Regulation: AK20	Subject Code: 20AES0504	Subject Name : Data Structures Lab	AY: 2020-2021
Data Structures Laboratory Manual			

Year : I B.Tech	Semester : II	Branch of Study : CSE/CIC/AIDS/ECE		
Code:20AES0504	Subject Name: **DATA STRUCTURES LAB**	L T P 0 0 3	Credits:1.5	

Course Objectives:

1. To introduce to the different data structures
2. To elucidate how the data structure selection influences the algorithm complexity
3. To explain the different operations that can be performed on different data structures
4. To introduce to the different search and sorting algorithms.

Laboratory Experiments

1. String operations using array of pointers
2. Searching Algorithms (With the Number of Key Comparisons) Sequential, Binary and Fibonacci Search Algorithms.
3. Sorting Algorithms: Insertion Sort, Selection Sort, Shell Sort, Bubble Sort, Quick Sort, Heap Sort, Merge Sort, and Radix Sort. Using the system clock, compute the time taken for sorting of elements. The time for other operations like I/O etc should not be considered while computing time.
4. Implementation of Singly Linked List, Doubly Linked List, Circular Linked List
5. Stack implementation using arrays
6. Stack implementation using linked lists
7. Queue implementation using arrays. Implement different forms of queue. While implementing you should be able to store elements equal to the size of the queue. No positions should be left blank.
8. Queue implementation using linked lists
9. Creation of binary search tree, performing operations insertion, deletion, and traversal.
10. Breadth first search
11. Depth first search
12. Travelling sales man problem
13. File operations
14. Indexing of a file
15. Reversing the links (not just displaying) of a linked list.
16. Consider a linked list consisting of name of a person and gender as a node. Arrange the linked list using _Ladies first‘ principle. You may create new linked lists if necessary.
17. An expression can be represented in three ways: infix, prefix and postfix. All the forms are necessary

in different contexts. Write modules to convert from one form to another form.

18. A table can be defined as a collection of rows and columns. Each row and column may have a label. Different values are stored in the cells of the table. The values can be of different data types. Numerical operations like summation, average etc can be performed on rows/columns which contain numerical data. Such operations are to be prevented on data which is not numeric. User may like to insert row/columns in the already existing table. User may like to remove row/column. Create table data type and support different operations on it.

Course Outcomes:

1. Select the data structure appropriate for solving the problem
2. Implement searching and sorting algorithms
3. Design new data types
4. Illustrate the working of stack and queue
5. Organize the data in the form of files

List of COs	PO no. and keyword	Competency Indicator	Performance Indicator
CO1	PO1: Engineering Knowledge	1.4	1.4.1
CO2	PO 2: Problem analysis	2.2	2.2.4
CO3	PO1: Engineering Knowledge	1.3	1.3.1
CO4	PO1: Engineering Knowledge	1.4	1.4.1
CO5	PO1: Engineering Knowledge	1.4	1.4.1

Regulation: AK20	Subject Code: 20AES0504	Subject Name : Data Structures Lab	AY: 2020-2021
Data Structures Laboratory Manual			

1. AIM: Write a C Program to implement String operations using array of pointers

```
//array of strings
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
int main() {
 // declaring the string pointer array as well as a char array
 char *names[50], a[50];
 int length, i,n;
 char *x;

        printf("\nArray of pointers to strings program:");
        printf("\nHow many no of stings do you want:");
        scanf("%d",&n);
        printf("\nEnter %d strings:\n",n);

 // taking inputs in char array as well
 // as copying them to the string pointer array
 for (i = 0; i < n; i++)
 {
        fflush(stdin);
  gets(a); // taking values in char array

  length = strlen(a);
        // used malloc to allocate dynamic memory. l+1 to store "\0".
  x = (char *)malloc(length + 1);
  if(x==NULL)
  {
        printf("\nmemory insufficient:");
                return;
        }
  strcpy(x, a);
 names[i]= x;
 }

 printf("\nThe strings are:");

 for (i = 0; i < n; i++)
  printf("\n%s",names[i]);

 return 0;
}
/*
```

```
OUTPUT:
Array of pointers to strings program:
How many no of stings do you want:5

Enter 5 strings:
R RUPADEVI SINGH
T SREENIVASULA REDDY
M KIRAN MONI
T SAI KUMAR
G RAMAKRISHNA

The strings are:
R RUPADEVI SINGH
T SREENIVASULA REDDY
M KIRAN MONI
T SAI KUMAR
G RAMAKRISHNA
*/
```

2. AIM: Write a C Program to implement Array Operations

```
//array operations
#include<stdio.h>
#define MAX 100
int array[MAX], pos, i, n, value,ch;
int main()
{
        void create_array();
        void insert_beg();
  void insert_end();
  void insert_pos();
  void display();
  void delete_beg();
  void delete_end();
  void delete_pos();

  while(1)
  {
    printf("\n---- Array Operations Menu ----");
    printf("\n1. Create Array\n2.Insert\n3.Display\n4.Delete\n5.Exit\n\n");
    printf("Enter your choice(1-5):");
    scanf("%d",&ch);

    switch(ch)
    {
      case 1:create_array();
             break;
```

```
                        case 2:
            if(n<=MAX)
            {

                                                printf("\n---- Insert Menu ----");
                    printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specified
position\n4.Exit");
                    printf("\n\nEnter your choice(1-4):");
                                                scanf("%d",&ch);

                    switch(ch)
                    {
                      case 1: insert_beg();
                            break;
                      case 2: insert_end();
                            break;
                      case 3: insert_pos();
                            break;

                      case 4: exit(0);
                      default: printf("\nWrong Choice!!");
                    }
            }
            else exit(0);
                  break;

      case 3: display();
            break;

      case 4: printf("\n---- Delete Menu ----");
            printf("\n1.Delete from beginning\n2.Delete from end\n3.Delete from specified position");
            printf("\n\nEnter your choice(1-4:)");
            scanf("%d",&ch);

            switch(ch)
            {
              case 1: delete_beg();
                    break;
              case 2: delete_end();
                    break;
              case 3: delete_pos();
                    break;

              case 4: exit(0);
              default: printf("\nWrong Choice!!");
```

```
            }
            break;
        case 5: exit(0);
            default: printf("\nWrong Choice!!");
    }
  }
return 0;
}
void display()
{
        printf("\nResultant array is:\n ");
    for(i = 0; i < n; i++)
    printf("%d  ", array[i]);
    return;
}
 void create_array()
 {

  printf("\nEnter Size of the Array:");
  scanf("%d",&n);

  printf("\nEnter %d Elements\n", n);
  for(i = 0; i < n; i++)
    scanf("%d", &array[i]);

        printf("\nInitial Array Elements:\n", n);
  for(i = 0; i < n; i++)
    printf("%5d",array[i]);
}
void insert_beg()
{
        printf("\nEnter the value to insert:");
  scanf("%d", &value);

  // shifting the elements from (position to n) to right
  for(i = n-1; i >= 0; i--)
    array[i+1] = array[i];

  array[0] = value;    // inserting the given value
        n=n+1;
}

void insert_end()
{
        printf("\nEnter the value to insert:");
```

```
   scanf("%d", &value);
   array[n]=value;
   n=n+1;
}
void insert_pos()
{
        printf("\nEnter the location where you want to insert new element:");
   scanf("%d", &pos);

   printf("\nEnter the value to insert:");
   scanf("%d", &value);
        if(pos<0 || pos>n)
        {
                printf("\nInvalid Position Retry....");
                return;
        }
   // shifting the elements from (position to n) to right
   for(i = n-1; i >= pos-1; i--)
      array[i+1] = array[i];

   array[pos - 1] = value;    // inserting the given value
        n=n+1;
        return;
}
void delete_beg()
{
        // shifting the elements from (position n to 0) to left
   for(i = 0; i < n; i++)
      array[i] = array[i+1];
   n=n-1;
   return;
}

void delete_end()
{
        n=n-1;
        return;
}
void delete_pos()
{
        printf("\nEnter the location where you want to Delete element:");
   scanf("%d", &pos);
        if(pos<0 || pos>n)
        {
                printf("\nInvalid Position Retry....");
```

```
                return;
            }
    // shifting the elements from (position to n) to right
    n=n-1;
    printf("\nPos=%d\t n=%d",pos,n);
            for(i = pos-1; i < n; i++)
        array[i] = array[i+1];

            return;
}
/*
OUTPUT:
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):1

Enter Size of the Array:5

Enter 5 Elements
15 87 585 -55 -2

Initial Array Elements:
  15   87  585  -55   -2
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):2

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):1
```

```
Enter the value to insert:1000

---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):3

Resultant array is:
 1000  15  87  585  -55  -2
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):2

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):2

Enter the value to insert:5000

---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):3

Resultant array is:
 1000  15  87  585  -55  -2  5000
---- Array Operations Menu ----
1. Create Array
```

```
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):2

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):3

Enter the location where you want to insert new element:5

Enter the value to insert:777

---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):3

Resultant array is:
 1000  15  87  585  777  -55  -2  5000
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):4

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)1
```

```
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):3

Resultant array is:
 15  87  585  777  -55  -2  5000
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):4

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)2

---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):3

Resultant array is:
 15  87  585  777  -55  -2
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit
```

```
Enter your choice(1-5):4

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)3

Enter the location where you want to Delete element:4

---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):3

Resultant array is:
 15  87  585  -55  -2
---- Array Operations Menu ----
1. Create Array
2.Insert
3.Display
4.Delete
5.Exit

Enter your choice(1-5):2
Array Reached the Maximum Size.....Program terminated
*/
```

