

ABHAY PANDEY 2100320120004

Enter the number of elements in the stack 3

*********Stack operations using array*********

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

1

Enter the value?3

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

1

Enter the value?4

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

1

Enter the value?5

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

2

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

3

4

3

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

3

4

3

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

3

4

3

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

1

Enter the value?6

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

3

6

4

3

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

4

Exiting....

```c
// Program for Decimal to Any Base Conversion

#include<stdio.h>

#include<stdlib.h>

int main(){

printf("ABHAY PANDEY  2100320120004\n");

int a[10],n,i;

system ("cls");

printf("Enter the number to convert: ");

scanf("%d",&n);

for(i=0;n>0;i++)

{

a[i]=n%2;

n=n/2;

}

printf("\nBinary of Given Number is=");

for(i=i-1;i>=0;i--)

{

printf("%d",a[i]);

}

return 0;

}
```

OUTPUT:

```
Enter the number to convert: 34


Binary of Given Number is=100010
```

```c
// Program to check the validity of Parenthesized Arithmetic Expression using Stack
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

// function prototypes
void push(char);
void pop();
void find_top();

int main()
{
            printf("ABHAY PANDEY  2100320120004\n");
            int i;
            char a[100];
            printf("enter expression\n");
            scanf("%s", &a);
            for (i = 0; a[i] != '\0';i++)
            {
                if (a[i] == '(')
                {
                    push(a[i]);
                }
                else if (a[i] == ')')
                {
                    pop();
                }
            }
            find_top();
```

```
        return 0;
}

// to push elements in stack
void push(char a)
{
        stack[top] = a;
        top++;
}

// to pop elements from stack
void pop()
{
        if (top == -1)
        {
            printf("expression is invalid\n");
            exit(0);
        }
        else
        {
            top--;
        }
}

// to find top element of stack
void find_top()
{
        if (top == -1)
            printf("\nexpression is valid\n");
        else
            printf("\nexpression is invalid\n");
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

enter expression

hello world

expression is valid

LAB - 13

// Program to check the validity of Bracketed Arithmetic Expression using Stack

```
#include <stdio.h>

#include <stdlib.h>

#define bool int


// structure of a stack node

struct sNode {

	char data;

	struct sNode* next;

};


// Function to push an item to stack

void push(struct sNode** top_ref, int new_data);


// Function to pop an item from stack

int pop(struct sNode** top_ref);


// Returns 1 if character1 and character2 are matching left

// and right Brackets

bool isMatchingPair(char character1, char character2)

{

	if (character1 == '(' && character2 == ')')

		return 1;

	else if (character1 == '{' && character2 == '}')

		return 1;

	else if (character1 == '[' && character2 == ']')

		return 1;

	else

		return 0;

}
```

```
// Return 1 if expression has balanced Brackets
bool areBracketsBalanced(char exp[])
{
	int i = 0;

	// Declare an empty character stack
	struct sNode* stack = NULL;

	// Traverse the given expression to check matching
	// brackets
	while (exp[i])
	{
		// If the exp[i] is a starting bracket then push
		// it
		if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[')
			push(&stack, exp[i]);

		// If exp[i] is an ending bracket then pop from
		// stack and check if the popped bracket is a
		// matching pair*/
		if (exp[i] == '}' || exp[i] == ')'
			|| exp[i] == ']') {

			// If we see an ending bracket without a pair
			// then return false
			if (stack == NULL)
				return 0;

			// Pop the top element from stack, if it is not
			// a pair bracket of character then there is a
			// mismatch.
			// his happens for expressions like {(})
```

```
                    else if (!isMatchingPair(pop(&stack), exp[i]))
                        return 0;
                }
                i++;
        }

        // If there is something left in expression then there
        // is a starting bracket without a closing
        // bracket
        if (stack == NULL)
                return 1; // balanced
        else
                return 0; // not balanced
}

// Driver code
int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
        char exp[100] = "{()}[]";

        // Function call
        if (areBracketsBalanced(exp))
                printf("Balanced");
        else
                printf("Not Balanced");
        return 0;
}

// Function to push an item to stack
void push(struct sNode** top_ref, int new_data)
{
```

```
	// allocate node
	struct sNode* new_node
		= (struct sNode*)malloc(sizeof(struct sNode));

	if (new_node == NULL) {
		printf("Stack overflow n");
		getchar();
		exit(0);
	}

	// put in the data
	new_node->data = new_data;

	// link the old list off the new node
	new_node->next = (*top_ref);

	// move the head to point to the new node
	(*top_ref) = new_node;
}

// Function to pop an item from stack
int pop(struct sNode** top_ref)
{
	char res;
	struct sNode* top;

	// If stack is empty then error
	if (*top_ref == NULL) {
		printf("Stack overflow n");
		getchar();
		exit(0);
	}
```

```
        else {
                top = *top_ref;
                res = top->data;
                *top_ref = top->next;
                free(top);
                return res;
        }
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Balanced

```
// Program to check if the given number is a palindrome using stacks
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 50

int top = -1, front = 0;
int stack[MAX];
void push(char);
void pop();

int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
  int i, choice;
  char s[MAX], b;
  while (1)
  {
    printf("1-enter string\n2-exit\n");
    printf("enter your choice\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the String\n");
      scanf("%s", s);
      for (i = 0;s[i] != '\0';i++)
      {
        b = s[i];
        push(b);
      }
      for (i = 0;i < (strlen(s) / 2);i++)
```

```
        {
          if (stack[top] == stack[front])
          {
            pop();
            front++;
          }
          else
          {
            printf("%s is not a palindrome\n", s);
            break;
          }
        }
        if ((strlen(s) / 2)  ==  front)
          printf("%s is palindrome\n",  s);
        front  =  0;
        top  =  -1;
        break;
      case 2:
        exit(0);
      default:
        printf("enter correct choice\n");
      }
    }
    return 0;
}

/* to push a character into stack */
void push(char a)
{
    top++;
    stack[top]  =  a;
}
```

```
/* to delete an element in stack */
void pop()
{
   top--;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

1-enter string

2-exit

enter your choice

1

Enter the String

hellel

hellel is not a palindrome

1-enter string

2-exit

enter your choice

1

Enter the String

mom

mom is palindrome

1-enter string

2-exit

enter your choice

2

```c
// Program to Reverse the given String using Stack
#include <stdio.h>
#include <string.h>

#define max 100
int top,stack[max];

void push(char x){

      // Push(Inserting Element in stack) operation
      if(top == max-1){
          printf("stack overflow");
      }  else {
          stack[++top]=x;
      }

}

void pop(){
    // Pop (Removing element from stack)
      printf("%c",stack[top--]);
}



int main()
{
    printf("ABHAY PANDEY  2100320120004\n");
   char str[]="sri lanka";
   int len = strlen(str);
   int i;

   for(i=0;i<len;i++)
```

```
        push(str[i]);


    for(i=0;i<len;i++)
      pop();
      return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

aknal irs

LAB – 14

```
// Program for Postfix Evaluation
#include <stdio.h>
#include <ctype.h>

#define MAXSTACK 100 /* for max size of stack */
#define POSTFIXSIZE 100 /* define max number of charcters in postfix expression */

/* declare stack and its top pointer to be used during postfix expression
evaluation*/
int stack[MAXSTACK];
int top = -1; /* because array index in C begins at 0 */
/* can be do this initialization somewhere else */

/* define push operation */
void push(int item)
{

    if (top >= MAXSTACK - 1) {
        printf("stack over flow");
        return;
    }
    else {
        top = top + 1;
        stack[top] = item;
    }
}

/* define pop operation */
int pop()
```

```
{
    int item;
    if (top < 0) {
        printf("stack under flow");
    }
    else {
        item = stack[top];
        top = top - 1;
        return item;
    }
}

/* define function that is used to input postfix expression and to evaluate it */
void EvalPostfix(char postfix[])
{

    int i;
    char ch;
    int val;
    int A, B;

    /* evaluate postfix expression */
    for (i = 0; postfix[i] != ')'; i++) {
        ch = postfix[i];
        if (isdigit(ch)) {
            /* we saw an operand,push the digit onto stack
ch - '0' is used for getting digit rather than ASCII code of digit */
            push(ch - '0');
        }
        else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
```

```
        /* we saw an operator
* pop top element A and next-to-top elemnet B
* from stack and compute B operator A
*/
        A = pop();
        B = pop();

        switch (ch) /* ch is an operator */
        {
        case '*':
           val = B * A;
           break;

        case '/':
           val = B / A;
           break;

        case '+':
           val = B + A;
           break;

        case '-':
           val = B - A;
           break;
        }

        /* push the value obtained above onto the stack */
        push(val);
     }
  }
```

```
    printf(" \n Result of expression evaluation : %d \n", pop());
}

int main()
{
    printf("ABHAY PANDEY  2100320120004\n");
    int i;

    /* declare character array to store postfix expression */
    char postfix[POSTFIXSIZE];
    printf("ASSUMPTION: There are only four operators(*, /, +, -) in an expression and operand is single digit only.\n");
    printf(" \nEnter postfix expression,\npress right parenthesis ')' for end expression : ");

    /* take input of postfix expression from user */

    for (i = 0; i <= POSTFIXSIZE - 1; i++) {
        scanf("%c", &postfix[i]);

        if (postfix[i] == ')') /* is there any way to eliminate this if */
        {
            break;
        } /* and break statement */
    }

    /* call function to evaluate postfix expression */

    EvalPostfix(postfix);

    return 0;
```

}

OUTPUT:

ABHAY PANDEY 2100320120004

ASSUMPTION: There are only four operators(*, /, +, -) in an expression and operand is single digit only.

Enter postfix expression,

press right parenthesis ')' for end expression : 123*4+56(1+3)

Result of expression evaluation : 3

```
//Program for Prefix Evaluation
#include<stdio.h>
#include<conio.h>
#include<string.h>
int stk[10];
int top=-1;
void push(int);
int pop();
int main()
{
        printf("ABHAY PANDEY 2100320120004\n");

char prefix[10];
int len,val,i,opr1,opr2,res;

printf("Enter the prefix Expression :");
gets(prefix);
len=strlen(prefix);
for(i=len-1;i>=0;i--)
{
switch(get_type(prefix[i]))
{
case 0:
val=prefix[i]-'0';
push(val);
break;

case 1: opr1=pop();
opr2=pop();
switch(prefix[i])
{
case '+': res=opr1+opr2;
```

```
break;
case '-': res=opr1-opr2;
break;
case '*': res=opr1*opr2;
break;
case '/': res=opr1/opr2;
break;
}
push(res);

}
}
printf("Result is %d",stk[0]);
getch();
return 0;
}

void push(int val)
{
stk[++top]=val;
}
int pop()
{
return(stk[top--]);
}
int get_type(char c)
{
if(c=='+'||c=='-'||c=='*'||c=='/')
return 1;
else
return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Input : -+8/632

Output : 8

```
// Program for Infix to Postfix Coversion
#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()
{
    printf("ABHAY PANDEY  2100320120004\n");
```

```
    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;
}
```

OUTPUT:

Enter the expression : a+b*c

a b c * +

```
// Program for Infix to Prefix Coversion
#include <stdio.h>

void selection(int arr[], int n)
{
    int i, j, small;

    for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
    {
        small = i; //minimum element in unsorted array

        for (j = i+1; j < n; j++)
        if (arr[j] < arr[small])
            small = j;
// Swap the minimum element with the first element
    int temp = arr[small];
    arr[small] = arr[i];
    arr[i] = temp;
    }
}

void printArr(int a[], int n) /* function to print the array */
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}

int main()
{
    printf("ABHAY PANDEY  2100320120004\n");
    int a[] = { 12, 31, 25, 8, 32, 17 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
```

```
    printArr(a, n);
    selection(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

*+ABC

LAB – 15

// Program for implementation of 2 stacks using a single Array

```
#include <stdio.h>
#define SIZE 20
 int array[SIZE];  // declaration of array type variable.
int top1 = -1;
int top2 = SIZE;

//Function to push data into stack1
void push1 (int data)
{
// checking the overflow condition
  if (top1 < top2 - 1)
  {
      top1++;
    array[top1] = data;
  }
  else
  {
    printf ("Stack is full");
  }
}
// Function to push data into stack2
void push2 (int data)
{
// checking overflow condition
if (top1 < top2 - 1)
  {
    top2--;
    array[top2] = data;
  }
  else
```

```
  {
    printf ("Stack is full..\n");
  }
}

//Function to pop data from the Stack1
void pop1 ()
{
// Checking the underflow condition
 if (top1 >= 0)
  {
    int popped_element = array[top1];
    top1--;

    printf ("%d is being popped from Stack 1\n", popped_element);
  }
  else
  {
    printf ("Stack is Empty \n");
  }
}
// Function to remove the element from the Stack2
void pop2 ()
{
// Checking underflow condition
if (top2 < SIZE)
  {
      int popped_element = array[top2];
    top2--;

    printf ("%d is being popped from Stack 1\n", popped_element);
  }
```

```
  else
  {
    printf ("Stack is Empty!\n");
  }
}

//Functions to Print the values of Stack1
void display_stack1 ()
{
  int i;
  for (i = top1; i >= 0; --i)
  {
    printf ("%d ", array[i]);
  }
  printf ("\n");
}
// Function to print the values of Stack2
void display_stack2 ()
{
  int i;
  for (i = top2; i < SIZE; ++i)
  {
    printf ("%d ", array[i]);
  }
  printf ("\n");
}

int main()
{
  printf("ABHAY PANDEY  2100320120004\n");
  int ar[SIZE];
  int i;
```

```
  int num_of_ele;

  printf ("We can push a total of 20 values\n");

  //Number of elements pushed in stack 1 is 10
  //Number of elements pushed in stack 2 is 10

// loop to insert the elements into Stack1
for (i = 1; i <= 10; ++i)
  {
    push1(i);
    printf ("Value Pushed in Stack 1 is %d\n", i);
  }
// loop to insert the elements into Stack2.
for (i = 11; i <= 20; ++i)
  {
    push2(i);
    printf ("Value Pushed in Stack 2 is %d\n", i);
  }

  //Print Both Stacks
  display_stack1 ();
 display_stack2 ();

  //Pushing on Stack Full
  printf ("Pushing Value in Stack 1 is %d\n", 11);
  push1 (11);

  //Popping All Elements from Stack 1
  num_of_ele = top1 + 1;
  while (num_of_ele)
  {
```

```
    pop1 ();
    --num_of_ele;
  }

  // Trying to Pop the element From the Empty Stack
  pop1 ();

  return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

We can push a total of 20 values

Value Pushed in Stack 1 is 1

Value Pushed in Stack 1 is 2

Value Pushed in Stack 1 is 3

Value Pushed in Stack 1 is 4

Value Pushed in Stack 1 is 5

Value Pushed in Stack 1 is 6

Value Pushed in Stack 1 is 7

Value Pushed in Stack 1 is 8

Value Pushed in Stack 1 is 9

Value Pushed in Stack 1 is 10

Value Pushed in Stack 2 is 11

Value Pushed in Stack 2 is 12

Value Pushed in Stack 2 is 13

Value Pushed in Stack 2 is 14

Value Pushed in Stack 2 is 15

Value Pushed in Stack 2 is 16

Value Pushed in Stack 2 is 17

Value Pushed in Stack 2 is 18

Value Pushed in Stack 2 is 19

Value Pushed in Stack 2 is 20

10 9 8 7 6 5 4 3 2 1

20 19 18 17 16 15 14 13 12 11

Pushing Value in Stack 1 is 11

Stack is full10 is being popped from Stack 1

9 is being popped from Stack 1

8 is being popped from Stack 1

7 is being popped from Stack 1

6 is being popped from Stack 1

5 is being popped from Stack 1

4 is being popped from Stack 1

3 is being popped from Stack 1

2 is being popped from Stack 1

1 is being popped from Stack 1

Stack is Empty

```c
// Program for Finding Minimum in the Stack
#include <stdio.h>
int main()
{
            printf("ABHAY PANDEY  2100320120004\n");
   int q;
   scanf("%d",&q);
   int stack[q],stackmin[q];
   int top=-1,topmin=-1;
   while(q--)
   {
      int x;scanf("%d",&x);
      if(x==1)
      {
         int y;scanf("%d",&y);
         stack[++top]=y;
         if(topmin==-1)
          stackmin[++topmin]=y;
         else if(y<=stackmin[topmin])
            stackmin[++topmin]=y;
      }
      else if(x==2)
      {
         if(top==-1)
            printf("-1\n");
         else
         {
            if(stack[top]==stackmin[topmin])
               topmin--;
            //printf("%d\n",stack[top]);
            top--;}
      }
```

```
        else if(x==3)
        {
            if(top==-1)
                printf("-1\n");
            else
                printf("%d\n",stack[top]);}
            else
            {
                if(top==-1)
                    printf("-1\n");
                else
                    printf("%d\n",stackmin[topmin]);}
    }
    return 0;
 }
```

OUTPUT:

ABHAY PANDEY 2100320120004

3

-1

1

2

5

2

3

2

```
// Program for Sorting of stack

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

// Stack is represented using linked list
struct stack {
	int data;
	struct stack* next;
};

// Utility function to initialize stack
void initStack(struct stack** s) { *s = NULL; }

// Utility function to check if stack is empty
int isEmpty(struct stack* s)
{
	if (s == NULL)
		return 1;
	return 0;
}

// Utility function to push an item to stack
void push(struct stack** s, int x)
{
	struct stack* p = (struct stack*)malloc(sizeof(*p));

	if (p == NULL) {
		fprintf(stderr, "Memory allocation failed.\n");
		return;
	}
```

```
    p->data = x;
    p->next = *s;
    *s = p;
}

// Utility function to remove an item from stack
int pop(struct stack** s)
{
    int x;
    struct stack* temp;

    x = (*s)->data;
    temp = *s;
    (*s) = (*s)->next;
    free(temp);

    return x;
}

// Function to find top item
int top(struct stack* s) { return (s->data); }

// Recursive function to insert an item x in sorted way
void sortedInsert(struct stack** s, int x)
{
    // Base case: Either stack is empty or newly inserted
    // item is greater than top (more than all existing)
    if (isEmpty(*s) || x > top(*s)) {
        push(s, x);
        return;
    }
```

```
	// If top is greater, remove the top item and recur
	int temp = pop(s);
	sortedInsert(s, x);

	// Put back the top item removed earlier
	push(s, temp);
}

// Function to sort stack
void sortStack(struct stack** s)
{
	// If stack is not empty
	if (!isEmpty(*s)) {
		// Remove the top item
		int x = pop(s);

		// Sort remaining stack
		sortStack(s);

		// Push the top item back in sorted stack
		sortedInsert(s, x);
	}
}

// Utility function to print contents of stack
void printStack(struct stack* s)
{
	while (s) {
		printf("%d ", s->data);
		s = s->next;
	}
	printf("\n");
```

```
}

// Driver code
int main(void)
{
	printf("ABHAY PANDEY  2100320120004\n");
	struct stack* top;

	initStack(&top);
	push(&top, 30);
	push(&top, -5);
	push(&top, 18);
	push(&top, 14);
	push(&top, -3);

	printf("Stack elements before sorting:\n");
	printStack(top);

	sortStack(&top);
	printf("\n\n");

	printf("Stack elements after sorting:\n");
	printStack(top);

	return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
Stack elements before sorting:
-3 14 18 -5 30
```

Stack elements after sorting:

30 18 14 -3 -5

```
// Program for implementation of Multiple stack in one Array
#include<stdio.h>
#include<conio.h>
#define MAX_X 5
#define MAX_Y 5
int topx=-1;
int topy=10;

/*Begin of push_x*/
void push_x(int *stack)
{
	int info;

	if(topx>=(MAX_X-1))
	{	printf("\n\nStack OverFlow");
		return;
	}
	else
	{	printf("\n\nEnter The info To Push");
		scanf("%d",&info);
		topx++;
		stack[topx]=info;
	}}
/*End of push_x*/
/*Begin of push_y*/
void push_y(int *stack)
{
	int info;

	if(topy<=(MAX_Y))
	{
		printf("\n\nStack OverFlow");
```

```
                return;
        }
        else
        {
                printf("\n\nEnter The info To Push");
                scanf("%d",&info);
                topy--;
                stack[topy]=info;
        }
}
/*End of push_y*/
/*Begin of pop_x*/
void pop_x(int *stack)
{       if(topx==-1)
        {
                printf("Stack X is Underflow");
                return;
        }
        else
        {
          printf("Item Poped from stack X is:%d\n",stack[topx]);

          topx--;
        }
}
/*End of pop_x*/
/*Begin of pop_y*/
void pop_y(int *stack)
{       if(topy==10)
        {printf("Stack y is Underflow");
        return;
        }
```

```
            else
            { printf("Item Poped from stack Y is:%d\n",stack[topy]);
              topy++;
            }}
/*End of pop_y*/
/*Begin of display_x*/
void display_x(int *stack)
{
            int i;
            if(topx==-1)
            {
                        printf("Stack X is Empty");
                        return;
            }
            else
            { for(i=topx;i>=0;i--)
               {printf("%d,",stack[i]);}
               printf("\n");
            }}
/*End of display_x*/
/*Begin of display_y*/
void display_y(int *stack)
{
            int i;
            if(topy==10)
            {printf("Stack Y is Empty");
                        return;}
            else
            {for(i=topy;i<=9;i++)
               {
                        printf("%d,",stack[i]);
               }
```

