Maharaja Surajmal Institute

Affiliated to GGSIPU & NAAC 'A' grade accredited

DEPARTMENT OF COMPUTER APPLICATIONS

DATA STRUCTURE & ALGORITHMS USING C

PRACTICAL FILE

SUBJECT CODE – BCA

SUBMITTED BY:

Name: Aman Tripathi

Enroll No: 05221202021

Sem: 2nd Sec: A (2nd shift)

SUBMITTED TO:

Mrs Tarunim Sharma

Assistant Professor

INDEX

S.No.	Practical	Sign
1.	WAP to find the greatest among 3 numbers using a conditional operator.	
2.	WAP to print a table of a number using a do-while loop.	
3.	WAP to calculate the factorial of a number.	
4.	WAP to print Fibonacci series using function.	
5.	WAP to enter the given list of numbers and find how many are positive, negative or zero.	
6.	WAP to sort a given list of no in ascending order and print the original and sorted list using the function.	
7.	WAP to calculate addition, subtraction, and multiplication of matrices.	
8.	WAP to find the largest/smallest element of a matrix.	
9.	WAP to calculate the sum of each row and column and the total of all elements of the matrix.	
10.	WAP to search elements from an array using linear search.	
11.	WAP to search elements from an array using binary search.	
12.	WAP to check whether numbers are a palindrome.	
13.	WAP to calculate the sum of digits of the number.	
14.	WAP for Bubble Sort to Sort Elements in An Order.	
15.	WAP for insertion Sort to Sort Elements in An Order.	
16.	WAP to for selection Sort to Sort Elements in An Order.	
17.	WAP to mergeSort to Sort Elements in An Order.	
18.	WAP to find lower and upper triangular matrices.	
19.	WAP to check whether an entered is lower triangular.	
20.	WAP of Single Linked List and perform the operations on it like(Insert, display, length, delete,..etc)	
21.	WAP of Double Linked List and perform the operations on it like(Insert, display, length, delete,..etc).	
22.	WAP of Circular Single Linked List and perform the operations on it like(Insert, display, length, delete,..etc).	

23.	WAP of Circular Double Linked List and perform the operations on it like(Insert, display, length, delete,..etc).	
24.	WAP of static implementation of a stack.	
25.	WAP of Dynamic implementation of a stack.	
26.	WAP of Static implementation of Linear Queue.	
27.	WAP of Dynamic implementation of Linear Queue.	
28.	WAP of Static implementation of Circular Queue.	
29.	WAP of Dynamic implementation of Circular Queue.	
30.	WAP of Static implementation of Double-Ended Queue.	
31.	WAP of Input Restricted Double Ended Queue.	
32.	WAP of Output Restricted Double Ended Queue.	
33.	WAP of Dynamic implementation of Double-Ended Queue.	
34.	WAP of Static Implementation of Priority Queue.	
35.	WAP of Dynamic Implementation of Priority Queue.	
36.	WAP of Binary tree traversal (Inorder, Preorder & Postorder).	

Practical 1

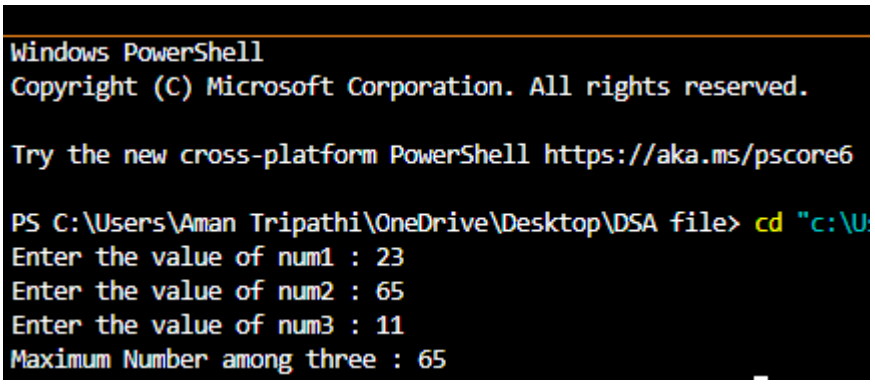
Code:-

// Q 1. WAP to find the greatest among 3 numbers using conditional operator.

```
#include <stdio.h>
void main()
{
    int num1 = 0;
    int num2 = 0;
    int num3 = 0;
    int max = 0;

    // input from user
    printf("Enter the value of num1 : ");
    scanf("%d", &num1);
    printf("Enter the value of num2 : ");
    scanf("%d", &num2);
    printf("Enter the value of num3 : ");
    scanf("%d", &num3);

    // logic for program
    max = (num1 > num2)
     ? (num1 > num3 ? num1 : num3)
                    : (num2 > num3 ? num2 : num3);
```

```
    // Output of Program
    printf("Maximum Number among three : %d", max);
    printf("\n");
}
```

Output:-



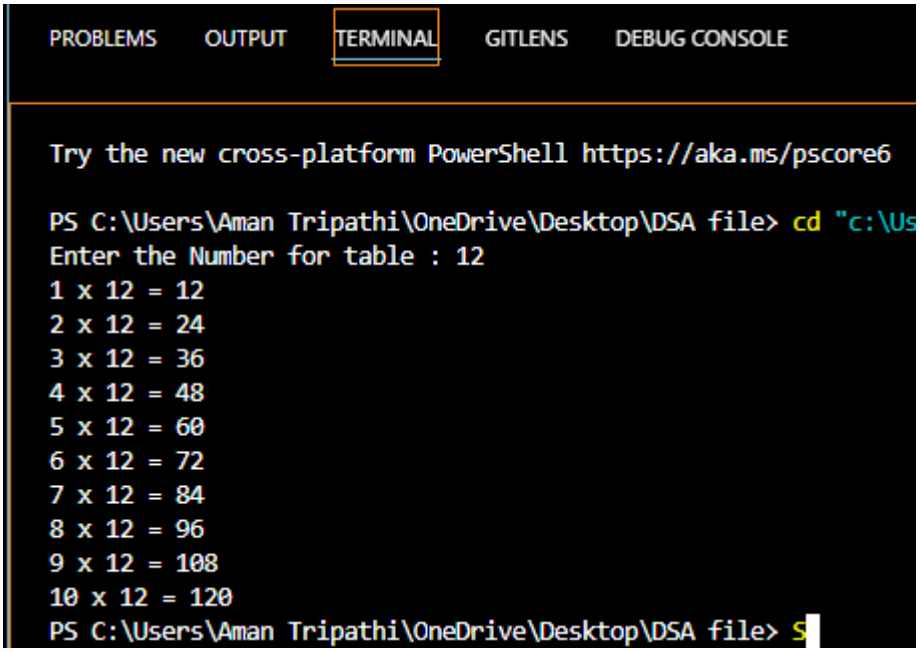
Practical 2

Code:-

```
/*
Q 2. WAP to print table of a number using do while loop.
*/


#include <stdio.h>
void main()
{
    int num = 0;
    // User Input
    printf("Enter the Number for table : ");
    scanf("%d", &num);

    // Logic
    int i = 1;
    int temp = 0;
    do
    {
        temp = i * num;
        printf("%d x %d = %d", i, num, temp);
        printf("\n");
        i++;
    } while (i <= 10);
    temp = 0;
```

}

Output:-

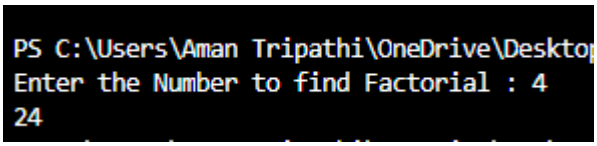

PROBLEMS
OUTPUT
TERMINAL
GITLENS
DEBUG CONSOLE
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\Aman Tripathi\OneDrive\Desktop\DSA file> cd "c:\Us
Enter the Number for table : 12
1 x 12 = 12
2 x 12 = 24
3 x 12 = 36
4 x 12 = 48
5 x 12 = 60
6 x 12 = 72
7 x 12 = 84
8 x 12 = 96
9 x 12 = 108
10 x 12 = 120
PS C:\Users\Aman Tripathi\OneDrive\Desktop\DSA file> S

Practical 3

Code:-

```
// Q 3. WAP to calculate the factorial of a number.
#include <stdio.h>
int fact(int num)
{
    if (num == 0)
        return 1;
    else
        return num * fact(num - 1);
}
void main()
{
    int num = 0;
    printf("Enter the Number to find Factorial : ");
    scanf("%d", &num);
    int fac = fact(num);
    printf("%d", fac);
}
```

Output:-



Practical 4

Code:-

```
// Q 4. WAP to print Fibonacci series using function.

// 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

#include <stdio.h>

void fibonacci(int range)

{

   int a = 0, b = 1, c;

   while (a <= range)

   {

      printf("%d\t", a);

      c = a + b;

      a = b;

      b = c;

   }

}

void main()

{

   int limit = 0;
```

```
    printf("Enter the Limit of the series : ");
    scanf("%d", &limit);
    printf("The fibonacci series is : ");
    fibonacci(limit);
}
```