3. AIM: Searching Algorithms (With the Number of Key Comparisons) Sequential, Binary and Fibonacci Search Algorithms

//linear search

```
#include<stdio.h>
int linear_search(int *, int , int );
int count=0;
int linear_search(int *arr, int n, int key)
{
int i;
                for(i = 0; i < n; i++)
                {
                count++;
```

```
                    if (arr[i] == key)
                    return i;
                    }
return -1;
}

int main(void)
{
int i, pos;
int n, key;

          printf("\n Linear Search Program:");
         printf("\n*****************************");
          printf("\nEnter size of the array:");
          scanf("%d",&n);
   int * arr = ( int*)(malloc(sizeof(int) * n));
          printf("\nEnter %d Elements into the array:\n",n);
   for (i = 0; i < n; i++)
      scanf("%d",&arr[i]);
          printf("\nOriginal List is:\n");
          for (i = 0; i < n; i++)
      printf("%5d",arr[i]);
   printf("\nEnter key value to be search:");
   scanf("%d",&key);
pos = linear_search(arr, n, key);
if(pos==-1)
    {
        printf("\nElement is not found in the array !\n");
    }
    else
    {
        printf("\nElement %d is present at position %d",key, pos+1);
    }
printf("\nThe no. of key comparisions is %d\n",count);

return 0;

}
```

OUTPUT:

```
Linear Search Program:
*****************************
Enter size of the array:8

Enter 8 Elements into the array:
10 15 879 -55 858 -77 999 0
```

```
Original List is:
  10  15  879  -55  858  -77  999   0
Enter key value to be search:0

Element 0 is present at position 8
The no. of key comparisions is 8
```

```
//binary search
#include<stdio.h>

int count=0;
int binary_search(int *arr, int l, int h,int key)
{
while (l<=h) {
int mid = (l+h) / 2;
count++;

if (arr[mid] == key)
return mid;

else if(arr[mid] > key)
h=mid-1;

else l=mid+1;
}
return -1;
 }


int main(void)
{
int i, pos;
int n, key;

          printf("\n Binary Search Program:");
          printf("\n*******************************");
          printf("\nEnter size of the array:");
          scanf("%d",&n);
   int * arr = ( int*)(malloc(sizeof(int) * n));
          printf("\nEnter %d Elements into the array ( Enter in Sorted Order Only....):\n",n);
   for (i = 0; i < n; i++)
      scanf("%d",&arr[i]);
          printf("\nOriginal List is:\n");
          for (i = 0; i < n; i++)
```

```
      printf("%5d",arr[i]);
   printf("\nEnter key value to be search:");
   scanf("%d",&key);

pos = binary_search(arr,0,n-1, key);
if(pos==-1)
    {
        printf("\nElement is not found in the array !\n");
    }
    else
    {
        printf("\nElement %d is present at position %d",key, pos+1);
    }
printf("\nThe no. of key comparisions is %d\n",count);

return 0;

}
```

OUTPUT:

```
Binary Search Program:
******************************
Enter size of the array:8

Enter 8 Elements into the array ( Enter in Sorted Order Only....):
-777 -5 0 5 777 888 989 1020

Original List is:
 -777   -5    0    5  777  888  989 1020
Enter key value to be search:1020

Element 1020 is present at position 8
The no. of key comparisions is 4
```

// C program for Fibonacci Search

```
#include <stdio.h>
int count=0;
// Utility function to find minimum of two elements
int min(int x, int y) { return (x <= y) ? x : y; }

/* Returns index of x if present, else returns -1 */
int fibonacci_Search(int arr[], int x, int n)
{ count++;
          /* Initialize fibonacci numbers */
          int fibmm2 = 0; // (m-2)'th Fibonacci No.
          int fibmm1 = 1; // (m-1)'th Fibonacci No.
```

```
int fibM = fibmm2 + fibmm1; // m'th Fibonacci

/* fibM is going to store the smallest Fibonacci
Number greater than or equal to n */
while (fibM < n) {
        fibmm2 = fibmm1;
        fibmm1 = fibM;
        fibM = fibmm2 + fibmm1;
}

// Marks the eliminated range from front
int offset = -1;

/* while there are elements to be inspected. Note that
we compare arr[fibMm2] with x. When fibM becomes 1,
fibMm2 becomes 0 */
while (fibM > 1) {
        // Check if fibMm2 is a valid location
        int i = min(offset + fibmm2, n - 1);

        /* If x is greater than the value at index fibMm2,
        cut the subarray array from offset to i */
        if (arr[i] < x) {
                fibM = fibmm1;
                fibmm1 = fibmm2;
                fibmm2 = fibM - fibmm1;
                offset = i;
        }

        /* If x is greater than the value at index fibMm2,
        cut the subarray after i+1 */
        else if (arr[i] > x) {
                fibM = fibmm2;
                fibmm1 = fibmm1 - fibmm2;
                fibmm2 = fibM - fibmm1;
        }

        /* element found. return index */
        else
                return i;
}

/* comparing the last element with x */
if (fibmm1 && arr[offset + 1] == x)
        return offset + 1;
```

```
        /*element not found. return -1 */
        return -1;
}

/* driver function */
int main(void)
{
        int i, pos;
   int n, key;

        printf("\nFibonacci Search Program:");
        printf("\n*******************************");
        printf("\nEnter size of the array:");
        scanf("%d",&n);
   int * arr = ( int*)(malloc(sizeof(int) * n));
        printf("\nEnter %d Elements into the array:\n",n);
   for (i = 0; i < n; i++)
      scanf("%d",&arr[i]);

        printf("\nOriginal List is:\n");
        for(i=0;i<n;i++)
        printf("%5d",arr[i]);
        printf("\nEnter the key value to be searched:");
        scanf("%d",&key);
        pos = fibonacci_Search(arr, key, n);
if(pos>=0)
        printf("\nElement %d is present at position %d",key, pos+1);
else
        printf("\n %d isn't present in the array",key);
printf("\nThe no. of key comparisions: %d\n",count);
        return 0;
}
OUTPUT:
Binary Search Program:
*******************************
Fibonacci Search Program:
*******************************
Enter size of the array:15

Enter 15 Elements into the array:
1 3 5 7 9 51 64 53 -88 -99 -745 519 537 841 9863

Original List is:
   1   3   5   7   9  51  64  53 -88 -99 -745  519  537  841 9863
```

```
Enter the key value to be searched:9863

Element 9863 is present at position 15
The no. of key comparisions: 1
```

4. AIM: Write a C Program to implement Sorting Algorithms: Insertion Sort, Selection Sort, Shell Sort, Bubble Sort, Quick Sort, Heap Sort, Merge Sort, and Radix Sort. Using the system clock, compute the time taken for sorting of elements. The time for other operations like I/O etc should not be considered while computing time.

//bubble sort

```
#include <stdio.h>
#include <time.h>
#include <limits.h>
void bubble_sort(int *, int );
void swap(int *, int *);

void bubble_sort(int *arr, int size)
{
   int i, j;
   for (i = 0;  i < size; i++)
   {
      for (j = 0; j < size - 1; j++)
      {
         if (arr[j] > arr[j+1])
            swap(&arr[j], &arr[j+1]);

      }
   }
}
void swap(int *a, int *b)
{
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
int main()
{
   int i,n;
   clock_t tstart,tend;
   printf("\n Bubble Sort OR Sinking Sort OR Exchange Sort");
   printf("\n****************************************");
          printf("\nEnter size of the array:");
   scanf("%d", &n);
```

```
   int *arr=(int *)malloc(sizeof(int)*n);
   if(arr==NULL)
   {
          printf("\n Insufficient Memory");
          exit(0);
          }
   printf("\nEnter %d numbers : ",n);
   for (i = 0; i < n; i++)
      scanf("%d", &arr[i]);
   printf("\nOriginal array is:\n ");
   for (i = 0; i < n; i++)
      printf(" %5d ", arr[i]);
          tstart=clock();
   bubble_sort(arr, n);
   tend=clock();

   printf("\nSorted array in Ascending Order:\n ");

   for (i = 0; i < n; i++)
      printf(" %5d ", arr[i]);
   printf("\nTime taken in iterative Bubble Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);
          return 0;

}
```

OUTPUT:

```
Bubble Sort OR Sinking Sort OR Exchange Sort
****************************************
Enter size of the array:10

Enter 10 numbers : 151 254 689 0 1 -888 -794 5841 5 4

Original array is:
   151   254   689     0     1  -888  -794  5841     5     4
Sorted array in Ascending Order:
  -888  -794     0     1     4     5   151   254   689  5841
Time taken in iterative Bubble Sort: 0.000000s
```

/* C Program to sort an array in ascending order using Insertion Sort */

```
#include <stdio.h>
#include <time.h>
#include <limits.h>
void InsertionSort(int [], int);
void swap(int *, int *);

//Selection sort function
```

```
void InsertionSort(int arr[], int n)
{
int i,j,temp;
for (i = 1 ; i <= n - 1; i++)
  {
            j = i;
       while ( j > 0 && arr[j-1] > arr[j])
       {
         /* temp    = arr[j];
          arr[j]   = arr[j-1];
          arr[j-1] = temp; */
          swap(&arr[j],&arr[j-1]);
          j--;
       }
  }
}
/* Function to swap two variables */
void swap(int *a, int *b)
{
  int temp;
  temp = *a;
  *a = *b;
  *b = temp;
}

int main()
{
  int n, i, j, temp;
  int arr[100];
        clock_t tstart,tend;
  printf("\nInsertion Sort");
  printf("\n*****************************************");
  printf("\nEnter size of the array:");
  scanf("%d", &n);

        printf("\nEnter %d integers in to the array:\n", n);
  for (i = 0; i < n; i++)
  scanf("%d", &arr[i]);

  printf("\nOriginal array is:\n ");
  for (i = 0; i < n; i++)
    printf("%5d", arr[i]);

        tstart=clock();
  InsertionSort(arr, n);
```

```
    tend=clock();



    printf("\nSorted list in ascending order:\n");
    for (i = 0; i <= n - 1; i++)
    printf("%5d", arr[i]);
    printf("\nTime taken in iterative Insertion Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);
    return 0;
}
OUTPUT:
Insertion Sort
*****************************************
Enter size of the array:10

Enter 10 integers in to the array:
15 52 -201 -101 512 789 4 0 3 65

Original array is:
   15   52 -201 -101  512  789    4    0    3   65
Sorted list in ascending order:
 -201 -101    0    3    4   15   52   65  512  789
Time taken in iterative Insertion Sort: 0.000000s
```