```
            printf("\n");
        }       }
/*End of display_y*/
/*Begin of main*/
int main()
{       int choice;
        char ch;
        int stack[MAX_X+MAX_Y];
do
        {   printf("1.Push_X\n2.Push_Y\n");
                printf("\n3.Pop_X\n4.Pop_Y\n");
                printf("\n5.Display_X\n6.Display_Y\n");
                printf("\n7.Exit");
                printf("\n\nEnter Choice");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: push_x(stack);break;
                        case 2: push_y(stack);break;

                        case 3: pop_x(stack);break;
                        case 4: pop_y(stack);break;

                        case 5: display_x(stack);break;
                        case 6: display_y(stack);break;
                        case 7: break;
                        default: printf("Wrong Option...");
                }
        }while(choice!=7);
        return 0;
}
```

OUTPUT:

1.Push_X

2.Push_Y

3.Pop_X

4.Pop_Y

5.Display_X

6.Display_Y

7.Exit

Enter Choice1

Enter The info To Push2

1.Push_X

2.Push_Y

3.Pop_X

4.Pop_Y

5.Display_X

6.Display_Y

7.Exit

Enter Choice1

Enter The info To Push3

1.Push_X

2.Push_Y

3.Pop_X

4.Pop_Y

5.Display_X

6.Display_Y

7.Exit

Enter Choice5

3,2,

1.Push_X

2.Push_Y

3.Pop_X

4.Pop_Y

5.Display_X

6.Display_Y

7.Exit

Enter Choice5

3,2,

1.Push_X

2.Push_Y

3.Pop_X

4.Pop_Y

5.Display_X

6.Display_Y

7.Exit

Enter Choice7

LAB – 16

// Program of Array Implementaion of Linear Queue

```
#include<stdio.h>
#define n 5
int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
   int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
   printf("Queue using Array");
   printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");
   while(ch)
   {
      printf("\nEnter the Choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         if(rear==x)
            printf("\n Queue is Full");
         else
         {
            printf("\n Enter no %d:",j++);
            scanf("%d",&queue[rear++]);
         }
         break;
      case 2:
         if(front==rear)
         {
            printf("\n Queue is empty");
         }
         else
         {
```

```
            printf("\n Deleted Element is %d",queue[front++]);
            x++;
          }
          break;
        case 3:
          printf("\nQueue Elements are:\n ");
          if(front==rear)
            printf("\n Queue is Empty");
          else
          {
            for(i=front; i<rear; i++)
            {
              printf("%d",queue[i]);
              printf("\n");
            }
            break;
          case 4:
            return 0;
          default:
            printf("Wrong Choice: please see the options");
          }
        }
      }
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Queue using Array

1.Insertion

2.Deletion

3.Display

4.Exit

Enter the Choice:1

Enter no 1:2

Enter the Choice:1

Enter no 2:3

Enter the Choice:3

Queue Elements are:

2

3

Enter the Choice:2

Deleted Element is 2

Enter the Choice:2

Deleted Element is 3

```c
// Program of Array Implementaion of CircularQueue
# include<stdio.h>
# define MAX 5

int cqueue_arr[MAX];
int front = -1;
int rear = -1;

/*Begin of insert*/
void insert(int item)
{
	if((front == 0 && rear == MAX-1) || (front == rear+1))
	{
		printf("Queue Overflow \n");
		return;
	}
	if (front == -1)  /*If queue is empty */
	{
		front = 0;
		rear = 0;
	}
	else
	{
		if(rear == MAX-1)		/*rear is at last position of queue */
			rear = 0;
		else
			rear = rear+1;
	}
	cqueue_arr[rear] = item ;
}
/*End of insert*/
```

```
/*Begin of del*/
void del()
{
	if (front == -1)
	{
		printf("Queue Underflow\n");
		return ;
	}
	printf("Element deleted from queue is : %d\n",cqueue_arr[front]);
	if(front == rear) /* queue has only one element */
	{
		front = -1;
		rear=-1;
	}
	else
	{
		if(front == MAX-1)
			front = 0;
		else
			front = front+1;
	}
}
/*End of del() */

/*Begin of display*/
void display()
{
	int front_pos = front,rear_pos = rear;
	if(front == -1)
	{
		printf("Queue is empty\n");
		return;
```

```
	}
	printf("Queue elements :\n");
	if( front_pos <= rear_pos )
		while(front_pos <= rear_pos)
		{
			printf("%d ",cqueue_arr[front_pos]);
			front_pos++;
		}
	else
	{
		while(front_pos <= MAX-1)
		{
			printf("%d ",cqueue_arr[front_pos]);
			front_pos++;
		}
		front_pos = 0;
		while(front_pos <= rear_pos)
		{
			printf("%d ",cqueue_arr[front_pos]);
			front_pos++;
		}
	}
	printf("\n");
}
/*End of display*/

/*Begin of main*/
int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
	int choice,item;
	do
```

```
	{
		printf("1.Insert\n");
		printf("2.Delete\n");
		printf("3.Display\n");
		printf("4.Quit\n");

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

		switch(choice)
		{
			case 1 :
				printf("Input the element for insertion in queue : ");
				scanf("%d", &item);

				insert(item);
				break;
			case 2 :
				del();
				break;
			case 3:
				display();
				break;
			case 4:
				break;
				default:
				printf("Wrong choice\n");
		}
	}while(choice!=4);

	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

1.Insert

2.Delete

3.Display

4.Quit

Enter your choice : 1

Input the element for insertion in queue : 2

1.Insert

2.Delete

3.Display

4.Quit

Enter your choice : 1

Input the element for insertion in queue : 5

1.Insert

2.Delete

3.Display

4.Quit

Enter your choice : 3

Queue elements :

2 5

1.Insert

2.Delete

3.Display

4.Quit

Enter your choice : 4

```
// Program for ArrayImplementation of Double Ended Queue
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define size 5

int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
    int arr[size],R=-1,F=0,te=0,ch,n,i,x;

    for(;;)         // An infinite loop
    {
        system("cls");              // for clearing the screen
        printf("F=%d  R=%d\n\n",F,R);
        printf("1. Add Rear\n");
        printf("2. Delete Rear\n");
        printf("3. Add Front\n");
        printf("4. Delete Front\n");
        printf("5. Display\n");
        printf("6. Exit\n");
        printf("Enter Choice: ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                if(te==size)
                {
                    printf("Queue is full");
                    getch();      // pause the loop to see the message
                }
```

```
    else
    {
       printf("Enter a number ");
       scanf("%d",&n);
       R=(R+1)%size;
       arr[R]=n;
       te=te+1;
    }
    break;

 case 2:
    if(te==0)
    {
       printf("Queue is empty");
       getch();        // pause the loop to see the message
    }
    else
    {
       if(R==-1)
       {
          R=size-1;
       }
       printf("Number Deleted From Rear End = %d",arr[R]);
       R=R-1;
       te=te-1;
       getch();        // pause the loop to see the number
    }
    break;

 case 3:
    if(te==size)
    {
```

```
        printf("Queue is full");
        getch();        // pause the loop to see the message
    }
    else
    {
        printf("Enter a number ");
        scanf("%d",&n);
        if(F==0)
        {
            F=size-1;
        }
        else
        {
            F=F-1;
        }
        arr[F]=n;
        te=te+1;
    }
    break;

case 4:
    if(te==0)
    {
        printf("Queue is empty");
        getch();        // pause the loop to see the message
    }
    else
    {
        printf("Number Deleted From Front End = %d",arr[F]);
        F=(F+1)%size;
        te=te-1;
        getch();        // pause the loop to see the number
```

```
            }
            break;

        case 5:
            if(te==0)
            {
                printf("Queue is empty");
                getch();        // pause the loop to see the message
            }
            else
            {
                x=F;
                for(i=1; i<=te; i++)
                {
                    printf("%d ",arr[x]);
                    x=(x+1)%size;
                }
                getch();        // pause the loop to see the numbers
            }
            break;

        case 6:
            exit(0);
            break;

        default:
            printf("Wrong Choice");
            getch();            // pause the loop to see the message
      }
   }
   return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

F=0 R=2

1. Add Rear

2. Delete Rear

3. Add Front

4. Delete Front

5. Display

6. Exit

Enter Choice: 5

3 3 4

LAB – 17

```
// Program for Array Implementation of Priority Queue (Ascending Array)
#include<stdio.h>
#include<limits.h>
#define MAX 100


// denotes where the last item in priority queue is
// initialized to -1 since no item is in queue
int idx = -1;

// pqVal holds data for each index item
// pqPriority holds priority for each index item
int pqVal[MAX];
int pqPriority[MAX];



int isEmpty(){
   return idx == -1;
}

int isFull(){
   return idx == MAX - 1;
}
// enqueue just adds item to the end of the priority queue | O(1)
void enqueue(int data, int priority)
{
   if(!isFull()){

      // Increase the index
      idx++;
```

```
        // Insert the element in priority queue
        pqVal[idx] = data;
        pqPriority[idx] = priority;
    }
}

// returns item with highest priority
// NOTE: Max Priority Queue High priority number means higher priority | O(N)
int peek()
{
    // Note : Max Priority, so assigned min value as initial value
    int maxPriority = INT_MIN;
    int indexPos = -1;

    // Linear search for highest priority
    for (int i = 0; i <= idx; i++) {
        // If two items have same priority choose the one with
        // higher data value
        if (maxPriority == pqPriority[i] && indexPos > -1 && pqVal[indexPos] < pqVal[i])
        {
            maxPriority = pqPriority[i];
            indexPos = i;
        }

        // note: using MAX Priority so higher priority number
        // means higher priority
        else if (maxPriority < pqPriority[i]) {
            maxPriority = pqPriority[i];
            indexPos = i;
        }
    }
```

```
    // Return index of the element where
    return indexPos;
}

// This removes the element with highest priority
// from the priority queue | O(N)
void dequeue()
{
    if(!isEmpty())
    {
        // Get element with highest priority
        int indexPos = peek();

        // reduce size of priority queue by first
        // shifting all elements one position left
        // from index where the highest priority item was found
        for (int i = indexPos; i < idx; i++) {
            pqVal[i] = pqVal[i + 1];
            pqPriority[i] = pqPriority[i + 1];
        }

        // reduce size of priority queue by 1
        idx--;
    }
}

void display(){
    for (int i = 0; i <= idx; i++) {
        printf("(%d, %d)\n",pqVal[i], pqPriority[i]);
    }
}
```

```
// Driver Code
int main()
{
        printf("ABHAY PANDEY  2100320120004n");
   // To enqueue items as per priority
   enqueue(5, 1);
   enqueue(10, 3);
   enqueue(15, 4);
   enqueue(20, 5);
   enqueue(500, 2);

   printf("Before Dequeue : \n");
   display();

   // Dequeue the top element
   dequeue(); // 20 dequeued
   dequeue(); // 15 dequeued

   printf("\nAfter Dequeue : \n");
   display();

   return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
Before Dequeue :
(5, 1)
(10, 3)
(15, 4)
(20, 5)
(500, 2)
```

After Dequeue :

(5, 1)

(10, 3)

(500, 2)

```
// Program for Stack implementation using Queue
#include <stdio.h>
#include <stdlib.h>

void push1(int);
void push2(int);
int pop1();
int pop2();
void enqueue();
void dequeue();
void display();
void create();

int st1[100], st2[100];
int top1 = -1, top2 = -1;
int count = 0;

int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
   int ch;

   printf("\n1 - Enqueue element into queue");
   printf("\n2 - Dequeu element from queue");
   printf("\n3 - Display from queue");
   printf("\n4 - Exit");
   create();
   while (1)
   {
      printf("\nEnter choice");
      scanf("%d", &ch);
      switch (ch)
```

```
        {
        case 1:
            enqueue();
            break;
        case 2:
            dequeue();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Wrong choice");
        }
    }
    return 0;
}

/*Function to create a queue*/
void create()
{
    top1 = top2 = -1;
}

/*Function to push the element on to the stack*/
void push1(int data)
{
    st1[++top1] = data;
}

/*Function to pop the element from the stack*/
```

```
int pop1()
{
   return(st1[top1--]);
}

/*Function to push an element on to stack*/
void push2(int data)
{
   st2[++top2] = data;
}

/*Function to pop an element from th stack*/

int pop2()
{
   return(st2[top2--]);
}

/*Function to add an element into the queue using stack*/
void enqueue()
{
   int data, i;

   printf("Enter data into queue");
   scanf("%d", &data);
   push1(data);
   count++;
}

/*Function to delete an element from the queue using stack*/

void dequeue()
```

```
{
    int i;

    for (i = 0;i <= count;i++)
    {
        push2(pop1());
    }
    pop2();
    count--;
    for (i = 0;i <= count;i++)
    {
        push1(pop2());
    }
}

/*Function to display the elements in the stack*/

void display()
{
    int i;

    for (i = 0;i <= top1;i++)
    {
        printf(" %d ", st1[i]);
    }
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

1 - Enqueue element into queue

2 - Dequeu element from queue

3 - Display from queue

4 - Exit

Enter choice1

Enter data into queue2

Enter choice1

Enter data into queue5

Enter choice1

Enter data into queue7

Enter choice3

2 5 7

Enter choice4

```c
// Program for Queue implementation using Stack
#include<stdio.h>
#define N 5
int stack1[5], stack2[5]; // declaration of two stacks
// declaration of top variables.
int top1=-1, top2=-1;
int count=0;
// inserting the elements in stack1.
void push1(int data)
{
// Condition to check whether the stack1 is full or not.
 if(top1==N-1)
{
  printf("\n Stack is overflow...");
}
else
{
  top1++;   // Incrementing the value of top1
  stack1[top1]=data;  // pushing the data into stack1
}
}
// Removing the elements from the stack1.
int pop1()
{
// Condition to check whether the stack1 is empty or not.
if(top1==-1)
{
  printf("\nStack is empty..");
}
else
{
  int a=stack1[top1];  // Assigning the topmost value of stack1 to 'a' variable.
```

```
  top1--;  // decrementing the value of top1.
  return a;
}
}
// pushing the data into the stack2.
void push2(int x)
{
//  Condition to check whether the stack2 is full or not
if(top2==N-1)
{
  printf("\nStack is full..");
}
else
{
   top2++;   // incrementing the value of top2.
   stack2[top2]=x;  // assigning the 'x' value to the Stack2

}
}
// Removing the elements from the Stack2
int pop2()
{
  int element = stack2[top2];  // assigning the topmost value to element
  top2--;  // decrement the value of top2
  return element;
}
void enqueue(int x)
{
   push1(x);
   count++;
}
void dequeue()
```

```
{
   if((top1==-1) && (top2==-1))
{
   printf("\nQueue is empty");
}
else
{
  for(int i=0;i<count;i++)
  {
     int element = pop1();
     push2(element);
  }
int b= pop2();
printf("\nThe dequeued element is %d", b);
printf("\n");
count--;
for(int i=0;i<count;i++)
{
   int a = pop2();
   push1(a);
}
}}
void display()
{
   for(int i=0;i<=top1;i++)
  {
     printf("%d , ", stack1[i]);
  }
}
int main()
{
   printf("ABHAY PANDEY  2100320120004\n");
```

```
  enqueue(10);
  enqueue(20);
  enqueue(30);
  dequeue();
  enqueue(40);
  display();
  return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

The dequeued element is 10

20 , 30 , 40

LAB – 18

```
// Program for Linear Linked List Primitive operations
#include <stdio.h>
#include <stdlib.h>

// Create a node
struct Node {
  int data;
  struct Node* next;
};

// Insert at the beginning
void insertAtBeginning(struct Node** head_ref, int new_data) {
  // Allocate memory to a node
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

  // insert the data
  new_node->data = new_data;

  new_node->next = (*head_ref);

  // Move head to new node
  (*head_ref) = new_node;
}

// Insert a node after a node
void insertAfter(struct Node* prev_node, int new_data) {
  if (prev_node == NULL) {
  printf("the given previous node cannot be NULL");
  return;
  }
```

```
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node->data = new_data;
  new_node->next = prev_node->next;
  prev_node->next = new_node;
}

// Insert the the end
void insertAtEnd(struct Node** head_ref, int new_data) {
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  struct Node* last = *head_ref; /* used in step 5*/

  new_node->data = new_data;
  new_node->next = NULL;

  if (*head_ref == NULL) {
  *head_ref = new_node;
  return;
  }

  while (last->next != NULL) last = last->next;

  last->next = new_node;
  return;
}

// Delete a node
void deleteNode(struct Node** head_ref, int key) {
  struct Node *temp = *head_ref, *prev;

  if (temp != NULL && temp->data == key) {
  *head_ref = temp->next;
  free(temp);
```

```
 return;
 }
 // Find the key to be deleted
 while (temp != NULL && temp->data != key) {
 prev = temp;
 temp = temp->next;
 }

 // If the key is not present
 if (temp == NULL) return;

 // Remove the node
 prev->next = temp->next;

 free(temp);
}

// Search a node
int searchNode(struct Node** head_ref, int key) {
 struct Node* current = *head_ref;

 while (current != NULL) {
 if (current->data == key) return 1;
 current = current->next;
 }
 return 0;
}

// Sort the linked list
void sortLinkedList(struct Node** head_ref) {
 struct Node *current = *head_ref, *index = NULL;
 int temp;
```

```
  if (head_ref == NULL) {
  return;
  } else {
  while (current != NULL) {
    // index points to the node next to current
    index = current->next;

    while (index != NULL) {
    if (current->data > index->data) {
      temp = current->data;
      current->data = index->data;
      index->data = temp;
    }
    index = index->next;
    }
    current = current->next;
  }
  }
}

// Print the linked list
void printList(struct Node* node) {
  while (node != NULL) {
  printf(" %d ", node->data);
  node = node->next;
  }
}

// Driver program
int main() {
	printf("ABHAY PANDEY  2100320120004\n");
```

```
  struct Node* head = NULL;

  insertAtEnd(&head, 1);
  insertAtBeginning(&head, 2);
  insertAtBeginning(&head, 3);
  insertAtEnd(&head, 4);
  insertAfter(head->next, 5);

  printf("Linked list: ");
  printList(head);

  printf("\nAfter deleting an element: ");
  deleteNode(&head, 3);
  printList(head);

  int item_to_find = 3;
  if (searchNode(&head, item_to_find)) {
  printf("\n%d is found", item_to_find);
  } else {
  printf("\n%d is not found", item_to_find);
  }

  sortLinkedList(&head);
  printf("\nSorted List: ");
  printList(head);
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Linked list: 3 2 5 1 4

After deleting an element: 2 5 1 4

3 is not found

Sorted List: 1 2 4 5

//Program for creation of Linked List header file and test of basic functions through that

```
// C program for a Header Linked List
#include <malloc.h>
#include <stdio.h>

// Structure of the list
```

```
struct link {
	int info;
	struct link* next;
};

// Empty List
struct link* start = NULL;

// Function to create a header linked list
struct link* create_header_list(int data)
{

	// Create a new node
	struct link *new_node, *node;
	new_node = (struct link*)
		malloc(sizeof(struct link));
	new_node->info = data;
	new_node->next = NULL;

	// If it is the first node
	if (start == NULL) {

		// Initialize the start
		start = (struct link*)
			malloc(sizeof(struct link));
		start->next = new_node;
	}
	else {

		// Insert the node in the end
		node = start;
		while (node->next != NULL) {
```

```
                    node = node->next;
            }
            node->next = new_node;
    }
    return start;
}

// Function to display the
// header linked list
struct link* display()
{
    struct link* node;
    node = start;
    node = node->next;
    while (node != NULL) {
            printf("%d ", node->info);
            node = node->next;
    }
    printf("\n");
    return start;
}

// Driver code
int main()
{
  printf("ABHAY PANDEY  2100320120004\n");
    // Create the list
    create_header_list(11);
    create_header_list(12);
    create_header_list(13);

    // Print the list
```

```
    display();
    create_header_list(14);
    create_header_list(15);

    // Print the list
    display();

    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

11 12 13

11 12 13 14 15

```
//Program for finding count of Nodes in Linked List
#include <stdio.h>
//linked list node structure
struct node{
int data;
struct node* next;
};
struct node* head;
void insert(int data){
/* Allocate memory*/
struct node* temp = (struct node*)malloc(sizeof(struct node));
temp->data = data;
temp->next = head;
head = temp;
}
void print(){
struct node* temp = head;
int count=0;
/* Traverse the linked list and maintain the count. */
while(temp != NULL){
temp = temp->next;
count++;
}
printf("\n Total no. of nodes is %d",count);
}
int main(){
        printf("ABHAY PANDEY  2100320120004\n");
head = NULL;
insert(2);
insert(4);
/* calling print function to print the count of node. */
print();
```

```
return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Total no. of nodes is 2

```
// Program for concatenation of Linear Linked List
#include <stdio.h>
#include <stdlib.h>

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

display(struct node *head)
{
   if(head == NULL)
   {
      printf("NULL\n");
   }
   else
   {
      printf("%d\n", head -> data);
      display(head->next);
   }
}

void concatenate(struct node *a,struct node *b)
{
   if( a != NULL && b!= NULL )
   {
      if (a->next == NULL)
         a->next = b;
      else
         concatenate(a->next,b);
   }
```

```
    else
    {
        printf("Either a or b is NULL\n");
    }
}

int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
    struct node *prev,*a, *b, *p;
    int n,i;
    printf ("number of elements in a:");
    scanf("%d",&n);
    a=NULL;
    for(i=0;i<n;i++)
    {
        p=malloc(sizeof(struct node));
        scanf("%d",&p->data);
        p->next=NULL;
        if(a==NULL)
            a=p;
        else
            prev->next=p;
        prev=p;
    }
    printf ("number of elements in b:");
    scanf("%d",&n);
    b=NULL;
    for(i=0;i<n;i++)
    {
        p=malloc(sizeof(struct node));
        scanf("%d",&p->data);
```