Output:-

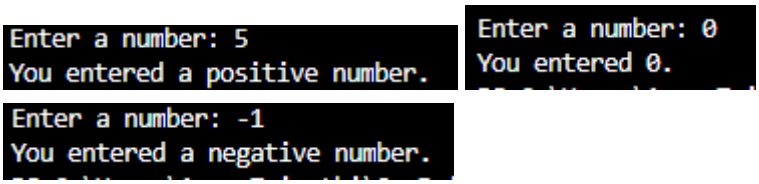
Practical 5

Code:-

```
/*
Q 5. WAP to enter given list of numbers and find how many
positive, negative or zero.
*/
#include <stdio.h>
int main()
{
    double num;
    printf("Enter a number: ");
    scanf("%lf", &num);

    if (num < 0.0)
```

```
        printf("You entered a negative number.");
    else if (num > 0.0)
        printf("You entered a positive number.");
    else
        printf("You entered 0.");
    return 0;
}
```

Output:-



Practical 6

Code:-

```
// Q 6. WAP to sort a given list of no in ascending order and print
#include <stdio.h>
// Sort the array in ascending order
void sort(int arr[], int length)
{
    int temp = 0;
    for (int i = 0; i < length; i++)
    {
        for (int j = i + 1; j < length; j++)
        {
```

```
            if (arr[i] > arr[j])
            {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
int main()
{
    // Initialize array
    int n = 0;
    printf("Enter the Size of array : ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the Element of the Array: ");
    for (int i = 0; i < n; i++)
    { scanf("%d", &arr[i]); }
    // Calculate length of array arr
    int length = sizeof(arr) / sizeof(arr[0]);
    // Displaying elements of original array
    printf("Elements of original array: \n");
    for (int i = 0; i < length; i++)
    { printf("%d ", arr[i]); }
    sort(arr, length);
    printf("\n");
```

```
    // Displaying elements of array after sorting
    printf("Elements of array sorted in ascending order: \n");
    for (int i = 0; i < length; i++)
    {
        printf("%d ", arr[i]);
    }  return 0;
}
```

Output:-

```
Enter the Size of array : 5
Enter the Element of the Array: 2
1
3
4
5
Elements of original array:
2 1 3 4 5
Elements of array sorted in ascending order:
1 2 3 4 5
```

Practical 7

Code:-

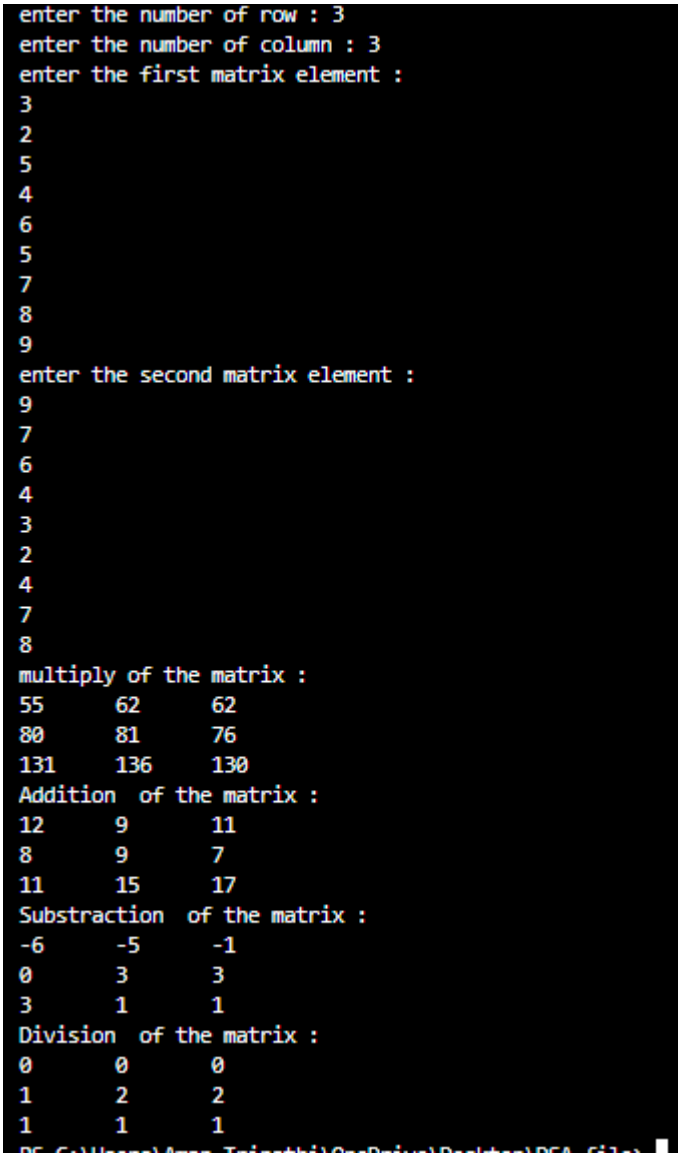
```
/*Q 7. WAP to calculate addition, subtraction, multiplication of
matrix.*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int a[10][10], b[10][10], mul[10][10], r, c, i, j, k;
```

```
system("cls");
printf("enter the number of row : ");
scanf("%d", &r);
printf("enter the number of column : ");
scanf("%d", &c);
printf("enter the first matrix element : \n");
for (i = 0; i < r; i++)
{
   for (j = 0; j < c; j++)
   {
      scanf("%d", &a[i][j]);
   }
}
printf("enter the second matrix element : \n");
for (i = 0; i < r; i++)
{
   for (j = 0; j < c; j++)
   {
      scanf("%d", &b[i][j]);
   }
}

printf("multiply of the matrix : \n");
for (i = 0; i < r; i++)
{
   for (j = 0; j < c; j++)
   {
```

```
        mul[i][j] = 0;
        for (k = 0; k < c; k++)
        {
            mul[i][j] += a[i][k] * b[k][j];
        }
    }
}

// for printing result
for (i = 0; i < r; i++)
{
    for (j = 0; j < c; j++)
    {
        printf("%d\t", mul[i][j]);
    }
    printf("\n");
}

// Addition of Matrix
printf("Addition  of the matrix : \n");
for (i = 0; i < r; i++)
{
    for (j = 0; j < c; j++)
    {
        printf("%d\t", a[i][j] + b[i][j]);
    }
    printf("\n");
```

```
    }
    // Substraction of Matrix
    printf("Substraction  of the matrix : \n");
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            printf("%d\t", a[i][j] - b[i][j]);
        }
        printf("\n");
    }
    //  Division of Matrix
    printf("Division  of the matrix : \n");
    for (i = 0; i < r; i++)
    {
        for (j = 0; j < c; j++)
        {
            printf("%d\t", a[i][j] / b[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Output:-


enter the number of row : 3
enter the number of column : 3
enter the first matrix element :
3
2
5
4
6
5
7
8
9
enter the second matrix element :
9
7
6
4
3
2
4
7
8
multiply of the matrix :
55 62 62
80 81 76
131 136 130
Addition of the matrix :
12 9 11
8 9 7
11 15 17
Substraction of the matrix :
-6 -5 -1
0 3 3
3 1 1
Division of the matrix :
0 0 0
1 2 2
1 1 1

Practical 8

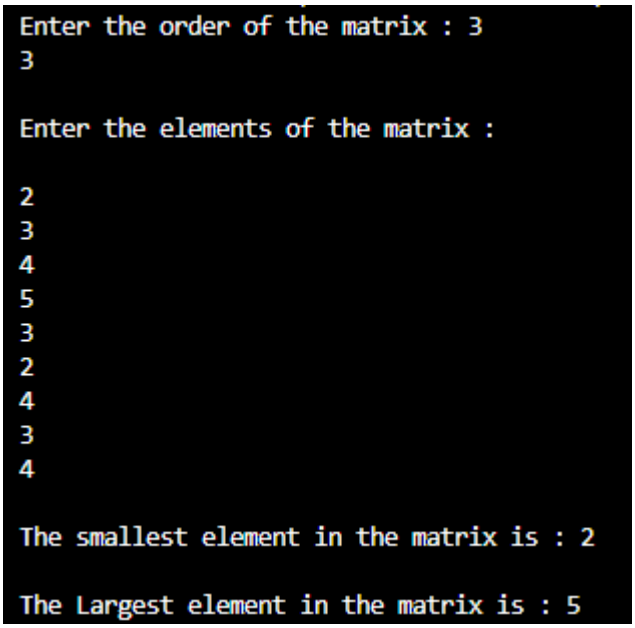
Code:-

// Q 8. WAP to find largest/smallest element of matrix.

```
#include <stdio.h>
void main()
{
    int mat[10][10];
    int i, j, row, col, small, big;

    printf("Enter the order of the matrix : ");
    scanf("%d %d", &row, &col);
    printf("\nEnter the elements of the matrix : \n\n");
    for (i = 0; i < row; i++)
        for (j = 0; j < col; j++)
            scanf("%d", &mat[i][j]);
    big = mat[0][0];
    small = mat[0][0];
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < col; j++)
        {
            if (mat[i][j] < small)
                small = mat[i][j];
            if (mat[i][j] > big)
```

```
            big = mat[i][j];
        }
    }
    printf("\nThe smallest element in the matrix is : %d\n\n", small);
    printf("The Largest element in the matrix is : %d", big);
}
```

Output:-



Practical 9

Code:-

```
// Q 9. WAP to calculate sum of each rows and columns and total
//of all elements of the matrix.
#include <stdio.h>

int main()
{
   int rows, cols, sumRow, sumCol;



   //Initialize matrix a
   int a[][3] = {
                  {1, 2, 3},
                  {4, 5, 6},
                  {7, 8, 9}
               };

   //Calculates number of rows and columns present in given matrix
   rows = (sizeof(a)/sizeof(a[0]));
   cols = (sizeof(a)/sizeof(a[0][0]))/rows;
```

```
    //Calculates sum of each row of given matrix
    for(int i = 0; i < rows; i++){
        sumRow = 0;
        for(int j = 0; j < cols; j++){
          sumRow = sumRow + a[i][j];
        }
        printf("Sum of %d row: %d\n", (i+1), sumRow);
    }

    //Calculates sum of each column of given matrix
    for(int i = 0; i < cols; i++){
        sumCol = 0;
        for(int j = 0; j < rows; j++){
          sumCol = sumCol + a[j][i];
        }
        printf("Sum of %d column: %d\n", (i+1), sumCol);
    }

    return 0;
}
```

Output:-

```
Sum of 1 row: 6
Sum of 2 row: 15
Sum of 3 row: 24
Sum of 1 column: 12
Sum of 2 column: 15
Sum of 3 column: 18
```

Practical 10

Code:-

```
// Q 10. WAP to search element form array using linear search.