// C Program to Implement Selection Sort

```
#include <stdio.h>
#include <time.h>
#include <limits.h>
void selectionSort(int arr[], int size);
void swap(int *a, int *b);
/*
 * Selection sort function
 */
void selectionSort(int arr[], int size)
{
    int i, j;
    for (i = 0 ;  i < size;i++)
    {
        for (j = i ; j < size; j++)
        {
            if (arr[i] > arr[j])
                swap(&arr[i], &arr[j]);
        }
    }
}
```

```
/* Function to swap two variables */
void swap(int *a, int *b)
{
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
/*
 * Main Function
 */
int main()
{
   int array[10], i, size;
   clock_t tstart,tend;
   printf("\nSelection Sort Program");
   printf("\n****************************************");
         printf("\nHow many numbers you want to sort:");
   scanf("%d", &size);
   printf("\nEnter %d numbers:\n", size);
   for (i = 0; i < size; i++)
      scanf("%d", &array[i]);

   printf("\nOriginal array is:\n ");
   for (i = 0; i < size; i++)
      printf("%5d", array[i]);

         tstart=clock();
   selectionSort(array, size);
   tend=clock();
   printf("\nSorted array in Ascending Order:\n ");

   for (i = 0; i < size;i++)
      printf("%5d", array[i]);
   printf("\nTime taken in iterative Selection Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);
   return 0;
}
OUTPUT:
Selection Sort Program
****************************************
How many numbers you want to sort:10

Enter 10 numbers:
12 -89 65 123 847 586 951 753 -878 0
```

```
Original array is:
   12  -89   65  123  847  586  951  753 -878    0
Sorted array in Ascending Order:
 -878  -89    0   12   65  123  586  753  847  951
Time taken in iterative Selection Sort: 0.000000s
```

//quick sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define MAX 100

void swap(int *, int *);
int partion(int [], int , int );
void quick_sort(int [], int , int );


void swap(int *a, int *b)
{
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
}
int partion(int arr[], int p, int r)
{
   int pivot = arr[r];
   int i = p - 1;
   int j;
   for (j = p; j < r; j++)
   {
      if (arr[j] < pivot)
      {
         i++;
         swap(&arr[i], &arr[j]);
      }

   }
   swap(&arr[i+1], &arr[r]);
   return i + 1;
}

void quick_sort(int arr[], int p, int q)
{
```

```
    int j;
    if (p < q)
    {
        j = partion(arr, p, q);
        quick_sort(arr, p, j - 1);
        quick_sort(arr, j + 1, q);
    }
}

int main()
{
    int i,j,arr[MAX],size;
    clock_t tstart,tend;


    printf("\nQuick Sort OR Partition Exchange Sort");
    printf("\n****************************************");

            printf("\nEnter size of the array:");
    scanf("%d", &size);
            printf("\nEnter %d integers in to the array:\n", size);
    for(i = 0; i < size; i++)
      scanf("%d", &arr[i]);

            printf("\nOriginal array is:\n ");
    for (i = 0,j=1; i < size; i++,j++)
    printf("\n%d", arr[i]);

            tstart=(1000.0*clock());
            printf("\nQuick Sort Algorithm Start Time:%lf",tstart);
            quick_sort(arr, 0, size-1); //function to sort the elements of array
    tend=(1000.0*clock());
            printf("\nQuick Sort Algorithm End Time:%lf",tend);

            printf("\nSorted list in ascending order:\n");
            for (i = 0,j=1; i < size; i++,j++)
    printf("\n%d", arr[i]);

    printf("\nTime taken in Recursive Quick Sort: %lfs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);
            return 0;
}
```

OUTPUT:

```
Quick Sort OR Partition Exchange Sort
****************************************
Enter size of the array:10
```

```
Enter 10 integers in to the array:
10 54 56 -89 -501 -55 0 1 456 123

Original array is:

10
54
56
-89
-501
-55
0
1
456
123
Quick Sort Algorithm Start Time:-0.000000
Quick Sort Algorithm End Time:0.000000
Sorted list in ascending order:

-501
-89
-55
0
1
10
54
56
123
456
Time taken in Recursive Quick Sort: 0.000000s
```

//C Program to Input Few Numbers & Perform Merge Sort on them using Recursion

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define MAX 100

void mergeSort(int [], int, int, int);
void partition(int [],int, int);
void random_shuffle(int []);

int main()
{
```

```
    int list[MAX];
    int i, size;
    clock_t tstart,tend;


            printf("\nMerge Sort Program");
            printf("\n*****************************************");
    printf("\nEnter size of the array:");
   scanf("%d", &size);

            printf("\nEnter %d integers in to the array:\n", size);
  for(i = 0; i < size; i++)
 scanf("%d", &list[i]);
 //   size=MAX;
    printf("\nOriginal array is:\n ");
    for(i = 0; i < size; i++)
    printf("\n%d  ",list[i]);

            tstart=(1000.0*clock());
            partition(list, 0, size - 1);
    tend=(1000.0*clock());

            printf("\nSorted list in ascending order:\n");
    for(i = 0;i < size; i++)
    printf("\n%d  ",list[i]);
    printf("\nTime taken in Merge Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);

   return 0;
}

void partition(int list[],int low,int high)
{
    int mid;

    if(low < high)
    {
       mid = (low + high) / 2;
       partition(list, low, mid);
       partition(list, mid + 1, high);
       mergeSort(list, low, mid, high);
    }
}

void mergeSort(int list[],int low,int mid,int high)
{
```

```
    int i, mi, k, lo, temp[MAX];

    lo = low;
    i = low;
    mi = mid + 1;
    while ((lo <= mid) && (mi <= high))
    {
        if (list[lo] <= list[mi])
        {
            temp[i] = list[lo];
            lo++;
        }
        else
        {
            temp[i] = list[mi];
            mi++;
        }
        i++;
    }
    if (lo > mid)
    {
        for (k = mi; k <= high; k++)
        {
            temp[i] = list[k];
            i++;
        }
    }
    else
    {
        for (k = lo; k <= mid; k++)
        {
            temp[i] = list[k];
            i++;
        }
    }

    for (k = low; k <= high; k++)
    list[k] = temp[k];

}
```

OUTPUT:

```
Merge Sort Program
****************************************
Enter size of the array:10
```

```
Enter 10 integers in to the array:
10 584 756 321 -55 -77 485 0 15 94

Original array is:

10
584
756
321
-55
-77
485
0
15
94
Sorted list in ascending order:

-77
-55
0
10
15
94
321
485
584
756
Time taken in Merge Sort: 0.000000s
```

//C Program to sort an array using Shell Sort technique

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define MAX 100
void shellsort(int[], int);


void shellsort(int arr[], int num)
{
   int i, j, k, tmp;
   for (i = num / 2; i > 0; i = i / 2)
   {
      for (j = i; j < num; j++)
      {
```

```
        for(k = j - i; k >= 0; k = k - i)
        {
           if (arr[k+i] >= arr[k])
              break;
           else
           {
              tmp = arr[k];
              arr[k] = arr[k+i];
              arr[k+i] = tmp;
           }
        }
     }
  }
}
int main()
{
  int arr[MAX];
  int i,  n;
  clock_t tstart,tend;
  printf("\nShell Sort Program");
          printf("\n*****************************************");
          printf("\nEnter size of the array:");
  scanf("%d", &n);

          printf("\nEnter %d integers in to the array:\n", n);
  for (i = 0; i < n; i++)
  scanf("%d", &arr[i]);

  printf("\nOriginal array is:\n ");
  for (i = 0; i < n; i++)
     printf("%5d", arr[i]);

          tstart=clock();
          shellsort(arr, n);
          tend=clock();

  printf("\nSorted list in ascending order:\n");
  for (i = 0; i <n; i++)
  printf("%5d", arr[i]);
  printf("\nTime taken in Shell Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);
  return 0;
}
OUTPUT:
Shell Sort Program
*****************************************
```

```
Enter size of the array:10

Enter 10 integers in to the array:
0 5 -5 -10 545 685 -789 514 562 101

Original array is:
    0   5   -5  -10  545  685 -789  514  562  101
Sorted list in ascending order:
 -789  -10   -5    0    5  101  514  545  562  685
Time taken in Shell Sort: 0.000000s
```

//counting sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define MAX 100
int B[MAX], C[MAX];
void countingsort(int [], int , int );


void countingsort(int arr[], int k, int n)
{
   int i, j;
   for (i = 0; i <= k; i++)
         C[i] = 0;
   for (j =1; j <= n; j++)
         C[arr[j]] = C[arr[j]] + 1;
   for (i = 1; i <= k; i++)
         C[i] = C[i] + C[i-1];
   for (j = n; j >= 1; j--)
   {
         B[C[arr[j]]] = arr[j];
         C[arr[j]] = C[arr[j]] - 1;
   }
 }

int main()
{
   int n,i,k = 0, arr[MAX];
   clock_t tstart,tend;
   printf("\nCounting Sort Program");
           printf("\n*****************************************");
   printf("\nEnter size of the array:");
   scanf("%d", &n);
```

```
   printf("\nEnter %d integers in to the array:\n", n);
   for ( i = 1; i <= n; i++)
           scanf("%d", &arr[i]);

           printf("\nOriginal array is:\n ");
           for ( i = 1; i <= n; i++)
           printf("%5d  ", arr[i]);

           for ( i = 1; i <= n; i++)
   {
      if (arr[i] > k)
       k = arr[i];
   }

           tstart=clock();
           countingsort(arr, k, n);
   tend=clock();

           printf("\nSorted list in ascending order:\n");
   for(i = 1;i <=n; i++)
   printf("%5d",B[i]);
   printf("\nTime taken in Counting Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);
           return 0;

}
```