```
    p->next=NULL;
    if(b==NULL)
      b=p;
    else
      prev->next=p;
    prev=p;
  }
  concatenate(a,b);
  return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
number of elements in a:2
34
45
number of elements in b:3
12
34
56
67
dash: 2: 67: not found
```

```
// Program to implement Linear search.
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *next;
}  *first = NULL;

void Create(int A[], int n)
{
   int i;
   struct Node *t, *last;
   first = (struct Node *) malloc(sizeof (struct Node));
   first->data = A[0];
   first->next = NULL;
   last = first;

   for (i = 1; i < n; i++)
   {
      t = (struct Node *) malloc(sizeof (struct Node));
      t->data = A[i];
      t->next = NULL;
      last->next = t;
      last = t;
   }
}

struct Node* LSearch(struct Node *p, int key)
{
   while (p != NULL)
   {
```

```
        if (key == p->data)
         return p;
        p = p->next;
    }
    return NULL;
}

struct Node* RSearch(struct Node *p, int key)
{
    if (p == NULL)
        return NULL;
    if (key == p->data)
        return p;
    return RSearch (p->next, key);
}

int main()
{
            printf("ABHAY PANDEY  2100320120004\n");
    struct Node *temp;
    int A[] = { 8, 3, 7, 12 };
    Create(A, 4);
    temp = LSearch(first, 7);
    printf (" %d", temp->data);
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

7

LAB – 19

// Program to insert an item at any given position in the linked List

```
#include<stdio.h>

#include<stdlib.h>


struct Node

{

  int data;

  struct Node *next;

};


// required for insertAfter

int getCurrSize (struct Node *node)

{

  int size = 0;


  while (node != NULL)

    {

      node = node->next;

      size++;

    }

  return size;

}


//function to insert after nth node

void insertPosition (int pos, int data, struct Node **head)

{

  int size = getCurrSize (*head);


  struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));

  newNode->data = data;

  newNode->next = NULL;
```

```
  // Can't insert if position to insert is greater than size of Linked List
  // can insert after negative pos
  if (pos < 0 || pos > size)
    printf ("Invalid position to insert\n");

  // inserting first node
  else if (pos == 0)
    {
      newNode->next = *head;
      *head = newNode;
    }

  else
    {
      // temp used to traverse the Linked List
      struct Node *temp = *head;

      // traverse till the nth node
      while (--pos)
            temp = temp->next;

      // assign newNode's next to nth node's next
      newNode->next = temp->next;
      // assign nth node's next to this new node
      temp->next = newNode;
      // newNode inserted b/w 3rd and 4th node
    }
}

void display (struct Node *node)
{
```

```
  printf ("Linked List : ");

  // as linked list will end when Node is Null
  while (node != NULL)
    {
      printf ("%d ", node->data);
      node = node->next;
    }
  printf ("\n");
}

int main ()
{
   printf("ABHAY PANDEY  2100320120004\n");
  //creating 4 pointers of type struct Node
  //So these can point to address of struct type variable
  struct Node *head = NULL;
  struct Node *node2 = NULL;
  struct Node *node3 = NULL;
  struct Node *node4 = NULL;

  // allocate 3 nodes in the heap
  head = (struct Node *) malloc (sizeof (struct Node));
  node2 = (struct Node *) malloc (sizeof (struct Node));
  node3 = (struct Node *) malloc (sizeof (struct Node));
  node4 = (struct Node *) malloc (sizeof (struct Node));


  head->data = 10;                    // data set for head node
  head->next = node2;                 // next pointer assigned to address of node2
```

```
node2->data = 20;
node2->next = node3;

node3->data = 30;
node3->next = node4;

node4->data = 40;
node4->next = NULL;


display (head);

//Inserts data: 15 after the 1st node
insertPosition (1, 15, &head);

//Inserts data: 25 after the 3rd node
insertPosition (3, 25, &head);

//Inserts data: 35 after the 5th node
insertPosition (5, 35, &head);

//Inserts data: 25 after the 7th node
insertPosition (7, 45, &head);

display (head);

// Invalid: can't insert after -2 pos
insertPosition (-2, 100, &head);

// Invalid: Current size 8, trying to enter after 10th pos
insertPosition (10, 200, &head);
```

```
  return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Linked List : 10 20 30 40

Linked List : 10 15 20 25 30 35 40 45

Invalid position to insert

Invalid position to insert

```
// Program for Creation of Copy of the Linked list
// C program for the above approach
#include <stdio.h>
#include <stdlib.h>

// Node for linked list
struct Node {
	int data;
	struct Node* next;
};

// Function to print given linked list
void printList(struct Node* head)
{
	struct Node* ptr = head;
	while (ptr) {

		printf("%d -> ", ptr->data);
		ptr = ptr->next;
	}

	printf("NULL");
}

// Function to create a new node
void insert(struct Node** head_ref, int data)
{
	// Allocate the memory for new Node
	// in the heap and set its data
	struct Node* newNode
		= (struct Node*)malloc(
			sizeof(struct Node));
```

```
        newNode->data = data;

        // Set the next node pointer of the
        // new Node to point to the current
        // node of the list
        newNode->next = *head_ref;

        // Change the pointer of head to point
        // to the new Node
        *head_ref = newNode;
}

// Function to create a copy of a linked list
struct Node* copyList(struct Node* head)
{
        if (head == NULL) {
                return NULL;
        }
        else {

                // Allocate the memory for new Node
                // in the heap and set its data
                struct Node* newNode
                        = (struct Node*)malloc(
                                sizeof(struct Node));

                newNode->data = head->data;

                // Recursively set the next pointer of
                // the new Node by recurring for the
                // remaining nodes
```

```
        newNode->next = copyList(head->next);

        return newNode;
    }
}

// Function to create the new linked list
struct Node* create(int arr[], int N)
{
    // Pointer to point the head node
    // of the singly linked list
    struct Node* head_ref = NULL;

    // Construct the linked list
    for (int i = N - 1; i >= 0; i--) {

        insert(&head_ref, arr[i]);
    }

    // Return the head pointer
    return head_ref;
}

// Function to create both the lists
void printLists(struct Node* head_ref,
                struct Node* dup)
{

    printf("Original list: ");

    // Print the original linked list
    printList(head_ref);
```

```c
            printf("\nDuplicate list: ");

            // Print the duplicate linked list
            printList(dup);
}

// Driver Code
int main(void)
{
    printf("ABHAY PANDEY  2100320120004n");
            // Given nodes value
            int arr[] = { 1, 2, 3, 4, 5 };
            int N = sizeof(arr) / sizeof(arr[0]);

            // Head of the original Linked list
            struct Node* head_ref = create(arr, N);

            // Head of the duplicate Linked List
            struct Node* dup = copyList(head_ref);

            printLists(head_ref, dup);

            return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Original list: 1 -> 2 -> 3 -> 4 -> 5 -> NULL

Duplicate list: 1 -> 2 -> 3 -> 4 -> 5 -> NULL

```
// Program for counting nodes containing even and odd information.
// C program to segregate even and odd nodes in a
// Linked List
#include <stdio.h>
#include <stdlib.h>

/* a node of the singly linked list */
struct Node
{
	int data;
	struct Node *next;
};

void segregateEvenOdd(struct Node **head_ref)
{
	struct Node *end = *head_ref;
	struct Node *prev = NULL;
	struct Node *curr = *head_ref;

	/* Get pointer to the last node */
	while (end->next != NULL)
		end = end->next;

	struct Node *new_end = end;

	/* Consider all odd nodes before the first even node
	and move then after end */
	while (curr->data %2 != 0 && curr != end)
	{
		new_end->next = curr;
		curr = curr->next;
		new_end->next->next = NULL;
```

```
        new_end = new_end->next;
}

// 10->8->17->17->15
/* Do following steps only if there is any even node */
if (curr->data%2 == 0)
{
        /* Change the head pointer to point to first even node */
        *head_ref = curr;

        /* now current points to the first even node */
        while (curr != end)
        {
                if ( (curr->data)%2 == 0 )
                {
                        prev = curr;
                        curr = curr->next;
                }
                else
                {
                        /* break the link between prev and current */
                        prev->next = curr->next;

                        /* Make next of curr as NULL */
                        curr->next = NULL;

                        /* Move curr to end */
                        new_end->next = curr;

                        /* make curr as new end of list */
                        new_end = curr;
```

```
                        /* Update current pointer to next of the moved node */
                        curr = prev->next;
                    }
                }
            }

            /* We must have prev set before executing lines following this
            statement */
            else prev = curr;

            /* If there are more than 1 odd nodes and end of original list is
            odd then move this node to end to maintain same order of odd
            numbers in modified list */
            if (new_end!=end && (end->data)%2 != 0)
            {
                prev->next = end->next;
                end->next = NULL;
                new_end->next = end;
            }
            return;
        }

        /* UTILITY FUNCTIONS */
        /* Function to insert a node at the beginning */
        void push(struct Node** head_ref, int new_data)
        {
            /* allocate node */
            struct Node* new_node =
                (struct Node*) malloc(sizeof(struct Node));

            /* put in the data */
            new_node->data = new_data;
```

```
	/* link the old list of the new node */
	new_node->next = (*head_ref);

	/* move the head to point to the new node */
	(*head_ref) = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct Node *node)
{
	while (node!=NULL)
	{
		printf("%d ", node->data);
		node = node->next;
	}
}

/* Driver program to test above functions*/
int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
	/* Start with the empty list */
	struct Node* head = NULL;

	/* Let us create a sample linked list as following
	0->2->4->6->8->10->11 */

	push(&head, 11);
	push(&head, 10);
	push(&head, 8);
	push(&head, 6);
```

```
	push(&head, 4);
	push(&head, 2);
	push(&head, 0);

	printf("\nOriginal Linked list \n");
	printList(head);

	segregateEvenOdd(&head);

	printf("\nModified Linked list \n");
	printList(head);

	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Original Linked list

0 2 4 6 8 10 11

Modified Linked list

0 2 4 6 8 10 11

// Program for Creation of Ascending Order Linear Linked List

```
#include<stdio.h>

#include<stdlib.h>


typedef struct node

{

 int data;

 struct node*next;

}node;


void add(node**s,int num)

{

 node*temp;

 if(*s==NULL||num<(*s)->data)

 {

  temp=(node*)malloc(sizeof(node));

  temp->data=num;

  temp->next=*s;

  *s=temp;

 }

 else

 {

  temp=*s;

  while(temp!=NULL)

  {

   if(temp->data<=num&&(temp->next==NULL || temp->next->data>num))

   {

    node*temp1=(node*)malloc(sizeof(node));

    temp1->data=num;

    temp1->next=temp->next;

    temp->next=temp1;

    return;
```

```
   }
   temp=temp->next;
  }
 }

}
void traverse(node*s)
{
 while(s!=NULL)
 {
  printf("%d\t",s->data);
  s=s->next;
 }
}

int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
 node*n=NULL;

 add(&n,5);
 add(&n,7);
 add(&n,18);
 add(&n,12);
 add(&n,78);
 add(&n,-13);
 traverse(n);
 return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

-13 5 7 12 18 78

```c
// Program for Merging two sorted Linked List/unsoted link list
#include <stdio.h>
#include <stdlib.h>

struct Node
{
   int data;
   struct Node *next;
} *temp = NULL, *first = NULL, *second = NULL, *third = NULL, *last = NULL;

struct Node* Create (int A[], int n)
{
   int i;
   struct Node *t, *last;
   temp = (struct Node *) malloc(sizeof(struct Node));
   temp->data = A[0];
   temp->next = NULL;
   last = temp;

   for (i = 1; i < n; i++)
   {
      t = (struct Node *) malloc(sizeof(struct Node));
      t->data = A[i];
      t->next = NULL;
      last->next = t;
      last = t;
   }
   return temp;
}

void Display(struct Node *p)
{
```

```
    while (p != NULL)
    {
        printf ("%d ", p->data);
        p = p->next;
    }
}

void Merge(struct Node *first, struct Node *second)
{
    if (first->data < second->data)
    {
        third = last = first;
        first = first->next;
        last->next = NULL;
    }
    else
    {
        third = last = second;
        second = second->next;
        last->next = NULL;
    }

    while (first != NULL && second != NULL)
    {
        if (first->data < second->data)
        {
            last->next = first;
            last = first;
            first = first->next;
            last->next = NULL;
        }
        else
```

```
        {
          last->next = second;
          last = second;
          second = second->next;
          last->next = NULL;
        }
    }

    if (first != NULL)
        last->next = first;
    else
        last->next = second;
}

int main()
{
            printf("ABHAY PANDEY  2100320120004\n");
    int A[] = { 3, 4, 7, 9 };
    int B[] = { 2, 5, 6, 8 };
    first = Create (A, 4);
    second = Create (B, 4);

    printf ("1st Linked List: ");
    Display (first);

    printf ("\n2nd Linked List: ");
    Display (second);

    Merge (first, second);

    printf ("\n\nMerged Linked List: \n");
    Display (third);
```

```
  return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

1st Linked List: 3 4 7 9

2nd Linked List: 2 5 6 8

Merged Linked List:

2 3 4 5 6 7 8 9

```
// Program for finding difference of two linked list (consider lists as sets)
print("ABHAY PANDEY  2100320120004\n");
class Node:
	def __init__(self, data):
		self.data = data
		self.next = None

# Linked List
class linked_list:
	def __init__(self):
		self.head = None

	# Function to insert a node
	# at the end of Linked List
	def append(self, data):
		temp = Node(data)
		if self.head == None:
			self.head = temp
		else:
			p = self.head
			while p.next != None:
				p = p.next
			p.next = temp

	# Function to find the middle
	# node of the Linked List
	def get_mid(self, head):
		if head == None:
			return head
		slow = fast = head
		while fast.next != None \
		and fast.next.next != None:
```

```
        slow = slow.next
        fast = fast.next.next
    return slow

# Recursive method to merge the
# two half after sorting
def merge(self, l, r):
    if l == None:return r
    if r == None:return l

    if l.data<= r.data:
        result = l
        result.next = \
            self.merge(l.next, r)
    else:
        result = r
        result.next = \
            self.merge(l, r.next)
    return result

# Recursive method to divide the
# list into two half until 1 node left
def merge_sort(self, head):
    if head == None or head.next == None:
        return head
    mid = self.get_mid(head)
    next_to_mid = mid.next
    mid.next = None
    left = self.merge_sort(head)
    right = self.merge_sort(next_to_mid)
    sorted_merge = self.merge(left, right)
    return sorted_merge
```

```
    # Function to print the list elements
    def display(self):
        p = self.head
        while p != None:
            print(p.data, end =' ')
            p = p.next
        print()

# Function to get the difference list
def get_difference(p1, p2):
    difference_list = linked_list()
    # Scan the lists
    while p1 != None and p2 != None:

        # Condition to check if the
        # Data of the both pointer are
        # same then move ahead
        if p2.data == p1.data:
            p1 = p1.next
            p2 = p2.next

        # Condition to check if the
        # Data of the first pointer is
        # greater than second then
        # move second pointer ahead
        elif p2.data<p1.data:
            p2 = p2.next

        # Condition when first pointer
        # data is greater than the
        # second pointer then append
```

```
        # into the difference list and move
        else:
            difference_list.append(p1.data)
            p1 = p1.next

    # If end of list2 is reached,
    # there may be some nodes in
    # List 1 left to be scanned,
    # they all will be inserted
    # in the difference list
    if p2 == None:
        while p1:
            difference_list.append(p1.data)
            p1 = p1.next

    return difference_list

# Driver Code
if __name__ == '__main__':
    # Linked List 1
    list1 = linked_list()
    list1.append(2)
    list1.append(6)
    list1.append(8)
    list1.append(1)

    # Linked List 2
    list2 = linked_list()
    list2.append(4)
    list2.append(1)
    list2.append(9)
```

```
    # Sort both the linkedlists
    list1.head = list1.merge_sort(
                                  list1.head
                                  )
    list2.head = list2.merge_sort(
                                  list2.head
                                  )

    # Get difference list
    result = get_difference(
                            list1.head, list2.head
                            )
    if result.head:
        result.display()

    # if difference list is empty,
    # then lists are equal
    else:
        print('Lists are equal')
```

OUTPUT:

ABHAY PANDEY 2100320120004

2 6 8

LAB – 20

//Program for Finding the Middle element of a singly linked list in one pass

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

// Link list node
struct Node
{
	int data;
	struct Node* next;
};

// Function to get the middle of
// the linked list
void printMiddle(struct Node *head)
{
	struct Node *slow_ptr = head;
	struct Node *fast_ptr = head;

	if (head!=NULL)
	{
		while (fast_ptr != NULL &&
			fast_ptr->next != NULL)
		{
			fast_ptr = fast_ptr->next->next;
			slow_ptr = slow_ptr->next;
		}
		printf("The middle element is [%d]",
				slow_ptr->data);
	}
}
```

```
void push(struct Node** head_ref,
          int new_data)
{
    // Allocate node
    struct Node* new_node =
          (struct Node*) malloc(sizeof(struct Node));

    // Put in the data
    new_node->data = new_data;

    // Link the old list off the new node
    new_node->next = (*head_ref);

    // Move the head to point to the new node
    (*head_ref) = new_node;
}

// A utility function to print a given
// linked list
void printList(struct Node *ptr)
{
    while (ptr != NULL)
    {
          printf("%d->", ptr->data);
          ptr = ptr->next;
    }
    printf("NULL");
}

// Driver code
int main()
```

```
{
	printf("ABHAY PANDEY  2100320120004\n");
	// Start with the empty list
	struct Node* head = NULL;
	int i;

	for (i = 5; i > 0; i--)
	{
		push(&head, i);
		printList(head);
		printMiddle(head);
	}
	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

5->NULLThe middle element is [5]4->5->NULLThe middle element is [5]3->4->5->NULLThe middle element is [4]2->3->4->5->NULLThe middle element is [4]1->2->3->4->5->NULLThe middle element is [3]

```
//Program to perform Binary Search on the Linked List
#include<stdio.h>
#include<stdlib.h>

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

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

// function to find out middle element
struct Node* middle(struct Node* start,struct Node* last)
{
   if (start == NULL)
      return NULL;

   struct Node* slow = start;
   struct Node* fast = start -> next;

   while (fast != last)
   {
      fast = fast -> next;
      if (fast != last)
      {
```

```
            slow = slow -> next;
            fast = fast -> next;
        }
    }

    return slow;
}

// Function for implementing the Binary
// Search on linked list
struct Node* binarySearch(struct Node *head, int value)
{
    struct Node* start = head;
    struct Node* last = NULL;

    do
    {
        // Find middle
        struct Node* mid = middle(start, last);

        // If middle is empty
        if (mid == NULL)
            return NULL;

        // If value is present at middle
        if (mid -> data == value)
            return mid;

        // If value is more than mid
        else if (mid -> data < value)
            start = mid -> next;
        // If the value is less than mid.
```

```
        else
          last = mid;

    } while (last == NULL ||
            last != start);

    // value not present
    return NULL;
}
// Driver Code
int main()
{
              printf("ABHAY PANDEY  2100320120004\n");
    struct Node *head = newNode(1);
    head->next = newNode(4);
    head->next->next = newNode(7);
    head->next->next->next = newNode(8);
    head->next->next->next->next = newNode(9);
    head->next->next->next->next->next = newNode(10);
    int value = 8;
    if (binarySearch(head, value) == NULL)
        printf("Value not present\n");
    else
        printf("Present");
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Present

//Program for Reversing the Linear Linked List

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

```
#include <stdlib.h>

/* Link list node */
struct Node {
	int data;
	struct Node* next;
};

/* Function to reverse the linked list */
static void reverse(struct Node** head_ref)
{
	struct Node* prev = NULL;
	struct Node* current = *head_ref;
	struct Node* next = NULL;
	while (current != NULL) {
		// Store next
		next = current->next;

		// Reverse current node's pointer
		current->next = prev;

		// Move pointers one position ahead.
		prev = current;
		current = next;
	}
	*head_ref = prev;
}

/* Function to push a node */
void push(struct Node** head_ref, int new_data)
{
	struct Node* new_node
```

```
                = (struct Node*)malloc(sizeof(struct Node));
        new_node->data = new_data;
        new_node->next = (*head_ref);
        (*head_ref) = new_node;
}

/* Function to print linked list */
void printList(struct Node* head)
{
        struct Node* temp = head;
        while (temp != NULL) {
                printf("%d ", temp->data);
                temp = temp->next;
        }
}

/* Driver code*/
int main()
{
         printf("ABHAY PANDEY  2100320120004\n");
        /* Start with the empty list */
        struct Node* head = NULL;

        push(&head, 20);
        push(&head, 4);
        push(&head, 15);
        push(&head, 85);

        printf("Given linked list\n");
        printList(head);
        reverse(&head);
        printf("\nReversed linked list \n");
```

```
    printList(head);

    getchar();

}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Given linked list

85 15 4 20

Reversed linked list

20 4 15 85

```
//Program to print Linked List contents in reverse order
#include<stdio.h>
#include<stdlib.h>

struct node                                   //code for making a node
{
  int data;
  struct node *next;
};

void display (struct node *head)//method for reverse display nodes
{
  if (head == NULL)
    return;

  // print the list after head node
  display (head->next);

  // After everything else is printed, print head
  printf ("%d  ", head->data);
}


int main ()
{
        printf("ABHAY PANDEY  2100320120004\n");
  struct node *prev, *head, *p;
  int n, i;
  printf ("Enter size of Linked List: ");
  scanf ("%d", &n);
  head = NULL;
  for (i = 0; i < n; i++)
```