#include <stdio.h>

int search(int arr[], int n, int x)

{

    int i;

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

        if (arr[i] == x)

            return i;

    return -1;

}

int main(void)

{

    int arr[] = {2, 3, 4, 10, 40};

    int x = 10;

    int n = sizeof(arr) / sizeof(arr[0]);

    // Function call
```

```
    int result = search(arr, n, x);
    (result == -1)
        ? printf("Element is not present in array")
        : printf("Element is present at index %d", result);
    return 0;
}
```

Output:-

```
Element is present at index 3
```

Practical 11

Code:-

```
// Q 11. WAP to search element form array using binary search.

#include <stdio.h>

int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        int m = l + (r - l) / 2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;
```

```
        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }
    return -1;
}

int main(void)
{
    int arr[] = {2, 3, 4, 10, 40};
    int n = sizeof(arr) / sizeof(arr[0]);
    int x = 10;
    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? printf("Element is not present"
                            " in array")
                   : printf("Element is present at "
                            "index %d",
                            result);
    return 0;
}
```

Output:-

```
Element is present at index 3
```

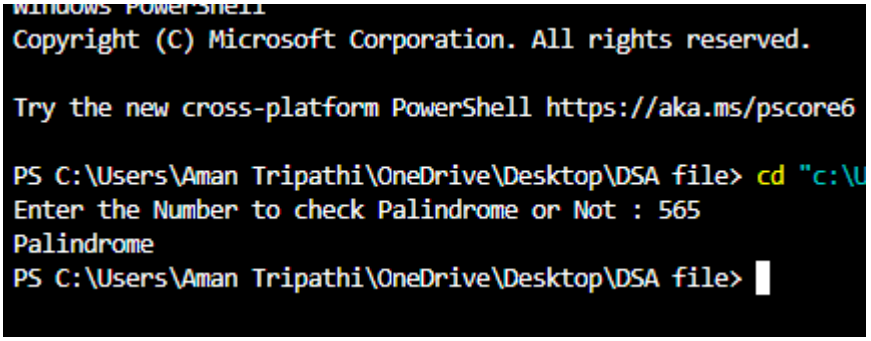
Practical 12

Code:-

```
// Q 12. WAP to check weather number is palindrome.

#include <stdio.h>

int palindrome(int num)

{

   int temp = 0;

   int duplicate = num;

   while (num > 0)

   {

      int d = num % 10;

      temp = temp * 10 + d;
```

```
        num = num / 10;
    }
    // printf("%d",temp);
    if (duplicate == temp)
    {
        return 1;
    }
    else
        return 0;
}
void main()
{
    int num = 0;
    printf("Enter the Number to check Palindrome or Not : ");
    scanf("%d", &num);
    // palindrome(num);
    int ans = palindrome(num);
    if (ans == 0)
        printf("Not Palindrome");
    else if (ans == 1)
        printf("Palindrome");
}
```

Output:-



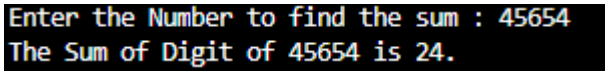
Practical 13

Code:-

```
//  Q 13. WAP to calculate sum of digit of number.

#include <stdio.h>

int sumOfDigit(int num)

{
```

```
    int sum = 0;
    while (num > 0)
    {
        int d = num % 10;
        sum += d;
        num /= 10;
    }
    return sum;
}
int main(int argc, char const *argv[])
{
    int num = 0;
    printf("Enter the Number to find the sum : ");
    scanf("%d", &num);
    int sum = sumOfDigit(num);
    printf("The Sum of Digit of %d is %d.", num, sum);
    return 0;
}
```

Output:-



Practical 14

Code:-

```
// Q14. WAP to for Bubble Sort to Sort Elements in An Order.

#include <stdio.h>


void print(int a[], int n) // function to print array elements

{

   int i;

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

   {

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

   }

}

void bubble(int a[], int n) // function to implement bubble sort

{

   int i, j, temp;

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

   {

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

      {

         if (a[j] < a[i])

         {

            temp = a[i];

            a[i] = a[j];

            a[j] = temp;
```

```
            }
        }
    }
}
void main()
{
    int i, j, temp;
    int a[5] = {10, 35, 32, 13, 26};
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    print(a, n);
    bubble(a, n);
    printf("\nAfter sorting array elements are - \n");
    print(a, n);
}
```

Output:-

```
Before sorting array elements are -
10 35 32 13 26
After sorting array elements are -
10 13 26 32 35
```

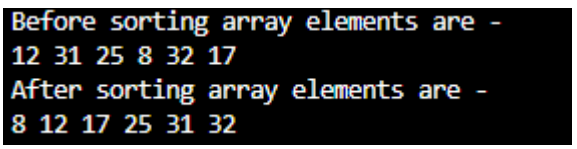
Practical 15

Code:-

```
// Q15. WAP to for insertion Sort to Sort Elements in An Order.

#include <stdio.h>


void insert(int a[], int n) /* function to sort an aay with insertion sort */

{

   int i, j, temp;

   for (i = 1; i < n; i++)

   {

      temp = a[i];

      j = i - 1;


      while (j >= 0 && temp <= a[j]) /* Move the elements greater than temp to one position ahead from their current position*/

      {

         a[j + 1] = a[j];

         j = j - 1;

      }

      a[j + 1] = 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()
{
    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);
    insert(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);

    return 0;
}
```

Output:-



Practical 16

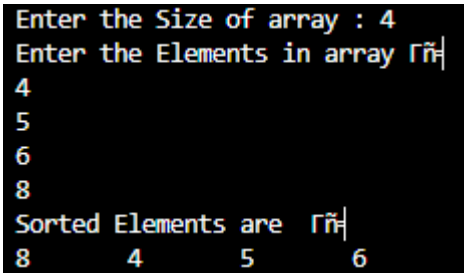
Code:-

```
// WAP to for selection Sort to Sort Elements in An Order.


#include <stdio.h>
void swap(int arr[], int first, int second)
{
   int temp = arr[first];
   arr[first] = arr[second];
   arr[second] = temp;
}
int getMaxIndex(int arr[], int start, int end)
{
   int max = start;
   for (int i = start; i < end; i++)
   {
      if (arr[max] < arr[i])
      {
```

```
            max = i;
        }
    }
    return max;
}
void selectionSort(int arr[], int length)
{

    for (int i = 0; i < length; i++)
    {
        // find the max item in remaining array and swap with correct with correct index
        int last = length - i - 1;
        int maxIndex = getMaxIndex(arr, 0, last);
        swap(arr, maxIndex, last);
    }
}

void main()

{
    int size = 0;
    printf("Enter the Size of array : ");
    scanf("%d", &size);
    int array[size];
    int length = sizeof(array) / sizeof(int);
    printf("Enter the Elements in array ↘");
    printf("\n");
```

```
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &array[i]);
    }
    selectionSort(array, size);
    printf("Sorted Elements are ↘ ");
    printf("\n");
    for (int i = 0; i < size; i++)
    {
        printf("%d\t", array[i]);
    }
}
```

Output:-



Practical 17

Code:-

```
// Q17. WAP to for merge Sort to Sort Elements in An Order.