OUTPUT:

```
Counting Sort Program
****************************************
Enter size of the array:8

Enter 8 integers in to the array:
55 66 44 11 22 33 88 99

Original array is:
   55    66    44    11    22    33    88    99
Sorted list in ascending order:
  11   22   33   44   55   66   88   99
Time taken in Counting Sort: 0.000000s
```

//Radix sort

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define MAX 100
```

```
int getMax(int [], int);
void countSort(int [], int , int );
void radixsort(int [], int );

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

void countSort(int arr[], int n, int exp) {
   int output[n]; // output array
   int i, count[10] = { 0 };

   // Store count of occurrences in count[]
   for (i = 0; i < n; i++)
      count[(arr[i] / exp) % 10]++;

   for (i = 1; i < 10; i++)
      count[i] += count[i - 1];

   // Build the output array
   for (i = n - 1; i >= 0; i--) {
      output[count[(arr[i] / exp) % 10] - 1] = arr[i];
      count[(arr[i] / exp) % 10]--;
   }

   for (i = 0; i < n; i++)
      arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using Radix Sort
void radixsort(int arr[], int n) {
   int m = getMax(arr, n);

   int exp;
   for (exp = 1; m / exp > 0; exp *= 10)
      countSort(arr, n, exp);
}

int main() {
```

```
   int arr[MAX],n,i;
          clock_t tstart,tend;
   printf("\nRadix Sort Program");
          printf("\n*****************************************");
          printf("\nEnter size of the array:");
   scanf("%d", &n);

          printf("\nEnter %d integers in to the array:\n", n);
   for (i = 0; i < n; i++)
   scanf("%d", &arr[i]);

   printf("\nOriginal array is:\n ");
   for (i = 0; i < n; i++)
      printf("%5d", arr[i]);

          tstart=clock();
          radixsort(arr, n);
   tend=clock();

   printf("\nSorted list in ascending order:\n");
   for (i = 0; i <n; i++)
   printf("%5d", arr[i]);
   printf("\nTime taken in Radix Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);

   return 0;
}
OUTPUT:
Radix Sort Program
*****************************************
Enter size of the array:10

Enter 10 integers in to the array:
11 210 710 855 755 322 456 586 960 120

Original array is:
   11  210  710  855  755  322  456  586  960  120
Sorted list in ascending order:
  11  120  210  322  456  586  710  755  855  960
Time taken in Radix Sort: 0.000000s
```

//C Program to sort an array using Shell Sort technique

```
 #include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define MAX 100
```

```
void shellsort(int[], int);

void shellsort(int arr[], int num)
{
   int i, j, k, tmp;
   for (i = num / 2; i > 0; i = i / 2)
   {
      for (j = i; j < num; j++)
      {
         for(k = j - i; k >= 0; k = k - i)
         {
            if (arr[k+i] >= arr[k])
               break;
            else
            {
               tmp = arr[k];
               arr[k] = arr[k+i];
               arr[k+i] = tmp;
            }
         }
      }
   }
}
int main()
{
   int arr[MAX];
   int i,  n;
   clock_t tstart,tend;
   printf("\nShell Sort Program");
            printf("\n*****************************************");
            printf("\nEnter size of the array:");
   scanf("%d", &n);

            printf("\nEnter %d integers in to the array:\n", n);
   for (i = 0; i < n; i++)
   scanf("%d", &arr[i]);

   printf("\nOriginal array is:\n ");
   for (i = 0; i < n; i++)
      printf("%5d", arr[i]);

            tstart=clock();
            shellsort(arr, n);
            tend=clock();
```

```
   printf("\nSorted list in ascending order:\n");
   for (i = 0; i <n; i++)
   printf("%5d", arr[i]);
   printf("\nTime taken in Shell Sort: %.6fs\n", ((double)(tend - tstart) )/ CLOCKS_PER_SEC);
   getch();
          return 0;
}
```

OUTPUT:

```
Shell Sort Program
****************************************
Enter size of the array:6

Enter 6 integers in to the array:
33 66 55 66 33 12

Original array is:
  33  66  55  66  33  12
Sorted list in ascending order:
 12  33  33  55  66  66
Time taken in Shell Sort: 0.000000s
```

5. AIM: Write a C Program to Implement Singly Linked List, Doubly Linked List, Circular Linked List

//Singly Linked List(SLL)

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

struct node
{
   int data;
   struct node *next;
}*head=NULL,*newnode,*temp,*prev;

int main()
{
   int ch;
   void insert_beg();
   void insert_end();
   int insert_pos();
   void display();
   void delete_beg();
   void delete_end();
   int delete_pos();
   struct node * create_node();
```

```
while(1)
{
   printf("\n---- Singly Linked List(SLL) Menu ----");
   printf("\n1.Insert\n2.Traverse OR Display\n3.Delete\n4.Exit\n");
   printf("\nEnter your choice(1-4):");
   scanf("%d",&ch);

   switch(ch)
   {
      case 1:
            printf("\n---- Insert Menu ----");
            printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specified position\n4.Exit");
            printf("\n\nEnter your choice(1-4):");
                                          scanf("%d",&ch);

            switch(ch)
            {
               case 1: insert_beg();
                     break;
               case 2: insert_end();
                     break;
               case 3: insert_pos();
                     break;

               case 4: exit(0);
               default: printf("\nWrong Choice!!");
            }
            break;

      case 2: display();
            break;

      case 3: printf("\n---- Delete Menu ----");
            printf("\n1.Delete from beginning\n2.Delete from end\n3.Delete from specified position");
            printf("\n\nEnter your choice(1-4:)");
            scanf("%d",&ch);

            switch(ch)
            {
               case 1: delete_beg();
                     break;
               case 2: delete_end();
                     break;
               case 3: delete_pos();
```

```
                    break;

                case 4: exit(0);
                default: printf("\nWrong Choice!!");
            }
            break;
        case 4: exit(0);
            default: printf("\nWrong Choice!!");
    }
  }
  return 0;
}
struct node * create_node()
{
        int num;
        newnode= (struct node*)malloc(sizeof(struct node));
  //printf("\nEnter data:");
  //scanf("%d",&num);
  newnode->data=0;
  newnode->next=NULL;
  return newnode;
}
void insert_beg()
{
  int num;
  newnode=create_node();

  printf("\nEnter data:");
  scanf("%d",&num);
  newnode->data=num;

  if(head==NULL)      //If list is empty
  {
    newnode->next=NULL;
    head=newnode;
  }
  else
  {
    newnode->next=head;
    head=newnode;
  }
}

void insert_end()
{
```

```
    int num;
    newnode=create_node();
    printf("Enter data:");
    scanf("%d",&num);
    newnode->data=num;
    newnode->next=NULL;

    if(head==NULL)      //If list is empty
    {
       head=newnode;
    }
    else
    {
      temp=head;
       while(temp->next!=NULL)
       temp=temp->next;
       temp->next=newnode;
    }
}

int insert_pos()
{
    int pos,i,num;
    if(head==NULL)
    {
       printf("List is empty!!");
       return 0;
    }

    newnode=create_node();
    printf("Enter data:");
    scanf("%d",&num);
    printf("Enter position to insert:");
    scanf("%d",&pos);
    newnode->data=num;

    temp=head;
    for(i=1;i<pos-1;i++)
    {
       if(temp->next==NULL)
       {
          printf("There are less elements!!");
          return 0;
       }
```

```
    temp=temp->next;
  }

  newnode->next=temp->next;
  temp->next=newnode;
  return 0;
}

void display()
{
  if(head==NULL)
  {
    printf("List is empty!!");return;
  }
  else
  {
    temp=head;
    printf("The linked list is:\n");
    while(temp!=NULL)
    {
      printf("%d->",temp->data);
      temp=temp->next;
    }
    printf("NULL");
  }
}

void delete_beg()
{
  if(head==NULL)
  {
    printf("The list is empty!!");return;
  }
  else
  {
    temp=head;
    head=head->next;
    printf("Deleted element is %d",temp->data);
    free(temp);
  }
}

void delete_end()
{
  if(head==NULL)
```

```
  {
    printf("The list is empty!!");return;
  }
  else if(head->next==NULL)
  {
        temp=head;
              printf("Deleted element is %d",temp->data);
        head=NULL;
        free(temp);
        }

        else
  {
    temp=head;
    while(temp->next!=NULL)
    {
        prev=temp;
        temp=temp->next;
              }
    prev->next=NULL;
    printf("Deleted element is %d",temp->data);
    free(temp);
  }
}

int delete_pos()
{
  int pos,i=1;
  printf("Enter position to delete:");
  scanf("%d",&pos);

  if(head==NULL)
  {
    printf("List is empty!!");
    return 0;
  }
  temp=head;
  if(i==pos)
  {
        printf("\nUse Delete at Begining");
              return;
        }
        else if(temp->next->next==NULL)
        {
              printf("\nUse Delete at Ending");
```

```
                return;
            }

            for(i=1;i<pos;i++)
    {
        if(temp->next==NULL)
        {
            printf("There are less elements!!");
            return 0;
        }
        prev=temp;
                temp=temp->next;
    }


    prev->next=temp->next;
    printf("Deleted element is %d",temp->data);
    free(temp);
    return 0;
}
```