```
  {
    p = malloc (sizeof (struct node));
    printf ("Enter the data: ");
    scanf ("%d", &p->data);
    p->next = NULL;
    if (head == NULL)
        head = p;
    else
        prev->next = p;
    prev = p;
  }
 display (head);
 return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
Enter size of Linked List: 5
Enter the data: 1
2Enter the data:
34
Enter the data: 56
Enter the data: 67
Enter the data: 78
78  67  56  34  1
```

```
//Program for Pair wise swap of elements in linked list

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

/* A linked list node */
struct Node {
	int data;
	struct Node* next;
};

/*Function to swap two integers at addresses a and b */
void swap(int* a, int* b);

/* Function to pairwise swap elements of a linked list */
void pairWiseSwap(struct Node* head)
{
	struct Node* temp = head;

	/* Traverse further only if there are at-least two nodes left */
	while (temp != NULL && temp->next != NULL) {
		/* Swap data of node with its next node's data */
		swap(&temp->data, &temp->next->data);

		/* Move temp by 2 for the next pair */
		temp = temp->next->next;
	}
}

/* UTILITY FUNCTIONS */
/* Function to swap two integers */
void swap(int* a, int* b)
```

```
{
	int temp;
	temp = *a;
	*a = *b;
	*b = temp;
}

/* Function to add a node at the beginning of Linked List */
void push(struct Node** head_ref, int new_data)
{
	/* allocate node */
	struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

	/* put in the data */
	new_node->data = new_data;

	/* link the old list of the new node */
	new_node->next = (*head_ref);

	/* move the head to point to the new node */
	(*head_ref) = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct Node* node)
{
	while (node != NULL) {
		printf("%d ", node->data);
		node = node->next;
	}
}
```

```
/* Driver program to test above function */
int main()
{
             printf("ABHAY PANDEY  2100320120004\n");
            struct Node* start = NULL;

            /* The constructed linked list is:
            1->2->3->4->5 */
            push(&start, 5);
            push(&start, 4);
            push(&start, 3);
            push(&start, 2);
            push(&start, 1);

            printf("Linked list before calling pairWiseSwap()\n");
            printList(start);

            pairWiseSwap(start);

            printf("\nLinked list after calling pairWiseSwap()\n");
            printList(start);
            return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Linked list before calling pairWiseSwap()

1 2 3 4 5

Linked list after calling pairWiseSwap()

2 1 4 3 5

//Program to find kth node from the last in a single link list

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

```
#include <stdlib.h>

/* Link list node */
typedef struct Node {
	int data;
	struct Node* next;
} Node;

/* Function to get the nth node from the last of a linked
* list*/
void printNthFromLast(Node* head, int N)
{
	int len = 0, i;
	Node* temp = head;

	// Count the number of nodes in Linked List
	while (temp != NULL) {
		temp = temp->next;
		len++;
	}

	// Check if value of N is not
	// more than length of the linked list
	if (len < N)
		return;
	temp = head;

	// Get the (len-N+1)th node from the beginning
	for (i = 1; i < len - N + 1; i++)
		temp = temp->next;
	printf("%d", temp->data);
	return;
```

```
}

void push(struct Node** head_ref, int new_data)
{
	/* Allocate node */
	Node* new_node = (Node*)malloc(sizeof(Node));

	/* Put in the data */
	new_node->data = new_data;

	/* link the old list of the new node */
	new_node->next = (*head_ref);

	/* move the head to point to the new node */
	(*head_ref) = new_node;
}

// Driver's Code
int main()
{
    printf("ABHAY PANDEY  2100320120004\n");
	/* Start with the empty list */
	struct Node* head = NULL;

	// create linked 35->15->4->20
	push(&head, 20);
	push(&head, 4);
	push(&head, 15);
	push(&head, 35);

	// Function call
	printNthFromLast(head, 4);
```

```
	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

35

```
//Program for Sorting the Linear Linked List
#include <stdio.h>

//Represent a node of the singly linked list
struct node{
   int data;
   struct node *next;
};

//Represent the head and tail of the singly linked list
struct node *head, *tail = NULL;

//addNode() will add a new node to the list
void addNode(int data) {
   //Create a new node
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   newNode->next = NULL;

   //Checks if the list is empty
   if(head == NULL) {
      //If list is empty, both head and tail will point to new node
      head = newNode;
      tail = newNode;
   }
   else {
      //newNode will be added after tail such that tail's next will point to newNode
      tail->next = newNode;
      //newNode will become new tail of the list
      tail = newNode;
   }
}
```

```
    //sortList() will sort nodes of the list in ascending order
    void sortList() {
        //Node current will point to head
        struct node *current = head, *index = NULL;
        int temp;

        if(head == NULL) {
            return;
        }
        else {
            while(current != NULL) {
                //Node index will point to node next to current
                index = current->next;

                while(index != NULL) {
                    //If current node's data is greater than index's node data, swap the data between them
                    if(current->data > index->data) {
                        temp = current->data;
                        current->data = index->data;
                        index->data = temp;
                    }
                    index = index->next;
                }
                current = current->next;
            }
        }
    }

//display() will display all the nodes present in the list
void display() {
    //Node current will point to head
```

```c
    struct node *current = head;
    if(head == NULL) {
        printf("List is empty \n");
        return;
    }
    while(current != NULL) {
        //Prints each node by incrementing pointer
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main()
{
     printf("ABHAY PANDEY  2100320120004\n");
    //Adds data to the list
    addNode(9);
    addNode(7);
    addNode(2);
    addNode(5);
    addNode(4);

    //Displaying original list
    printf("Original list: \n");
    display();

    //Sorting list
    sortList();

    //Displaying sorted list
    printf("Sorted list: \n");
```

```
    display();


    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Original list:

9 7 2 5 4

Sorted list:

2 4 5 7 9

LAB - 21

```
// Program to Detect if there is ay cycle in the linked list, starting point of cycle, length of cycle
#include <stdio.h>
#include <stdlib.h>

/* A structure of linked list node */
struct node {
  int data;
  struct node *next;
} *head;

void initialize(){
   head = NULL;
}

/*
Given a Inserts a node in front of a singly linked list.
*/
void insert(int num) {
   /* Create a new Linked List node */
   struct node* newNode = (struct node*) malloc(sizeof(struct node));
   newNode->data  = num;
   /* Next pointer of new node will point to head node of linked list  */
   newNode->next = head;
   /* make new node as new head of linked list */
   head = newNode;
   printf("Inserted Element : %d\n", num);
}

void findloop(struct node *head) {
   struct node *slow, *fast;
   slow = fast = head;
```

```
    while(slow && fast && fast->next) {
        /* Slow pointer will move one node per iteration whereas
        fast node will move two nodes per iteration */
        slow = slow->next;
        fast  = fast->next->next;
        if (slow == fast) {
           printf("Linked List contains a loop\n");
           return;
        }
    }
    printf("No Loop in Linked List\n");
}
/*
 Prints a linked list from head node till tail node
*/
void printLinkedList(struct node *nodePtr) {
  while (nodePtr != NULL) {
     printf("%d", nodePtr->data);
     nodePtr = nodePtr->next;
     if(nodePtr != NULL)
         printf("-->");
  }
}

int main() {
              printf("ABHAY PANDEY  2100320120004\n");
    initialize();
    /* Creating a linked List*/
    insert(8);
    insert(3);
    insert(2);
```

```
    insert(7);
    insert(9);

    /* Create loop in linked list. Set next pointer of last node to second node from head */
    head->next->next->next->next->next = head->next;

    findloop(head);
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Inserted Element : 8

Inserted Element : 3

Inserted Element : 2

Inserted Element : 7

Inserted Element : 9

Linked List contains a loop

```
//   Program for Delete duplicate nodes in the Linked List
#include <stdio.h>

//Represent a node of the singly linked list
struct node{
   int data;
   struct node *next;
};

//Represent the head and tail of the singly linked list
struct node *head, *tail = NULL;

//addNode() will add a new node to the list
void addNode(int data) {
   //Create a new node
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   newNode->data = data;
   newNode->next = NULL;

   //Checks if the list is empty
   if(head == NULL) {
      //If list is empty, both head and tail will point to new node
      head = newNode;
      tail = newNode;
   }
   else {
      //newNode will be added after tail such that tail's next will point to newNode
      tail->next = newNode;
      //newNode will become new tail of the list
      tail = newNode;
   }
}
```

```
//removeDuplicate() will remove duplicate nodes from the list
void removeDuplicate() {
   //Node current will point to head
   struct node *current = head, *index = NULL, *temp = NULL;

   if(head == NULL) {
      return;
   }
   else {
      while(current != NULL){
         //Node temp will point to previous node to index.
         temp = current;
         //Index will point to node next to current
         index = current->next;

         while(index != NULL) {
            //If current node's data is equal to index node's data
            if(current->data == index->data) {
               //Here, index node is pointing to the node which is duplicate of current node
               //Skips the duplicate node by pointing to next node
               temp->next = index->next;
            }
            else {
               //Temp will point to previous node of index.
               temp = index;
            }
            index = index->next;
         }
         current = current->next;
      }
   }
```

```
}

//display() will display all the nodes present in the list
void display() {
   //Node current will point to head
   struct node *current = head;
   if(head == NULL) {
      printf("List is empty \n");
      return;
   }
   while(current != NULL) {
      //Prints each node by incrementing pointer
      printf("%d ", current->data);
      current = current->next;
   }
   printf("\n");
}

int main()
{
   printf("ABHAY PANDEY  2100320120004\n");
   //Adds data to the list
   addNode(1);
   addNode(2);
   addNode(3);
   addNode(2);
   addNode(2);
   addNode(4);
   addNode(1);

   printf("Originals list: \n");
   display();
```

```
    //Removes duplicate nodes
    removeDuplicate();

    printf("List after removing duplicates: \n");
    display();

    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Originals list:

1 2 3 2 2 4 1

List after removing duplicates:

1 2 3 4

```
// Program for Linked List Implementaion of Priority Queue

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

// Node
typedef struct node {
	int data;

	// Lower values indicate higher priority
	int priority;

	struct node* next;

} Node;

// Function to Create A New Node
Node* newNode(int d, int p)
{
	Node* temp = (Node*)malloc(sizeof(Node));
	temp->data = d;
	temp->priority = p;
	temp->next = NULL;

	return temp;
}

// Return the value at head
int peek(Node** head)
{
	return (*head)->data;
}
```

```
// Removes the element with the
// highest priority from the list
void pop(Node** head)
{
	Node* temp = *head;
	(*head) = (*head)->next;
	free(temp);
}

// Function to push according to priority
void push(Node** head, int d, int p)
{
	Node* start = (*head);

	// Create new Node
	Node* temp = newNode(d, p);

	// Special Case: The head of list has lesser
	// priority than new node. So insert new
	// node before head node and change head node.
	if ((*head)->priority > p) {

		// Insert New Node before head
		temp->next = *head;
		(*head) = temp;
	}
	else {

		// Traverse the list and find a
		// position to insert new node
		while (start->next != NULL &&
```

```
                start->next->priority < p) {
                start = start->next;
            }

            // Either at the ends of the list
            // or at required position
            temp->next = start->next;
            start->next = temp;
        }
}

// Function to check is list is empty
int isEmpty(Node** head)
{
        return (*head) == NULL;
}

// Driver code
int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
        // Create a Priority Queue
        // 7->4->5->6
        Node* pq = newNode(4, 1);
        push(&pq, 5, 2);
        push(&pq, 6, 3);
        push(&pq, 7, 0);

        while (!isEmpty(&pq)) {
            printf("%d ", peek(&pq));
            pop(&pq);
        }
```

```
	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

7 4 5 6

// Program to arrange the consonats and vowel nodes of the linked list it in such a way that all the vowels nodes come before the consonats while maintaining the order of their arrival

```
#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

/* A linked list node */

struct Node

{

	char data;

	struct Node *next;

};


/* Function to add new node to the List */

struct Node *newNode(char key)

{

	struct Node *temp = (struct Node*)malloc(sizeof(struct Node));

	temp->data = key;

	temp->next = NULL;

	return temp;

}


// utility function to print linked list

void printlist(struct Node *head)

{

	if (! head)

	{

		printf("Empty list \n");

		return;

	}

	while (head != NULL)

	{
```

```
		printf("%c",head->data);
		if (head->next)
		printf("->");
		head = head->next;
	}
	printf("\n");
}

// utility function for checking vowel
bool isVowel(char x)
{
	return (x == 'a' || x == 'e' || x == 'i' ||
				x == 'o' || x == 'u');
}

/* function to arrange consonants and
vowels nodes */
struct Node *arrange(struct Node *head)
{
	struct Node *newHead = head;

	// for keep track of vowel
struct     Node *latestVowel;

	struct Node *curr = head;

	// list is empty
	if (head == NULL)
		return NULL;

	// We need to discover the first vowel
	// in the list. It is going to be the
```

```
// returned head, and also the initial
// latestVowel.
if (isVowel(head->data))

        // first element is a vowel. It will
        // also be the new head and the initial
        // latestVowel;
        latestVowel = head;

else
{

        // First element is not a vowel. Iterate
        // through the list until we find a vowel.
        // Note that curr points to the element
        // *before* the element with the vowel.
        while (curr->next != NULL &&
                !isVowel(curr->next->data))
                curr = curr->next;



        // This is an edge case where there are
        // only consonants in the list.
        if (curr->next == NULL)
                return head;

        // Set the initial latestVowel and the
        // new head to the vowel item that we found.
        // Relink the chain of consonants after
        // that vowel item:
        // old_head_consonant->consonant1->consonant2->
        // vowel->rest_of_list becomes
```

```
        // vowel->old_head_consonant->consonant1->
        // consonant2->rest_of_list
        latestVowel = newHead = curr->next;
        curr->next = curr->next->next;
        latestVowel->next = head;
}

// Now traverse the list. Curr is always the item
// *before* the one we are checking, so that we
// can use it to re-link.
while (curr != NULL && curr->next != NULL)
{
        if (isVowel(curr->next->data))
        {
                // The next discovered item is a vowel
                if (curr == latestVowel)
                {
                        // If it comes directly after the
                        // previous vowel, we don't need to
                        // move items around, just mark the
                        // new latestVowel and advance curr.
                        latestVowel = curr = curr->next;
                }
                else
                {

                        // But if it comes after an intervening
                        // chain of consonants, we need to chain
                        .
                        struct Node *temp = latestVowel->next;

                        // Chain in new vowel
```

```
                    latestVowel->next = curr->next;

                    // Advance latestVowel
                    latestVowel = latestVowel->next;

                    // Remove found vowel from previous place
                    curr->next = curr->next->next;

                    // Re-link chain of consonants after latestVowel
                    latestVowel->next = temp;
                }
            }
            else
            {

                // No vowel in the next element, advance curr.
                curr = curr->next;
            }
        }
        return newHead;
}

// Driver code
int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
        struct Node *head = newNode('a');
        head->next = newNode('b');
        head->next->next = newNode('c');
        head->next->next->next = newNode('e');
        head->next->next->next->next = newNode('d');
        head->next->next->next->next->next = newNode('o');
```

```
    head->next->next->next->next->next->next = newNode('x');
    head->next->next->next->next->next->next->next = newNode('i');

    printf("Linked list before :\n");
    printlist(head);

    head = arrange(head);

    printf("Linked list after :\n");
    printlist(head);

    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Linked list before :

a->b->c->e->d->o->x->i

Linked list after :

a->e->o->i->b->c->d->x

```
// Program for Deletion of all occuraces of x from Linked List

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

// A linked list node
typedef struct Node {
	int data;
	struct Node* next;
} Node;

/* Given a reference (pointer to pointer) to the head of a
list and an int, inserts a new node on the front of the
list. */
void push(struct Node** head_ref, int new_data)
{
	struct Node* new_node
		= (struct Node*)malloc(sizeof(struct Node));
	new_node->data = new_data;
	new_node->next = (*head_ref);
	(*head_ref) = new_node;
}

/* Given a reference (pointer to pointer) to the head of a
list and a key, deletes all occurrence of the given key
in linked list */
Node* deleteKey(Node* head, int x)
{
	if (!head)
		return head;
	// Until the head data is equal to the key move the head
	// pointer
```

```
    while (head && head->data == x)
        head = head->next;
    Node *curr = head, *prev = NULL;
    while (curr) {
        if (curr->data == x)
            prev->next = curr->next;
        else
            prev = curr;
        curr = curr->next;
    }
    return head;
}

// This function prints contents of linked list starting
// from the given node
void printList(Node* node)
{
    while (node != NULL) {
        printf(" %d ", node->data);
        node = node->next;
    }
}

// Driver code
int main()
{
    printf("ABHAY PANDEY  2100320120004\n");
    // Start with the empty list
    struct Node* head = NULL;

    push(&head, 7);
    push(&head, 2);
```

```
	push(&head, 3);
	push(&head, 2);
	push(&head, 8);
	push(&head, 1);
	push(&head, 2);
	push(&head, 2);

	int key = 2; // key to delete

	puts("Created Linked List: ");
	printList(head);

	// Function call
	head = deleteKey(head, key);
	if (!head)
		printf("\nNo element present in the Linked list\n");
	else {
		printf("\nLinked List after Deletion is:\n");
		printList(head);
	}
	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Created Linked List:

2 2 1 8 2 3 2 7

Linked List after Deletion is:

1 8 3 7

// Program to Delete kth node from end of a linked list in a single scan and O(n) time

```
#include <assert.h>

#include <stdio.h>

#include <stdlib.h>


/* Link list node */

typedef struct Node {

	int data;

	struct Node* next;

} Node;


Node* deleteNode(Node* head, int key)

{


	// We will be using this pointer for holding address

	// temporarily while we delete the node

	Node* temp;


	// First pointer will point to the head of the linked

	// list

	Node* first = head;


	// Second pointer will point to the Nth node from the

	// beginning

	Node* second = head;

	for (int i = 0; i < key; i++) {


		// If count of nodes in the given linked list is <=N

		if (second->next == NULL) {


			// If count = N i.e. delete the head node
```

```
                if (i == key - 1) {
                    temp = head;
                    head = head->next;
                    free(temp);
                }
                return head;
            }
            second = second->next;
        }

        // Increment both the pointers by one until
        // second pointer reaches the end
        while (second->next != NULL) {
            first = first->next;
            second = second->next;
        }

        // First must be pointing to the Nth node from the end
        // by now So, delete the node first is pointing to
        temp = first->next;
        first->next = first->next->next;
        free(temp);
        return head;
}

/* Function to insert a node at the beginning of the
linked list */
void push(Node** head_ref, int new_data)
{
        /* allocate node */
        Node* new_node = (Node*)malloc(sizeof(Node));
        /* put in the data */
```

```
	new_node->data = new_data;
	/* link the old list off the new node */
	new_node->next = (*head_ref);
	/* move the head to point to the new node */
	(*head_ref) = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct Node* node)
{
	while (node != NULL) {
		printf("%d ", node->data);
		node = node->next;
	}
}

// Driver program
int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
	struct Node* head = NULL;
	push(&head, 7);
	push(&head, 1);
	push(&head, 3);
	push(&head, 2);
	printf("Created Linked list is:\n");
	printList(head);
	int n = 1;
	deleteNode(head, n);
	printf("\nLinked List after Deletion is:\n");
	printList(head);
	return 0;
```

}

OUTPUT:

ABHAY PANDEY 2100320120004

Created Linked list is:

2 3 1 7

Linked List after Deletion is:

2 3 1

LAB – 22

// Program to find out the addition of two given link list 125+85 =210 1->2->5 8->5

```
#include <stdio.h>

#include <stdlib.h>


// A linked List Node

struct Node {

	int data;

	struct Node* next;

};


typedef struct Node node;


/* A utility function to insert a

node at the beginning of

* linked list */

void push(struct Node** head_ref, int new_data)

{

	/* allocate node */

	struct Node* new_node

		= (struct Node*)malloc(sizeof(struct Node));


	/* put in the data */

	new_node->data = new_data;


	/* link the old list of the new node */

	new_node->next = (*head_ref);


	/* move the head to point to the new node */

	(*head_ref) = new_node;
```

```
}

/* A utility function to print linked list */
void printList(struct Node* node)
{
	while (node != NULL) {
		printf("%d ", node->data);
		node = node->next;
	}
	printf("n");
}

// A utility function to swap two pointers
void swapPointer(Node** a, Node** b)
{
	node* t = *a;
	*a = *b;
	*b = t;
}

/* A utility function to get size
of linked list */
int getSize(struct Node* node)
{
	int size = 0;
	while (node != NULL) {
		node = node->next;
		size++;
	}
	return size;
}
```

```
// Adds two linked lists of same
// size represented by head1
// and head2 and returns head of
// the resultant linked list.
// Carry is propagated while
// returning from the recursion
node* addSameSize(Node* head1,
                                        Node* head2, int* carry)
{
          // Since the function assumes
          // linked lists are of same
          // size, check any of the two
          // head pointers
          if (head1 == NULL)
                    return NULL;

          int sum;

          // Allocate memory for sum
          // node of current two nodes
          Node* result = (Node*)malloc(sizeof(Node));

          // Recursively add remaining nodes
          // and get the carry
          result->next
                    = addSameSize(head1->next,
                                        head2->next, carry);

          // add digits of current nodes
          // and propagated carry
          sum = head1->data + head2->data + *carry;
          *carry = sum / 10;
```

```
    sum = sum % 10;