#include <stdio.h>

#include <stdlib.h>
```

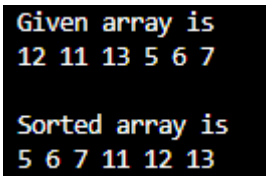
```
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
```

```
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there
    are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* l is for left index and r is right index of the
sub-array of arr to be sorted */
```

```
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

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

/* UTILITY FUNCTIONS */
/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver code */
int main()
```

```
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Output:-



Practical 18 & 19

Code:-

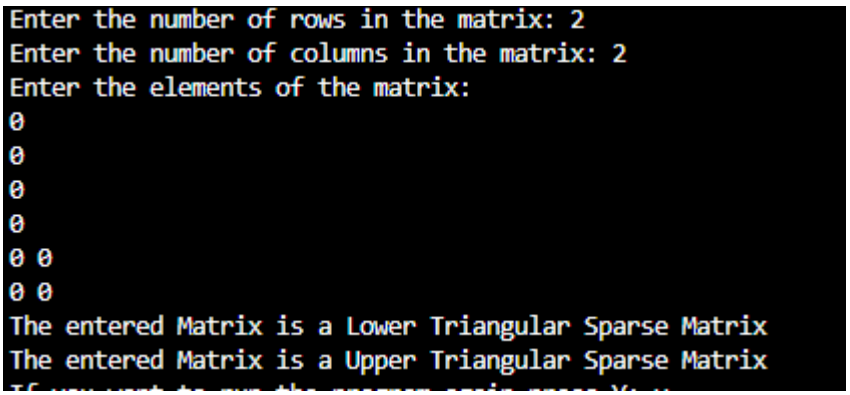
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char ans;
    ans='Y';
    while(ans=='Y'||ans=='y'){
    int rows,cols;
    printf("Enter the number of rows in the matrix: ");
    scanf("%d", &rows);
    printf("Enter the number of columns in the matrix: ");
    scanf("%d", &cols);
    int arr[rows+1][cols+1];
    int flag, flag1;
    flag=0;
    flag1=0;
    printf("Enter the elements of the matrix: \n");
    for(int i=0;i<rows;++i)
    {
        for(int j=0;j<cols;++j)
        {
            scanf("%d", &arr[i][j]);
```

```
    }
}
for(int i=0;i<rows;++i)
{
    for(int j=0;j<cols;++j)
    {
        printf("%d ", arr[i][j]);
    }
    printf("\n");
}
for(int i=0;i<rows;++i)
{
    for(int j=0;j<cols;++j)
    {
        if(j>i && arr[i][j]!=0)
        {
            flag=1;
            break;
        }
    }
}

if(flag==0)
{
    printf("The entered Matrix is a Lower Triangular Sparse Matrix\n");
}
else
```

```
{
   printf("The entered Matrix is not a Lower Triangular Sparse Matrix\n");
}

for(int i=0;i<rows;++i)
{
   for(int j=0;j<cols;++j)
   {
      if(j<i &&  arr[i][j]!=0)
      {
         flag1=1;
         break;
      }
   }
}

if(flag1==0)
{
   printf("The entered Matrix is a Upper Triangular Sparse Matrix\n");
}
else
{
   printf("The entered Matrix is not a Upper Triangular Sparse Matrix\n");
}

printf("If you want to run the program again press Y: ");
scanf(" %c", &ans);
```

```
    if(ans!='Y'&& ans!='y')
    {
        printf("Exiting the program");
        exit(0);
    }
    printf("\n");
}
    return 0;
}
```

Output:-



Practical 20

Code:-

```
 #include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *link;
};
struct node *root = NULL;
int len;

void append()
{
   struct node *temp;
   temp = (struct node *)malloc(sizeof(struct node));
   printf("Enter the Node Data: ");
   scanf("%d", &temp->data);
   temp->link = NULL;
   if (root == NULL)
   {
      root = temp;
   }
   else
   {
```

```
        struct node *p;
        p = root;
        while (p->link != NULL)
        {
          p=  p->link;
        }
        p->link = temp;
    }
}
int length()
{
    int count = 0;
    struct node *temp;
    temp = root;
    while (temp != NULL)
    {
        count++;
        temp = temp->link;
    }
    return count;
}
void display()
{
    struct node *temp;
    temp = root;
    if (temp == NULL)
    {
```

```
        printf("List Empty \n\n");
    }
    else
    {
        while (temp != NULL)
        {
            printf("%d   ", temp->data);
            temp = temp->link;
        }
        printf("\n\n");
    }
}
void addAfter()
{
    struct node *temp;
    struct node *p;
    int loc, len, c = 1;
    printf("Enter Location: ");
    scanf("%d",&loc);
    len = length();
    if (loc > len)
    {
        printf("Invalid Location\n");
        printf("Currently List is having %d nodes", len);
    }
    else
    {
```

```
        p = root;
        while (c < loc)
        {
            p = p->link;
            c++;
        }
        temp = (struct node *)malloc(sizeof(struct node));
        printf("Enter Node Data :");
        scanf("%d",&temp->data);
        temp->link = p->link;
        p->link = temp;
    }
}
void addAtBegin(){
printf("Sorry! No function found...");
}
void delete(){
    struct node * temp;
    int loc;
printf("Enter Location to delete: ");
scanf("%d",&loc);
if (loc>length())
{
  printf("Invalid Location \n");
}
else if (loc==1)
{
```

```
    temp=root;
    root=temp->link;
    temp->link=NULL;
    free(temp);

}
else{
    struct node * p = root,*q;
    int i = 1;
    while (i<loc-1)
    {
        p=p->link;
        i++;

    }
    q=p->link;
    p->link=q->link;
    q->link=NULL;
    free(q);


}
}
void main()
{
    int ch;
    while (1)
```

```
{
    printf("Single Linkedlist operations: \n");
    printf("1. Append \n");
    printf("2. Add at beginning \n");
    printf("3. Add after \n");
    printf("4. Length \n");
    printf("5. Display \n");
    printf("6. Delete \n");
    printf("7. Quit \n");
    printf("Enter Your Choice :");
    scanf("%d", &ch);
    switch (ch)
    {
    case 1:
        append();
        break;
    case 2:
        addAtBegin();
        break;
    case 3:
        addAfter();
        break;
    case 4:
        len = length();
        printf("Length: %d\n", len);
        break;
    case 5:
```

```
            display();
            break;
        case 6:
            delete ();
            break;
        case 7:
            exit(1);
            break;
        default:
            printf("Invalid Input \n");
            break;
        }
    }
}
```

Output:-

```
Single Linkedlist operations:
1. Append
2. Add at beginning
3. Add after
4. Length
5. Display
6. Delete
7. Quit
Enter Your Choice :1
Enter the Node Data: 56
Single Linkedlist operations:
1. Append
2. Add at beginning
3. Add after
4. Length
5. Display
6. Delete
7. Quit
Enter Your Choice :1
Enter the Node Data: 56
Single Linkedlist operations:
1. Append
2. Add at beginning
3. Add after
4. Length
5. Display
6. Delete
7. Quit
Enter Your Choice :1
Enter the Node Data: 68
```

```
Single Linkedlist operations:
1. Append
2. Add at beginning
3. Add after
4. Length
5. Display
6. Delete
7. Quit
Enter Your Choice :156
Invalid Input
Single Linkedlist operations:
1. Append
2. Add at beginning
3. Add after
4. Length
5. Display
6. Delete
7. Quit
Enter Your Choice :1
Enter the Node Data: 67
Single Linkedlist operations:
1. Append
2. Add at beginning
3. Add after
4. Length
5. Display
6. Delete
7. Quit
Enter Your Choice :5
56   56   68   67
```

Practical 21

Code:-

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

struct node
{
   int data;
   struct node *left;
   struct node *right;
};

struct node *root=NULL;
struct node *tail=NULL;
int len;

void append();
void addatbegin();
void display();
int length();
void delaloc();
void addafter();

int main()
```

```
{
    int ch;
    while(1)
    {
        printf("\nDouble Linked List Operations: \n");
        printf("1. Append\n");
        printf("2. Add At Beginning\n");
        printf("3. Length\n");
        printf("4. Display\n");
        printf("5. Add after a Particular Location\n");
        printf("6. Delete at a Certain Location\n");
        printf("7. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:append();
            break;
            case 2:addatbegin();
            break;
            case 3:len=length();
            printf("Length is: %d\n", len);
            break;
            case 4:display();
            break;
            case 5:addafter();
            break;
```

```
            case 6: delaloc();
                    display();
            break;
            case 7:exit(1);
            default:printf("Invalid Input");
        }
    }

    return 0;
}

void append()
{
    struct node *temp;
    temp=(struct node*)malloc (sizeof(struct node));
    printf("Enter node data: ");
    scanf("%d", &temp->data);
    temp->left=NULL;
    temp->right=NULL;
    if(root==NULL)
    {
        root=temp;
    }
    else
    {
        struct node *p;
        p=root;
```

```
        while(p->right!=NULL)
        {
            p=p->right;
        }
        p->right=temp;
        temp->left=p;
        tail=temp;
    }
}

void addatbegin()
{
    struct node *temp;
    temp=(struct node*)malloc(sizeof(struct node));
    printf("Enter node data: ");
    scanf("%d", &temp->data);
    temp->left=NULL;
    temp->right=NULL;
    if(root==NULL)
    {
        root=temp;
    }
    else
    {
        temp->right=root;
        root->left=temp;
        root=temp;
```

```
    }
}

int length()
{
    struct node *temp=root;
    int count=0;
    while(temp!=NULL)
    {
        ++count;
        temp=temp->right;
    }
    return count;
}

void display()
{
    struct node *temp=root;
    if(temp==NULL)
    {
        printf("List is Empty\n");
    }
    else
    {
        while(temp!=NULL)
        {
            printf("%d ",temp->data);
```

```
            temp=temp->right;
        }
    }
}



void addafter()
{
    struct node *temp, *p;
    int loc,len,i=1;
    printf("Enter Location: ");
    scanf("%d", &loc);
    len=length();
    if(loc>len)
    {
        printf("Invalid Location\n");
    }
    else
    {
        temp=(struct node*)malloc(sizeof(struct node));
        printf("Enter node data: ");
        scanf("%d", &temp->data);
        temp->left=NULL;
        temp->right=NULL;
        p=root;
        while(i<loc)
        {
```

```
            p=p->right;
            ++i;
        }
        temp->right=p->right;
        p->right->left=temp;
        temp->left=p;
        p->right=temp;
    }
}


void delaloc()
{
    int loc,len;
    printf("Enter the location to delete from: ");
    scanf("%d", &loc);
    len=length();
    if(len==0)
    {
        printf("Empty List\n");
    }
    else if(loc>len)
    {
        printf("Invalid Location\n");
    }
    else if(root==tail)
    {
```

```
    struct node *temp;
    temp=root;
    root=NULL;
    tail=NULL;
    free(temp);
}
else if(loc==1)
{
    struct node *temp;
    temp=root;
    root=root->right;
    root->left=0;
    free(temp);
}
else if(loc==len)
{
    struct node *temp;
    temp=tail;
    tail=tail->left;
    tail->right=0;
    free(temp);
}
else
{
    struct node *p, *q;
    p=root;
    int i=1;
```

```
        while(i<loc-1)
        {
            p=p->right;
            ++i;
        }
        q=p->right;
        p->right=q->right;
        q->right->left=p;
        free(q);

    }
}
```