OUTPUT:

```
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):1

Enter data:10

---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit
```

```
Enter your choice(1-4):2
The linked list is:
10->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):2
Enter data:100

---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
The linked list is:
10->100->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):3
Enter data:20
```

```
Enter position to insert:2

---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
The linked list is:
10->20->100->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):3
Enter data:30
Enter position to insert:3

---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
The linked list is:
10->20->30->100->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1
```

```
---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit

Enter your choice(1-4):2
Enter data:200

---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
The linked list is:
10->20->30->100->200->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)1
Deleted element is 10
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
The linked list is:
20->30->100->200->
---- Singly Linked List(SLL) Menu ----
1.Insert
```

```
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)2
Deleted element is 200
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
The linked list is:
20->30->100->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)3
Enter position to delete:2
Deleted element is 30
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
```

```
The linked list is:
20->100->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)2
Deleted element is 100
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
The linked list is:
20->
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position

Enter your choice(1-4:)1
Deleted element is 20
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
```

```
4.Exit

Enter your choice(1-4):2
List is empty!!
---- Singly Linked List(SLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):
```

Doubly Linked List program

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

struct node
{
   int data;
   struct node *prev;
          struct node *next;
};
struct node *head=NULL,*newnode=NULL,*temp,*temp1;


int main()
{
   int ch;
   void insert_beg();
   void insert_end();
   void insert_pos();
   void display();
   void delete_beg();
   void delete_end();
   void delete_pos();
          struct node * create_node();

   while(1)
   {
      printf("\n---- Doubly Linked List(DLL) Menu ----");
      printf("\n1.Insert\n2.Traverse OR Display\n3.Delete\n4.Exit\n");
      printf("\nEnter your choice(1-4):");
      scanf("%d",&ch);

      switch(ch)
```

```
    {
      case 1:
            printf("\n---- Insert Menu ----");
            printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specific position \n  ( position
value between start position and end position)\n4.Exit");
            printf("\nEnter your choice(1-4):");
                                          scanf("%d",&ch);

            switch(ch)
            {
              case 1: insert_beg();
                    break;
              case 2: insert_end();
                    break;
              case 3: insert_pos();
                    break;

              case 4: exit(0);
              default: printf("\nWrong Choice!!");
            }
            break;

      case 2: display();
            break;

      case 3: printf("\n---- Delete Menu ----");
            printf("\n1.Delete from beginning\n2.Delete from end\n3.Delete from specified
position\n4.Exit");
            printf("\nEnter your choice(1-4:)");
            scanf("%d",&ch);

            switch(ch)
            {
              case 1: delete_beg();
                    break;
              case 2: delete_end();
                    break;
              case 3: delete_pos();
                    break;
                                                case 4: exit(0);
              default: printf("\nWrong Choice!!");
            }
            break;
      case 4: exit(0);
            default: printf("\nWrong Choice!!");
```

```
      }
   }
   return 0;
}
struct node * create_node()
{
          int num;
          newnode= (struct node*)malloc(sizeof(struct node));
   newnode->data=0;
   newnode->next=NULL;
   return newnode;
}
void insert_beg()
{
   int num;
   newnode=create_node();
   printf("\nEnter data:");
   scanf("%d",&num);
   newnode->data=num;

   if(head==NULL)      //If list is empty
   {
      newnode->next=NULL;
      newnode->prev=NULL;
      head=newnode;
   }
   else
   {
      newnode->next=head;
      newnode->prev=head->prev;
      head->prev=newnode;
      head=newnode;
   }
}
void insert_end()
{
   int num;
   newnode=create_node();
   printf("Enter data:");
   scanf("%d",&num);
   newnode->data=num;
   //newnode->next=NULL;

   if(head==NULL)      //If list is empty
   {
```

```
    newnode->next=NULL;
    newnode->prev=NULL;
    head=newnode;
  }
  else
  {
        temp=head;
    while(temp->next!=NULL)
    temp=temp->next;

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

void insert_pos()
{
  int pos,i,num;
  if(head==NULL)
  {
    printf("List is empty!!!");
    return 0;
  }

  newnode=create_node();
  printf("Enter data:");
  scanf("%d",&num);
  printf("Enter position to insert:");
  scanf("%d",&pos);
  newnode->data=num;

  temp=head;

  for(i=1;i<pos;i++)
  {
    if(temp->next==NULL)
    {
      printf("There are less elements!!!");
      return 0;
    }
                temp1=temp;
    temp=temp->next;
  }
```

```
   temp1->next=newnode;
   newnode->prev=temp1;

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

}

void display()
{
   if(head==NULL)
   {
      printf("List is empty!!!");return;
   }
   else
   {
      temp=temp1=head;
      printf("\nThe linked list is LEFT TO RIGHT:");
      printf(" NULL<-");
                  while(temp!=NULL)
      {
         temp1=temp;
                        printf("%d->",temp->data);
         temp=temp->next;

      }
      printf(" NULL");

                  printf("\nThe linked list is RIGHT TO LEFT:");
      printf(" NULL<-");
                  while(temp1!=head)
      {
         printf("%d->",temp1->data);
         temp1=temp1->prev;
      }
      printf("%d->",temp1->data);
      printf(" NULL");
   }
}

void delete_beg()
{
   if(head==NULL)
   {
      printf("\nThe list is empty!!!");return;
```

```
  }
  else if(head->next==NULL)
  {
        temp=head;
        head=NULL;
              printf("\nThe list Contains one element that is deleted");
              free(temp);
              return;
        }
        else
  {
    temp=head;
    head=head->next;
    head->prev=NULL;
    printf("\nDeleted element is %d",temp->data);
    free(temp);
  }
}

void delete_end()
{
  if(head==NULL)
  {
    printf("\nThe list is empty!!!");return;
  }
  else if(head->next==NULL)
  {
        temp=head;
              printf("\nDeleted element is %d",temp->data);
        head=NULL;
        free(temp);
        }

        else
  {
    temp=head;
    while(temp->next!=NULL)
    {
        temp1=temp;
        temp=temp->next;
              }
    temp1->next=NULL;
    printf("\nDeleted element is %d",temp->data);
    free(temp);
  }
```

```
}

void delete_pos()
{
   int pos,i=1;
   printf("Enter position to delete:");
   scanf("%d",&pos);

   if(head==NULL)
   {
      printf("List is empty!!!");
      return 0;
   }
   temp=head;
   if(i==pos)
   {
         printf("\nUse Delete at Begining");
                  return;
         }
         else if(temp->next->next==NULL)
         {
                  printf("\nUse Delete at Ending");
                  return;
         }

         for(i=1;i<pos;i++)
   {
      if(temp->next==NULL)
      {
         printf("There are less elements!!!");
         return 0;
      }
      temp1=temp;
                  temp=temp->next;
   }


   temp1->next=temp->next;
   temp->next->prev=temp1;
   printf("Deleted element is %d",temp->data);
   free(temp);
   return 0;
}
OUTPUT:
---- Doubly Linked List(DLL) Menu ----
```

```
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specific position
  ( position value between start position and end position)
4.Exit
Enter your choice(1-4):2
Enter data:20

---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->20-> NULL
The linked list is RIGHT TO LEFT: NULL<-20->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specific position
  ( position value between start position and end position)
4.Exit
Enter your choice(1-4):3
Enter data:11
Enter position to insert:2

---- Doubly Linked List(DLL) Menu ----
```

```
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->11->20-> NULL
The linked list is RIGHT TO LEFT: NULL<-20->11->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specific position
  ( position value between start position and end position)
4.Exit
Enter your choice(1-4):3
Enter data:12
Enter position to insert:3

---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->11->12->20-> NULL
The linked list is RIGHT TO LEFT: NULL<-20->12->11->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1
```

```
---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specific position
 ( position value between start position and end position)
4.Exit
Enter your choice(1-4):2
Enter data:100

---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->11->12->20->100-> NULL
The linked list is RIGHT TO LEFT: NULL<-100->20->12->11->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specific position
 ( position value between start position and end position)
4.Exit
Enter your choice(1-4):1

Enter data:500

---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
```

```
The linked list is LEFT TO RIGHT: NULL<-500->10->11->12->20->100-> NULL
The linked list is RIGHT TO LEFT: NULL<-100->20->12->11->10->500-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
4.Exit
Enter your choice(1-4:)1

Deleted element is 500
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->11->12->20->100-> NULL
The linked list is RIGHT TO LEFT: NULL<-100->20->12->11->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
4.Exit
Enter your choice(1-4:)2

Deleted element is 100
---- Doubly Linked List(DLL) Menu ----
```

```
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->11->12->20-> NULL
The linked list is RIGHT TO LEFT: NULL<-20->12->11->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
4.Exit
Enter your choice(1-4:)3
Enter position to delete:3
Deleted element is 12
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->11->20-> NULL
The linked list is RIGHT TO LEFT: NULL<-20->11->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
```

```
2.Delete from end
3.Delete from specified position
4.Exit
Enter your choice(1-4:)3
Enter position to delete:2
Deleted element is 11
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-10->20-> NULL
The linked list is RIGHT TO LEFT: NULL<-20->10-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
4.Exit
Enter your choice(1-4:)1

Deleted element is 10
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT: NULL<-20-> NULL
The linked list is RIGHT TO LEFT: NULL<-20-> NULL
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
```

```
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
4.Exit
Enter your choice(1-4:)1

The list Contains one element that is deleted
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):2
List is empty!!!
---- Doubly Linked List(DLL) Menu ----
1.Insert
2.Traverse OR Display
3.Delete
4.Exit