    // Assigne the sum to current
    // node of resultant list
    result->data = sum;

    return result;
}

// This function is called after
// the smaller list is added
// to the bigger lists's sublist
// of same size. Once the
// right sublist is added, the
// carry must be added toe left
// side of larger list to get
// the final result.
void addCarryToRemaining(Node* head1,
                         Node* cur, int* carry,
                         Node** result)
{
    int sum;

    // If diff. number of nodes are
    // not traversed, add carry
    if (head1 != cur) {
        addCarryToRemaining(head1->next,
                            cur, carry,
                            result);

        sum = head1->data + *carry;
        *carry = sum / 10;
```

```
		sum %= 10;

		// add this node to the front of the result
		push(result, sum);
	}
}

// The main function that adds two
// linked lists represented
// by head1 and head2. The sum of
// two lists is stored in a
// list referred by result
void addList(Node* head1,
			Node* head2, Node** result)
{
	Node* cur;

	// first list is empty
	if (head1 == NULL) {
		*result = head2;
		return;
	}

	// second list is empty
	else if (head2 == NULL)
	{
		*result = head1;
		return;
	}

	int size1 = getSize(head1);
	int size2 = getSize(head2);
```

```
    int carry = 0;

    // Add same size lists
    if (size1 == size2)
        *result = addSameSize(head1, head2, &carry);

    else {
        int diff = abs(size1 - size2);

        // First list should always be
        // larger than second
        // list. If not, swap pointers
        if (size1 < size2)
            swapPointer(&head1, &head2);

        // move diff. number of nodes in first list
        for (cur = head1; diff--; cur = cur->next)
            ;

        // get addition of same size lists
        *result = addSameSize(cur,head2, &carry);

        // get addition of remaining first list and carry
        addCarryToRemaining(head1, cur, &carry, result);
    }

    // if some carry is still there, add a new node to the
    // front of the result list. e.g. 999 and 87
    if (carry)
        push(result, carry);
}
```

```
// Driver code
int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
	Node *head1 = NULL, *head2 = NULL, *result = NULL;

	int arr1[] = { 9, 9, 9 };
	int arr2[] = { 1, 8 };

	int size1 = sizeof(arr1) / sizeof(arr1[0]);
	int size2 = sizeof(arr2) / sizeof(arr2[0]);

	// Create first list as 9->9->9
	int i;
	for (i = size1 - 1; i >= 0; --i)
		push(&head1, arr1[i]);

	// Create second list as 1->8
	for (i = size2 - 1; i >= 0; --i)
		push(&head2, arr2[i]);

	addList(head1, head2, &result);

	printList(result);

	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

1 0 1 7 n

```
// Program to find out the substraction of two given link list

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

// A linked List Node
typedef struct Node {
	int data;
	struct Node* next;
} Node;

// A utility which creates Node.
Node* newNode(int data)
{
	Node* temp = (Node*)malloc(sizeof(Node));
	temp->data = data;
	temp->next = NULL;
	return temp;
}

/* A utility function to get length
of linked list */
int getLength(Node* Node)
{
	int size = 0;
	while (Node != NULL) {
		Node = Node->next;
		size++;
	}
	return size;
}
```

```
/* A Utility that padds zeros in front of the
Node, with the given diff */
Node* paddZeros(Node* sNode, int diff)
{
	if (sNode == NULL)
		return NULL;

	Node* zHead = newNode(0);
	diff--;
	Node* temp = zHead;
	while (diff--) {
		temp->next = newNode(0);
		temp = temp->next;
	}
	temp->next = sNode;
	return zHead;
}

/* Subtract LinkedList Helper is a recursive function,
move till the last Node, and subtract the digits and
create the Node and return the Node. If d1 < d2, we
borrow the number from previous digit. */
static bool borrow;
Node* subtractLinkedListHelper(Node* l1, Node* l2)
{

	if (l1 == NULL && l2 == NULL && borrow == 0)
		return NULL;

	Node* previous = subtractLinkedListHelper(
		l1 ? l1->next : NULL, l2 ? l2->next : NULL);
```

```
        int d1 = l1->data;
        int d2 = l2->data;
        int sub = 0;

        /* if you have given the value to next digit then
        reduce the d1 by 1 */
        if (borrow) {
                d1--;
                borrow = false;
        }

        /* If d1 < d2, then borrow the number from previous
        digit. Add 10 to d1 and set borrow = true; */
        if (d1 < d2) {
                borrow = true;
                d1 = d1 + 10;
        }

        /* subtract the digits */
        sub = d1 - d2;

        /* Create a Node with sub value */
        Node* current = newNode(sub);

        /* Set the Next pointer as Previous */
        current->next = previous;

        return current;
}

/* This API subtracts two linked lists and returns the
```

linked list which shall have the subtracted result. */

```
Node* subtractLinkedList(Node* l1, Node* l2)
{
	// Base Case.
	if (l1 == NULL && l2 == NULL)
		return NULL;

	// In either of the case, get the lengths of both
	// Linked list.
	int len1 = getLength(l1);
	int len2 = getLength(l2);

	Node *lNode = NULL, *sNode = NULL;

	Node* temp1 = l1;
	Node* temp2 = l2;

	// If lengths differ, calculate the smaller Node
	// and padd zeros for smaller Node and ensure both
	// larger Node and smaller Node has equal length.
	if (len1 != len2) {
		lNode = len1 > len2 ? l1 : l2;
		sNode = len1 > len2 ? l2 : l1;
		sNode = paddZeros(sNode, abs(len1 - len2));
	}

	else {
		// If both list lengths are equal, then calculate
		// the larger and smaller list. If 5-6-7 & 5-6-8
		// are linked list, then walk through linked list
		// at last Node as 7 < 8, larger Node is 5-6-8
		// and smaller Node is 5-6-7.
```

```
		while (l1 && l2) {
			if (l1->data != l2->data) {
				lNode = l1->data > l2->data ? temp1 : temp2;
				sNode = l1->data > l2->data ? temp2 : temp1;
				break;
			}
			l1 = l1->next;
			l2 = l2->next;
		}
	}
	// If both lNode and sNode still have NULL value,
	// then this means that the value of both of the given
	// linked lists is the same and hence we can directly
	// return a node with value 0.
	if (lNode == NULL && sNode == NULL) {
		return newNode(0);
	}
	// After calculating larger and smaller Node, call
	// subtractLinkedListHelper which returns the subtracted
	// linked list.
	borrow = false;
	return subtractLinkedListHelper(lNode, sNode);
}

/* A utility function to print linked list */
void printList(struct Node* Node)
{
	while (Node != NULL) {
		printf("%d ", Node->data);
		Node = Node->next;
	}
	printf("\n");
```

```
}

// Driver program to test above functions
int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
	Node* head1 = newNode(1);
	head1->next = newNode(0);
	head1->next->next = newNode(0);

	Node* head2 = newNode(1);

	Node* result = subtractLinkedList(head1, head2);

	printList(result);

	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

0 9 9

```
// Program for Polynomial Addition using Linked List
#include<stdio.h>
#include<stdlib.h>

struct Node {
   int coeff;
   int pow;
   struct Node* next;
};

// Function to create new node
void create_node(int x, int y, struct Node** temp)
{
   struct Node *r, *z;
   z = *temp;
   if (z == NULL) {
      r = (struct Node*)malloc(sizeof(struct Node));
      r->coeff = x;
      r->pow = y;
      *temp = r;
      r->next = (struct Node*)malloc(sizeof(struct Node));
      r = r->next;
      r->next = NULL;
   }
   else {
      r->coeff = x;
      r->pow = y;
      r->next = (struct Node*)malloc(sizeof(struct Node));
      r = r->next;
      r->next = NULL;
   }
}
```

```
// Function Adding two polynomial numbers
void polyadd(struct Node* poly1, struct Node* poly2,
          struct Node* poly)
{
   while (poly1->next && poly2->next) {
      if (poly1->pow > poly2->pow) {
         poly->pow = poly1->pow;
         poly->coeff = poly1->coeff;
         poly1 = poly1->next;
      }

      else if (poly1->pow < poly2->pow) {
         poly->pow = poly2->pow;
         poly->coeff = poly2->coeff;
         poly2 = poly2->next;
      }
      else {
         poly->pow = poly1->pow;
         poly->coeff = poly1->coeff + poly2->coeff;
         poly1 = poly1->next;
         poly2 = poly2->next;
      }

      // Dynamically create new node
      poly->next
         = (struct Node*)malloc(sizeof(struct Node));
      poly = poly->next;
      poly->next = NULL;
   }
   while (poly1->next || poly2->next) {
      if (poly1->next) {
```

```
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if (poly2->next) {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
}

// Display Linked list
void show(struct Node* node)
{
    while (node->next != NULL) {
        printf("%dx^%d", node->coeff, node->pow);
        node = node->next;
        if (node->coeff >= 0) {
            if (node->next != NULL)
                printf("+");
        }
    }
}

// Driver code
int main()
{
```

```
        printf("ABHAY PANDEY  2100320120004\n");
    struct Node *poly1 = NULL, *poly2 = NULL, *poly = NULL;

    // Create first list of 5x^2 + 4x^1 + 2x^0
    create_node(5, 2, &poly1);
    create_node(4, 1, &poly1);
    create_node(2, 0, &poly1);

    // Create second list of -5x^1 - 5x^0
    create_node(-5, 1, &poly2);
    create_node(-5, 0, &poly2);

    printf("1st Number: ");
    show(poly1);

    printf("\n2nd Number: ");
    show(poly2);

    poly = (struct Node*)malloc(sizeof(struct Node));

    // Function add two polynomial numbers
    polyadd(poly1, poly2, poly);

    // Display resultant List
    printf("\nAdded polynomial: ");
    show(poly);

    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

1st Number: 5x^2+4x^1+2x^0

2nd Number: -5x^1-5x^0

Added polynomial: 5x^2-1x^1-3x^0

```c
// Program for Polynomial Multiplication using Linked List
#include <stdio.h>
#include <stdlib.h>

typedef struct Node
{
   // Define useful field of Node
   int data;
   int power;
   struct Node * next;
}Node;

Node * getNode(int data, int power)
{
   // Create dynamic memory of Node
   Node * ref = (Node * ) malloc(sizeof(Node));
   if (ref == NULL)
   {
      // Failed to create memory
      return NULL;
   }
   ref->data = data;
   ref->power = power;
   ref->next = NULL;
   return ref;
}
// Update node value
void updateRecord(Node * ref, int data, int power)
{
   ref->data = data;
   ref->power = power;
}
```

```
typedef struct MultiplyPolynomial
{
   // Define useful field of MultiplyPolynomial
   struct Node * head;
}MultiplyPolynomial;

MultiplyPolynomial * getMultiplyPolynomial()
{
   // Create dynamic memory of MultiplyPolynomial
   MultiplyPolynomial * ref = (MultiplyPolynomial * )
                malloc(sizeof(MultiplyPolynomial));
   if (ref == NULL)
   {
      // Failed to create memory
      return NULL;
   }
   ref->head = NULL;
   return ref;
}
// Insert Node element
void insert(MultiplyPolynomial * ref, int data, int power)
{
   if (ref->head == NULL)
   {
      // Add first node
      ref->head = getNode(data, power);
   }
   else
   {
      Node * node = NULL;
      Node * temp = ref->head;
      Node * location = NULL;
```

```
        // Find the valid new node location
        while (temp != NULL && temp->power >= power)
        {
            location = temp;
            temp = temp->next;
        }
        if (location != NULL && location->power == power)
        {
            // When polynomial power already exists
            // Then add current add to previous data
            location->data = location->data + data;
        }
        else
        {
            node = getNode(data, power);
            if (location == NULL)
            {
                // When add node in begining
                node->next = ref->head;
                ref->head = node;
            }
            else
            {
                // When adding node in intermediate
                // location or end location
                node->next = location->next;
                location->next = node;
            }
        }
    }
}
// Perform multiplication of given polynomial
```

```
MultiplyPolynomial * multiplyPolynomials(
  MultiplyPolynomial * ref, MultiplyPolynomial * other)
{
    // Define some useful variable
    MultiplyPolynomial * result = getMultiplyPolynomial();
    // Get first node of polynomial
    Node * poly1 = ref->head;
    Node * temp = other->head;
    int power_value = 0;
    int coefficient = 0;
    // Execute loop until when polynomial are exist
    while (poly1 != NULL)
    {
        temp = other->head;
        while (temp != NULL)
        {
            // Get result info
            power_value = poly1->power + temp->power;
            coefficient = poly1->data * temp->data;
            insert(result, coefficient, power_value);
            // Visit to next node
            temp = temp->next;
        }
        // Visit to next node
        poly1 = poly1->next;
    }
    // return first node
    return result;
}
// Display given polynomial nodes
void display(MultiplyPolynomial * ref)
{
```

```
    if (ref->head == NULL)
    {
        printf("Empty Polynomial ");
    }
    printf(" ");
    Node * temp = ref->head;
    while (temp != NULL)
    {
        if (temp != ref->head)
        {
            printf(" + %d", temp->data);
        }
        else
        {
            printf("%d",temp->data);
        }
        if (temp->power != 0)
        {
            printf("x^%d", temp->power);
        }
        // Visit to next node
        temp = temp->next;
    }
    printf("\n");
}
int main()
{
            printf("ABHAY PANDEY  2100320120004\n");
    MultiplyPolynomial * a = getMultiplyPolynomial();
    MultiplyPolynomial * b = getMultiplyPolynomial();
    // Add node in polynomial A
    insert(a, 9, 3);
```

```
    insert(a, 4, 2);
    insert(a, 3, 0);
    insert(a, 7, 1);
    insert(a, 3, 4);
    // Add node in polynomial b
    insert(b, 7, 3);
    insert(b, 4, 0);
    insert(b, 6, 1);
    insert(b, 1, 2);
    // Display Polynomial nodes
    printf("\n Polynomial A\n");
    display(a);
    printf(" Polynomial B\n");
    display(b);
    MultiplyPolynomial * result = multiplyPolynomials(a, b);
    // Display calculated result
    printf(" Result\n");
    display(result);
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
 Polynomial A
 3x^4 + 9x^3 + 4x^2 + 7x^1 + 3
 Polynomial B
 7x^3 + 1x^2 + 6x^1 + 4
```

Result

21x^7 + 66x^6 + 55x^5 + 119x^4 + 88x^3 + 61x^2 + 46x^1 + 12

LAB – 23

// Program for Circular Linked List Primitive Operations

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

struct Node {
   int coeff;
   int pow;
   struct Node* next;
};

// Function to create new node
void create_node(int x, int y, struct Node** temp)
{
   struct Node *r, *z;
   z = *temp;
   if (z == NULL) {
      r = (struct Node*)malloc(sizeof(struct Node));
      r->coeff = x;
      r->pow = y;
      *temp = r;
      r->next = (struct Node*)malloc(sizeof(struct Node));
      r = r->next;
      r->next = NULL;
   }
   else {
      r->coeff = x;
      r->pow = y;
      r->next = (struct Node*)malloc(sizeof(struct Node));
      r = r->next;
      r->next = NULL;
   }
```

```
}

// Function Adding two polynomial numbers
void polyadd(struct Node* poly1, struct Node* poly2,
          struct Node* poly)
{
   while (poly1->next && poly2->next) {
      if (poly1->pow > poly2->pow) {
         poly->pow = poly1->pow;
         poly->coeff = poly1->coeff;
         poly1 = poly1->next;
      }

      else if (poly1->pow < poly2->pow) {
         poly->pow = poly2->pow;
         poly->coeff = poly2->coeff;
         poly2 = poly2->next;
      }
      else {
         poly->pow = poly1->pow;
         poly->coeff = poly1->coeff + poly2->coeff;
         poly1 = poly1->next;
         poly2 = poly2->next;
      }

      // Dynamically create new node
      poly->next
         = (struct Node*)malloc(sizeof(struct Node));
      poly = poly->next;
      poly->next = NULL;
   }
   while (poly1->next || poly2->next) {
```

```
        if (poly1->next) {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        if (poly2->next) {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        poly->next
            = (struct Node*)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
}

// Display Linked list
void show(struct Node* node)
{
    while (node->next != NULL) {
        printf("%dx^%d", node->coeff, node->pow);
        node = node->next;
        if (node->coeff >= 0) {
            if (node->next != NULL)
                printf("+");
        }
    }
}

// Driver code
int main()
```

```
{
        printf("ABHAY PANDEY  2100320120004\n");
    struct Node *poly1 = NULL, *poly2 = NULL, *poly = NULL;

    // Create first list of 5x^2 + 4x^1 + 2x^0
    create_node(5, 2, &poly1);
    create_node(4, 1, &poly1);
    create_node(2, 0, &poly1);

    // Create second list of -5x^1 - 5x^0
    create_node(-5, 1, &poly2);
    create_node(-5, 0, &poly2);

    printf("1st Number: ");
    show(poly1);

    printf("\n2nd Number: ");
    show(poly2);

    poly = (struct Node*)malloc(sizeof(struct Node));

    // Function add two polynomial numbers
    polyadd(poly1, poly2, poly);

    // Display resultant List
    printf("\nAdded polynomial: ");
    show(poly);

    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

1st Number: 5x^2+4x^1+2x^0

2nd Number: -5x^1-5x^0

Added polynomial: 5x^2-1x^1-3x^0

```
// Program for concatenation of Circular Linked List
#include<stdio.h>
#include<stdlib.h>

struct node
{
   int info;
   struct node *link;
};

struct node *create_list(struct node *last);
void display(struct node *last);
struct node *addtoempty(struct node *last,int data );
struct node *addatend(struct node *last,int data);
struct node *concat(struct node *last1,struct node *last2);

int main( )
{
        printf("ABHAY PANDEY  2100320120004\n");
   struct node *last1=NULL,*last2=NULL;
   last1=create_list(last1);
   last2=create_list(last2);
   printf("First list is :  ");
   display(last1);
   printf("Second list is :  ");
   display(last2);
   last1=concat(last1, last2);
   printf("Concatenated list is  : ");
   display(last1);
   return 0;
}
```

```
struct node *concat( struct node *last1,struct node *last2)
{
   struct node *ptr;
   if(last1==NULL)
   {
      last1=last2;
      return last1;
   }
   if(last2==NULL )
      return last1;
   ptr=last1->link;
   last1->link=last2->link;
   last2->link=ptr;
   last1=last2;
   return last1;
}
struct node *create_list(struct node *last)
{
   int i,n;
   int data;
   printf("Enter the number of nodes : ");
   scanf("%d",&n);
   last=NULL;
   if(n==0)
      return last;
   printf("Enter the element to be inserted : ");
   scanf("%d",&data);
   last=addtoempty(last,data);

   for(i=2;i<=n;i++)
   {
      printf("Enter the element to be inserted : ");
```

```
        scanf("%d",&data);
        last=addatend(last,data);
    }
    return last;
}

void display(struct node *last)
{
    struct node *p;
    if(last==NULL)
    {
        printf("List is empty\n");
        return;
    }
    p=last->link;  /*p points to first node*/
    do
    {
        printf("%d ", p->info);
        p=p->link;
    }while(p!=last->link);
    printf("\n");
}

struct node *addtoempty(struct node *last,int data)
{
    struct node *tmp;
    tmp = (struct node *)malloc(sizeof(struct node));
    tmp->info = data;
    last = tmp;
    last->link = last;
    return last;
}
```

```
struct node *addatend(struct node *last,int data)
{
   struct node *tmp;
   tmp = (struct node *)malloc(sizeof(struct node));
   tmp->info = data;
   tmp->link = last->link;
   last->link = tmp;
   last = tmp;
   return last;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
Enter the number of nodes : 2
Enter the element to be inserted : 4
Enter the element to be inserted : 2
Enter the number of nodes : 5
Enter the element to be inserted : 3
Enter the element to be inserted : 2
Enter the element to be inserted : 2
Enter the element to be inserted : 5
Enter the element to be inserted :
6
First list is :  4 2
Second list is :  3 2 2 5 6
Concatenated list is  : 4 2 3 2 2 5 6
```

```c
// Program for implementation of Josephus Problem
#include <stdio.h>
#include <stdlib.h>

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

void create(struct node **);
void display(struct node *);
int survivor(struct node **, int);

int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
    struct node *head = NULL;
    int survive, skip;

    create(&head);
    printf("The persons in circular list are:\n");
    display(head);
    printf("Enter the number of persons to be skipped: ");
    scanf("%d", &skip);
    survive = survivor(&head, skip);
    printf("The person to survive is : %d\n", survive);
    free(head);

    return 0;
}
```

```
int survivor(struct node **head, int k)
{
    struct node *p, *q;
    int i;

    q = p = *head;
    while (p->next != p)
    {
        for (i = 0; i < k - 1; i++)
        {
            q = p;
            p = p->next;
        }
        q->next = p->next;
        printf("%d has been killed.\n", p->num);
        free(p);
        p = q->next;
    }
    *head = p;

    return (p->num);
}

void create (struct node **head)
{
    struct node *temp, *rear;
    int a, ch;

    do
    {
        printf("Enter a number: ");
        scanf("%d", &a);
```

```
        temp = (struct node *)malloc(sizeof(struct node));
        temp->num = a;
        temp->next = NULL;
        if (*head == NULL)
        {
            *head = temp;
        }
        else
        {
            rear->next = temp;
        }
        rear = temp;
        printf("Do you want to add a number [1/0]? ");
        scanf("%d", &ch);
    } while (ch != 0);
    rear->next = *head;
}

void display(struct node *head)
{
    struct node *temp;

    temp = head;
    printf("%d   ", temp->num);
    temp = temp->next;
    while (head != temp)
    {
        printf("%d   ", temp->num);
        temp = temp->next;
    }
    printf("\n");
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Enter a number: 4

Do you want to add a number [1/0]? 1

Enter a number: 6

Do you want to add a number [1/0]? 1

Enter a number: 8

Do you want to add a number [1/0]? 1

Enter a number: 7

Do you want to add a number [1/0]? 1

Enter a number: 5

Do you want to add a number [1/0]? 1

Enter a number: 5

Do you want to add a number [1/0]? 1

Enter a number: 5

Do you want to add a number [1/0]? 1

Enter a number: 6

Do you want to add a number [1/0]? 1

Enter a number: 6

Do you want to add a number [1/0]? 0

The persons in circular list are:

4   6   8   7   5   5   5   6   6

Enter the number of persons to be skipped: 2

6 has been killed.

7 has been killed.

5 has been killed.

6 has been killed.

4 has been killed.

5 has been killed.

6 has been killed.

5 has been killed.

The person to survive is : 8

LAB – 24