Output:-

```
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 1
Enter node data: 21

Double Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 1
Enter node data: 21

Double Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 1
Enter node data: 32

Double Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 2
Enter node data: 43

Double Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 3
Length is: 4

Double Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 4
43 21 21 32
Double Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 5
Enter Location: 3
Enter node data: 43

Double Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
```

Practical 22

Code:-

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

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

struct node*root=NULL;
struct node *tail=NULL;
int len;

void append();
int length();
void display();
void addafter();
void reverse_list();
void addatbeg();
void sort1();
void sort2();
void delete();
```

```
void main()
{
   int ch;
   while(1)
   {
      printf("\nSingle Circular Linked List Operations: \n");
      printf("1. Append\n");
      printf("2. Add After\n");
      printf("3. Length\n");
      printf("4. Display\n");
      printf("5. Delete\n");
      printf("6. Add At Beginning\n");
      /*printf("8. Selection Sort\n");
      printf("9. Bubble Sort\n");*/
      printf("7. Quit\n");
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch(ch)
      {
         case 1:append();
         break;
         case 2:addafter();
         break;
         case 3:len=length();
         printf("Length is: %d\n", len);
         break;
```

```
            case 4:display();
            break;
            case 5:delete();
            break;
            case 6: addatbeg();
            break;
            case 7:exit(1);
            default:printf("Invalid Input");
        }
    }

}
void append()
{
    struct node *temp;
    temp=(struct node*)malloc(sizeof(struct node));
    printf("Enter the node data: ");
    scanf("%d", &temp->data);
    temp->link=NULL;
    if(root==NULL)
    {
        root=temp;
        tail=temp;
        temp->link=root;
    }
    else
    {
```

```
        struct node *p;
        p=root;
        while(p->link!=root)
        {
           p=p->link;
        }
        p->link=temp;
        temp->link=root;
        tail=temp;
    }
}

int length()
{
    int count=0;
    struct node *temp;
    temp=root;
    while(temp->link!=root)
    {
        ++count;
        temp=temp->link;
    }
    return count+1;
}

void display()
{
```

```
    struct node *temp;
    temp=root;
    if(temp==NULL)
    {
        printf("List is empty\n\n");
    }
    else
    {
        while(temp->link!=root)
        {
            printf("%d ", temp->data);
            temp=temp->link;
        }
        printf("%d ", temp->data);
        printf("\n\n");
    }
}

void addafter()
{
    struct node *temp;
    struct node *p;
    int loc, len,i;
    i=1;
    printf("Enter Location: ");
    scanf("%d", &loc);
    len=length();
```

```
    if(loc>len)
    {
        printf("Invalid Location\n");
        printf("Currently list is having %d nodes: ", len);
    }
    else
    {
        p=root;
        while(i<loc)
        {
            p=p->link;
            ++i;
        }
        temp=(struct node*)malloc(sizeof(struct node));
        printf("Enter node data: ");
        scanf("%d", &temp->data);
        temp->link=p->link;
        p->link=temp;
    }

}

void delete()
{
    struct node *temp;
    int loc;
    printf("Enter Location to Delete: ");
```

```
scanf("%d", &loc);
if(loc>length())
{
   printf("Invalid Location\n");
}
else if(loc==1)
{
   temp=root;
   root=temp->link;
   tail->link=root;
   free(temp);
}
else
{
   struct node *p=root;
   struct node *q;
   int i=1;
   while(i<loc-1)
   {
      p=p->link;
      ++i;
   }
   if(loc==length())
   {
      tail=p->link;
      q=p->link;
      p->link=q->link;
```

```
            free(q);
        }
        else
        {
            q=p->link;
            p->link=q->link;
            free(q);
        }
    }
}

void addatbeg()
{
    struct node *temp;
    temp=(struct node*)malloc(sizeof(struct node));
    printf("Enter data: ");
    scanf("%d", &temp->data);
    if(root==NULL)
    {
        root=temp;
    }
    else
    {
        temp->link=root;
        root=temp;
        tail->link=temp;
    }
```

}

Output:-

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 1
Enter the node data: 65
```

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 6
Enter data: 23
```

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 1
Enter the node data: 65
```

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 5
Enter Location to Delete: 3
```

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 2
Enter Location: 1
Enter node data: 65
```

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 4
23 65 65
```

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 4
65 65 65
```

```
Single Circular Linked List Operations:
1. Append
2. Add After
3. Length
4. Display
5. Delete
6. Add At Beginning
7. Quit
Enter your choice: 3
Length is: 3
```

Practical 23

Code:-

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

struct node
{
   int data;
   struct node *left;
   struct node *right;
};

struct node *root=NULL;
struct node *tail=NULL;
int len;

void append();
void display();
int length();
void addatbegin();
void addafter();
void delaloc();

int main()
{
```

```
int ch;
while(1)
{
   printf("\nDouble Circular Linked List Operations: \n");
   printf("1. Append\n");
   //printf("2. Add At Beginning\n");
   printf("2. Add At Beginning\n");
   printf("3. Length\n");
   printf("4. Display\n");
   printf("5. Add after a Particular Location\n");
   printf("6. Delete at a Certain Location\n");
   printf("7. Quit\n");
   printf("Enter your choice: ");
   scanf("%d", &ch);
   switch(ch)
   {
      case 1:append();
      break;
      case 2:addatbegin();
      break;
      case 3:len=length();
      printf("Length is: %d\n", len);
      break;
      case 4:display();
      break;
      case 5:addafter();
      break;
```

```
            case 6: delaloc();
                    display();
            break;
            case 7:exit(1);
            default:printf("Invalid Input");
        }
    }

    return 0;
}

void append()
{
    struct node *newnode;
    newnode=(struct node*)malloc(sizeof(struct node));
    printf("Enter data: ");
    scanf("%d", &newnode->data);
    newnode->left=NULL;
    newnode->right=NULL;
    if(tail==NULL)
    {
        tail=newnode;
        root=newnode;
        tail->right=newnode;
        root->left=newnode;
    }
    else
```

```
    {
        newnode->left=tail;
        newnode->right=root;
        tail->right=newnode;
        tail=newnode;

    }

}

void display()
{
    struct node *temp;
    if(root==NULL)
    {
        printf("Empty List\n");
    }
    else if(root==tail)
    {
        printf("%d\n", root->data);
    }
    else
    {
        temp=root;
        while(temp!=tail)
        {
            printf("%d ", temp->data);
```

```
            temp=temp->right;
        }
        printf("%d ", temp->data);
    }


}


int length()
{
    struct node *temp=root;
    int count=0;
    while(temp!=tail)
    {
        ++count;
        temp=temp->right;
    }
    return count+1;
}

void addatbegin()
{
    struct node *temp;
    temp=(struct node*)malloc(sizeof(struct node));
    printf("Enter node data: ");
    scanf("%d", &temp->data);
```

```
    temp->left=NULL;
    temp->right=NULL;
    if(root==NULL)
    {
        root=temp;
        tail=temp;
        temp->right=tail;
        temp->left=tail;
    }
    else
    {
        temp->left=tail;
        temp->right=root;
        root->left=temp;
        tail->right=temp;
        root=temp;

    }
}

void addafter()
{
    struct node *temp, *p;
    int loc,len,i=1;
    printf("Enter Location: ");
    scanf("%d", &loc);
    len=length();
```

```
if(loc>len)
{
   printf("Invalid Location\n");
}
else
{
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter node data: ");
   scanf("%d", &temp->data);
   temp->left=NULL;
   temp->right=NULL;
   p=root;
   while(i<loc)
   {
      p=p->right;
      ++i;
   }

   if(loc==len)
   {
      temp->right=p->right;
      p->right->left=temp;
      temp->left=p;
      p->right=temp;
      tail=temp;
   }
   else
```

```
        {
            temp->right=p->right;
            p->right->left=temp;
            temp->left=p;
            p->right=temp;
        }
    }
}

void delaloc()
{
    int loc,len;
    printf("Enter the location to delete from: ");
    scanf("%d", &loc);
    len=length();
    if(len==0)
    {
        printf("Empty List\n");
    }
    else if(loc>len)
    {
        printf("Invalid Location\n");
    }
    else if(root==tail)
    {
        struct node *temp;
        temp=root;
```

```
    root=NULL;
    tail=NULL;
    free(temp);
}
else if(loc==1)
{
    struct node *temp;
    temp=root;
    root=root->right;
    root->left=tail;
    free(temp);
}
else if(loc==len)
{
    struct node *temp;
    temp=tail;
    tail=tail->left;
    tail->right=root;
    free(temp);
}
else
{
    struct node *p, *q;
    p=root;
    int i=1;
    while(i<loc-1)
    {
```