Enter your choice(1-4):
```

// Circular Singly Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
}*head=NULL,*newnode,*temp,*prev;

int main()
{
   int ch;
   void insert_beg();
   void insert_end();
   void insert_pos();
   void display();
```

```
void delete_beg();
void delete_end();
void delete_pos();
        struct node * create_node();
while(1)
{
   printf("\n---- Circular Singly Linked List(CSLL) Menu ----");
   printf("\n1.Insert\n2.Display\n3.Delete\n4.Exit\n");
   printf("\nEnter your choice(1-4):");
   scanf("%d",&ch);

   switch(ch)
   {
      case 1:
            printf("\n---- Insert Menu ----");
            printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specified position\n4.Exit");
            printf("\nEnter your choice(1-4):");
                                          scanf("%d",&ch);

            switch(ch)
            {
               case 1: insert_beg();
                     break;
               case 2: insert_end();
                     break;
               case 3: insert_pos();
                     break;

               case 4: exit(0);
               default: printf("\nWrong Choice!!!");
            }
            break;

      case 2: display();
            break;

      case 3: printf("\n---- Delete Menu ----");
            printf("\n1.Delete from beginning\n2.Delete from end\n3.Delete from specified position");
            printf("\nEnter your choice(1-4:)");
            scanf("%d",&ch);

            switch(ch)
            {
               case 1: delete_beg();
                     break;
```

```
                case 2: delete_end();
                        break;
                case 3: delete_pos();
                        break;

                case 4: exit(0);
                default: printf("\nWrong Choice!!!");
              }
              break;
         case 4: exit(0);
              default: printf("\nWrong Choice!!!");
      }
   }
   return 0;
}

struct node * create_node()
{
           int num;
           newnode= (struct node*)malloc(sizeof(struct node));
   newnode->data=0;
   newnode->next=NULL;
   return newnode;
}
void insert_beg()
{
   int num;
   newnode=create_node();
   printf("\nEnter data:");
   scanf("%d",&num);
   newnode->data=num;

   if(head==NULL)       //If list is empty
   {
      head=newnode;
                  newnode->next=head;
   }
   else
   {
      temp=head;
      while(temp->next!=head)
           temp=temp->next;

                  newnode->next=head;
      temp->next=newnode;
```

```
      head=newnode;
   }
}
void insert_end()
{
   int num;
   newnode=create_node();
   printf("Enter data:");
   scanf("%d",&num);
   newnode->data=num;
   newnode->next=NULL;

   if(head==NULL)      //If list is empty
   {
      head=newnode;
                  newnode->next=head;
   }
   else
   {
         temp=head;
      while(temp->next!=head)
         temp=temp->next;

                  temp->next=newnode;
                  newnode->next=head;
   }
}

void insert_pos()
{
   int pos,i,num;
   if(head==NULL)
   {
      printf("List is empty!!");
      return 0;
   }

   newnode=create_node();
   printf("Enter data:");
   scanf("%d",&num);
   printf("Enter position to insert:");
   scanf("%d",&pos);
   newnode->data=num;

   temp=head;
```

```
   if(pos==1)
   {
          printf("\nUse insertion at Begining");
          return;
          }
          for(i=1;i<pos-1;i++)
   {
      if(temp->next==head)
      {
         printf("There are less elements!!");
         return ;
      }

      temp=temp->next;
   }

   newnode->next=temp->next;
   temp->next=newnode;
   return 0;
}

void display()
{
   struct node *temp=head;
          if(head==NULL)
   {
      printf("\nList is empty!!");return;
   }
   else
   {
      //temp=head;
      printf("\nThe linked list is:\nHEAD: ");

      while(temp->next != head)
      {

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

      }
      printf("%d -> :HEAD",temp->data);
   }
}

void delete_beg()
```

```
{
  struct node *temp=head;
        if(head==NULL)
  {
    printf("\nThe list is empty!!");return;
  }
  else if(temp->next==head)
  {
        head=NULL;
        printf("\nDeleted element is %d",temp->data);
        free(temp);
        }
        else
  {
    while(temp->next!=head)
        temp=temp->next;

                temp->next=head->next;
                printf("\nDeleted element is %d",head->data);
                free(head);
                head=temp->next;
                //free(temp);
  }
}

void delete_end()
{
  struct node *temp=head,*prev;
        if(head==NULL)
  {
    printf("The list is empty!!");return;
  }
  else if(head->next==head)
  {
        head=NULL;
        printf("\nDeleted element is %d",temp->data);
        free(temp);
        }

        else
  {
    while(temp->next!=head)
    {
        prev=temp;
        temp=temp->next;
```

```
                }
    prev->next=temp->next;
    printf("Deleted element is %d",temp->data);
    free(temp);
  }
}

void delete_pos()
{
  int pos,i=1;
  struct node *temp=head,*prev;
  printf("Enter position to delete:");
  scanf("%d",&pos);

  if(head==NULL)
  {
    printf("List is empty!!");
    return 0;
  }
  if(pos==1)
  {
        printf("\nUse Delete at Begining");
                return;
        }
        else if(temp->next->next==head)
        {
                printf("\nUse Delete at Ending");
                return;
        }

        for(i=1;i<pos;i++)
  {
    if(temp->next==head)
    {
      printf("There are less elements!!");
      return 0;
    }
    prev=temp;
                temp=temp->next;
  }


  prev->next=temp->next;
  printf("Deleted element is %d",temp->data);
  free(temp);
```

```
    return 0;
}
OUTPUT:

---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit
Enter your choice(1-4):1
---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):1
Enter data:10

---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):
2

The linked list is:
HEAD: 10 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):2
Enter data:50
```

```
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 10 ->50 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):3
Enter data:12
Enter position to insert:2

---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 10 ->12 ->50 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1
```

```
---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):1

Enter data:100

---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 100 ->10 ->12 ->50 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):2
Enter data:200

---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 100 ->10 ->12 ->50 ->200 -> :HEAD
```

```
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):3
Enter data:500
Enter position to insert:3

---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 100 ->10 ->500 ->12 ->50 ->200 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):3
Enter data:1000
Enter position to insert:6

---- Circular Singly Linked List(CSLL) Menu ----
```

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

Enter your choice(1-4):2

The linked list is:
HEAD: 100 ->10 ->500 ->12 ->50 ->1000 ->200 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)1

Deleted element is 100
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 10 ->500 ->12 ->50 ->1000 ->200 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
```

```
3.Delete from specified position
Enter your choice(1-4:)2
Deleted element is 200
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 10 ->500 ->12 ->50 ->1000 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)3
Enter position to delete:3
Deleted element is 12
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 10 ->500 ->50 ->1000 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3
```

```
---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)3
Enter position to delete:3
Deleted element is 50
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 10 ->500 ->1000 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)1

Deleted element is 10
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 500 ->1000 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
```

```
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)2
Deleted element is 1000
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is:
HEAD: 500 -> :HEAD
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)1

Deleted element is 500
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2
```

```
List is empty!!
---- Circular Singly Linked List(CSLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):
```

```
// Circular Doubly Linked List
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *prev;
         struct node *next;
};
struct node *head=NULL,*newnode=NULL,*temp,*temp1;


int main()
{
   int ch;
   void insert_beg();
   void insert_end();
   void insert_pos();
   void display();
   void delete_beg();
   void delete_end();
   void delete_pos();
         struct node * create_node();
   while(1)
   {
      printf("\n---- Circular Doubly Linked List(CDLL) Menu ----");
      printf("\n1.Insert\n2.Display\n3.Delete\n4.Exit\n");
      printf("\nEnter your choice(1-4):");
      scanf("%d",&ch);

      switch(ch)
      {
         case 1:
               printf("\n---- Insert Menu ----");
               printf("\n1.Insert at beginning\n2.Insert at end\n3.Insert at specified position\n4.Exit");
               printf("\nEnter your choice(1-4):");
```

```
                                        scanf("%d",&ch);

            switch(ch)
            {
               case 1: insert_beg();
                     break;
               case 2: insert_end();
                     break;
               case 3: insert_pos();
                     break;

               case 4: exit(0);
               default: printf("\nWrong Choice!!!");
            }
            break;

      case 2: display();
            break;

      case 3: printf("\n---- Delete Menu ----");
            printf("\n1.Delete from beginning\n2.Delete from end\n3.Delete from specified position");
            printf("\nEnter your choice(1-4:)");
            scanf("%d",&ch);

            switch(ch)
            {
               case 1: delete_beg();
                     break;
               case 2: delete_end();
                     break;
               case 3: delete_pos();
                     break;
                                                case 4: exit(0);
               default: printf("\nWrong Choice!!!");
            }
            break;
      case 4: exit(0);
            default: printf("\nWrong Choice!!!");
     }
   }
   return 0;
}
struct node * create_node()
{
         int num;
```

```
        newnode= (struct node*)malloc(sizeof(struct node));
   newnode->data=0;
   newnode->next=NULL;
   return newnode;
}
void insert_beg()
{
   int num;
   newnode=create_node();
   printf("\nEnter data:");
   scanf("%d",&num);
   newnode->data=num;

   if(head==NULL)      //If list is empty
   {
      head=newnode;
                  newnode->next=head;
      newnode->prev=head;
   }
   else
   {
      temp=head;
      while(temp->next!=head)
          temp=temp->next;

                  temp->next=newnode;
                  newnode->next=head;
      newnode->prev=temp;
      head->prev=newnode;
      head=newnode;
   }
}
void insert_end()
{
   int num;
   newnode=create_node();
   printf("\nEnter data:");
   scanf("%d",&num);
   newnode->data=num;
   //newnode->next=NULL;

   if(head==NULL)      //If list is empty
   {
      head=newnode;
                  newnode->next=head;
```

```
    newnode->prev=head;
  }
  else
  {
        temp=head;
    while(temp->next!=head)
        temp=temp->next;