```
// Program for Doubly linked list Primitive operations


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


// Linked List Node
struct node {
	int info;
	struct node *prev, *next;
};
struct node* start = NULL;


// Function to traverse the linked list
void traverse()
{
	// List is empty
	if (start == NULL) {
		printf("\nList is empty\n");
		return;
	}
	// Else print the Data
	struct node* temp;
	temp = start;
	while (temp != NULL) {
		printf("Data = %d\n", temp->info);
		temp = temp->next;
	}
}


// Function to insert at the front
// of the linked list
```

```
void insertAtFront()
{
	int data;
	struct node* temp;
	temp = (struct node*)malloc(sizeof(struct node));
	printf("\nEnter number to be inserted: ");
	scanf("%d", &data);
	temp->info = data;
	temp->prev = NULL;

	// Pointer of temp will be
	// assigned to start
	temp->next = start;
	start = temp;
}

// Function to insert at the end of
// the linked list
void insertAtEnd()
{
	int data;
	struct node *temp, *trav;
	temp = (struct node*)malloc(sizeof(struct node));
	temp->prev = NULL;
	temp->next = NULL;
	printf("\nEnter number to be inserted: ");
	scanf("%d", &data);
	temp->info = data;
	temp->next = NULL;
	trav = start;

	// If start is NULL
```

```
        if (start == NULL) {

                start = temp;
        }

        // Changes Links
        else {
                while (trav->next != NULL)
                        trav = trav->next;
                temp->prev = trav;
                trav->next = temp;
        }
}

// Function to insert at any specified
// position in the linked list
void insertAtPosition()
{
        int data, pos, i = 1;
        struct node *temp, *newnode;
        newnode = malloc(sizeof(struct node));
        newnode->next = NULL;
        newnode->prev = NULL;

        // Enter the position and data
        printf("\nEnter position : ");
        scanf("%d", &pos);


        // If start==NULL,
        if (start == NULL) {
                start = newnode;
```

```
                newnode->prev = NULL;
                newnode->next = NULL;
        }

        // If position==1,
        else if (pos == 1) {
        // this is author method its correct but we can simply call insertAtfront() function for this special case
        /* newnode->next = start;
                newnode->next->prev = newnode;
                newnode->prev = NULL;
                start = newnode; */
        // now this is improved by Jay Ghughriwala on geeksforgeeks
        insertAtFront();
        }

        // Change links
        else {
        printf("\nEnter number to be inserted: ");
        scanf("%d", &data);
        newnode->info = data;
        temp = start;
                while (i < pos - 1) {
                        temp = temp->next;
                        i++;
                }
                newnode->next = temp->next;
                newnode->prev = temp;
                temp->next = newnode;
                temp->next->prev = newnode;
        }
}
```

```
// Function to delete from the front
// of the linked list
void deleteFirst()
{
	struct node* temp;
	if (start == NULL)
		printf("\nList is empty\n");
	else {
		temp = start;
		start = start->next;
		if (start != NULL)
			start->prev = NULL;
		free(temp);
	}
}

// Function to delete from the end
// of the linked list
void deleteEnd()
{
	struct node* temp;
	if (start == NULL)
		printf("\nList is empty\n");
	temp = start;
	while (temp->next != NULL)
		temp = temp->next;
	if (start->next == NULL)
		start = NULL;
	else {
		temp->prev->next = NULL;
		free(temp);
	}
```

```
}

// Function to delete from any specified
// position from the linked list
void deletePosition()
{
	int pos, i = 1;
	struct node *temp, *position;
	temp = start;

	// If DLL is empty
	if (start == NULL)
		printf("\nList is empty\n");

	// Otherwise
	else {
		// Position to be deleted
		printf("\nEnter position : ");
		scanf("%d", &pos);

		// If the position is the first node
		if (pos == 1) {
			deleteFirst(); // im,proved by Jay Ghughriwala on GeeksforGeeks
			if (start != NULL) {
				start->prev = NULL;
			}
			free(position);
			return;
		}

		// Traverse till position
		while (i < pos - 1) {
```

```
                temp = temp->next;
                i++;
            }
            // Change Links
            position = temp->next;
            if (position->next != NULL)
                position->next->prev = temp;
            temp->next = position->next;

            // Free memory
            free(position);
        }
}

// Driver Code
int main()
{
         printf("ABHAY PANDEY  2100320120004\n");
        int choice;
        while (1) {

            printf("\n\t1 To see list\n");
            printf("\t2 For insertion at"
                " starting\n");
            printf("\t3 For insertion at"
                " end\n");
            printf("\t4 For insertion at "
                "any position\n");
            printf("\t5 For deletion of "
                "first element\n");
            printf("\t6 For deletion of "
                "last element\n");
```

```
printf("\t7 For deletion of "
        "element at any position\n");
printf("\t8 To exit\n");
printf("\nEnter Choice :\n");
scanf("%d", &choice);

switch (choice) {
case 1:
        traverse();
        break;
case 2:
        insertAtFront();
        break;
case 3:
        insertAtEnd();
        break;
case 4:
        insertAtPosition();
        break;
case 5:
        deleteFirst();
        break;
case 6:
        deleteEnd();
        break;
case 7:
        deletePosition();
        break;

case 8:
        exit(1);
        break;
```

```
        default:
            printf("Incorrect Choice. Try Again \n");
            continue;
        }
    }
    return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
    1 To see list
    2 For insertion at starting
    3 For insertion at end
    4 For insertion at any position
    5 For deletion of first element
    6 For deletion of last element
    7 For deletion of element at any position
    8 To exit

Enter Choice :
1
List is empty


    1 To see list
    2 For insertion at starting
    3 For insertion at end
    4 For insertion at any position
    5 For deletion of first element
    6 For deletion of last element
    7 For deletion of element at any position
    8 To exit

Enter Choice :
```

2

Enter number to be inserted: 2

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice :

1

Data = 2

1 To see list

2 For insertion at starting

3 For insertion at end

4 For insertion at any position

5 For deletion of first element

6 For deletion of last element

7 For deletion of element at any position

8 To exit

Enter Choice : 8

```c
// Program for Circular Doubly Linked List Primitive Operations
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void deletion_beginning();
void deletion_last();
void display();
void search();
int main ()
{
 printf("ABHAY PANDEY  2100320120004\n");
int choice =0;
    while(choice != 9)
    {
        printf("\n*********Main Menu*********\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n===============================================\n");
        printf("\n1.Insert in Beginning\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
last\n5.Search\n6.Show\n7.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
```

```
        insertion_beginning();
        break;
        case 2:
            insertion_last();
        break;
        case 3:
        deletion_beginning();
        break;
        case 4:
        deletion_last();
        break;
        case 5:
        search();
        break;
        case 6:
        display();
        break;
        case 7:
        exit(0);
        break;
        default:
        printf("Please enter valid choice..");
      }
    }
    return 0;
}
void insertion_beginning()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
```

```
  {
     printf("\nOVERFLOW");
  }
  else
  {
   printf("\nEnter Item value");
   scanf("%d",&item);
   ptr->data=item;
  if(head==NULL)
  {
    head = ptr;
    ptr -> next = head;
    ptr -> prev = head;
  }
  else
  {
     temp = head;
   while(temp -> next != head)
   {
      temp = temp -> next;
   }
   temp -> next = ptr;
   ptr -> prev = temp;
   head -> prev = ptr;
   ptr -> next = head;
   head = ptr;
  }
  printf("\nNode inserted\n");
}

}
void insertion_last()
```

```
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *) malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nOVERFLOW");
  }
  else
  {
     printf("\nEnter value");
     scanf("%d",&item);
      ptr->data=item;
     if(head == NULL)
     {
        head = ptr;
        ptr -> next = head;
        ptr -> prev = head;
     }
     else
     {
       temp = head;
       while(temp->next !=head)
       {
          temp = temp->next;
       }
       temp->next = ptr;
       ptr ->prev=temp;
       head -> prev = ptr;
    ptr -> next = head;
      }
  }
```

```
    printf("\nnode inserted\n");
}

void deletion_beginning()
{
   struct node *temp;
   if(head == NULL)
   {
      printf("\n UNDERFLOW");
   }
   else if(head->next == head)
   {
      head = NULL;
      free(head);
      printf("\nnode deleted\n");
   }
   else
   {
      temp = head;
      while(temp -> next != head)
      {
         temp = temp -> next;
      }
      temp -> next = head -> next;
      head -> next -> prev = temp;
      free(head);
      head = temp -> next;
   }

}
void deletion_last()
{
```

```
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != head)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = head;
        head -> prev = ptr -> prev;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
```

```
    }
    else
    {
        printf("\n printing values ... \n");

        while(ptr -> next != head)
        {

            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }

}

void search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
        printf("item found at location %d",i+1);
```

```
        flag=0;
        }
        else
        {
        while (ptr->next != head)
        {
          if(ptr->data == item)
          {
            printf("item found at location %d ",i+1);
            flag=0;
            break;
          }
          else
          {
            flag=1;
          }
          i++;
          ptr = ptr -> next;
        }
        }
        if(flag != 0)
        {
          printf("Item not found\n");
        }
    }

}
```

OUTPUT:

ABHAY PANDEY 2100320120004

*********Main Menu*********

Choose one option from the following list ...

===

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

1

Enter Item value5

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

1

Enter Item value6

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

1

Enter Item value

8

Node inserted

*********Main Menu*********

Choose one option from the following list ...

===

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

6

printing values ...

8

6

5

*********Main Menu*********

Choose one option from the following list ...

===

1.Insert in Beginning

2.Insert at last

3.Delete from Beginning

4.Delete from last

5.Search

6.Show

7.Exit

Enter your choice?

7

```
// Program for Linked List Implementation of Stacks
#include <stdio.h>
#include <stdlib.h>
void push();
void pop();
void display();
struct node
{
int val;
struct node *next;
};
struct node *head;

int main ()
{
     printf("ABHAY PANDEY  2100320120004\n");
    int choice=0;
    printf("\n*********Stack operations using linked list*********\n");
    printf("\n----------------------------------------------\n");
    while(choice != 4)
    {
        printf("\n\nChose one from the below options...\n");
        printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
        printf("\n Enter your choice \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
```

```
        case 2:
        {
            pop();
            break;
        }
        case 3:
        {
            display();
            break;
        }
        case 4:
        {
            printf("Exiting....");
            break;
        }
        default:
        {
            printf("Please Enter valid choice ");
        }
    };
}
return 0;
}
void push ()
{
    int val;
    struct node *ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("not able to push the element");
    }
    else
```

```
    {
        printf("Enter the value");
        scanf("%d",&val);
        if(head==NULL)
        {
            ptr->val = val;
            ptr -> next = NULL;
            head=ptr;
        }
        else
        {
            ptr->val = val;
            ptr->next = head;
            head=ptr;

        }
        printf("Item pushed");

    }
}

void pop()
{
    int item;
    struct node *ptr;
    if (head == NULL)
    {
        printf("Underflow");
    }
    else
    {
        item = head->val;
```

```
        ptr = head;
        head = head->next;
        free(ptr);
        printf("Item popped");

    }
}
void display()
{
    int i;
    struct node *ptr;
    ptr=head;
    if(ptr == NULL)
    {
        printf("Stack is empty\n");
    }
    else
    {
        printf("Printing Stack elements \n");
        while(ptr!=NULL)
        {
            printf("%d\n",ptr->val);
            ptr = ptr->next;
        }
    }
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

*********Stack operations using linked list*********

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

1

Enter the value3

Item pushed

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

1

Enter the value6

Item pushed

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

3

Printing Stack elements

6

3

Chose one from the below options...

1.Push

2.Pop

3.Show

4.Exit

Enter your choice

4

Exiting....

```c
// Program for Linked List Implementaion of Queue

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

// A linked list (LL) node to store a queue entry
struct QNode {
	int key;
	struct QNode* next;
};

// The queue, front stores the front node of LL and rear
// stores the last node of LL
struct Queue {
	struct QNode *front, *rear;
};

// A utility function to create a new linked list node.
struct QNode* newNode(int k)
{
	struct QNode* temp
		= (struct QNode*)malloc(sizeof(struct QNode));
	temp->key = k;
	temp->next = NULL;
	return temp;
}

// A utility function to create an empty queue
struct Queue* createQueue()
{
	struct Queue* q
		= (struct Queue*)malloc(sizeof(struct Queue));
```

```
	q->front = q->rear = NULL;
	return q;
}

// The function to add a key k to q
void enQueue(struct Queue* q, int k)
{
	// Create a new LL node
	struct QNode* temp = newNode(k);

	// If queue is empty, then new node is front and rear
	// both
	if (q->rear == NULL) {
		q->front = q->rear = temp;
		return;
	}

	// Add the new node at the end of queue and change rear
	q->rear->next = temp;
	q->rear = temp;
}

// Function to remove a key from given queue q
void deQueue(struct Queue* q)
{
	// If queue is empty, return NULL.
	if (q->front == NULL)
		return;

	// Store previous front and move front one node ahead
	struct QNode* temp = q->front;
```

```
            q->front = q->front->next;

            // If front becomes NULL, then change rear also as NULL
            if (q->front == NULL)
                        q->rear = NULL;

            free(temp);
}
// Driver code
int main()
{
             printf("ABHAY PANDEY  2100320120004\n");
            struct Queue* q = createQueue();
            enQueue(q, 10);
            enQueue(q, 20);
            deQueue(q);
            deQueue(q);
            enQueue(q, 30);
            enQueue(q, 40);
            enQueue(q, 50);
            deQueue(q);
            printf("Queue Front : %d \n", ((q->front != NULL) ? (q->front)->key : -1));
            printf("Queue Rear : %d", ((q->rear != NULL) ? (q->rear)->key : -1));
            return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Queue Front : 40

Queue Rear : 50

// Program for Linked List implementation of Double Ended Queue

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

```
#include <stdlib.h>

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

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

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

 /*
  * create sentinel(dummy head & tail) that
  * helps us to do insertion and deletion
  * operation at front and rear so easily.  And
  * these dummy head and tail wont get deleted
  * till the end of execution of this program
  */

void createSentinels() {
    head = createNode(0);
    tail = createNode(0);
    head->next = tail;
    tail->prev = head;
}

/* insertion at the front of the queue */
```

```
void enqueueAtFront(int data) {
    struct node *newnode, *temp;
    newnode = createNode(data);
    temp = head->next;
    head->next = newnode;
    newnode->prev = head;
    newnode->next = temp;
    temp->prev = newnode;
}

/*insertion at the rear of the queue */
void enqueueAtRear(int data) {
    struct node *newnode, *temp;
    newnode = createNode(data);
    temp = tail->prev;
    tail->prev = newnode;
    newnode->next = tail;
    newnode->prev = temp;
    temp->next = newnode;
}

/* deletion at the front of the queue */
void dequeueAtFront() {
    struct node *temp;
    if (head->next == tail) {
        printf("Queue is empty\n");
    } else {
        temp = head->next;
        head->next = temp->next;
        temp->next->prev = head;
        free(temp);
    }
```

```
      return;
}



/* deletion at the rear of the queue */

void dequeueAtRear()  {
      struct node *temp;
      if (tail->prev == head) {
              printf("Queue is empty\n");
      } else {
              temp = tail->prev;
              tail->prev = temp->prev;
              temp->prev->next = tail;
              free(temp);
      }
      return;
}

/* display elements present in the queue */
void display() {
      struct node *temp;

      if (head->next == tail) {
              printf("Queue is empty\n");
              return;
      }

      temp = head->next;
      while (temp != tail) {
              printf("%-3d", temp->data);
              temp = temp->next;
```

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

int main() {
        printf("ABHAY PANDEY  2100320120004\n");
    int data, ch;
    createSentinels();
    while (1) {
        printf("1. Enqueue at front\n2. Enqueue at rear\n");
        printf("3. Dequeue at front\n4. Dequeue at rear\n");
        printf("5. Display\n6. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter the data to insert:");
                scanf("%d", &data);
                enqueueAtFront(data);
                break;

            case 2:
                printf("Enter ur data to insert:");
                scanf("%d", &data);
                enqueueAtRear(data);
                break;

            case 3:
                dequeueAtFront();
                break;

            case 4:
```

```
                dequeueAtRear();
                break;

            case 5:
                display();
                break;

            case 6:
                exit(0);

            default:
                printf("Pls. enter correct option\n");
                break;
        }
    }
    return 0;
 }
```

OUTPUT:

ABHAY PANDEY 2100320120004

1. Enqueue at front

2. Enqueue at rear

3. Dequeue at front

4. Dequeue at rear

5. Display

6. Exit

Enter your choice:1

Enter the data to insert:3

1. Enqueue at front

2. Enqueue at rear

3. Dequeue at front

4. Dequeue at rear

5. Display

6. Exit

Enter your choice:1

Enter the data to insert:5

1. Enqueue at front

2. Enqueue at rear

3. Dequeue at front

4. Dequeue at rear

5. Display

6. Exit

Enter your choice:5

5 3

1. Enqueue at front

2. Enqueue at rear

3. Dequeue at front

4. Dequeue at rear

5. Display

6. Exit

Enter your choice:6

LAB – 25

// Program for Pre-Order, In-Order, Post-Order Traversal

```
#include <stdio.h>

#include <stdlib.h>

/* A binary tree node has data, pointer to left child

and a pointer to right child */

struct node {

	int data;

	struct node* left;

	struct node* right;

};

/* Helper function that allocates a new node with the

given data and NULL left and right pointers. */

struct node* newNode(int data)

{

	struct node* node

		= (struct node*)malloc(sizeof(struct node));

	node->data = data;

	node->left = NULL;

	node->right = NULL;

	return (node);

}

/* Given a binary tree, print its nodes in inorder*/

void printInorder(struct node* node)

{
```

```
	if (node == NULL)
		return;

	/* first recur on left child */
	printInorder(node->left);

	/* then print the data of node */
	printf("%d ", node->data);

	/* now recur on right child */
	printInorder(node->right);
}


/* Driver code*/
int main()
{
	 printf("ABHAY PANDEY  2100320120004\n");
	struct node* root = newNode(1);
	root->left = newNode(2);
	root->right = newNode(3);
	root->left->left = newNode(4);
	root->left->right = newNode(5);

	// Function call
	printf("\nInorder traversal of binary tree is \n");
	printInorder(root);

	getchar();
	return 0;
```

}

OUTPUT:

ABHAY PANDEY 2100320120004

Inorder traversal of binary tree is

4 2 5 1 3

```
// Recursive Creation of Binary Tree
#include<stdio.h>

typedef struct node
{
int data;
struct node *left;
struct node *right;
} node;

node *create()
{
node *p;
int x;
printf("Enter data(-1 for no data):");
scanf("%d",&x);
if(x==-1)
return NULL;
p=(node*)malloc(sizeof(node));
p->data=x;
printf("Enter left child of %d:\n",x);
p->left=create();

printf("Enter right child of %d:\n",x);
p->right=create();
return p;
}

void preorder(node *t) //address of root node is passed in t
{
```

```
if(t!=NULL)
{
printf("\n%d",t->data); //visit the root
preorder(t->left); //preorder traversal on left subtree
preorder(t->right); //preorder traversal om right subtree
}
}

int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
node *root;
root=create();
printf("\nThe preorder traversal of tree is:\n");
preorder(root);
return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
Enter data(-1 for no data):2
Enter left child of 2:
Enter data(-1 for no data):3
Enter left child of 3:
Enter data(-1 for no data):6
Enter left child of 6:
Enter data(-1 for no data):7
Enter left child of 7:
Enter data(-1 for no data):8
Enter left child of 8:
Enter data(-1 for no data):8
```

Enter left child of 8:

Enter data(-1 for no data):5

Enter left child of 5:

Enter data(-1 for no data):2

Enter left child of 2:

Enter data(-1 for no data):12

Enter left child of 12:

Enter data(-1 for no data):34

Enter left child of 34:

Enter data(-1 for no data):45

Enter left child of 45:

Enter data(-1 for no data):67

Enter left child of 67:

Enter data(-1 for no data):89

Enter left child of 89:

Enter data(-1 for no data):90

Enter left child of 90:

Enter data(-1 for no data):-1

Enter right child of 90:

Enter data(-1 for no data):34

Enter left child of 34:

Enter data(-1 for no data):23

Enter left child of 23:

Enter data(-1 for no data):23

4Enter left child of 23:

Enter data(-1 for no data):5

6Enter left child of 5:

Enter data(-1 for no data):7

Enter left child of 7:

Enter data(-1 for no data):23

Enter left child of 23:

Enter data(-1 for no data):90

Enter left child of 90:

Enter data(-1 for no data):

34

Enter left child of 4:

Enter data(-1 for no data):23

Enter left child of 23:

Enter data(-1 for no data):12

Enter left child of 12:

Enter data(-1 for no data):35

```
// Program to find Node Count in the Binary Tree
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int info;
   struct node *left, *right;
};
struct node *createnode(int key)
{
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
   newnode->info = key;
   newnode->left = NULL;
   newnode->right = NULL;
   return(newnode);
}
static int count = 0;
int countnodes(struct node *root)
{
   if(root != NULL)
   {
      countnodes(root->left);
      count++;
      countnodes(root->right);
   }
   return count;
}
/*
 * Main Function
 */
```

```
int main()

{

            printf("ABHAY PANDEY  2100320120004\n");

   /* Creating first Tree. */

   struct node *newnode = createnode(25);

   newnode->left = createnode(27);

   newnode->right = createnode(19);

   newnode->left->left = createnode(17);

   newnode->left->right = createnode(91);

   newnode->right->left = createnode(13);

   newnode->right->right = createnode(55);

   /* Sample Tree 1:

    *              25

    *            /    \

    *          27     19

    *          / \    / \

    *        17  91   13 55

    */

   printf("Number of nodes in tree 1 = %d ",countnodes(newnode));

   printf("\n");

   count = 0;


   /* Creating second Tree. */

   struct node *node = createnode(1);

   node->right = createnode(2);

   node->right->right = createnode(3);

   node->right->right->right = createnode(4);

   node->right->right->right->right = createnode(5);

   /* Sample Tree 2:   Right Skewed Tree (Unbalanced).

    *             1
```

```
 *           \
 *            2
 *             \
 *              3
 *               \
 *                4
 *                 \
 *                  5
 */
printf("Number of nodes in tree 2 = %d ",countnodes(node));
printf("\n");
count = 0;

/* Creating third Tree. */
struct node *root = createnode(15);
/* Sample Tree 3- Tree having just one root node.
 *          15
 */
printf("Number of nodes in tree 3 = %d",countnodes(root));
return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Number of nodes in tree 1 = 7

Number of nodes in tree 2 = 5

Number of nodes in tree 3 = 1

```
// Program to find count of nodes having 1 child, 2 children and leaf nodes

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

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct node
{
	int data;
	struct node* left;
	struct node* right;
};

/* Function to get the count of leaf nodes in a binary tree*/
unsigned int getLeafCount(struct node* node)
{
if(node == NULL)
	return 0;
if(node->left == NULL && node->right==NULL)
	return 1;
else
	return getLeafCount(node->left)+
		getLeafCount(node->right);
}

/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
struct node* newNode(int data)
{
```

```
struct node* node = (struct node*)malloc(sizeof(struct node));
node->data = data;
node->left = NULL;
node->right = NULL;

return(node);
}

/*Driver program to test above functions*/
int main()
{
	printf("ABHAY PANDEY  2100320120004\n");
/*create a tree*/
struct node *root = newNode(1);
root->left        = newNode(2);
root->right       = newNode(3);
root->left->left = newNode(4);
root->left->right = newNode(5);
/*get leaf count of the above created tree*/
printf("Leaf count of the tree is %d", getLeafCount(root));

getchar();
return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Leaf count of the tree is 3

```
// Program to Find the height of the Binary Tree
#include <stdio.h>
#include <stdlib.h>
```

```
struct node {
   int data;
   struct node* left;
   struct node* right;
};

int height(struct node* node)
{
   if (node == NULL)
      return 0;
   else {

      int leftHeight = height(node->left);
      int rightHeight = height(node->right);


      if (leftHeight > rightHeight)
         return (leftHeight + 1);
      else
         return (rightHeight + 1);
   }
}


struct node* newNode(int data)
{
   struct node* node
      = (struct node*)malloc(sizeof(struct node));
   node->data = data;
```

```
    node->left = NULL;
    node->right = NULL;

    return (node);
}

int main()
{
        printf("ABHAY PANDEY  2100320120004\n");
    struct node* root = newNode(10);

    root->left = newNode(20);
    root->right = newNode(30);
    root->left->left = newNode(40);
    root->left->right = newNode(50);

    printf("Height of tree is %d", height(root));

    getchar();
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Height of tree is 3

LAB – 26

// write a program or function to find the sum all nodes in a given binary tree.

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

//Represent a node of binary tree
struct node{
   int data;
   struct node *left;
   struct node *right;
};

//Represent the root of binary tree
struct node *root = NULL;

//createNode() will create a new node
struct node* createNode(int data){
   //Create a new node
   struct node *newNode = (struct node*)malloc(sizeof(struct node));
   //Assign data to newNode, set left and right children to NULL
   newNode->data = data;
   newNode->left = NULL;
   newNode->right = NULL;

   return newNode;
}

//calculateSum() will calculate the sum of all the nodes present in the binary tree
int calculateSum(struct node *temp){
   int sum, sumLeft, sumRight;
   sum = sumRight = sumLeft = 0;
```

```
    //Check whether tree is empty
    if(root == NULL) {
        printf("Tree is empty\n");
        return 0;
    }
    else {
        //Calculate the sum of nodes present in left subtree
        if(temp->left != NULL)
            sumLeft = calculateSum(temp->left);

        //Calculate the sum of nodes present in right subtree
        if(temp->right != NULL)
              sumRight = calculateSum(temp->right);

        //Calculate the sum of all nodes by adding sumLeft, sumRight and root node's data
        sum = temp->data + sumLeft + sumRight;
        return sum;
   }
}

int main()
{
    printf("ABHAY PANDEY  2100320120004\n");
    //Add nodes to the binary tree
    root = createNode(5);
    root->left = createNode(2);
    root->right = createNode(9);
    root->left->left = createNode(1);
    root->right->left = createNode(8);
    root->right->right = createNode(6);

    //Display the sum of all the nodes in the given binary tree
```

```
    printf("Sum of all nodes of binary tree: %d", calculateSum(root));
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Sum of all nodes of binary tree: 31

```
// Program to Find if the given Binary Tree is complete

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_Q_SIZE 500

/* A binary tree node has data, a pointer to left child
and a pointer to right child */
struct node {
	int data;
	struct node* left;
	struct node* right;
};

/* function prototypes for functions needed for Queue data
structure. A queue is needed for level order traversal */
struct node** createQueue(int*, int*);
void enQueue(struct node**, int*, struct node*);
struct node* deQueue(struct node**, int*);
bool isQueueEmpty(int* front, int* rear);

/* Given a binary tree, return true if the tree is complete
else false */
bool isCompleteBT(struct node* root)
{
	// Base Case: An empty tree is complete Binary Tree
	if (root == NULL)
		return true;

	// Create an empty queue
	int rear, front;
```

```
struct node** queue = createQueue(&front, &rear);

// Create a flag variable which will be set true
// when a non full node is seen
bool flag = false;

// Do level order traversal using queue.
enQueue(queue, &rear, root);
while (!isQueueEmpty(&front, &rear)) {
	struct node* temp_node = deQueue(queue, &front);

	/* Check if left child is present*/
	if (temp_node->left) {
		// If we have seen a non full node, and we see a
		// node with non-empty left child, then the
		// given tree is not a complete Binary Tree
		if (flag == true)
			return false;

		enQueue(queue, &rear,
				temp_node->left); // Enqueue Left Child
	}
	else // If this a non-full node, set the flag as
		// true
		flag = true;

	/* Check if right child is present*/
	if (temp_node->right) {
		// If we have seen a non full node, and we see a
		// node with non-empty right child, then the
		// given tree is not a complete Binary Tree
		if (flag == true)
```

```
                    return false;

                enQueue(
                    queue, &rear,
                    temp_node->right); // Enqueue Right Child
            }
            else // If this a non-full node, set the flag as
                // true
                flag = true;
        }

        // If we reach here, then the tree is complete Binary
        // Tree
        return true;
}

/*UTILITY FUNCTIONS*/
struct node** createQueue(int* front, int* rear)
{
        struct node** queue = (struct node**)malloc(
            sizeof(struct node*) * MAX_Q_SIZE);

        *front = *rear = 0;
        return queue;
}

void enQueue(struct node** queue, int* rear,
                struct node* new_node)
{
        queue[*rear] = new_node;
        (*rear)++;
}
```

```
struct node* deQueue(struct node** queue, int* front)
{
        (*front)++;
        return queue[*front - 1];
}

bool isQueueEmpty(int* front, int* rear)
{
        return (*rear == *front);
}

/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
struct node* newNode(int data)
{
        struct node* node
                = (struct node*)malloc(sizeof(struct node));
        node->data = data;
        node->left = NULL;
        node->right = NULL;

        return (node);
}

/* Driver program to test above functions*/
int main()
{
         printf("ABHAY PANDEY  2100320120004\n");
        /* Let us construct the following Binary Tree which
        is not a complete Binary Tree
                        1
```

```
            / \
            2       3
            / \       \
            4 5       6
        */

        struct node* root = newNode(1);
        root->left = newNode(2);
        root->right = newNode(3);
        root->left->left = newNode(4);
        root->left->right = newNode(5);
        root->right->right = newNode(6);

        if (isCompleteBT(root) == true)
            printf("Complete Binary Tree");
        else
            printf("NOT Complete Binary Tree");

        return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

NOT Complete Binary Tree

```
// Program for Level Order Traversal

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

/* A binary tree node has data,
pointer to left child
and a pointer to right child */
struct node {
	int data;
	struct node *left, *right;
};

/* Function prototypes */
void printCurrentLevel(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);

/* Function to print level order traversal a tree*/
void printLevelOrder(struct node* root)
{
	int h = height(root);
	int i;
	for (i = 1; i <= h; i++)
		printCurrentLevel(root, i);
}

/* Print nodes at a current level */
void printCurrentLevel(struct node* root, int level)
{
```

```
	if (root == NULL)
		return;
	if (level == 1)
		printf("%d ", root->data);
	else if (level > 1) {
		printCurrentLevel(root->left, level - 1);
		printCurrentLevel(root->right, level - 1);
	}
}

/* Compute the "height" of a tree -- the number of
	nodes along the longest path from the root node
	down to the farthest leaf node.*/
int height(struct node* node)
{
	if (node == NULL)
		return 0;
	else {
		/* compute the height of each subtree */
		int lheight = height(node->left);
		int rheight = height(node->right);

		/* use the larger one */
		if (lheight > rheight)
			return (lheight + 1);
		else
			return (rheight + 1);
	}
}
```

```
/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
struct node* newNode(int data)
{
	struct node* node
		= (struct node*)malloc(sizeof(struct node));
	node->data = data;
	node->left = NULL;
	node->right = NULL;

	return (node);
}

/* Driver program to test above functions*/
int main()
{
	 printf("ABHAY PANDEY  2100320120004\n");
	struct node* root = newNode(1);
	root->left = newNode(2);
	root->right = newNode(3);
	root->left->left = newNode(4);
	root->left->right = newNode(5);

	printf("Level Order traversal of binary tree is \n");
	printLevelOrder(root);

	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Level Order traversal of binary tree is

1 2 3 4 5

LAB – 27

```
// Write a program to create a copy of the given Binary Tree
#include <stdio.h>
#include <limits.h>

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

struct node* getNewNode(int data) {
  /* dynamically allocate memory for a new node */
  struct node* newNode = (struct node*)malloc(sizeof(struct node));

  /* populate data in new Node */
  newNode->data = data;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}

/*
This function returns below tree
          1
         / \
       9    12
      / \    \
     4  50    -7
    / \
   18  9
```

```
*/
struct node* generateBTree(){
   // Root Node
   struct node* root =  getNewNode(1);

   root->left = getNewNode(9);
   root->right = getNewNode(12);

   root->left->left = getNewNode(4);
   root->left->right = getNewNode(50);
   root->right->right = getNewNode(-7);

   root->left->left->left = getNewNode(18);
   root->left->left->right = getNewNode(9);

   return root;
}

/* Returns a tree which is exact copy of passed tree */
struct node* cloneBinaryTree(struct node *root){
   if(root == NULL)
      return NULL;
   /* create a copy of root node */
   struct node* newNode = getNewNode(root->data);
   /* Recursively create clone of left and right sub tree */
   newNode->left = cloneBinaryTree(root->left);
   newNode->right = cloneBinaryTree(root->right);
   /* Return root of cloned tree */
   return newNode;
}

/*
```

```
 Prints inOrder Traversal of a binary tree
*/
void inOrderTraversal(struct node *nodeptr){
    if(nodeptr != NULL){
        /* First, recursively prints in Order traversal of left sub-tree */
        inOrderTraversal(nodeptr->left);
        /* Prints current node */
        printf("%d ", nodeptr->data);
        /* Recursively prints in Order traversal of right sub-tree */
        inOrderTraversal(nodeptr->right);
    }
}
int main() {
            printf("ABHAY PANDEY  2100320120004\n");
    struct node *clone, *root = generateBTree();

 /*InOrder traversal of original tree */
    printf("Original Tree\n");
    inOrderTraversal(root);

    clone = cloneBinaryTree(root);

 /*InOrder traversal of clone tree */
    printf("\nClone Tree\n");
    inOrderTraversal(clone);

    getchar();
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Original Tree

18 4 9 9 50 1 12 -7

Clone Tree

18 4 9 9 50 1 12 -7

```
// write a program to check if the given tree is BST or not.
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct node {
	int data;
	struct node* left;
	struct node* right;
};

/* Helper function that allocates a new node with the
given data and NULL left and right pointers. */
struct node* newNode(int data)
{
	struct node* node
		= (struct node*)malloc(sizeof(struct node));
	node->data = data;
	node->left = NULL;
	node->right = NULL;

	return (node);
}

int maxValue(struct node* node)
{
	if (node == NULL) {
		return 0;
```

```
	}

	int leftMax = maxValue(node->left);
	int rightMax = maxValue(node->right);

	int value = 0;
	if (leftMax > rightMax) {
		value = leftMax;
	}
	else {
		value = rightMax;
	}

	if (value < node->data) {
		value = node->data;
	}

	return value;
}

int minValue(struct node* node)
{
	if (node == NULL) {
		return 1000000000;
	}

	int leftMax = minValue(node->left);
	int rightMax = minValue(node->right);

	int value = 0;
```

```
	if (leftMax < rightMax) {
		value = leftMax;
	}
	else {
		value = rightMax;
	}

	if (value > node->data) {
		value = node->data;
	}

	return value;
}

/* Returns true if a binary tree is a binary search tree */
int isBST(struct node* node)
{
	if (node == NULL)
		return 1;

	/* false if the max of the left is > than us */
	if (node->left != NULL
		&& maxValue(node->left) > node->data)
		return 0;

	/* false if the min of the right is <= than us */
	if (node->right != NULL
		&& minValue(node->right) < node->data)
		return 0;
```

```
	/* false if, recursively, the left or right is not a BST
	*/
	if (!isBST(node->left) || !isBST(node->right))
		return 0;

	/* passing all that, it's a BST */
	return 1;
}

/* Driver code*/
int main()
{
	 printf("ABHAY PANDEY  2100320120004\n");
	struct node* root = newNode(4);
	root->left = newNode(5);
	root->right = newNode(6);
// root->left->left = newNode(1);
	//root->left->right = newNode(3);

	// Function call
	if (isBST(root))
		printf("Is BST");
	else
		printf("Not a BST");

	getchar();
	return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Not a BST

```
// write a program to implement Insertion and Search operation in BST (Iterative)
#include<stdio.h>
#include<stdlib.h>
void insert(int);
struct node
{
   int data;
   struct node *left;
   struct node *right;
};
struct node *root;
int main ()
{
   int choice,item;
   do
   {
      printf("\nEnter the item which you want to insert?\n");
      scanf("%d",&item);
      insert(item);
      printf("\nPress 0 to insert more ?\n");
      scanf("%d",&choice);
   }while(choice == 0);
          return 0;
}
void insert(int item)
{
   struct node *ptr, *parentptr , *nodeptr;
   ptr = (struct node *) malloc(sizeof (struct node));
   if(ptr == NULL)
   {
```

```
   printf("can't insert");
}
else
{
ptr -> data = item;
ptr -> left = NULL;
ptr -> right = NULL;
if(root == NULL)
{
   root = ptr;
   root -> left = NULL;
   root -> right = NULL;
}
else
{
   parentptr = NULL;
   nodeptr = root;
   while(nodeptr != NULL)
   {
      parentptr = nodeptr;
      if(item < nodeptr->data)
      {
         nodeptr = nodeptr -> left;
      }
      else
      {
         nodeptr = nodeptr -> right;
      }
   }
   if(item < parentptr -> data)
```

```
        {
            parentptr -> left = ptr;
        }
        else
        {
            parentptr -> right = ptr;
        }
    }
    printf("Node Inserted");
    }
}
```

OUTPUT:

Enter the item which you want to insert?

4

Node Inserted

Press 0 to insert more ?

0

Enter the item which you want to insert?

2

Node Inserted

Press 0 to insert more ?

LAB – 28

```
// Program to find the diameter of the Binary Tree (distance between the farthest node)
// Recursive optimized C program to find the diameter of a
// Binary Tree
#include <stdio.h>
#include <stdlib.h>

// A binary tree node has data, pointer to left child
// and a pointer to right child
struct node {
	int data;
	struct node *left, *right;
};

// function to create a new node of tree and returns pointer
struct node* newNode(int data);

// returns max of two integers
int max(int a, int b) { return (a > b) ? a : b; }

// function to Compute height of a tree.
int height(struct node* node);

// Function to get diameter of a binary tree
int diameter(struct node* tree)
{
	// base case where tree is empty
	if (tree == NULL)
		return 0;

	// get the height of left and right sub-trees
	int lheight = height(tree->left);
```

```
	int rheight = height(tree->right);

	// get the diameter of left and right sub-trees
	int ldiameter = diameter(tree->left);
	int rdiameter = diameter(tree->right);

	// Return max of following three
	// 1) Diameter of left subtree
	// 2) Diameter of right subtree
	// 3) Height of left subtree + height of right subtree +
	// 1

	return max(lheight + rheight + 1,
				max(ldiameter, rdiameter));
}

// UTILITY FUNCTIONS TO TEST diameter() FUNCTION

// The function Compute the "height" of a tree. Height is
// the number f nodes along the longest path from the root
// node down to the farthest leaf node.
int height(struct node* node)
{
	// base case tree is empty
	if (node == NULL)
		return 0;

	// If tree is not empty then height = 1 + max of left
	// height and right heights
	return 1 + max(height(node->left), height(node->right));
}
```

```
// Helper function that allocates a new node with the
// given data and NULL left and right pointers.
struct node* newNode(int data)
{
	struct node* node
		= (struct node*)malloc(sizeof(struct node));
	node->data = data;
	node->left = NULL;
	node->right = NULL;

	return (node);
}

// Driver Code
int main()
{
    printf("ABHAY PANDEY  2100320120004\n");
	/* Constructed binary tree is
			1
			/ \
		2	 3
		/ \
	4	 5
	*/
	struct node* root = newNode(1);
	root->left = newNode(2);
	root->right = newNode(3);
	root->left->left = newNode(4);
	root->left->right = newNode(5);

	// Function Call
	printf("Diameter of the given binary tree is %d\n",
```

```
                        diameter(root));

            return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Diameter of the given binary tree is 4

```
// write a program to implement deletion in BST.
// C program to demonstrate
// delete operation in binary
// search tree
#include <stdio.h>
#include <stdlib.h>

struct node {
	int key;
	struct node *left, *right;
};

// A utility function to create a new BST node
struct node* newNode(int item)
{
	struct node* temp
		= (struct node*)malloc(sizeof(struct node));
	temp->key = item;
	temp->left = temp->right = NULL;
	return temp;
}

// A utility function to do inorder traversal of BST
void inorder(struct node* root)
{
	if (root != NULL) {
		inorder(root->left);
		printf("%d ", root->key);
		inorder(root->right);
	}
}
```

```
/* A utility function to
insert a new node with given key in
* BST */
struct node* insert(struct node* node, int key)
{
	/* If the tree is empty, return a new node */
	if (node == NULL)
		return newNode(key);

	/* Otherwise, recur down the tree */
	if (key < node->key)
		node->left = insert(node->left, key);
	else
		node->right = insert(node->right, key);

	/* return the (unchanged) node pointer */
	return node;
}

/* Given a non-empty binary search
tree, return the node
with minimum key value found in
that tree. Note that the
entire tree does not need to be searched. */
struct node* minValueNode(struct node* node)
{
	struct node* current = node;

	/* loop down to find the leftmost leaf */
	while (current && current->left != NULL)
		current = current->left;
```

```
	return current;
}

/* Given a binary search tree
and a key, this function
deletes the key and
returns the new root */
struct node* deleteNode(struct node* root, int key)
{
	// base case
	if (root == NULL)
		return root;

	// If the key to be deleted
	// is smaller than the root's
	// key, then it lies in left subtree
	if (key < root->key)
		root->left = deleteNode(root->left, key);

	// If the key to be deleted
	// is greater than the root's
	// key, then it lies in right subtree
	else if (key > root->key)
		root->right = deleteNode(root->right, key);

	// if key is same as root's key,
	// then This is the node
	// to be deleted
	else {
		// node with only one child or no child
		if (root->left == NULL) {
			struct node* temp = root->right;
```

```
                free(root);
                return temp;
            }
            else if (root->right == NULL) {
                struct node* temp = root->left;
                free(root);
                return temp;
            }

            // node with two children:
            // Get the inorder successor
            // (smallest in the right subtree)
            struct node* temp = minValueNode(root->right);

            // Copy the inorder
            // successor's content to this node
            root->key = temp->key;

            // Delete the inorder successor
            root->right = deleteNode(root->right, temp->key);
        }
        return root;
}

// Driver Code
int main()
{
         printf("ABHAY PANDEY  2100320120004\n");
        /* Let us create following BST
                50
            /        \
            30        70
```

```
            / \ / \
        20 40 60 80 */
        struct node* root = NULL;
        root = insert(root, 50);
        root = insert(root, 30);
        root = insert(root, 20);
        root = insert(root, 40);
        root = insert(root, 70);
        root = insert(root, 60);
        root = insert(root, 80);

        printf("Inorder traversal of the given tree \n");
        inorder(root);

        printf("\nDelete 20\n");
        root = deleteNode(root, 20);
        printf("Inorder traversal of the modified tree \n");
        inorder(root);

        printf("\nDelete 30\n");
        root = deleteNode(root, 30);
        printf("Inorder traversal of the modified tree \n");
        inorder(root);

        printf("\nDelete 50\n");
        root = deleteNode(root, 50);
        printf("Inorder traversal of the modified tree \n");
        inorder(root);

        return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Inorder traversal of the given tree

20 30 40 50 60 70 80

Delete 20

Inorder traversal of the modified tree

30 40 50 60 70 80

Delete 30

Inorder traversal of the modified tree

40 50 60 70 80

Delete 50

Inorder traversal of the modified tree

40 60 70 80