```
            p=p->right;
            ++i;
        }
        q=p->right;
        p->right=q->right;
        q->right->left=p;
        free(q);

    }
}
```

Output:-

```
Double Circular Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 2
Enter node data: 56

Double Circular Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 1
Enter data: 63

Double Circular Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 1
Enter data: 54

Double Circular Linked List Operatic
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 5
Enter Location: 5
Invalid Location
```

```
Double Circular Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 1
Enter data: 58

Double Circular Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 3
Length is: 4

Double Circular Linked List Operations:
1. Append
2. Add At Beginning
3. Length
4. Display
5. Add after a Particular Location
6. Delete at a Certain Location
7. Quit
Enter your choice: 4
56 63 54 58
```

Practical 24

Code:-

```
#include <stdio.h>

#include <stdlib.h>

// #define Max 5;

int Max = 5;

int stack[5];

int top = -1;


int isEmpty() { return (top == -1) ? 1 : 0; }

int isFull() { return (top == Max - 1) ? 1 : 0; }


void push(int data)

{

   if (isFull() == 1)

   {

      printf("Overflow");

   }

   else

   {

      top += 1;

      stack[top] = data;

      printf("Push Successfully");

   }

}
```

```
int pop()
{
    int temp;
    if (isEmpty() == 0)
    {
        temp = stack[top];
        top -= 1;
    }
    else
        printf("Underflow");
    return temp;
}
void peek()
{
    if (isEmpty() == 0)
    {
        printf("%d", stack[top]);
    }
    else
        printf("Stack is Empty");
}
void display()
{
    if (isEmpty() == 0)
    {
        for (int i = top; i >= 0; i--)
        {
```

```
            printf("%d \n", stack[i]);
        }
    }
}
void main()
{
    int input;
    int data;
    while (1)
    {
        printf(" \n0. Exit \n");
        printf("1. Push \n");
        printf("2. Pop \n");
        printf("3. peek \n");
        printf("4. Display \n");
        printf("5. Check Empty or not \n");
        printf("6. Check Full or not \n");
        scanf("%d", &input);
        switch (input)
        {
        case 0:
            exit(0);
            break;
        case 1:
            printf("Enter data to push: ");
            scanf("%d", &data);
            push(data);
```

```
    break;
case 2:
    printf("Deleted element is %d.", pop());
    break;
case 3:
    peek();
    break;
case 4:
    display();
    break;
case 5:
    if (isEmpty() == 1)
        printf("Stack is Empty");
    else
        printf("Stack is not Empty");
    break;
case 6:
    if (isFull() == 1)
    {
        printf("Stack is full");
    }
    else
        printf("Stack is not full");

    break;
default:
    printf("Wrong input");
```

```
            break;
        }
    }
}
```

Output:-

```
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
1
Enter data to push: 56
Push Successfully
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
2
Deleted element is 56.
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
1
Enter data to push: 69
Push Successfully
```

```
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
3
85
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
5
Stack is not Empty
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
6
Stack is not full
```

Practical 25

Code:-

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

struct stack
{
   int data;
   struct stack *link;
};
struct stack *top = 0;
void push(int item)
{
   struct stack *newNode;
   newNode = (struct stack *)malloc(sizeof(struct stack));
   newNode->data = item;
   newNode->link = top;
   top = newNode;
}
int pop()
{
   struct stack *temp;
   temp = top;
   if (top == NULL)
      printf("Stack is Empty");
```

```
    else
    {
        top = temp->link;
        // top= top->link;
        free(temp);
    }
    return top->data;
}
void peek()
{
    if (top == NULL)
    {

        printf("Stack is Empty");
    }
    else
    {
        printf("%d", top->data);
    }
}
void display()
{
    struct stack *temp = top;
    while (temp != NULL)
    {
        printf("%d \n", temp->data);
        temp = temp->link;
```

```
    }
}
void main()
{
    int input;
    int data;
    while (1)
    {
        printf(" \n0. Exit \n");
        printf("1. Push \n");
        printf("2. Pop \n");
        printf("3. peek \n");
        printf("4. Display \n");
        printf("5. Check Empty or not \n");
        printf("6. Check Full or not \n");
        scanf("%d", &input);
        switch (input)
        {
        case 0:
            exit(0);
            break;
        case 1:
            printf("Enter data to push: ");
            scanf("%d", &data);
            push(data);
            break;
        case 2:
```

```
            printf("Deleted element is %d.", pop());
            break;
        case 3:
            peek();
            break;
        case 4:
            display();
            break;
        default:
            printf("Wrong input");
            break;
        }
    }
}
```

Output:-

```
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
1
Enter data to push: 56
Push Successfully
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
2
Deleted element is 56.
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
1
Enter data to push: 69
Push Successfully
```

```
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
3
85
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
5
Stack is not Empty
0. Exit
1. Push
2. Pop
3. peek
4. Display
5. Check Empty or not
6. Check Full or not
6
Stack is not full
```

Practical 26

Code:-

```
#include <stdio.h>

#define n 5

int main()

{

   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:
            exit(0);
        default:
            printf("Wrong Choice: please see the options");
        }
      }
   }
   return 0;
}
```

Output:-

```
Queue using Array
1.Insertion
2.Deletion
3.Display
4.Exit
Enter the Choice:1

 Enter no 1:56

Enter the Choice:2

 Deleted Element is 56
Enter the Choice:1

 Enter no 2:66

Enter the Choice:3

Queue Elements are:
 66
```

Practical 27

Code:-

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

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

struct node *front = NULL;
struct node *rear = NULL;

void display();
void enqueue(int);
void dequeue();

int main()
{
   int n, ch;
   do
   {
      printf("\n\nQueue Menu\n1. Add \n2. Remove\n3. Display\n0. Exit");
      printf("\nEnter Choice 0-3? : ");
```

```
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                printf("\nEnter number ");
                scanf("%d", &n);
                enqueue(n);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
        }
    }while (ch != 0);
}

void enqueue(int item)
{
    struct node *nptr = malloc(sizeof(struct node));
    nptr->data = item;
    nptr->next = NULL;
    if (rear == NULL)
    {
        front = nptr;
        rear = nptr;
```

```
    }
    else
    {
        rear->next = nptr;
        rear = rear->next;
    }
}

void display()
{
    struct node *temp;
    temp = front;
    printf("\n");
    while (temp != NULL)
    {
        printf("%d\t", temp->data);
        temp = temp->next;
    }
}

void dequeue()
{
    if (front == NULL)
    {
        printf("\n\nqueue is empty \n");
    }
    else
```

```
    {
        struct node *temp;
        temp = front;
        front = front->next;
        printf("\n\n%d deleted", temp->data);
        free(temp);
    }
}
```

Output:-

```
Queue Menu
1. Add
2. Remove
3. Display
0. Exit
Enter Choice 0-3? : 1

Enter number 236


Queue Menu
1. Add
2. Remove
3. Display
0. Exit
Enter Choice 0-3? : 1

Enter number 265


Queue Menu
1. Add
2. Remove
3. Display
0. Exit
Enter Choice 0-3? : 3

236     265
```

```
Queue Menu
1. Add
2. Remove
3. Display
0. Exit
Enter Choice 0-3? : 2
```

Practical 28

Code:-

```
/*static circular queue*/

#include <stdio.h>

#define size 5


void insertq(int[], int);

void deleteq(int[]);

void display(int[]);


int front =  - 1;

int rear =  - 1;


int main()

{

   int n, ch;

   int queue[size];

   do

   {

      printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0. Exit");

      printf("\nEnter Choice 0-3? : ");

      scanf("%d", &ch);

      switch (ch)

      {

         case 1:
```

```
            printf("\nEnter number: ");
            scanf("%d", &n);
            insertq(queue, n);
            break;
        case 2:
            deleteq(queue);
            break;
        case 3:
            display(queue);
            break;
      }
   }while (ch != 0);
}


void insertq(int queue[], int item)
{
   if ((front == 0 && rear == size - 1) || (front == rear + 1))
   {
      printf("queue is full");
      return;
   }
   else if (rear ==  - 1)
   {
      rear++;
      front++;
   }
```

```
    else if (rear == size - 1 && front > 0)
    {
        rear = 0;
    }
    else
    {
        rear++;
    }
    queue[rear] = item;
}

void display(int queue[])
{
    int i;
    printf("\n");
    if (front > rear)
    {
        for (i = front; i < size; i++)
        {
            printf("%d ", queue[i]);
        }
        for (i = 0; i <= rear; i++)
            printf("%d ", queue[i]);
    }
    else
    {
        for (i = front; i <= rear; i++)
```

```
            printf("%d ", queue[i]);
    }
}

void deleteq(int queue[])
{
    if (front ==  - 1)
    {
        printf("Queue is empty ");
    }
    else if (front == rear)
    {
        printf("\n %d deleted", queue[front]);
        front =  - 1;
        rear =  - 1;
    }
    else
    {
        printf("\n %d deleted", queue[front]);
        front++;
    }
}
```

Output:-

```
 Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 1

Enter number: 2


 Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 1

Enter number: 2


 Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 3

2 2
```

```
 Circular Queue:
1. Insert
2. Delete
3. Display
0. Exit
Enter Choice 0-3? : 2