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

void insert_pos()
{
  int pos,i,num;
  if(head==NULL)
  {
    printf("\nList is empty!!!");
    return 0;
  }

  newnode=create_node();
  printf("\nEnter data:");
  scanf("%d",&num);
  printf("\nEnter position to insert:");
  scanf("%d",&pos);
  newnode->data=num;

  temp=head;
  for(i=1;i<pos;i++)
  {
    if(temp->next==head)
    {
      printf("There are less elements!!!");
      return 0;
    }
                temp1=temp;
    temp=temp->next;
  }

  temp1->next=newnode;
  newnode->prev=temp1;
```

```
        newnode->next=temp;
   temp->prev=newnode;

}

void display()
{
   if(head==NULL)
   {
      printf("\nList is empty!!!");return;
   }
   else
   {
      temp=temp1=head;
      printf("\nThe linked list is LEFT TO RIGHT:\nHEAD: ");
      while(temp->next!=head)
      {
         temp1=temp;
                        printf("%d->",temp->data);
         temp=temp->next;

      }
      printf("%d-> :HEAD",temp->data);

   }
}

void delete_beg()
{
   if(head==NULL)
   {
      printf("\nThe list is empty!!!");return;
   }
   else if(head->next==head)
   {
         printf("\nDeleted element is %d",head->data);
               head=NULL;
         free(head);
         }
         else
   {
      temp=head;
      while(temp->next!=head)
         temp=temp->next;
```

```
    temp->next=head->next;
    head->next->prev=temp;
    printf("\nDeleted element is %d",head->data);
    free(head);
    head=temp->next;
  }
}

void delete_end()
{
  if(head==NULL)
  {
    printf("\nThe list is empty!!!");return;
  }
  else if(head->next==head)
  {
        temp=head;
              printf("\nDeleted element is %d",temp->data);
        head=NULL;
        free(temp);
        }

        else
  {
    temp=head;
    while(temp->next!=head)
    {
        temp1=temp;
        temp=temp->next;
              }
    temp1->next=head;//temp->prev->next=temp->next;
    head->prev=temp1;//temp->prev;
    printf("\nDeleted element is %d",temp->data);
    free(temp);
  }
}

void delete_pos()
{
  int pos,i=1;
  printf("Enter position to delete:");
  scanf("%d",&pos);

  if(head==NULL)
```

```
  {
    printf("\nList is empty!!!");
    return 0;
  }
  temp=head;
  if(pos==1)
  {
        printf("\nUse Delete at Begining");
              return;
        }
        else if(temp->next->next==head)
        {
              printf("\nUse Delete at Ending");
              return;
        }

        for(i=1;i<pos;i++)
  {
    if(temp->next==head)
    {
      printf("\nThere are less elements!!!");
      return 0;
    }
    temp1=temp;
              temp=temp->next;
  }


  temp1->next=temp->next;
  temp->next->prev=temp1;
  printf("\nDeleted element is %d",temp->data);
  free(temp);
  return 0;
}
```

OUTPUT:

```
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
```

```
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):1

Enter data:100

---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 100-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):2

Enter data:500

---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 100->500-> :HEAD
```

```
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):3

Enter data:110

Enter position to insert:2

---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 100->110->500-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):1

Enter data:1000
```

```
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 1000->100->110->500-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):1

---- Insert Menu ----
1.Insert at beginning
2.Insert at end
3.Insert at specified position
4.Exit
Enter your choice(1-4):2

Enter data:2000

---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 1000->100->110->500->2000-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3
```

```
---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)1

Deleted element is 1000
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 100->110->500->2000-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)2

Deleted element is 2000
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 100->110->500-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
```

```
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3
---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)1

Deleted element is 100
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

The linked list is LEFT TO RIGHT:
HEAD: 110->500-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)2

Deleted element is 500
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2
```

```
The linked list is LEFT TO RIGHT:
HEAD: 110-> :HEAD
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):3

---- Delete Menu ----
1.Delete from beginning
2.Delete from end
3.Delete from specified position
Enter your choice(1-4:)1

Deleted element is 110
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):2

List is empty!!!
---- Circular Doubly Linked List(CDLL) Menu ----
1.Insert
2.Display
3.Delete
4.Exit

Enter your choice(1-4):
```

6. AIM: Write a C Program to implement Stack using arrays

```
//stack by using static array
#include <stdio.h>
int stack[100],i,j,choice=0,n=0,top=-1;
int isStackFull();
void push();
int isStackEmpty();
void pop();
void peep();
void show();
int main ()
```

```
{
  printf("\n*********Stack Operations Using Static Array*********");
        printf("\n-----------------------------------------------");
  printf("\nEnter the size of the STACK:");
  scanf("%d",&n);

  do
  {
    printf("\n...Chose one from the below options...");
    printf("\n1.Push\n2.Pop\n3.Peep/Peek\n4.Show OR Display\n5.Exit");
    printf("\n Enter your choice:");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1:
      {
        push();
        break;
      }
      case 2:
      {
        pop();
        break;
      }
      case 3:
      {
        peep();
        break;
      }
      case 4:
      {
        show();
        break;
      }
      case 5:
      {
        printf("\nPlease Wait Exiting....");
        break;
      }
      default:
      {
        printf("\nPlease Enter valid choice ...  ");
      }
    };
  } while(choice != 5) ;
```

```
return 0;
}
 int isStackFull()
 {
    if(top == n - 1)// n is size of the array
       return 1;
    else return 0;
 }
void push ()
{
   int val;
   if(isStackFull())
       printf("\nSTACK Is Full !!!");
   else
   {
      printf("\nEnter the value?");
      scanf("%d",&val);
      top = top +1;
      stack[top] = val;
                  printf("\nElement is pushed in to the STACK");
   }
}
 int isStackEmpty()
 {
    if(top == -1)
       return 1;
    else return 0;
 }
void pop ()
{
   if(isStackEmpty())
       printf("\nStack Is Empty !!!");
   else
   {top = top -1;
          printf("\nElement is Popped from the STACK");
          }
}
void peep()
{
          if(isStackEmpty())
          printf("\nStack Is Empty !!!");
          else printf("\nTOP Index =%d and Value=%d",top,stack[top]);
}
void show()
{  if(isStackEmpty())
```

```
        printf("\nStack Is Empty !!!");
   else
   {
                printf("\nStack Contains: ");
          for (i=top;i>=0;i--)
             printf("\n%5d",stack[i]);

        }
}
```

OUTPUT:

```
*********Stack Operations Using Static Array*********
-----------------------------------------------
Enter the size of the STACK:5

...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:1

Enter the value?100

Element is pushed in to the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:1
```

```
Enter the value?200

Element is pushed in to the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  200
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:1

Enter the value?300

Element is pushed in to the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  300
  200
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:1
```

```
Enter the value?400

Element is pushed in to the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:1

Enter the value?500

Element is pushed in to the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  500
  400
  300
  200
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:1

STACK Is Full !!!
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:600
```

```
Please Enter valid choice ...
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  500
  400
  300
  200
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:1
STACK Is Full !!!
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:2
Element is Popped from the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  400
  300
  200
  100
...Chose one from the below options...
```

```
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:2

Element is Popped from the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  300
  200
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:2

Element is Popped from the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  200
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
```

```
5.Exit
 Enter your choice:2

Element is Popped from the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Contains:
  100
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:2

Element is Popped from the STACK
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:4

Stack Is Empty !!!
...Chose one from the below options...
1.Push
2.Pop
3.Peep/Peek
4.Show OR Display
5.Exit
 Enter your choice:
```

7. AIM: Write a C Program to implement Stack using Linked Lists

```
// STACK by using Linked Lists
#include<stdio.h>
#include<stdlib.h>

struct Node
```

```
{
  int data;
  struct Node *next;
}*top = NULL;

void push(int);
void pop();
void display();
void peep();

int main()
{
  int choice, value;

  printf("\n... Stack using Linked List ...");
  while(1){
    printf("\n****** MENU ******");
    printf("\n1. Push\n2. Pop\n3. Peep / Peak\n4. Display OR Travese\n5. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
          case 1: printf("\nEnter the value to be insert: ");
                  scanf("%d", &value);
                  push(value);
                  break;
          case 2: pop(); break;
          case 3: peep(); break;
          case 4: display();break;
          case 5: exit(0);
          default: printf("\nWrong selection!!! Please try again!!!");
    }
  }
return 0;
}
void push(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  if(newNode==NULL)
  {
                  printf("\n Insufficient Memory ...");
                  exit(1);
  }
  else
  {
```

```
                newNode->data = value;
            if(top == NULL)
              newNode->next = NULL;
            else
              newNode->next = top;
            top = newNode;
            printf("\nElement is pushed in to the STACK !!!\n");
  }

}
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\nDeleted element is: %d", temp->data);
    top = temp->next;
    free(temp);
  }
}
void peep()
{
          if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else printf("\n Topmost( PEEP/PEAK) Element in the Stack is: %d",top->data);
}
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\n Stack contains:\n");
          while(temp->next != NULL)
          {
                printf("%d--->",temp->data);
                temp = temp -> next;
    }
    printf("%d--->NULL",temp->data);
  }
}
```

OUTPUT:

```
... Stack using Linked List ...
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 1

Enter the value to be insert: 100

Element is pushed in to the STACK !!!

****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 4

 Stack contains:
100--->NULL
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 1

Enter the value to be insert: 200

Element is pushed in to the STACK !!!

****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 4

 Stack contains:
200--->100--->NULL
```

```
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 1

Enter the value to be insert: 300

Element is pushed in to the STACK !!!