```c
// Write a Program for BST insertion (using Recursion)
#include <stdio.h>
#include <stdlib.h>

struct node{
   int data;
   struct node *left;
   struct node *right;
};
struct node *root = NULL;

struct node *newNode(int data){
   struct node *temp = (struct node *)malloc(sizeof(struct node));
   temp->data = data;
   temp->left = NULL;
   temp->right = NULL;
   return temp;
}

//SEARCH FUNCTION 1: this does not work correctly
void search(struct node *t,int data){
   if(t){
      if(data > t->data){
         search(t->right,data);
      }else{
         search(t->left,data);
      }
   }else{
      t = newNode(data);
   }
}
```

```
//SEARCH FUNCTION 2: this works fine and inserts the element correctly
void search2(struct node *t, int data){
   if(data < t->data && t->left != NULL){
      search(t->left, data);
   }else if(data < t->data && t->left == NULL){
      t->left = newNode(data);
   }else if(data > t->data && t->right != NULL){
      search(t->right,data);
   }else{
      t->right = newNode(data);
   }
}

void insertNode(int data){
   if(!root){
      root = newNode(data);
      return;
   }
   search(root, data);
}

void inorder(struct node *t){
   if(t){
      if(t->left){
         inorder(t->left);
      }
      printf("%d ->", t->data);
      if(t->right){
         inorder(t->right);
      }
   }
}
```

```
int main(){
            printf("ABHAY PANDEY  2100320120004\n");
   int step, data;
   while(1){
      printf("1. Insert element\n");
      printf("2. Print tree\n");
      scanf("%d",&step);
      switch(step){
         case 1: printf("enter element to be inserted\n");
            scanf("%d",&data);
            insertNode(data);
            break;
         case 2:inorder(root);
            printf("\n");
            break;
      }
   }
   return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
1. Insert element
2. Print tree
1
enter element to be inserted
2
1. Insert element
2. Print tree
1
enter element to be inserted
5
```

1. Insert element

2. Print tree

1

enter element to be inserted

6

1. Insert element

2. Print tree

18

1. Insert element

2. Print tree

1

enter element to be inserted

99

1. Insert element

2. Print tree

2

2 ->

1. Insert element

2. Print tree

2

2 ->

1. Insert element

2. Print tree

```
// write a program to perform insertion operation for AVL tree.
// C program to insert a node in AVL tree
#include<stdio.h>
#include<stdlib.h>

// An AVL tree node
struct Node
{
	int key;
	struct Node *left;
	struct Node *right;
	int height;
};

// A utility function to get the height of the tree
int height(struct Node *N)
{
	if (N == NULL)
		return 0;
	return N->height;
}

// A utility function to get maximum of two integers
int max(int a, int b)
{
	return (a > b)? a : b;
}

/* Helper function that allocates a new node with the given key and
	NULL left and right pointers. */
struct Node* newNode(int key)
{
```

```
	struct Node* node = (struct Node*)
							malloc(sizeof(struct Node));
	node->key = key;
	node->left = NULL;
	node->right = NULL;
	node->height = 1; // new node is initially added at leaf
	return(node);
}

// A utility function to right rotate subtree rooted with y
// See the diagram given above.
struct Node *rightRotate(struct Node *y)
{
	struct Node *x = y->left;
	struct Node *T2 = x->right;

	// Perform rotation
	x->right = y;
	y->left = T2;

	// Update heights
	y->height = max(height(y->left),
					height(y->right)) + 1;
	x->height = max(height(x->left),
					height(x->right)) + 1;

	// Return new root
	return x;
}

// A utility function to left rotate subtree rooted with x
// See the diagram given above.
```

```
struct Node *leftRotate(struct Node *x)
{
	struct Node *y = x->right;
	struct Node *T2 = y->left;

	// Perform rotation
	y->left = x;
	x->right = T2;

	// Update heights
	x->height = max(height(x->left),
					height(x->right)) + 1;
	y->height = max(height(y->left),
					height(y->right)) + 1;

	// Return new root
	return y;
}

// Get Balance factor of node N
int getBalance(struct Node *N)
{
	if (N == NULL)
		return 0;
	return height(N->left) - height(N->right);
}

// Recursive function to insert a key in the subtree rooted
// with node and returns the new root of the subtree.
struct Node* insert(struct Node* node, int key)
{
	/* 1. Perform the normal BST insertion */
```

```
if (node == NULL)
        return(newNode(key));

if (key < node->key)
        node->left = insert(node->left, key);
else if (key > node->key)
        node->right = insert(node->right, key);
else // Equal keys are not allowed in BST
        return node;

/* 2. Update height of this ancestor node */
node->height = 1 + max(height(node->left),
                        height(node->right));

/* 3. Get the balance factor of this ancestor
        node to check whether this node became
        unbalanced */
int balance = getBalance(node);

// If this node becomes unbalanced, then
// there are 4 cases

// Left Left Case
if (balance > 1 && key < node->left->key)
        return rightRotate(node);

// Right Right Case
if (balance < -1 && key > node->right->key)
        return leftRotate(node);

// Left Right Case
if (balance > 1 && key > node->left->key)
```

```
	{
		node->left = leftRotate(node->left);
		return rightRotate(node);
	}

	// Right Left Case
	if (balance < -1 && key < node->right->key)
	{
		node->right = rightRotate(node->right);
		return leftRotate(node);
	}

	/* return the (unchanged) node pointer */
	return node;
}

// A utility function to print preorder traversal
// of the tree.
// The function also prints height of every node
void preOrder(struct Node *root)
{
	if(root != NULL)
	{
		printf("%d ", root->key);
		preOrder(root->left);
		preOrder(root->right);
	}
}

/* Driver program to test above function*/
int main()
{
```

```
        printf("ABHAY PANDEY  2100320120004\n");
struct Node *root = NULL;

/* Constructing tree given in the above figure */
root = insert(root, 10);
root = insert(root, 20);
root = insert(root, 30);
root = insert(root, 40);
root = insert(root, 50);
root = insert(root, 25);

/* The constructed AVL Tree would be
                  30
            / \
            20 40
            / \     \
      10 25 50
*/
printf("Preorder traversal of the constructed AVL"" tree is \n");
preOrder(root);
return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Preorder traversal of the constructed AVL tree is

30 20 10 25 40 50

LAB – 29

```
// Program for Heap Sort

#include <stdio.h>

// Function to swap the position of two elements
```

```
void swap(int* a, int* b)
{

    int temp = *a;

    *a = *b;

    *b = temp;
}

// To heapify a subtree rooted with node i
// which is an index in arr[].
// n is size of heap
void heapify(int arr[], int N, int i)
{
    // Find largest among root, left child and right child

    // Initialize largest as root
    int largest = i;

    // left = 2*i + 1
    int left = 2 * i + 1;

    // right = 2*i + 2
    int right = 2 * i + 2;

    // If left child is larger than root
    if (left < N && arr[left] > arr[largest])

        largest = left;
```

```
	// If right child is larger than largest
	// so far
	if (right < N && arr[right] > arr[largest])

		largest = right;

	// Swap and continue heapifying if root is not largest
	// If largest is not root
	if (largest != i) {

		swap(&arr[i], &arr[largest]);

		// Recursively heapify the affected
		// sub-tree
		heapify(arr, N, largest);
	}
}

// Main function to do heap sort
void heapSort(int arr[], int N)
{

	// Build max heap
	for (int i = N / 2 - 1; i >= 0; i--)

		heapify(arr, N, i);

	// Heap sort
	for (int i = N - 1; i >= 0; i--) {

		swap(&arr[0], &arr[i]);
```

```
                // Heapify root element to get highest element at
                // root again
                heapify(arr, i, 0);
        }
}

// A utility function to print array of size n
void printArray(int arr[], int N)
{
        for (int i = 0; i < N; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

// Driver's code
int main()
{
         printf("ABHAY PANDEY  2100320120004\n");
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int N = sizeof(arr) / sizeof(arr[0]);

        // Function call
        heapSort(arr, N);
        printf("Sorted array is\n");
        printArray(arr, N);
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Sorted array is

5 6 7 11 12 13

```
//Program for Heap Implementation of Priority Queue
#include <stdio.h>

int tree_array_size = 20;
int heap_size = 0;
const int INF = 100000;

void swap( int *a, int *b ) {
  int t;
  t = *a;
  *a = *b;
  *b = t;
}

//function to get right child of a node of a tree
int get_right_child(int A[], int index) {
  if((((2*index)+1) < tree_array_size) && (index >= 1))
    return (2*index)+1;
  return -1;
}

//function to get left child of a node of a tree
int get_left_child(int A[], int index) {
    if(((2*index) < tree_array_size) && (index >= 1))
        return 2*index;
    return -1;
}

//function to get the parent of a node of a tree
int get_parent(int A[], int index) {
  if ((index > 1) && (index < tree_array_size)) {
    return index/2;
```

```
  }
  return -1;
}

void max_heapify(int A[], int index) {
  int left_child_index = get_left_child(A, index);
  int right_child_index = get_right_child(A, index);

  // finding largest among index, left child and right child
  int largest = index;

  if ((left_child_index <= heap_size) && (left_child_index>0)) {
    if (A[left_child_index] > A[largest]) {
      largest = left_child_index;
    }
  }

  if ((right_child_index <= heap_size && (right_child_index>0))) {
    if (A[right_child_index] > A[largest]) {
      largest = right_child_index;
    }
  }

  // largest is not the node, node is not a heap
  if (largest != index) {
    swap(&A[index], &A[largest]);
    max_heapify(A, largest);
  }
}

void build_max_heap(int A[]) {
  int i;
```

```
  for(i=heap_size/2; i>=1; i--) {
    max_heapify(A, i);
  }
}

int maximum(int A[]) {
  return A[1];
}

int extract_max(int A[]) {
  int maxm = A[1];
  A[1] = A[heap_size];
  heap_size--;
  max_heapify(A, 1);
  return maxm;
}

void increase_key(int A[], int index, int key) {
  A[index] = key;
  while((index>1) && (A[get_parent(A, index)] < A[index])) {
    swap(&A[index], &A[get_parent(A, index)]);
    index = get_parent(A, index);
  }
}

void decrease_key(int A[], int index, int key) {
  A[index] = key;
  max_heapify(A, index);
}

void insert(int A[], int key) {
  heap_size++;
```

```
  A[heap_size] = -1*INF;
  increase_key(A, heap_size, key);
}

void print_heap(int A[]) {
  int i;
  for(i=1; i<=heap_size; i++) {
    printf("%d\n",A[i]);
  }
  printf("\n");
}

int main() {
        printf("ABHAY PANDEY  2100320120004\n");
  int A[tree_array_size];
  insert(A, 20);
  insert(A, 15);
  insert(A, 8);
  insert(A, 10);
  insert(A, 5);
  insert(A, 7);
  insert(A, 6);
  insert(A, 2);
  insert(A, 9);
  insert(A, 1);

  print_heap(A);

  increase_key(A, 5, 22);
  print_heap(A);

  decrease_key(A, 1, 13);
```

```
  print_heap(A);

  printf("%d\n\n", maximum(A));
  printf("%d\n\n", extract_max(A));

  print_heap(A);

  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  printf("%d\n", extract_max(A));
  return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

```
20
15
8
10
5
7
6
2
9
1

22
```

20

8

10

15

7

6

2

9

1

20

15

8

10

13

7

6

2

9

1

20

20

15

13

8

10

1

7

6

2

9

15

13

10

9

8

7

6

2

1

//Program for BFS on a Graph

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40

struct queue {
  int items[SIZE];
  int front;
  int rear;
};

struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);

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

struct node* createNode(int);

struct Graph {
  int numVertices;
  struct node** adjLists;
  int* visited;
};
```

```
// BFS algorithm
void bfs(struct Graph* graph, int startVertex) {
  struct queue* q = createQueue();

  graph->visited[startVertex] = 1;
  enqueue(q, startVertex);

  while (!isEmpty(q)) {
    printQueue(q);
    int currentVertex = dequeue(q);
    printf("Visited %d\n", currentVertex);

    struct node* temp = graph->adjLists[currentVertex];

    while (temp) {
      int adjVertex = temp->vertex;

      if (graph->visited[adjVertex] == 0) {
        graph->visited[adjVertex] = 1;
        enqueue(q, adjVertex);
      }
      temp = temp->next;
    }
  }
}

// Creating a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
```

```
  return newNode;
}

// Creating a graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));
  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }

  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}
```

```
// Create a queue
struct queue* createQueue() {
  struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}

// Check if the queue is empty
int isEmpty(struct queue* q) {
  if (q->rear == -1)
    return 1;
  else
    return 0;
}

// Adding elements into queue
void enqueue(struct queue* q, int value) {
  if (q->rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}

// Removing elements from queue
int dequeue(struct queue* q) {
  int item;
```

```
  if (isEmpty(q)) {
    printf("Queue is empty");
    item = -1;
  } else {
    item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf("Resetting queue ");
      q->front = q->rear = -1;
    }
  }
  return item;
}

// Print the queue
void printQueue(struct queue* q) {
  int i = q->front;

  if (isEmpty(q)) {
    printf("Queue is empty");
  } else {
    printf("\nQueue contains \n");
    for (i = q->front; i < q->rear + 1; i++) {
      printf("%d ", q->items[i]);
    }
  }
}

int main() {
            printf("ABHAY PANDEY  2100320120004\n");
  struct Graph* graph = createGraph(6);
  addEdge(graph, 0, 1);
```

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

  bfs(graph, 0);
  return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
Queue contains
0 Resetting queue Visited 0
Queue contains
2 1 Visited 2
Queue contains
1 4 Visited 1
Queue contains
4 3 Visited 4
Queue contains
3 Resetting queue Visited 3
l
dash: 2: l: not found
1
dash: 3: 1: not found
```

//Program for DFS on a Graph

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

```
struct node {
  int vertex;
  struct node* next;
};

struct node* createNode(int v);

struct Graph {
  int numVertices;
  int* visited;

  // We need int** to store a two dimensional array.
  // Similary, we need struct node** to store an array of Linked lists
  struct node** adjLists;
};

// DFS algo
void DFS(struct Graph* graph, int vertex) {
  struct node* adjList = graph->adjLists[vertex];
  struct node* temp = adjList;

  graph->visited[vertex] = 1;
  printf("Visited %d \n", vertex);

  while (temp != NULL) {
    int connectedVertex = temp->vertex;

    if (graph->visited[connectedVertex] == 0) {
      DFS(graph, connectedVertex);
    }
    temp = temp->next;
  }
```

```
}

// Create a node
struct node* createNode(int v) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->vertex = v;
  newNode->next = NULL;
  return newNode;
}

// Create graph
struct Graph* createGraph(int vertices) {
  struct Graph* graph = malloc(sizeof(struct Graph));
  graph->numVertices = vertices;

  graph->adjLists = malloc(vertices * sizeof(struct node*));

  graph->visited = malloc(vertices * sizeof(int));

  int i;
  for (i = 0; i < vertices; i++) {
    graph->adjLists[i] = NULL;
    graph->visited[i] = 0;
  }
  return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
  // Add edge from src to dest
  struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
```

```
  graph->adjLists[src] = newNode;

  // Add edge from dest to src
  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}

// Print the graph
void printGraph(struct Graph* graph) {
  int v;
  for (v = 0; v < graph->numVertices; v++) {
    struct node* temp = graph->adjLists[v];
    printf("\n Adjacency list of vertex %d\n ", v);
    while (temp) {
      printf("%d -> ", temp->vertex);
      temp = temp->next;
    }
    printf("\n");
  }
}

int main() {
        printf("ABHAY PANDEY  2100320120004\n");
  struct Graph* graph = createGraph(4);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 2, 3);

  printGraph(graph);
```

```
  DFS(graph, 2);


  return 0;
}
```

OUTPUT:

```
ABHAY PANDEY  2100320120004
Adjacency list of vertex 0
 2 -> 1 ->

Adjacency list of vertex 1
 2 -> 0 ->

Adjacency list of vertex 2
 3 -> 1 -> 0 ->

Adjacency list of vertex 3
 2 ->
Visited 2
Visited 3
Visited 1
Visited 0
```

LAB -30

//Program to find the number of connected components in the undirected Graph

```
class Graph:
	print("ABHAY PANDEY  2100320120004\n");
	def __init__(self, V):

		# No. of vertices
		self.V = V
```

```
        # Pointer to an array containing
        # adjacency lists
        self.adj = [[] for i in range(self.V)]

    # Function to return the number of
    # connected components in an undirected graph
    def NumberOfconnectedComponents(self):

        # Mark all the vertices as not visited
        visited = [False for i in range(self.V)]

        # To store the number of connected
        # components
        count = 0

        for v in range(self.V):
            if (visited[v] == False):
                self.DFSUtil(v, visited)
                count += 1

        return count

    def DFSUtil(self, v, visited):

        # Mark the current node as visited
        visited[v] = True

        # Recur for all the vertices
        # adjacent to this vertex
        for i in self.adj[v]:
            if (not visited[i]):
                self.DFSUtil(i, visited)
```

```
    # Add an undirected edge
    def addEdge(self, v, w):

        self.adj[v].append(w)
        self.adj[w].append(v)

# Driver code
if __name__=='__main__':

    g = Graph(5)
    g.addEdge(1, 0)
    g.addEdge(2, 3)
    g.addEdge(3, 4)

    print(g.NumberOfconnectedComponents())
```

OUTPUT:

ABHAY PANDEY 2100320120004

2

```c
//Program for Warshall's Algorithm for APSP
#include <stdio.h>
#include <stdlib.h>

void floydWarshall(int **graph, int n)
{
   int i, j, k;
   for (k = 0; k < n; k++)
   {
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
            if (graph[i][j] > graph[i][k] + graph[k][j])
               graph[i][j] = graph[i][k] + graph[k][j];
         }
      }
   }
}

int main(void)
{
		printf("ABHAY PANDEY  2100320120004\n");
   int n, i, j;
   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   int **graph = (int **)malloc((long unsigned) n * sizeof(int *));
   for (i = 0; i < n; i++)
   {
      graph[i] = (int *)malloc((long unsigned) n * sizeof(int));
   }
   for (i = 0; i < n; i++)
```

```
{
    for (j = 0; j < n; j++)
    {
        if (i == j)
            graph[i][j] = 0;
        else
            graph[i][j] = 100;
    }
}
printf("Enter the edges: \n");
for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
    {
        printf("[%d][%d]: ", i, j);
        scanf("%d", &graph[i][j]);
    }
}
printf("The original graph is:\n");
for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
    {
        printf("%d ", graph[i][j]);
    }
    printf("\n");
}
floydWarshall(graph, n);
printf("The shortest path matrix is:\n");
for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
```

```
        {
            printf("%d ", graph[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Enter the number of vertices: 3

Enter the edges:

[0][0]: 12

[0][1]: 23

[0][2]: 34

[1][0]: 56

[1][1]: 45

[1][2]: 78

[2][0]: 35

[2][1]: 67

[2][2]: 58

The original graph is:

12 23 34

56 45 78

35 67 58

The shortest path matrix is:

12 23 34

56 45 78

35 58 58

```
//Program For Linked List Implementation of General Sparse Matrix
#include<stdio.h>
#include<stdlib.h>
int main(){
        printf("ABHAY PANDEY  2100320120004\n");
  int row,col,i,j,a[10][10],count = 0;
  printf("Enter row");
  scanf("%d",&row);
  printf("Enter Column");
  scanf("%d",&col);
  printf("Enter Element of Matrix1");
  for(i = 0; i < row; i++){
    for(j = 0; j < col; j++){
      scanf("%d",&a[i][j]);
    }
  }
  printf("Elements are:");
  for(i = 0; i < row; i++){
    for(j = 0; j < col; j++){
      printf("%d\t",a[i][j]);
    }
    printf("");
  }
  /*checking sparse of matrix*/
  for(i = 0; i < row; i++){
    for(j = 0; j < col; j++){
      if(a[i][j] == 0)
        count++;
    }
  }
```

```
  if(count > ((row * col)/2))
    printf("Matrix is a sparse matrix ");
  else
    printf("Matrix is not sparse matrix");
}
```

OUTPUT:

ABHAY PANDEY 2100320120004

Enter row2

Enter Column2

Enter Element of Matrix11

2

4

5

Elements are:1 2 4 5 Matrix is not sparse matrix