 2 deleted
```

Practical 29

Code:-

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
	int data;
	struct node* next;
};
struct node *f = NULL;
struct node *r = NULL;
void enqueue(int d) //Insert elements in Queue
{
	struct node* n;
	n = (struct node*)malloc(sizeof(struct node));
	n->data = d;
	n->next = NULL;
	if((r==NULL)&&(f==NULL))
	{
		f = r = n;
		r->next = f;
	}
	else
	{
		r->next = n;
```

```
                r = n;
                n->next = f;
        }
}
void dequeue() // Delete an element from Queue
{
        struct node* t;
        t = f;
        if((f==NULL)&&(r==NULL))
                printf("\nQueue is Empty");
        else if(f == r){
                f = r = NULL;
                free(t);
        }
        else{
                f = f->next;
                r->next = f;
                free(t);
        }


}
void print(){ // Print the elements of Queue
        struct node* t;
        t = f;
        if((f==NULL)&&(r==NULL))
                printf("\nQueue is Empty");
```

```
	else{
		do{
			printf("\n%d",t->data);
			t = t->next;
		}while(t != f);
	}
}
int main()
{
	int opt,n,i,data;
	printf("Enter Your Choice:-");
	do{
		printf("\n\n1 for Insert the Data in Queue\n2 for show the Data in Queue \n3 for Delete the data from the Queue\n0 for Exit");
		scanf("%d",&opt);
		switch(opt){
			case 1:
				printf("\nEnter the number of data");
				scanf("%d",&n);
				printf("\nEnter your data");
				i=0;
				while(i<n){
					scanf("%d",&data);
					enqueue(data);
					i++;
				}
				break;
```

```
                case 2:
                        print();
                        break;
                case 3:
                         dequeue();
                        break;
                case 0:
                        break;
                default:
                        printf("\nIncorrect Choice");

            }
        }while(opt!=0);
return 0;
}
```

Output:-

```
Enter the number of data 5

Enter your data
34
33
22
76
32


1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit 3


1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit 2

33
22
76
32

1 for Insert the Data in Queue
2 for show the Data in Queue
3 for Delete the data from the Queue
0 for Exit
```

Practical 30

Code:-

```
#include <stdio.h>
#define size 5
int deque[size];
int f = -1, r = -1;
//  insert_front function will insert the value from the front
void insert_front(int x)
{
   if((f==0 && r==size-1) || (f==r+1))
   {
      printf("Overflow");
   }
   else if((f==-1) && (r==-1))
   {
      f=r=0;
      deque[f]=x;
   }
   else if(f==0)
   {
      f=size-1;
      deque[f]=x;
   }
   else
   {
```

```
        f=f-1;
        deque[f]=x;
    }
}

// insert_rear function will insert the value from the rear
void insert_rear(int x)
{
    if((f==0 && r==size-1) || (f==r+1))
    {
        printf("Overflow");
    }
    else if((f==-1) && (r==-1))
    {
        r=0;
        deque[r]=x;
    }
    else if(r==size-1)
    {
        r=0;
        deque[r]=x;
    }
    else
    {
        r++;
        deque[r]=x;
    }
```

```
}

// display function prints all the value of deque.
void display()
{
   int i=f;
   printf("\nElements in a deque are: ");

   while(i!=r)
   {
      printf("%d ",deque[i]);
      i=(i+1)%size;
   }
    printf("%d",deque[r]);
}

// getfront function retrieves the first value of the deque.
void getfront()
{
   if((f==-1) && (r==-1))
   {
      printf("Deque is empty");
   }
   else
   {
      printf("\nThe value of the element at front is: %d", deque[f]);
```

```
    }

}

// getrear function retrieves the last value of the deque.
void getrear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else
    {
        printf("\nThe value of the element at rear is %d", deque[r]);
    }

}

// delete_front() function deletes the element from the front
void delete_front()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
```

```
        printf("\nThe deleted element is %d", deque[f]);
        f=-1;
        r=-1;

    }
     else if(f==(size-1))
     {
         printf("\nThe deleted element is %d", deque[f]);
         f=0;
     }
     else
     {
          printf("\nThe deleted element is %d", deque[f]);
          f=f+1;
     }
}

// delete_rear() function deletes the element from the rear
void delete_rear()
{
    if((f==-1) && (r==-1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
        printf("\nThe deleted element is %d", deque[r]);
```

```
        f=-1;
        r=-1;

    }
     else if(r==0)
     {
         printf("\nThe deleted element is %d", deque[r]);
         r=size-1;
     }
     else
     {
          printf("\nThe deleted element is %d", deque[r]);
          r=r-1;
     }
}

int main()
{
    insert_front(20);
    insert_front(10);
    insert_rear(30);
    insert_rear(50);
    insert_rear(80);
    display();  // Calling the display function to retrieve the values of deque
    getfront();  // Retrieve the value at front-end
    getrear();  // Retrieve the value at rear-end
    delete_front();
```

```
    delete_rear();
    display(); // calling display function to retrieve values after deletion
    return 0;
}
```

Output:-

```
Elements in a deque are: 10 20 30 50 80
The value of the element at front is: 10
The value of the element at rear is 80
The deleted element is 10
The deleted element is 80
Elements in a deque are: 20 30 50
```

Practical 31 & 32

Code:-

```
#include <stdio.h>

#define Size 5


int deque_arr[Size];

int front = -1;

int rear = -1;


/*Begin of insert_rear*/

void insert_rear()

{

   int added_item;

   if ((front == 0 && rear == Size - 1) || (front == rear + 1))

   {

      printf("Queue Overflow\n");

      return;

   }

   if (front == -1) /* if queue is initially empty */

   {

      front = 0;

      rear = 0;

   }

   else if (rear == Size - 1) /*rear is at last position of queue */

      rear = 0;
```

```
    else
        rear = rear + 1;

    printf("Input the element for adding in queue : ");
    scanf("%d", &added_item);
    deque_arr[rear] = added_item;
}
/*End of insert_rear*/

/*Begin of insert_front*/
void insert_front()
{
    int added_item;
    if ((front == 0 && rear == Size - 1) || (front == rear + 1))
    {
        printf("Queue Overflow \n");
        return;
    }
    if (front == -1) /*If queue is initially empty*/
    {
        front = 0;
        rear = 0;
    }
    else if (front == 0)
        front = Size - 1;
    else
        front = front - 1;
```

```
    printf("Input the element for adding in queue : ");
    scanf("%d", &added_item);
    deque_arr[front] = added_item;
}
/*End of insert_front*/

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

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

/*Begin of input_que*/
void display_queue()
{
   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 ", deque_arr[front_pos]);
        front_pos++;
    }
}
else
{
    while (front_pos <= Size - 1)
    {
        printf("%d ", deque_arr[front_pos]);
        front_pos++;
    }
    front_pos = 0;
    while (front_pos <= rear_pos)
    {
        printf("%d ", deque_arr[front_pos]);
        front_pos++;
    }
}
```

```
    printf("\n");
}
/*End of display_queue*/

/*Begin of input_que*/
void input_que()
{
    int choice;
    do
    {
        printf("1.Insert at rear\n");
        printf("2.Delete from front\n");
        printf("3.Delete from rear\n");
        printf("4.Display\n");
        printf("5.Quit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            insert_rear();
            break;
        case 2:
            delete_front();
            break;
        case 3:
```

```
            delete_rear();
            break;
        case 4:
            display_queue();
            break;
        case 5:
            break;
        default:
            printf("Wrong choice\n");
        }
    } while (choice != 5);
}
/*End of input_que*/

/*Begin of output_que*/
void output_que()
{
    int choice;
    do
    {
        printf("1.Insert at rear\n");
        printf("2.Insert at front\n");
        printf("3.Delete from front\n");
        printf("4.Display\n");
        printf("5.Quit\n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
```

```
        switch (choice)
        {
        case 1:
            insert_rear();
            break;
        case 2:
            insert_front();
            break;
        case 3:
            delete_front();
            break;
        case 4:
            display_queue();
            break;
        case 5:
            break;
        default:
            printf("Wrong choice\n");
        }
    } while (choice != 5);
}
/*End of output_que*/

/*Begin of main*/
main()
{
    int choice;
```

```
    printf("1.Input restricted dequeue\n");
    printf("2.Output restricted dequeue\n");
    printf("Enter your choice : ");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        input_que();
        break;
    case 2:
        output_que();
        break;
    default:
        printf("Wrong choice\n");
    }
}
/*End of main*/
```

Output:-

```
***** MAIN MENU *****
1.Input restricted deque
2.Output restricted deque
Enter your option : 1
INPUT RESTRICTED DEQUEUE
1.Insert at right
2.Delete from left
3.Delete from right
4.Display
5.Quit
Enter your option : 1
Enter the value to be added : 5
Enter the value to be added : 10
Enter your option : 2
The deleted element is : 5
Enter your option : 5
```