****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 4

 Stack contains:
300--->200--->100--->NULL
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 2

Deleted element is: 300
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 4

 Stack contains:
200--->100--->NULL
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
```

```
4. Display OR Travese
5. Exit
Enter your choice: 2

Deleted element is: 200
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 4

 Stack contains:
100--->NULL
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 3

 Topmost( PEEP/PEAK) Element in the Stack is: 100
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 4

 Stack contains:
100--->NULL
****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 2

Deleted element is: 100
****** MENU ******
1. Push
```

```
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice: 4

Stack is Empty!!!

****** MENU ******
1. Push
2. Pop
3. Peep / Peak
4. Display OR Travese
5. Exit
Enter your choice:
```

8. AIM: Queue implementation using arrays. Implement different forms of queue. While implementing you should be able to store elements equal to the size of the queue. No positions should be left blank.

```
// Queue implementation by using array
#include<stdio.h>
#include<conio.h>
#define SIZE 5

int front=-1;
int rear=-1;
int q[SIZE];

void enqueue();
void dequeue();
void peep();
void display();
int isFull();
int isEmpty();

void main()
{
        int choice;
do
        {
                printf("\n***************************************");
                printf("\n Linear Queue By Using Array");
                printf("\n***************************************");
                printf("\n 1. Enqueue");
                printf("\n 2. Dequeue");
                printf("\n 3. Peep / Peek");
```

```
                printf("\n 4. Display ");
                printf("\n 5. Exit");

                printf("\n Enter Your Choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1:
                                enqueue();
                                //display();
                                break;
                        case 2:
                                 dequeue();
                                 //display();
                                 break;
                        case 3:
                                 peep();
                                 break;
                        case 4:
                                display();
                                break;
                        case 5:
                                printf("End of Program....!!!!");
                                exit(1);
                        default: printf("\nWrong Choice Please Try Again.....");
                }
        }while(choice!=5);
}
int isFull()
{
        if(rear>=SIZE-1)
        return 1;
        else return 0;
}
void enqueue()
{
        int value;

        if(isFull())
        {
                printf("\n Queue overflow");
        }
        if(rear < SIZE-1)
        {
```

```
                printf("\n Enter Value to be inserted in to Queue:");
                scanf("%d",&value);

                q[++rear]=value;
                if(front==-1)
                front=0;// front=front+1;
        }

}
int isEmpty()
{
        if(front==-1)
        return 1;
        else return 0;
}
void dequeue()
{
        if(isEmpty())
        {
                printf("\nQueue Underflow");
                return;
        }
        else
        printf("\nDeleted Item:-->%d\n",q[front]);

        if(front==rear)
        front=rear=-1;
        else
        front=front+1;
}

void peep()
{
        if(isEmpty())
        {
                printf("\nQueue is empty....");
                return;
        }
        else printf("\nPeep Element in Queue is %d",q[front]);
 }
void display()
{
        int i;
        if(isEmpty())
        {
```

```
                printf("\nQueue is empty....");
                return;
        }
        printf("\nLinear Queue: FRONT");
        for(i=front;i<=rear;i++)
                printf("<--%d",q[i]);
        printf("<--REAR");
}
```

OUTPUT:

```
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:1

 Enter Value to be inserted in to Queue:100

**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Linear Queue: FRONT<--100<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:1

 Enter Value to be inserted in to Queue:200
```

```
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Linear Queue: FRONT<--100<--200<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:1

 Enter Value to be inserted in to Queue:300

**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Linear Queue: FRONT<--100<--200<--300<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:1

 Enter Value to be inserted in to Queue:400
```

```
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Linear Queue: FRONT<--100<--200<--300<--400<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:1

 Enter Value to be inserted in to Queue:500

**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Linear Queue: FRONT<--100<--200<--300<--400<--500<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:1
```

```
Queue overflow
**************************************
Linear Queue By Using Array
**************************************
1. Enqueue
2. Dequeue
3. Peep / Peek
4. Display
5. Exit
Enter Your Choice:2

Deleted Item:-->100

**************************************
Linear Queue By Using Array
**************************************
1. Enqueue
2. Dequeue
3. Peep / Peek
4. Display
5. Exit
Enter Your Choice:4

Linear Queue: FRONT<--200<--300<--400<--500<--REAR
**************************************
Linear Queue By Using Array
**************************************
1. Enqueue
2. Dequeue
3. Peep / Peek
4. Display
5. Exit
Enter Your Choice:2

Deleted Item:-->200

**************************************
Linear Queue By Using Array
**************************************
1. Enqueue
2. Dequeue
3. Peep / Peek
4. Display
5. Exit
Enter Your Choice:4
```

```
Linear Queue: FRONT<--300<--400<--500<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:2

Deleted Item:-->300

**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Linear Queue: FRONT<--400<--500<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:3

Peep Element in Queue is 400
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:2
```

```
Deleted Item:-->400

**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Linear Queue: FRONT<--500<--REAR
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:2

Deleted Item:-->500

**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
 Enter Your Choice:4

Queue is empty....
**************************************
 Linear Queue By Using Array
**************************************
 1. Enqueue
 2. Dequeue
 3. Peep / Peek
 4. Display
 5. Exit
```

Enter Your Choice:

9. AIM: Write a C Program to implement Queue using linked lists

```
// A C program to demonstrate linked list based implementation of queue
#include <stdio.h>
#include <stdlib.h>

// A linked list (LL) node to store a queue entry
struct Node {
        int key;
        struct Node* next;
};

// The queue, front stores the front node of LL and rear stores the
// last node of LL
struct Queue {
        int count;
        struct Node *front, *rear;
};

// A utility function to create a new linked list node.
struct Node* newNode()
{int k;
        struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
        if(temp==NULL)
  {
        printf("\nQueue is FUll Because Memory is not Sufficient");
        return;
        }
        else
        {
                printf("\nEnter value to be inserted in to the Linear Queue");
                scanf("%d",&k);
                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;
        q->count=0;
        return q;
```

```
}
int isEmpty(struct Queue* q)
{
        if(q->front==NULL && q->rear==NULL)
        return 1;
        else return 0;
}
// The function to add a key k to q
void enQueue(struct Queue* q)
{
        struct Node* temp = newNode();

                // If queue is empty, then new node is front and rear both
                if(isEmpty(q))//if (q->rear == NULL) {
                {
                        q->front = q->rear = temp;
                        q->count++;
                        return;
                }

                // Add the new node at the end of queue and change rear
                q->rear->next = temp;
                q->rear = temp;
                q->count++;
}
// Function to remove a key from given queue q
void deQueue(struct Queue* q)
{
        // If queue is empty, return NULL.
        if (q->front == NULL)
        {
                printf("\nLinear Queue is Empty");
                return;
        }

        // Store previous front and move front one node ahead
        struct Node* 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;
        q->count--;
        free(temp);
```

```
}
void peep(struct Queue* q)
{
   if (q->front==NULL)
   {
      printf("\nLinear Queue is Empty");
                  return;
   }
   else
      printf("\nThe front element is %d ", q->front->key);
}

// returns number of entries and displays the elements in queue
void display(struct Queue* q)
{
   struct Node *temp;
   temp = q->front;

   if (q->front==NULL)
   {
            printf("\nQueue is Empty \n");
                  return;
          }
          else
          {
                  printf("\nLinear Queue: FRONT");
                  while (temp)
            {
               printf("<-- %d  ", temp->key);
               temp = temp->next;

            }
            printf("<--REAR");
                  printf("\nSize of Linear Queue is %d", q->count);

          }
}
int main()
{
          int choice, value;
          struct Queue* q;
   while(1)
   {
      printf("\n*************************************");
                  printf("\nLinear Queue Using Singly Linked List");
```

```
    printf("\n**************************************");
                printf("\n1 : create an empty queue \n2 : Enqueue");
    printf("\n3 : Dequeue\n4 : isEmpty");
    printf("\n5 : Peep\n6 : Display\n7 : Exit");
    printf("\nEnter your choice:");
                scanf("%d", &choice);
    switch (choice)    // menu driven program
    {
    case 1:
      q = createQueue();
                        printf("\nEmpty queue is created");
      break;
    case 2:
      enQueue(q);
      display(q);
      break;
    case 3:
      deQueue(q);
      display(q);
      break;
    case 4:
      if(isEmpty(q))
      printf("\nLinear Queue is Empty");
      else printf("\n Linear Queue is not an Empty");
      break;
    case 5:
      peep(q);
      break;
    case 6:
      display(q);
      break;
    case 7:
      exit(0);
    default:
      printf("wrong choice\n");
      break;
    }
  }
}
```

OUTPUT:

```
************************************
Linear Queue Using Singly Linked List
************************************
1 : create an empty queue
```

```
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:1

Empty queue is created
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:2

Enter value to be inserted in to the Linear Queue100

Linear Queue: FRONT<-- 100  <--REAR
Size of Linear Queue is 1
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:2

Enter value to be inserted in to the Linear Queue200

Linear Queue: FRONT<-- 100  <-- 200  <--REAR
Size of Linear Queue is 2
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
```

```
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:2

Enter value to be inserted in to the Linear Queue300

Linear Queue: FRONT<-- 100  <-- 200  <-- 300  <--REAR
Size of Linear Queue is 3
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:5

The front element is 100
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:4

 Linear Queue is not an Empty
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
```

```
5 : Peep
6 : Display
7 : Exit
Enter your choice:3

Linear Queue: FRONT<-- 200  <-- 300  <--REAR
Size of Linear Queue is 2
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:3

Linear Queue: FRONT<-- 300  <--REAR
Size of Linear Queue is 1
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:3

Queue is Empty

**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
```

```
Enter your choice:4

Linear Queue is Empty
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:5

Linear Queue is Empty
**************************************
Linear Queue Using Singly Linked List
**************************************
1 : create an empty queue
2 : Enqueue
3 : Dequeue
4 : isEmpty
5 : Peep
6 : Display
7 : Exit
Enter your choice:
```