Practical 33

Code:-

```
#include<stdio.h>
#include<stdlib.h>
struct node {
    int data ;
    struct node *next ;
    struct node *prev ;

};
struct node *head ;
void insertatFront(int x){
    struct node *newnode = (struct node *)malloc(sizeof (struct node));
    newnode ->data = x;
    newnode->next = NULL;
    newnode->prev = NULL;
    if(head == NULL){
        head = newnode;

    }
    else{
        head ->prev = newnode;
        newnode->next = head;
        head = newnode;
    }
```

```
}
void insertAtEnd(int x){
    struct node *newnode ;
    newnode= (struct node*)malloc(sizeof(struct node));
    newnode->data = x;
    newnode->next = NULL ;
    newnode->prev = NULL;

    if(head == NULL){
        head = newnode;
    }
    else{
        struct node *temp ;
        temp = head ;
        while(temp->next != NULL){
            temp = temp ->next ;

        }
        temp ->next = newnode;
        newnode->prev = temp ;
        newnode->next = NULL;
    }

}

void deletefromfront(){
    struct node *temp = head;
```

```
    if(head == NULL){
        printf("The list is Empty !");
    }
    else{
        temp = head;
        head = head->next ;
        head->prev = NULL ;
        printf("\nThe deleted element is : %d\n",temp->data);
        free(temp);
    }
}
void deletefromend(){
    struct node *temp = head;
    if(head == NULL){
        printf("The list is Empty !");

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

        }
        temp->prev->next = NULL;
        free(temp);
```

```
    }
}

void insertAtposition(int x , int pos){
    int i = 1;
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = x;
    newnode->next = NULL;
    newnode->prev= NULL;
    if(head == NULL){
        head = newnode;
        newnode->prev = NULL ;
        newnode->next = NULL ;
    }
    else if(pos== 1){
        insertatFront(x);
    }
    else{
        struct node *temp = head;
        while(i<pos-1){
            temp = temp->next ;
            i++;
        }
        newnode->next = temp ->next ;
        newnode->prev = temp ;
        temp ->next = newnode;
        temp->next ->prev = newnode;
```

```
    }
    printf("The element get inserted at index %d\n",pos);
}

void deletefrompos(int pos){
    struct node *position ;

    int i = 1;
    if(head == NULL){
        printf("The list is Empty !");
    }
    else if (pos==1){
        deletefromfront();
    }
    else{
        struct node *temp = head;
        while (i < pos-1 ){
            temp = temp->next ;
            i++;
        }
        position= temp->next ;
        if(position->next != NULL){
            position->next ->prev = temp ;
        }
        temp->next =  position ->next ;
        printf("The node at index %d is %d deleted Now !\n",pos,position->data);
        free(position);
```

```
    }

}

void display(){
    struct node *temp ;
    if(head ==NULL){
        printf("The list is Empty ! ");
    }
    else{
        temp = head;
        while(temp!=NULL){
            printf("%d\n",temp ->data );
            temp = temp->next ;
        }
    }
}
int main(){
    insertatFront(10);
    insertatFront(20);
    insertatFront(30);
    insertAtEnd(100);
    insertAtEnd(200);
    insertAtEnd(300);
    insertAtposition(10000,3);
    printf("\nBefore deletion operation ! ---------\n");
```

```
    display();
    printf("After the deletion Operation !-----");
    // deletefromfront();
    // deletefromfront();
    deletefrompos(3);
    // deletefromend();
    // deletefromend();
    display();
    return 0;
}
```

Output:-

```
The element get inserted at index 3

Before deletion operation ! ---------
30
20
10000
10
100
200
300
After the deletion Operation !-----The node at index 3 is 10000 deleted Now !
30
20
10
100
200
300
PS C:\Users\Aman Tripathi\OneDrive\Desktop\DSA file\Stack_&_Queue>
```

Practical 34

Code:-

```
// C program to Demonstrate Priority Queue
#include <stdio.h>
#include <limits.h>
#define MAX 100
int idx = -1;
int pqVal[MAX];
int pqPriority[MAX];
int isEmpty()
{
    return idx == -1;
}
int isFull()
{
    return idx == MAX - 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;
    }
}

int peek()
{
    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;
        }
        else if (maxPriority < pqPriority[i])
        {
            maxPriority = pqPriority[i];
            indexPos = i;
        }
```

```
    }

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

void dequeue()
{
    if (!isEmpty())
    {
        // Get element with highest priority
        int indexPos = peek();
        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()
{
    // 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:-

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

After Dequeue :
(5, 1)
(10, 3)
(500, 2)
```

Practical 35

Code:-

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

struct node
{
   int priority;
   int info;
   struct node *link;
} *front = NULL;

void insert(int item, int item_priority);
int del();
void display();
int isEmpty();

int main()
{
   int choice, item, item_priority;
   while (1)
   {
      printf("\n1.Insert\n");
      printf("2.Delete\n");
      printf("3.Display\n");
      printf("4.Quit\n");
      printf("\nEnter your choice : ");
      scanf("%d", &choice);

      switch (choice)
      {
      case 1:
         printf("\nInput the item to be added in the queue : ");
         scanf("%d", &item);
         printf("\nEnter its priority : ");
         scanf("%d", &item_priority);
         insert(item, item_priority);
         break;
      case 2:
         printf("\nDeleted item is %d\n", del());
         break;
      case 3:
         display();
```

```
            break;
        case 4:
            exit(1);
        default:
            printf("\nWrong choice\n");
        } /*End of switch*/
    }    /*End of while*/

    return 0;
} /*End of main()*/

void insert(int item, int item_priority)
{
    struct node *tmp, *p;

    tmp = (struct node *)malloc(sizeof(struct node));
    if (tmp == NULL)
    {
        printf("\nMemory not available\n");
        return;
    }
    tmp->info = item;
    tmp->priority = item_priority;
    /*Queue is empty or item to be added has priority more than first element*/
    if (isEmpty() || item_priority < front->priority)
    {
        tmp->link = front;
        front = tmp;
    }
    else
    {
        p = front;
        while (p->link != NULL && p->link->priority <= item_priority)
            p = p->link;
        tmp->link = p->link;
        p->link = tmp;
    }
} /*End of insert()*/

int del()
{
    struct node *tmp;
    int item;
    if (isEmpty())
    {
        printf("\nQueue Underflow\n");
        exit(1);
    }
```

```
    else
    {
        tmp = front;
        item = tmp->info;
        front = front->link;
        free(tmp);
    }
    return item;
} /*End of del()*/

int isEmpty()
{
    if (front == NULL)
        return 1;
    else
        return 0;

} /*End of isEmpty()*/

void display()
{
    struct node *ptr;
    ptr = front;
    if (isEmpty())
        printf("\nQueue is empty\n");
    else
    {
        printf("\nQueue is :\n");
        printf("\nPriority       Item\n");
        while (ptr != NULL)
        {
            printf("%5d       %5d\n", ptr->priority, ptr->info);
            ptr = ptr->link;
        }
    }
} /*End of display() */
```

Output:-

```
1.Insert
2.Delete
3.Display
4.Quit

Enter your choice : 1

Input the item to be added in the queue : 32

Enter its priority : 2

1.Insert
2.Delete
3.Display
4.Quit

Enter your choice : 1

Input the item to be added in the queue : 23

Enter its priority : 1

1.Insert
2.Delete
3.Display
4.Quit

Enter your choice : 1

Input the item to be added in the queue : 23

Enter its priority : 1

1.Insert
2.Delete
3.Display
4.Quit
```

```
1.Insert
2.Delete
3.Display
4.Quit

Enter your choice : 3

Queue is :

Priority       Item
    1           23
    1           23
    2           32

1.Insert
2.Delete
3.Display
4.Quit

Enter your choice : 2

Deleted item is 23

1.Insert
2.Delete
3.Display
4.Quit

Enter your choice : 4
PS C:\Users\Aman Tripathi\O
```

Practical 36

```
#include <stdio.h>

// variable to store maximum number of nodes
int complete_node = 15;


// array to store the tree
char tree[] = {'\0', 'D', 'A', 'F', 'E', 'B', 'R', 'T', 'G', 'Q', '\0', '\0', 'V', '\0', 'J', 'L'};


int get_right_child(int index)
{
    // node is not null
    // and the result must lie within the number of nodes for a complete binary tree
    if (tree[index] != '\0' && ((2 * index) + 1) <= complete_node)
        return (2 * index) + 1;
    // right child doesn't exist
    return -1;
}


int get_left_child(int index)
{
    // node is not null
    // and the result must lie within the number of nodes for a complete binary tree
    if (tree[index] != '\0' && (2 * index) <= complete_node)
        return 2 * index;
    // left child doesn't exist
    return -1;
```

```
}

void preorder(int index)
{
   // checking for valid index and null node
   if (index > 0 && tree[index] != '\0')
   {
      printf(" %c ", tree[index]);      // visiting root
      preorder(get_left_child(index));  // visiting left subtree
      preorder(get_right_child(index)); // visiting right subtree
   }
}

void postorder(int index)
{
   // checking for valid index and null node
   if (index > 0 && tree[index] != '\0')
   {
      postorder(get_left_child(index));  // visiting left subtree
      postorder(get_right_child(index)); // visiting right subtree
      printf(" %c ", tree[index]);       // visiting root
   }
}

void inorder(int index)
{
   // checking for valid index and null node
```

```
    if (index > 0 && tree[index] != '\0')
    {
        inorder(get_left_child(index));  // visiting left subtree
        printf(" %c ", tree[index]);     // visiting root
        inorder(get_right_child(index)); // visiting right subtree
    }
}

int main()
{
    printf("Preorder:\n");
    preorder(1);
    printf("\nPostorder:\n");
    postorder(1);
    printf("\nInorder:\n");
    inorder(1);
    printf("\n");
    return 0;
}
```

Output:-

```
Preorder:
 D  A  E  G  Q  B  F  R  V  T  J  L
Postorder:
 G  Q  E  B  A  V  R  J  L  T  F  D
Inorder:
 G  E  Q  A  B  D  V  R  F  J  T  L
```